Российская академия наук
Суперкомпьютерный консорциум университетов России

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ (ПаВТ'2012)

Труды международной научной конференции

г. Новосибирск, 26 – 30 марта 2012 г.

Челябинск,
Издательский центр ЮУрГУ
2012

УДК 004.75
П 18

Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции (Новосибирск, 26 – 30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. 774 с.

ISBN 978-5-696-04237-4

Данный сборник содержит статьи, включенные в программу Международной научной конференции «Параллельные вычислительные технологии». Конференция проводится с 26 по 30 марта 2012 года. Подробную информацию о конференции можно найти в сети Интернет по адресу `http://agora.guru.ru/pavt`.

Отпечатано с авторских оригиналов.

Одобрено Советом факультета Вычислительной математики и информатики ЮУрГУ

Рецензенты:
В.В. Воеводин, член-корреспондент РАН,
В.И. Ухоботов, доктор физ.-мат. наук

Ответственные за выпуск:
Л.Б. Соколинский, доктор физ.-мат. наук,
К.С. Пан

Конференция проводится при поддержке
Российского фонда фундаментальных исследований

ISBN 978-5-696-04237-4

НАБЛЮДАТЕЛЬНЫЙ СОВЕТ

Бердышев В.И., академик РАН, ИММ УрО РАН, г. Екатеринбург
Ершов Ю.Л., академик РАН, председатель ОУС по математике и информатике, г. Новосибирск
Марчук Г.И., академик РАН, почетный директор ИВМ РАН, г. Москва
Михайленко Б.Г., академик РАН, директор ИВМиМГ СО РАН, г. Новосибирск
Моисеев Е.И., академик РАН, декан факультета ВМК МГУ, г. Москва
Савин Г.И., академик РАН, директор МСЦ РАН, г. Москва
Садовничий В.А., ректор МГУ, академик, вице-президент РАН, г. Москва
Четверушкин Б.Н., академик РАН, ИПМ РАН, г. Москва
Шокин Ю.И., академик РАН, директор ИВТ СО РАН, г. Новосибирск

ПРОГРАММНЫЙ КОМИТЕТ

Руководитель серии Международных суперкомпьютерных конференций в России:
Садовничий В.А., ректор МГУ, академик, вице-президент РАН

Председатель программного комитета:
Воеводин В.В., чл.-корр. РАН, НИВЦ МГУ, г. Москва

Сопредседатель программного комитета:
Соколинский Л.Б., д.ф.-м.н., НИУ ЮУрГУ, г. Челябинск

Ученый секретарь программного комитета:
Цымблер М.Л., к.ф.-м.н., НИУ ЮУрГУ, г. Челябинск

Члены программного комитета:
Абламейко С.В., чл.-корр. НАН РБ, ОИПИ НАН РБ, г. Минск
Акимова Е.Н., д.ф.-м.н., ИММ УрО РАН, г. Екатеринбург
Афанасьев А.П., д.ф.-м.н., ИСА РАН, г. Москва
Болдырев Ю.Я., д.т.н., НИУ СПбГПУ, г. Санкт-Петербург
Газизов Р.К, д.ф.-м.н., УГАТУ, г. Уфа
Гергель В.П., д.т.н., НИУ ННГУ, г. Нижний Новгород
Глинский Б.М., д.т.н., ИВМиМГ СО РАН, г. Новосибирск
Горячев В.Д., д.т.н., ТГТУ, г. Тверь
Гузев М.А., чл.-корр. РАН, ДВО РАН, г. Владивосток
Донгарра Дж. (J. Dongarra, University of Tennessee), США
Ильин В.П., д.ф.-м.н., ИВМиМГ СО РАН, г. Новосибирск
Лыкосов В.Н., чл.-корр. РАН, ИВМ РАН, г. Москва
Малышкин В.Э., д.т.н., ИВМиМГ СО РАН, г. Новосибирск
Мейер Х. (H. Meuer, ISC General Chair), Германия
Модорский В.Я., д.т.н., НИУ ПГТУ, г. Пермь
Немухин А.В., д.х.н., МГУ, г. Москва
Панюков А.В., д.ф.-м.н., НИУ ЮУрГУ, г. Челябинск
Попов Л.Д., д.ф.-м.н., ИММ УрО РАН, г. Екатеринбург
Самофалов В.В., к.т.н., Intel
Ситоле Х. (H. Sithole, Director of CHPC), ЮАР
Старченко А.В., д.ф.-м.н., НИУ ТГУ, г. Томск
Турлапов В.Е., д.т.н., НИУ ННГУ, г. Нижний Новгород
Якобовский М.В., д.ф.-м.н., ИММ РАН, г. Москва

ОРГАНИЗАЦИОННЫЙ КОМИТЕТ

Председатель организационного комитета:
Михайленко Б.Г., академик РАН, директор ИВМиМГ СО РАН, г. Новосибирск

Зам. председателя организационного комитета:
Глинский Б.М., исполнит. директор ЦКП ССКЦ ИВМиМГ СО РАН, г. Новосибирск
Лаврентьев М.М., проректор НГУ, г. Новосибирск

Члены организационного комитета:
Антонов А.С., с.н.с. НИВЦ МГУ, г. Москва
Брызгалов П.А., н.с. НИВЦ МГУ, г. Москва
Быринова Р.А., зам. главного бухгалтера ИВМиМГ СО РАН, г. Новосибирск
Воеводин Вад.В., н.с. НИВЦ МГУ, г. Москва
Воеводин Вл.В., зам. директора НИВЦ МГУ, г. Москва
Зернова Л.В., вед. программист ИВМиМГ СО РАН, г. Новосибирск
Иванова И.Н., вед. программист ИВМиМГ СО РАН, г. Новосибирск
Ильин В.П., г.н.с. ИВМиМГ СО РАН, г. Новосибирск
Косова С.Н., вед. инженер ИВМиМГ СО РАН, г. Новосибирск
Котелевский С.П., гл. специалист ИВМиМГ СО РАН, г. Новосибирск
Куликов И.М., н.с. ИВМиМГ СО РАН, г. Новосибирск
Кучин Н.В., гл. специалист ИВМиМГ СО РАН, г. Новосибирск
Лазарева Г.Г., н.с. ИВМиМГ СО РАН, г. Новосибирск
Марченко М.А., уч. секретарь ИВМиМГ СО РАН, г. Новосибирск
Пан К.С., программист кафедры системного программирования НИУ ЮУрГУ, г. Челябинск
Репина К.В., программист кафедры системного программирования НИУ ЮУрГУ, г. Челябинск
Соболев С.И., н.с. НИВЦ МГУ, г. Москва
Соколинский Л.Б., декан факультета Вычислительной математики и информатики НИУ ЮУрГУ, г. Челябинск
Усов А.Г., вед. программист ИВМиМГ СО РАН, г. Новосибирск
Уткина Л.И., ассистент кафедры системного программирования НИУ ЮУрГУ, г. Челябинск
Цымблер М.Л., доцент кафедры системного программирования НИУ ЮУрГУ, г. Челябинск
Черных И.Г., н.с. ИВМиМГ СО РАН, г. Новосибирск

ЭКСПЕРТЫ ПаВТ

Авербух В.Л., ИММ УрО РАН, г. Екатеринбург
Адинец А.В., НИВЦ МГУ, г. Москва
Аксенов А.А., ТЕСИС, г. Москва
Аксенова Е.В., ЮУрГУ, г. Челябинск
Антонов А.С., НИВЦ МГУ, г. Москва
Баландин Д.В., ННГУ, г. Нижний Новгород
Бандман О.Л., ИВМиМГ СО РАН, г. Новосибирск
Баркалов К.А., ННГУ, г. Нижний Новгород
Бородулин К.В., ЮУрГУ, г. Челябинск
Варламов Д.А., ИПХФ РАН, г. Черноголовка
Водопьянов В.В., УГАТУ, г. Уфа
Волохов В.М., ИПХФ РАН, г. Черноголовка
Вшивков В.А., ИВМиМГ СО РАН, г. Новосибирск
Городняя Л.В., ИВМиМГ СО РАН, г. Новосибирск
Губайдуллин И.М., ИНиК РАН, г. Уфа
Долганина Н.Ю., ЮУрГУ, г. Челябинск
Дорохов В.А., ЮУрГУ, г. Челябинск
Жуматий С.А., НИВЦ МГУ, г. Москва
Золотых Н.Ю., ННГУ, г. Нижний Новгород
Кетков Ю.Л., ННГУ, г. Нижний Новгород
Козинов Е.А., ННГУ, г. Нижний Новгород
Корж О.В., МГУ, г. Москва
Коротченко А.Г., ННГУ, г. Нижний Новгород
Костенецкий П.С., ЮУрГУ, г. Челябинск
Крюков В.А., ИПМ РАН, г. Москва
Линд Ю.Б., БашНИПИнефть, г. Уфа
Линёв А.Н., ННГУ, г. Нижний Новгород
Лукащук С.Ю., УГАТУ, г. Уфа
Лымарь Т.Ю., ЮУрГУ, г. Челябинск
Марчевский И.К., МГТУ, г. Москва
Медведев А.А., ЮУрГУ, г. Челябинск
Мееров И.Б., ННГУ, г. Нижний Новгород
Миниахметов Р.М., ЮУрГУ, г. Челябинск
Михайленко К.И., УГАТУ, г. Уфа
Мортиков Е.В., НИВЦ МГУ, г. Москва
Оленёв Н.Н., ВЦ РАН, г. Москва
Осипов Г.В., ННГУ, г. Нижний Новгород
Пан К.С., ЮУрГУ, г. Челябинск

Панюкова Т.А., ЮУрГУ, г. Челябинск
Пивушков А.В., ИПХФ РАН, г. Черноголовка
Половинкин А.Н., ННГУ, г. Нижний Новгород
Посыпкин М.А., ИСА РАН, г. Москва
Прокопьева Л.Ю., ИВТ СО РАН, г. Новосибирск
Радченко Г.И., ЮУрГУ, г. Челябинск
Репина К.В., ЮУрГУ, г. Челябинск
Романов С.Ю., НИВЦ МГУ, г. Москва
Сальников А.Н., ВМК МГУ, г. Москва
Сафина Ю.Н., ЮУрГУ, г. Челябинск
Свешников В.М., ИВМиМГ СО РАН, г. Новосибирск
Сенин А.В., ННГУ, г. Нижний Новгород
Соболев С.И., НИВЦ МГУ, г. Москва
Степаненко В.М., НИВЦ МГУ, г. Москва
Сухорослов О.В., ИСА РАН, г. Москва
Уткина Л.И., ЮУрГУ, г. Челябинск
Файзуллин Р.Т., ОмГТУ, г. Омск
Федорук М.П., ИВТ СО РАН, г. Новосибирск
Федянина Р.С., ЮУрГУ, г. Челябинск
Харченко С.А., ТЕСИС, г. Москва
Худякова Е.С., ЮУрГУ, г. Челябинск
Чеверда В.А., ИВМиМГ СО РАН, г. Новосибирск
Черноусов А.А., УГАТУ, г. Уфа
Шамакина А.В., ЮУрГУ, г. Челябинск
Шишков А.В., ННГУ, г. Нижний Новгород

СПОНСОРЫ

Платиновые спонсоры:
Корпорация Intel
Группа компаний РСК
Группа компаний Т-Платформы

Золотые спонсоры:
Компания ТЕСИС
Компания СофтПоинт
Корпорация IBM

Серебряные спонсоры:
Корпорация NVIDIA
Корпорация AMD
Корпорация Hewlett Packard
Компания CADFEM

ИНФОРМАЦИОННАЯ ПОДДЕРЖКА

Информационно-аналитический центр Parallel.ru
Газета «Поиск»
Журнал «CNews»
Журнал «CAD/CAM/CAE Observer»
Журнал «Rational Enterprise Management»
Журнал «Суперкомпьютеры»
Журнал «Вычисления в геологии»

Intel Direct Sparse Solver for Clusters, a research project for solving large sparse systems of linear algebraic equations

A. Kalinkin, K. Arturov

Intel Corporation

This research covers the Intel® Direct Sparse Solver for Clusters, the software that implements a direct method for solving the Ax=b equation with sparse symmetric matrix A on a cluster. This method, researched by Intel, is based on Cholesky decomposition. To achieve an efficient work balance on a large number of processes, the so-called "multifrontal" approach to Cholesky decomposition is implemented. This software implements parallelization that is based on nodes of the dependency tree and uses MPI, as well as parallelization inside a node of the tree that used OpenMP directives. The article provides a high-level description of the algorithm to distribute the work between both computational nodes and cores within a single node, as well as between different computational nodes. A series of experiments proves that this implementation causes no growth of the computational time and decreases the amount of memory needed for the computations.

1. Introduction

The paper describes a direct method based on Cholesky decomposition for solving the equation Ax=b with sparse symmetric matrix A. The positive-definite matrix A can be represented in terms of LL^T decomposition, in case of an indefinite matrix the decomposition is LDL^T, where the diagonal matrix D can be amended with extra "penalty" for additional stability of the decomposition. To achieve an efficient work balance on a large number of processes, the so-called "multifrontal" approach to Cholesky decomposition is proposed for the original matrix.

The multifrontal approach was proposed in the papers [1-7]. The decomposition algorithm implementation consists of several stages/ The initial matrix is subject to a reordering procedure [8-11] to represent it in the form of a dependency tree. Then the symbolic factorization takes place where the total number of nonzero elements is computed in LL^T. Then a factorization of the permuted matrix in the LL^T form takes place [2-3, 5].

This work is devoted to Intel® Direct Sparse Solver for Clusters package. This package implements parallelization based on nodes of the dependency tree using MPI as well as parallelization inside the node of tree using OpenMP directives. A variant of MPI-parallelization of Cholesky decomposition can be found in [1, 5]. However, as it will be shown later, such an algorithm becomes poorly scaled both in computational time and in the amount of memory used by each process if the total number of processes increases. To avoid scalability issues, we propose an algorithm, where one "tree" node is distributed among several computational nodes, and data transfers are interleaved with the computations preventing the growth of the computational time and reducing the amount of memory required for each process. This paper describes this algorithm.

The paper is organized as follows: Section 2 provides a brief description of the reordering step and describes parallel algorithms to distribute the work between both computational nodes and cores within a single node. Section 3 briefly describes the algorithm distributing a tree node between different computational nodes. Section 4 demonstrates on a series of experiments that this implementation does not result in the growth of the computational time and decreases the amount of memory needed for a process.

.

2. Main definitions and algorithms

In a general case, the algorithm of Cholesky decomposition can be presented in the following way:

Algorithm 0, Cholesky decomposition

```
L = A

for j = 1,size_of_matrix

 { for j = 1,i

  {L(i,j) = L(i,j)-L(i,k)L(k,j), k = 1,j-1

  if (i==j) L(i,j) = sqrt(L(i,j)

  if (i>j)  L(i,j) = L(i,j)/L(j,j)

  }

}
```

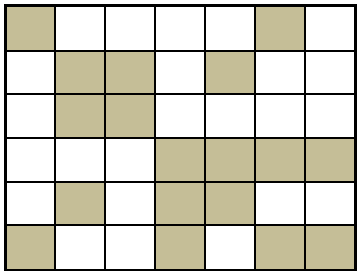


Figure 1. Original matrix non-zero pattern

Here A is a symmetric, positive-define matrix and L is the resulted lower-triangular matrix. If the initial matrix has a lot of zero elements (such kind of a matrix is called sparse), this algorithm can be rewritten in a more efficient way called multifrontal approach.

Suppose we have a sparse symmetric matrix A (Figure 1), where each grey block is in turn a sparse sub-matrix and each white one is a zero matrix. Using reordering algorithm procedures [12], this matrix can be reduced to the pattern as in Figure 2. A reordered matrix is essentially more convenient for computations than the initial one since Cholesky decomposition can start simultaneously from several entry points (for the matrix from Figure 2, 1st, 2nd, 4th and 5th rows of the matrix L can be calculated independently. For the original matrix from Figure 1, two rows only, namely, 1st and 4th, can be calculated independently. So, the reordering can provide an advantage for the algorithm implementation on parallel machines. While proceeding with Cholesky decomposition, the

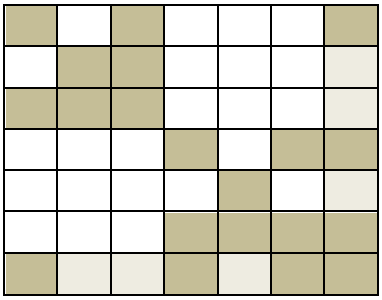


Figure 2. Non-zero pattern of the original matrix after reordering

non-zero pattern of the upper and lower triangular matrices in the final decomposition do not agree with the pattern of the original matrix, that is, additional non-zero blocks may appear (depicted as light-grey squares in the Figure 2). To be precise, a non-zero pattern of the matrix L in Cholesky decomposition is calculated at the symbolic factorization step before the main factorization step (stage). At this stage, we know the structure of the original matrix A after the reordering step and can calculate the non-zero pattern of the matrix L. At the same stage, the original matrix A stored in the sparse format is appended with zeros so that its non-zero pattern matches completely that of the matrix L. Henceforth, we will not distinguish the non-zero patterns of the matrices A and L. Moreover, it should be mentioned here that elements of the matrix L in the rows 3 and 6 can be computed only after the respective elements in the rows 1, 2 and 4, 5 are computed. The elements in the 7th row can be computed at last. This allows us to construct the dependency tree [1-2, 5-6] - a graph where each node corresponds to a single row of the matrix and each graph node can be computed only if its "children" (nodes on which it depends) are computed. The dependency tree for the matrix is given in the Figure 3a (the number in the center of a node shows the respective row number). For our example, an optimal distribution of the nodes between the computational processes is given in the Figure 3b, where the first bottom level of the nodes belongs to the processes with the numbers 0, 1, 2, …, the second level belongs to 0, 2, 4, …, the third belongs to 0, 4, 8, ..., etc.

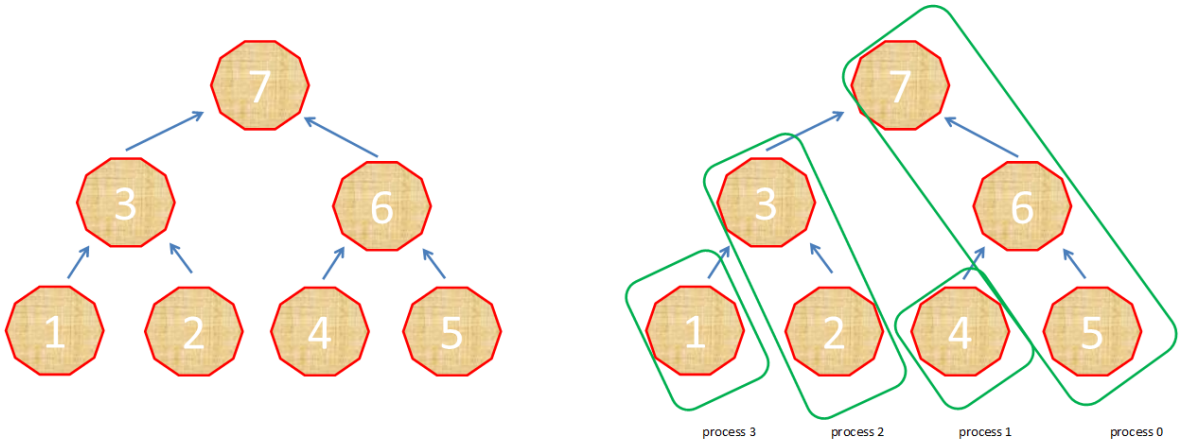


Figure 3a-b. Dependency tree sample distribution among processes

To compute elements of LL^T decomposition within a node Z of the dependency tree, it is necessary to compute the elements of LL^T decomposition in the nodes on which Z depends (its sub-trees). We call an update procedure to bring already computed elements to the node Z (actually, this procedure takes the elements from the sub-trees of Z multiplied by themselves and subtracted from elements of Z, so the main problem here is in a different non-zero pattern of Z and its sub-trees. The 4th line of the Algorithm 0 completes this operation with different L(i,k) being placed into different nodes of the tree, the last step is to calculate LL^T decomposition of elements placed into node Z. Algorithm 1 describes an implementation of the above mentioned piece of the global decomposition in terms of a pseudo-code:

Algorithm 1. Tree-parallelization in Cholesky decomposition

```
for current_level = 1, maximal_level

 {Z = node number that will be updated by this process

  for nodes of the tree with level smaller than the current_level

     {prepare an update of Z by multiplying elements of  LL^T  decomposition
elements lying within the current node by themselves}

 send own part of update of Z from each process to process which stores Z

 on process that stores Z compute LLT  elements of Z}
```

Consider in details an implementation of the algorithms called “compute elements of Z” and “prepare an update”.

Each tree node in turn can be represented as a sub-tree or as a square symmetric matrix. To use Algorithm 1, one needs to implement LL^T decomposition of each node on a single computational process. To compute Cholesky decomposition of each node of the tree, a similar algorithm could be implemented on a single computational process with several threads. If the number of threads is equal to the number of nodes of the tree on the bottom level, then LL^T factors for each node from the very bottom level of the tree are calculated by a single thread, on the next level each node can be calculated by 2 threads, on the next one by 4 threads, etc. However, such an algorithm becomes inefficient for the top nodes of the initial tree. Instead, Olaf Schenk suggested some modification of the standard algorithm of Cholesky decomposition for sparse symmetric matrices. The standard LL^T decomposition for sparse symmetric matrices can be described as follows:

Algorithm 2. Classical LL^T decomposition for sparse matrix

```
for row = 1, size_of_matrix

 {for column = 1, row

  {S = A[row, column]

  for all non-zero elements in the row before the column, S = S – A[row,
element]*A[element, column]

  // A[row, element]*A[element, column] have been computed

  If column<row A[row, column] = S/A[column, column]

  else A[row, column] = S^1/2

  }

}
```

In the paper [7], the following modification of Algorithm 2 for the computers with shared memory is proposed. Each computational thread sequentially selects a row or a set of k rows with similar structure (such set of rows is called supernode) *supr*, for which the following operations are performed.

Algorithm 3. Supernode modifications of Cholesky algorithm

```
for column = 1, supr

 {if column is not computed - wait, otherwise do {A[supr,*] = A[supr,*] –
A[column, *]*A[column, column]}

  // A[column, column] could be a square matrix

 }



A[supr, supr] = (A[supr, supr])^1/2

// if k ≠1, ½ means Cholesky decomposition of a dense matrix that can be
computed with Lapack functions

A[supr, supr+k,..,n] = A[supr, supr+k,..,n] * inv(A[supr, supr)],where
inv(B) mean inverse matrix B
```

In his paper, Olaf Schenk applies the algorithm to the entire matrix. In this paper, we apply it to the tree nodes only. Namely this provides an efficient Cholesky decomposition for a dependency tree node of the initial matrix.

The aforementioned Algorithm 3 describes an implementation of the procedure "compute elements of Z" in terms of the Algorithm 1. The procedure "prepare an update" differs only in a sense that each process modifies zero columns with the structure similar to *A[supr,*]* rather than *A[supr,*]* itself. After each process computes its update performing simple summation, we obtain A[supr,*] from which the computed elements of LL^T decomposition have been already deduced.

Thus, we have described the main steps of Algorithm 1. It has a number of drawbacks, however. First, the number of elements of the matrix L in each process differs significantly (for instance in the Figure 3b, the process 0 stores 3 tree nodes, whereas processes 1 and 3 have only one each). As a result, many processes may stay idle expecting the next task to work on. The size of the memory necessary for each process to store elements of the matrix L differs drastically. Following the idea of Algorithm 1, it can be demonstrated that all processes compute an update first, they start transferring data afterwards. This is inefficient since at this time the majority of processes are idle again. To avoid the idle time and distribute elements of LL^T-factors between processes more evenly, we propose an algorithm described in the next Section.

3. Asynchronous execution of processes

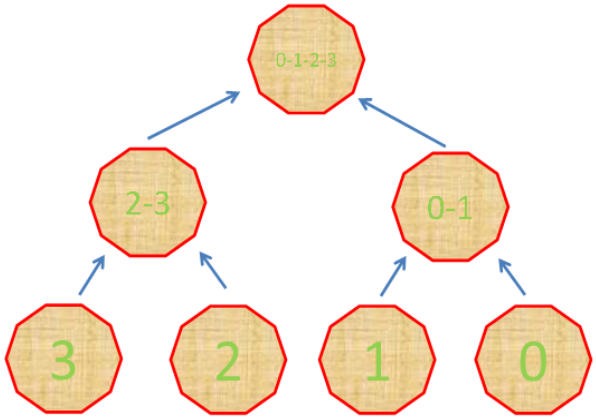


Figure 4. Distribution of nodes among processes

To avoid issues with uneven distribution of the elements of the matrix L, we propose a distribution method as in Figure 4 (digits in the decagons indicate the link of a node with a given process).

Figure 4 demonstrates the same dependence tree as in Figure 3 with the elements from each node of the tree being distributed in different manner between computational processes. At each tree level but the first (bottom) one, the elements of the matrix L are stored on several processes, e.g., at the second level all supernodes are distributed between two processes, at the third one – between four processes, etc. Supernodes from each node of the tree are distributed between n processes as follows: if the total number of supernodes in a certain tree node is m, then the first group of m_1 supernodes belongs to the first process, the next group of m_2 supernodes – to the second process, etc. Here $m_1+ m_1+\ldots+ m_n=m$. Note that the numbers m_1, m_2,.., m_n may vary that allows one to adjust them in order to provide better performance of the overall algorithm. Then Algorithm 1 is modified so that each process computes its part of the tree node Z. However, this idea does not provide a solution to the problem of keeping processes busy during the computations. Moreover, the problem becomes even bigger since parallel computation of the elements of the matrix L in a single tree node is virtually impossible – almost all supernodes in a single tree node are normally dependent on each other.

Note that each process can be executed by a modern computational node and, therefore, the node can consist of several dozens of individual computational threads. For individual processes to

send/receive the data and carry out the computations, we designate one thread in each process to be a "postman". A "postman" is a thread responsible for data transfer between the processes. Let us consider how Algorithm 2 changes.

3.1 Prepare an update

As it was stated in the Section 2, the Algorithm "prepare an update" is a modification of the Algorithm 3. As was mentioned before, to calculate LL^T factors from node Z of the dependence tree we need to calculate all LL^T factors from its sub-trees and take them into account during computations of LL^T factors of node Z. In general case, however, the node of the tree Z and its sub-tree are stored on different computational processes, so we cannot do it straightforwardly (node Z and its sub-tree have different non-zero pattern, for example). To resolve the issue, the following algorithm is proposed/ Each computational process i allocates matrix Z_i with the same non-zero pattern as the matrix corresponding to the node Z and fills it in with zero elements. Then, all elements from the sub-trees stored on the process i are taken into account in the matrix Z_i as if Z_i is Z (4^{th} line in the Algorithm 0). Further, the computed matrices Z_i are collected on the required process. Considering that we separated a postman thread, there is no need to compute an update first, and send it later, so computations and data transfers can be interleaved.

Algorithm 4. Asynchronous approach in Cholesky decomposition

```
if thread is a postman thread

   {Open a recipient to get updates

   for supr in A_loc

     if supernode supr is computed, send it to the respective process

     else wait until supernode is computed

   }

else

  {create A_loc(all elements in Z) // zero out the elements of node Z with
the same non-zero pattern

   for supr in A_loc

    { if column on the current process

      A_loc[supr,*] = A_loc[supr,*] – A[column, *]*A[column, column]

      // supernode supr is computed

    }

  }
```

It is apparent that having decreased the number of the threads involved in the computations we increased the total time of "update" computations in each process. Nevertheless, the experiments with a big number of threads show that the computational time increases insignificantly. It is also important to note that despite of the increase of the computation time to do the necessary "update", the transfer of the computed pieces between processes overlaps with these computations. Therefore, the total time spent in the new Algorithm is less compared to the Algorithm 3.

3.2 Compute elements of a tree node

It is much more interesting to consider a more complicated algorithm of computing the elements of the matrix L in a single node of the tree provided that this node is distributed among several processes. Let the elements of the node Z including the elements of the dependent sub-trees (children) that are not processed yet be distributed as it is shown in the Figure 5a, i.e. the first group of supernodes belongs to the first process, and the second group – to the second one, etc. So, each computational process stores only "grey" supernodes.

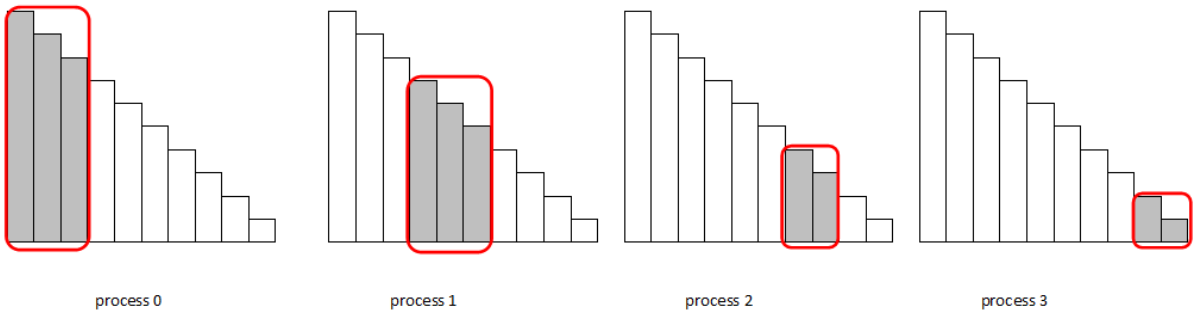


Figure 5a. Supernode distribution among the processes

It can be clearly seen that with such a distribution only the process 0 can start the computations. Thus, the first supernode can only be computed within it. However, after the supernode computation is done, it can be used by the computational threads of the process 0 for other (dependent) supernodes, second it can be sent by the postman thread to other processes, which in turn can use it in their (dependent) supernodes (see the Figure 5b, where blue arrows indicate communications between the processes, the green ones – the update of the supernodes on each process with the supernode received from the process number 0).

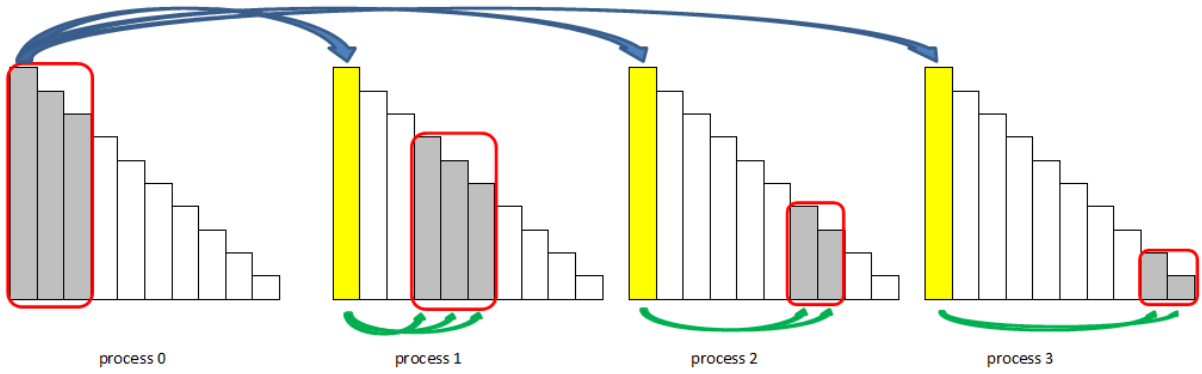


Figure 5b. Computational flow

With this example, it is apparent that if each process has more than 3 computational threads, some threads will simply lack a supernode to process, so making a postman out of one computational thread will have little effect on the overall efficiency if the thread count is big enough. Of course, one supernode can be processed with several threads, but this is not going to be considered in this paper.

4. Numerical experiments

All numerical experiments in this paper were carried on the Infiniband*-linked cluster consisting of 16 computational nodes; each node contains two Intel® Xeon® X5670 processors (12 cores in total) with 48Gb of RAM per node. The variable number of the computational threads within a node is created, i.e. only part of totally available threads is working within the nodes in the most cases.

4.1 Scaling of computational time

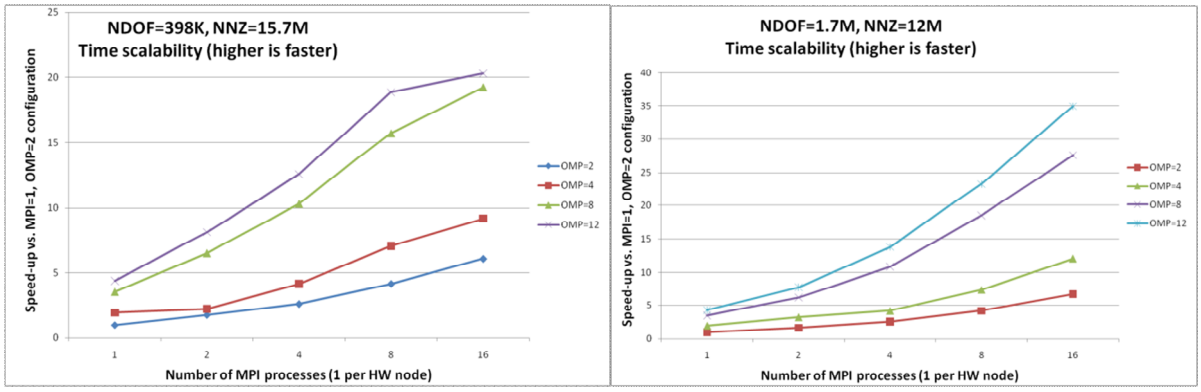


Figure 6a-b. Intel® Direct Sparse Solver for Clusters scalability of time

For this experiment, we selected either 7-diagonal matrix resulted from the approximation of a Helmholtz equation on a uniform grid with a positive coefficient (specific Helmholtz coefficient value is not crucial here since it has no effect on the matrix structure and only the accuracy of the solution obtained depends on it), or the matrix generated from the oil-filtration problem. The number of degrees of freedom (NDOF) for the first matrix is about 398K elements, for the second one is about 1.7M, and the number of nonzero elements (NNZ) in each matrix is 15.7M and 12M, respectively.

Figures 6a-6b show acceleration of Intel® Direct Sparse Solver for Clusters code on a different number of processes compared to the same program launched on 1 MPI process with 2 OpenMP threads on different matrices. The colored lines correspond to a different number of OpenMP threads used in the code. It can be seen from the Figures that the execution time reduces in all cases with the increase of the number of threads and processes. It can be readily seen that sometimes even super-linear acceleration takes place depending on the number of OpenMP threads that can be easily explained from the nature of the algorithm – one thread is used to send & receive data and rather often it falls out of the computations. For example, in the case of 2 threads, one thread is the postman and the other is the computational one, in the case of 4 threads, one thread is the postman and 3 others are computational ones. Based on the Figures, it can be concluded that it is recommended to exploit the computational threads on the node to the maximum and it is not recommended to have several computational processes per one node. From the Figure 6a, it is apparent that the acceleration of the combination 2 MPI x 8 OpenMP is larger than 4 MPI x 4 OpenMP, that in turn, is larger than 8MPI x 2 OpenMP. This perfectly matches the architecture features imposed by the modern computer systems, namely, the growing number of cores (threads) per computational node.

4.2 Memory scaling per node

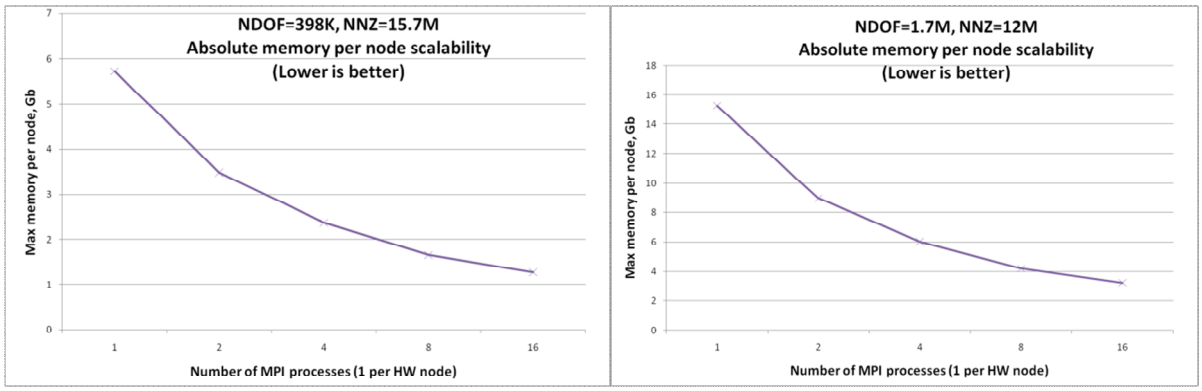


Figure 7a-b. Intel® Direct Sparse Solver for Clusters memory

We use the same matrices as before for the testing purposes. In the Figures 7a-7b, the maximal memory size needed for a computational node depending on the number of nodes is presented. Memory size that Intel® Direct Sparse Solver for Clusters needs for every computational node barely depends on the number of computational threads on it. Therefore, we present the data for the number of threads equal to 12 only. It is apparent that the memory size that every process needs demonstrates invers dependence on the number of processes (for the second matrix, the memory required decreased 5 times for 16 processes vs. 1 process). If the computational cluster has insufficient memory per node, it is still possible to solve the system of linear equations using Intel® Direct Sparse Solver for Clusters package increasing the number of nodes in the cluster. An example of this will be shown in the following

4.3 Solving a huge system of linear equations

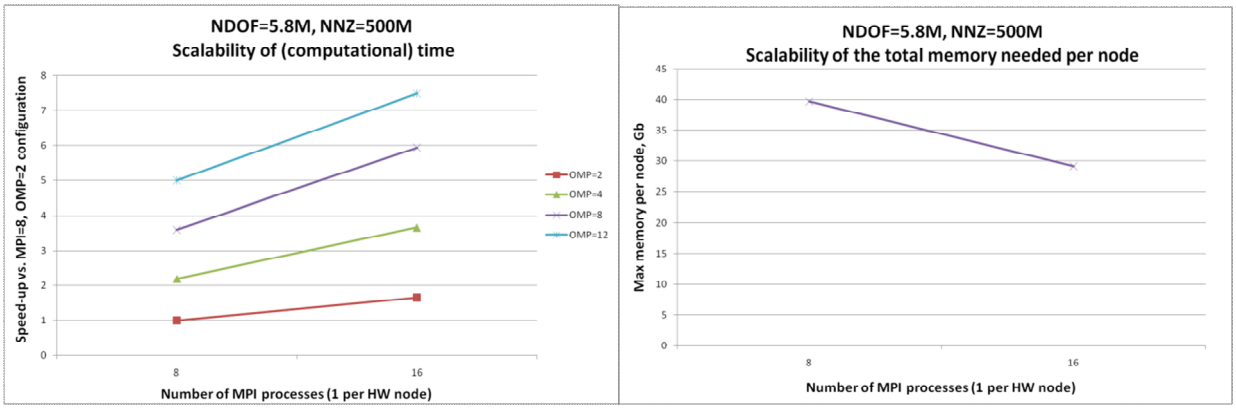


Figure 8a-b. Intel® Direct Sparse Solver for Clusters time and memory scalability, a huge system

In this Subsection, we chose a matrix of size 5.8M with more than half a billion non-zero elements for the experiment. Thanks to Intel® Direct Sparse Solver for Clusters memory scaling, we can solve this system on 8+ MPI processes. Note that almost 40 GB of memory is required per process in case of 8 MPI processes. For 16 processes, 29GB of memory is only required (see Figure 8b). In Figure 8a, the comparison of computational time with the configuration 8 MPI x 2 OpenMP is presented. It is clear that even for such a big matrix size and big number of MPI processes, Intel® Direct Sparse Solver for Clusters shows good scalability both in terms of OpenMP threads and MPI processes.

5. Conclusion

Within the frameworks of multifrontal approach, we proposed an efficient algorithm implementing all stages of Cholesky decomposition inside the node of the dependency tree for all processes on a distributed memory machine. This approach is implemented in Intel® Direct Sparse Solver for Clusters package, and numerical experiments show good scaling in computational time - proportional to the number of computational nodes used and the number of threads within them. Besides, this algorithm reduces the requirement for the memory size used by the algorithm on a single node when the number of processes grows. The experiments made confirm this.

Acknowledgement The authors would like to thank Sergey Gololobov for providing feedback that improved both style and content of the paper.

References

1. P.R.Amestoy, I.S.Duff, C.Vomel, Task scheduling in an asynchronous distributed memory multifrontal solver, SIAM Journal on Matrix Analysis and Applications, Vol 26(2) pp 544--565 (2005)

2. P. Amestoy, I.S. Duff, S. Pralet, C. Voemel, Adapting a parallel sparse direct solver to SMP architectures, Parallel Computing, 29 (11-12), pages 1645-1668.
3. P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet. Hybrid scheduling for the parallel solution of linear systems, Parallel Computing 32 (2): 136-156, 2006.
4. Amestoy, P. R. and Duff, I. S. 1993. Memory management issues in sparse multifrontal methods on multiprocessors. Int. J. Supercomput. Appl. 7, 64--82.
5. P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal on Matrix Analysis and Applications, 23[1], 15-41 (2001)
6. M. Bollhöfer and O. Schenk, Combinatorial Aspects in Sparse Direct Solvers, GAMM Mitteilungen, 29 (2006), pp. 342-367.
7. Olaf Schenk and Klaus G¨artner. On fast factorization pivoting methods for sparse symmetric indefinite systems. Technical report, Department of Computer Science, University of Basel, 2004
8. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. George Karypis and Vipin Kumar. 10th Intl. Parallel Processing Symposium, pp. 314 - 319, 1996.
9. A Parallel Algorithm for Multilevel Graph Partitioning and Sparse Matrix Ordering. George Karypis and Vipin Kumar. Journal of Parallel and Distributed Computing, Vol. 48, pp. 71 - 85, 1998
10. Parallel Multilevel Algorithms for Multi-Constraint Graph Partitioning. Kirk Schloegel, George Karypis, and Vipin Kumar. Euro-Par, pp: 296-310, 2000
11. George Karypis, Vipin Kumar: Parallel Multilevel Graph Partitioning. IPPS 1996: 314-319
12. http://glaros.dtc.umn.edu/gkhome/views/metis

libgpuvm: организация автоматического обмена данными между хостом и ГПУ

А.В. Адинец[1,2]

[1]НИВЦ МГУ им. М.В.Ломоносова
[2]Объединённый институт ядерных исследований

В настоящее время графические процессоры активно используются высокопроизводительными приложениями. Важным аспектом написания таких приложений является организация обмена данными между памятью хоста и ГПУ. В настоящее время такие обмены необходимо выполнять вручную, что затрудняет написание ГПУ-приложений. В данной работе предлагается подход к автоматизации передачи данных. Данные передаются на ГПУ при вызове ядра, после чего блокируются. Обратно они передаются при обработке страничного прерывания, вызванного обращением к ним на хосте. libgpuvm поддерживает данные, выделенные в динамической памяти, а также многопоточные приложения. OpenCL-совместимые ГПУ на ОС Linux поддерживаются без изменения кода их драйверов. Система предназначена прежде всего для интеграции поддержки ГПУ в языки программирования. В данной работе также описывается интеграция libgpuvm в систему NUDA.

1. Введение

В настоящее время графические процессорные устройства (ГПУ) активно используются для решения вычислительно ёмких задач. В дискретных ГПУ, наиболее мощных и популярных, память ГПУ отделена от памяти хоста. И хотя ГПУ может обращаться хост-памяти напрямую, как правило, это на порядок медленнее использования ГПУ-памяти. Поэтому данные необходимо копировать в ГПУ-память для обработки, а потом копировать обратно.

При использовании низкоуровневых средств программирования ГПУ, таких как CUDA [1] или OpenCL [2], программист организует обмен данными вручную. Однако с появлением высокоуровневой поддержки ГПУ в языках программирования, а также использующих ГПУ библиотеки возникает потребность в автоматизации процесса обмена данными. Способ организации обмена данных должен предоставлять простой интерфейс, и в то же время не приводить к большим накладным расходам.

Известен ряд способов автоматизации обмена данными между хостом и ГПУ. Самый простой заключается в копировании данных на ГПУ перед вызовом ядра, и копировании их обратно после его завершения. Мы назовём его *полным копированием*. Такой подход предоставляет простой интерфейс, однако приводит к большим накладным расходам. И хотя для ряда задач, например, задачи многих тел, накладные расходы малы, для остальных, таких как сеточные или итерационные методы, накладные расходы велики, так как там копирование данных на хост обычно не требуется.

Для решения проблемы накладных расходов также предолжено ряд путей. Один из них — директивы или аннотации, указывающие, что данные уже расположены в памяти ГПУ. Одной из форм таких директив являются *регионы данных*: данные внутри региона используются только на ГПУ, и не копируются обратно между вызовами ядра. Это снижает накладные расходы, поскольку тот же объём копирования теперь приходится на более количество вызовов ядер. Данный подход используется, например, в коммерческих компиляторах PGI [3] и CAPS HMPP [4], а также предложенный стандарт OpenACC [5]. Однако дополнительные директивы замусоривают код, кроме того, они требуют аннотирования параметров тех функций, которые работают с ГПУ. Компилятору также приходится отслеживать, какие массивы находятся на ГПУ, а какие нет.

Другим подходом является введение новых типов данных, похожих на массивы. Память для них выделяется на хосте и на ГПУ, а реализация обеспечивает автоматическую синхронизацию данных между ними. Такие типы могут быть введены в любой язык, поддерживающий классы и перегрузку оператора индексирования. Этот подход называется *автосинхронизация*. В дополнение к указателям на память хоста и ГПУ, такие массивы содержат информацию о том, какая копия данных в настоящий момент актуальна. При использовании массива в ГПУ-ядре данные актуализируются на ГПУ; при обращении к массиву на хосте — актуализируются на хосте. Если данные не являются актуальными выполняется копирование данных; если же они уже актуальны, то связанных с копированием накладных расходов удаётся избежать. Преимуществом данного подхода является простая и эффективная реализация отслеживания актуальности во время выполнения, что упрощает компилятор. Основным же недостатком подхода является необходимость введения новых типов, что в терминах реализации языка опять выливается в директивы и аннотации, и приводит к замусориванию программы.

Наш подход, называемый подходом *libgpuvm* [6], похож на подход автосинхронизации. Однако передача данных с ГПУ обратно на хост выполняется при обработке страничного прерывания в пользовательском режиме (проще говоря, обработке сигнала `SIGSEGV`), а не в перегруженном операторе индексации.

Это простая идея, и она может быть легко реализована, если необходимые данные выровнены на границу страницы. Такая реализация представлена в библиотеке GMAC [7], которая также предоставляет свой аллокатор памяти. Однако если речь идёт о поддержке ГПУ в языке программирования, массив может быть выделен не в той функции или библиотеке, в которой он используется, и скорее всего, не будет выравнен на границу страницы. libgpuvm поддерживает память, выделенную функции `malloc()` или подобного аллокатора динамической памяти. libgpuvm также поддерживает работу в многопоточных средах с несколькими ГПУ, а также работает с несколькими широко используемыми реализациями OpenCL без дополнительных изменений в коде уровня ядра.

Данная статья организована следующим образом. В разделе 2 описывается идея libgpuvm, а также интерфейс, который она предоставляет программам и системам времени выполнения языков. В разделе 3, описываются подробности реализации libgpuvm и её внутренние структуры данных. В разделе 4 описывается интеграция libgpuvm в систему времени выполнения языков программирования, на примере системы программирования ГПУ NUDA [8]. Наконец, в разделе 5 обсуждаются возникшие проблемы и направления дальнейшей работы.

2. Идея и интерфейс libgpuvm

Мы предполагаем, что приложение исполняется на *хосте*, многоядерном процессоре с общей памятью, и, вообще говоря, нескольких *устройствах* (ГПУ, но могут быть и многоядерные ЦПУ), каждый из которых имеет свою ОЗУ, логически отдельную от ОЗУ хоста. Приложение состоит из набора *потоков* на хосте, которые могут динамически порождаться и завершаться, а также *вызывать ядра* на устройствах. Любой поток на хосте может использовать любое устройство. *Хост-массив* — диапазон адресов ОЗУ хоста, определяемый начальным адресом и длиной (в байтах). Ввиду ограничений реализации, хост-массив должен быть выделен при помощи `malloc()` или подобного динамического аллокатора. Хост-массив не может быть выделен в стеке или глобальных данных программы. Хост-массивы могут использоваться либо несколькими потоками на хосте, либо на одном из устройств. Диапазоны адресов различных хост-массивов не пересекаются. С хост-массивом может быть ассоциирован *буфер на устройстве*, вообще говоря, на нескольких устройствах. На одном устройстве с одним хост-массивом может быть ассоциировано не более одного буфера. В настоящее время в качестве буферов на устройстве используются буфера OpenCL.

Для каждого массива будем рассматривать моменты начала и окончания работы ядер, которые его используют, и соответствующие промежутки времени. Будем говорить, что хост-массив *используется на хосте*, если происходит обращение по одному из его адресов на хосте. Хост-массив *используется на устройстве*, если на этом устройстве происходит обращение к ассоциированному с этим массивом буферу. Хост-массив *совместно используется корректно*, если в каждый промежуток времени он используется либо на хосте, возможно, несколькими потоками, или на одном из устройств. Если же хост-массив совместно используется одновременно на нескольких устройствах, или на каком-то устройстве и хосте, его *совместное использование некорректно*. Таким образом, хост-массив становится как бы единицей совместного использования, т.к. он может использоваться только в одном месте в каждый промежуток времени. Если совместное использование всех массивов корректно, libgpuvm обеспечивает корректность совместного использования в интуитивном смысле, т.е. как если бы массив был расположен в общей памяти, совместно используемой хостом и устройствами. Если же совместное использование массива некорректно, могут возникать гонки за ресурсы, даже если множества адресов, используемые хостом и различными устройствами, не пересекаются. В этом случае libgpuvm не гарантирует корректность описанной выше абстракции разделяемой памяти.

С каждым хост-массивом ассоциируется информация об актуальности. Информация хранится в ассоциативном массиве, который индексируется начальным адресом массива. Как и в подходе с автосинхронизацией, данные актуализируются на ГПУ, когда они используются в ГПУ-ядре, и актуализируются на хосте, когда к ним происходит обращение на хосте. Если данные уже актуальны в нужном месте, копирования не происходит, за счёт чего удаётся избежать накладных расходов. Однако механизм отслеживания обращения к данным на хосте другой. Когда завершается исполнение ГПУ-ядра, диапазон адресов хост-массива *защищается* на хосте при помощи системного вызова `mprotect()`. Когда после этого происходит обращение к данным на хосте, механизм защиты памяти ЦПУ генерирует страничное прерывание, которое отправляется вызвавшему его потоку в виде сигнала. Данные передаются обратно с ГПУ обратно на хост в обработчике сигнала, после чего исполнение хост-программы может продолжаться. Если данные совместно используются несколькими устройствами, они копируются с того устройства, которое содержит их актуальную копию, т.е. на котором они в последний раз использовались.

Для повышения гибкости упрощения интеграции с существующими библиотеками и средами времени выполнения libgpuvm является достаточно минималистичной: она обеспечивает только синхронизацию данных между хостом и устройствами, и всё, что для этого необходимо. Прочие функции, такие как инициализация устройств, выделение и освобождение памяти на устройстве и установка параметров ядра, выполняются самим приложением. Интерфейс libgpuvm отражает минималистичный дизайн самой библиотеки.

Прототипы библиотечных функций libgpuvm приведены на рис. 2. Все они начинаются с префикса `gpuvm_`, и возвращают код ошибки, или 0 в случае успешного завершения. Инициализация libgpuvm выполняется вызовом `gpuvm_init()`, в который передаётся список “устройств”, т.е. очередей команд OpenCL, ассоциированных с ними. В дальнейшем устройства идентифицируются по номеру в этом списке. Дополнительные флаги задают используемый интерфейс взаимодействия с устройством (сейчас только OpenCL), а также дополнительные настройки, например, выполнять ли сбор статистики. Инициализация выполняется только один раз. Статистику можно получить при помощи вызова `gpuvm_stat()`. Некоторые сведения доступны всегда, в то время как другие, например, суммарное время передачи данных, требует включения сбора статистики на очередят команд OpenCL и в библиотеке libgpuvm.

Для того, чтобы хост-массив можно было использовать на устройстве, его необходимо *связать* с буфером вызовом `gpuvm_link()`. Хост-массив может быть связан с несколькими буферами, но на одном устройстве у него может быть только один буфер. Флаги

указывают, расположены ли данные изначально на хосте или на устройстве. Когда хост-массив больше не требуется использовать на устройстве, его отребуется отсоединить вызовом `gpuvm_unlink()`.

Начало использования хост-массива на устройстве обозначается вызовом `gpuvm_kernel_begin()`. Флаги обозначают, будет ли массив использован только на чтение, или также и на запись (сейчас поддерживается только последнее). Массив будет актуализирован на соответствующем устройстве, и если потребуется, будет выполнено копирование данных. Окончание использования массива на устройстве обозначается вызовом `gpuvm_kernel_end()`. При этом будет установлена защита на диапазон адресов массива на хосте, если она не была установлена ранее, так что при следующем обращении к массиву на хосте произойдёт страничное прерывание. Между началом и окончанием использования массива на устройстве на этом устройстве может быть вызвано любое число ядер; но корректное использование массива на хосте гарантируется только после вызова `gpuvm_kernel-end()`.

```
// libgpuvm initialization
int gpuvm_init(unsigned ndevs, void** devs, int flags);

// linking a host array to a device buffer
int gpuvm_link(void *hostptr, size_t nbytes, unsigned idev,
  void *devbuf, int flags);

// unlinking a host array from the device buffer
int gpuvm_unlink(void *hostptr, unsigned idev);

// start using array on device
int gpuvm_kernel_begin(void *hostptr, unsigned idev, int flags);

// finish using array on device
int gpuvm_kernel_end(void *hostptr, unsigned idev);

// get some statistical information
int gpuvm_stat(int param, void *val);
```

Рис. 1. Прототипы библиотечных вызовов libgpuvm

3. Реализация libgpuvm

Идея обработки страничных прерываний в пользовательском режиме, хотя и звучит не обычно, сама по себе новой не является. Защита памяти широко распространена в ЦПУ уже достаточно давно, а ОС предоставляет информацию об этом в приложения, например, при помощи сигнала `SIGSEGV`. Защита памяти в современных ЦПУ обычно интегрирована в механизм страничной трансляции, и выполняется на уровне страниц. Размер страницы на процессорах x86 составляет 4 КБ; у других архитектур похожий минимальный размер страницы. И хотя в процессоре имеется поддержка страниц большего размера, по умолчанию ОС использует страницы минимального размера.

Таким образом, минимальная гранулярность защиты памяти равна размеру страницы. Соответственно, большинство приложений обработки страничных прерываний в пространстве пользователя требуют, чтобы данные в памяти занимали страницы целиком. Это, в свою очередь, требует использования специального аллокатора памяти. Однако память, с которой будет работать libgpuvm, выделена аллокатором по умолчанию, и в другой части

приложения; соответственно, нельзя рассчитывать на использование специального аллокатора. Поэтому libgpuvm должна поддерживать работу с любой динамически выделенной памятью; единственным требованием является отсутствие совместно используемых страниц со стеком или глобальными данными программы. Это требуется, чтобы установка защиты на диапазоны адресов не нарушала работу механизма вызова функций и обработки сигналов. Отсутствие требования к выравниванию приводит к двум типам проблем:

- **ложное совместное использование**, когда несколько массивов используют ту же самую страницу памяти, и поэтому ставить включать/отключать их защиту надо также совместно
- **вмешательство в работу OpenCL**, когда выделенная `malloc()`-ом память, используемая реалиацией OpenCL, оказывается в страницах, на которые libgpuvm ставит защиту

Реализация libgpuvm решает две эти проблемы в многопоточной среде с несколькими ГПУ. Для этого, в дополнение к хост-массивам, вводятся ещё два вида диапазонов адресов:

- *регионы защиты*, или просто *регионы*, диапазоны памяти, которые выравнены на границы страниц, покрывают страницы целиком, и служат единицей защиты памяти
- *подрегионы*, являющиеся пересечениями регионов и хост-массивов, которые служат единицей синхронизации между хостом и устройством; информация об актуальности поддерживается именно на уровне подрегионов

Диапазоны адресов разных регионов не пересекаются. Диапазоны адресов разных подрегионов не пересекаются. Каждый подрегион целиком содержится в одном регионе и хост-массиве. Регион может содержать несколько подрегионов, и должен содержать как минимум один подрегион. Подрегионы одно региона организованы в связанный список, отсортированный по адресу начала подрегиона. Каждый хост-массив содержит от одного до трёх подрегионов. Если хост-массив целиком помещается в одну страницу памяти, или занимает любое количество страниц целиком, он состоит из одного подрегиона. Иначе “средняя” часть массива, которая занимает целиком некоторое количество страниц, образует один подрегион, а оставшиеся “голова” и “хвост”, также образуют по одному подрегиону; одна из трёх частей в этом случае может отсутствовать.

Регионы организованы в *дерево регионов*, бинарное дерево, упорядоченное по адресу региона. Эта структура данных используется для поиска региона или хост-массива по адресу, получаемому в библиотечных вызовах или обработчиках сигналов. Для нормального функционирования libgpuvm память для её динамических структур данных выделяется отдельным аллокатором, который получает страницы от ОС. Таким образом, страницы данных, используемые libgpuvm, никогда не защищаются.

Поскольку финальное копирование данных в реализациях OpenCL выполняется в режиме пользователя, перед началом копирования необходимо снять защиту с региона памяти. В многопоточной среде это означает, что остальные потоки могут начать использовать данные до того, как они станут актуальными. Более того, если страничное прерывание возникнет в одном из рабочих потоков OpenCL, защита должна быть снята сразу для обеспечения нормальной работы приложения; таким образом, та же проблема возникает и в случае одного потока приложения.

Мы рассматривали два способа решения этой проблемы. Первый заключается в остановке всех потоков приложения перед снятием защиты, и их возобновлении после завершения копирования. Хотя это решение работает, остановка потоков является узким метсом в многопоточных приложениях. Вторым способом избежать снятия защиты перед копированием является запись данных по другому виртуальному адресу, который соответствует тем же страницам физической памяти. И хотя код в Linux-ядре для записи в файл

`/proc/self/mem` существует, сейчас он по умолчанию отключён [9]. Поэтому, для обеспечения стабильной работы, мы остановились на 1-м варианте, связанном с остановкой потоков.

Для остановки потоков текущая реализация использует полный список потоков процесса. На Linux его можно получить путём чтения каталога `/proc/self/task`. Поскольку чтение каталога каждый раз занимает много времени, а новые потоки порождаются редко, список потоков кэшируется. При каждой остановке потоков проверяется время обновления каталога, и он повторно читается только в случае, если он изменился с последнего раза.

С каждым потоком при помощи упорядоченного по его идентификатору двоичного дерева ассоциирован семафор. Чтобы остановить поток, ему отправляется сигнал реального времени, по которому вызывается зарегистрированный libgpuvm обработчик. Обработчик просто блокируется на семафоре потока; чтобы возобновить поток, семафор разблокируется.

Исполнение некоторых потоков не должно останавливаться. Например, не должны останавливаться рабочие потоки OpenCL, так как это приведёт к взаимным блокировкам. Отдельно хранится список идентификаторов потоков, которые не должны останавливаться; он проверяется, когда строится бинарное дерево с семафорами. Потоки OpenCL порождаются при создании контекстов и очередей; чтобы определить их идентификаторы, вводится специальный библиотечный вызов `gpuvm_pre_init()`, который должен быть вызван один раз до и один раз после создания контекстов и очередей. Оба раза он проверит список потоков, а а разницу занесёт как потоки, которые не останавливаются. Разумеется, рабочие потоки libgpuvm также не должны останавливаться. В настоящее время таких потока два, каждый обрабатывает запросы из своей очереди. *Поток снятия защиты* снимает защиту памяти с регионов, а также отвечает за остановку и запуск других потоков. *Поток копирования* инициирует передачу данных с устройства на хост, и дожидается её окончания.

Далее приводится описание работы библиотечных вызовов и обработчиков сигналов libgpuvm. Вызов `gpuvm_init()` инициализирует внутренние структуры данных libgpuvm, устанавливает обработчики сигналов и порождает рабочие потоки. Все остальные вызовы используют глобальную блокировку чтения-записи; это позволяет упростить архитектуру системы. Обработчик `SIGSEGV`, `gpuvm_kernel_begin()` и рабочие потоки ставят глобальную блокировку на чтение; остальные вызовы используют блокировку на запись. Кроме того, используется блокировка чтения-записи на бинарное дерево семафоров потоков. Поток снятия защиты блокирует блокирует её на запись, когда добавляются новые потоки блокируют её на чтение.

Вызов `gpuvm_link()` создаёт для новых хост-массивов внутреннюю структуру данных, разделяет их на подрегионы и находит регионы для вставки подрегионов или создаёт новые, а также ассоциирует буфер на устройстве с хост-массивом. Хост-массив также проверяется по дереву регионов; если уже существует массив с пересекающимся, но не полностью совпадающим диапазоном адресов, возвращается ошибка. Вызов `gpuvm_unlink()` снимает связь хост-массива с буфером устройства. Если это был последний буфер, ассоциированный с массивом, удаляется сам массив, а также связанные с ним подрегионы и, если в них больше нет подрегионов, регионы. Если диапазон адресов региона был защищён, защита снимается.

При вызове обработчика сигнала `SIGSEGV` по страничному прерыванию адрес обращения проверяется по дереву регионов. Если адрес не относится к региону libgpuvm, вызывается предыдущий обработчик сигнала. Если же адрес относится к обслуживаемому libgpuvm региону, регион передаётся потоку снятия защиты. После снятия защиты регион получает статус *ожидающего* и передаётся потоку копирования для актуализации. Поток копирования дожидается завершения копирования всех подрегионов, после чего ставит сообщение об этом в очередь потока снятия защиты. При получении этого сообщения поток снятия защиты уменьшает счётчик ожидающих регионов. Как только этот счётчик достигает нуля, остановленные потоки возобновляются.

Поток, на котором произошла обработка сигнала, дожидается снятия защиты. В случае, если это рабочий поток OpenCL, его исполнение возобновится сразу после снятия защиты, обеспечивая нормальную работу приложения. Если же это был поток приложения, он будет остановлен, и возобновлён только после актуализации данных.

По вызову `gpuvm_kernel_begin()`, данные актуализируются на устройстве. Для этого для каждого подрегиона выполняются следующие действия:

- если подрегион актуален на том же устройстве, копирования не выполняется
- если подрегион актуален на хосте, он копируется на устройство и обозначается как актуальный на устройстве
- если подрегион актуален на другом устройстве, он сначала актуализируется на хосте путём вызова страничного прерывания на подрегионе; после этого выполняются те же действия, что и в предыдущем пункте

По вызову `gpuvm_kernel_end()`, каждый из подрегионов хост-массива помечается как актуальный на устройстве, где он был использован, после чего на соответствующие регионы ставится защита, если она не была установлена.

4. Использование libgpuvm

В качестве примера, libgpuvm используется для обеспечения прозрачного совместного использования массивов между хостом и ГПУ с низкими накладными расходами в системе NUDA. NUDA (= Nemerle Unified Device Architecture) представлет собой систему расширения для языка Nemerle [10] для поддержки высокоуровневого программирования ГПУ. Ранее в NUDA использовался подход автосинхронизации, что потребовало введения новых типов для ГПУ-массивов. Для конечных пользователей эти типы были скрыты за аннотациями. И хотя это хорошо работает для массивов, которые используются только в той функции или классе, где они объявлены, в случае использования массива в нескольких функциях это выглядит не очень хорошо. У каждой такой функции требуется добавить дополнительные аннотации на параметры, а это публичная функция, требовалось создавать её версию, которая работает с обычными массивами.

С использованием libgpuvm стало возможным снизить накладные расходы на передачу данных, и при этом предоставить конечному пользователю интерфейс, не отличающийся от интерфейса обычных массивов. Для обеспечения поддержки libgpuvm в NUDA пришлось добавить дополнительный код, в основном для ленивого выделения буферов на устройстве и связывания массивов. В передаваемый на ГПУ код была добавлена поддержка обычных массивов в дополнение к ГПУ-массивам.

Одной из проблем, которые требовалось решить, было удаление выделенных на ГПУ массивов. Текущая версия NUDA работает со средой выполнения mono версии 2.0.1, в которой по умолчанию используется консервативный сборщик мусора Бёма [11]. Он не перемещает выделенные объекты в памяти, что упрощает интеграцию. Когда места для очередного буфера на устройстве становится недостаточно, сборщик мусора вызывается вручную. После этого, мы проходим по списку массивов, которые связаны с буферами на устройстве, и удаляем буфера тех из них, которые были собраны сборщиком мусора. Если даже после этого памяти для буфера на устройстве нет, выбрасывается исключение.

Здесь мы сравниваем подходы libgpuvm автосинхронизации и полного копирования на примере реализации поддержки массивов на ГПУ в NUDA. Сравниваются простота использования и накладные расходы. В качестве модельной задачи мы используем программу, которая выполняет фиксированное количество итераций вида $x^{n+1} = Ax^n + b$, т.е. последовательность умножения матрицы на вектор. Мы специально выбрали задачу, в которой

вычислительная нагрузка невелика по сравнению с объёмами передачи данных, чтобы лучше увидеть накладные расходы; в более вычислительно ёмких приложениях они будут ещё ниже. Код состоит из двух функций, первая из которых содержит внешний цикл по n, а вторая реализует умножение матрицы на вектор. Код для libgpuvm и полного копирования одинаковый и приведён на рис. 2. Код для автосинхронизации приведён на рис. 3. Код, реализующий инициализацию массивов, замеры времени и печать результата опущен для краткости. Мы также предполагаем, что матрица уже транспонирована, для более эффективной реализации на ГПУ. В представленном на рисунках коде `do` — макрос цикла произвольной размерности, `nuwork()` — макрос, отправляющий цикл на ГПУ и задающий размер блока рабочей группы, `<->` — макрос обмена значений переменных. Дополнительно, на рис. 3 можно видеть аннотации `nunew` и `nuarr`, применение которых соответственно к операции выделения памяти под массив и к объявлению переменных-массивов превращает их в операции и объявления ГПУ-массивов. Эти аннотации позволяют сделать код более удобным для чтения.

```
iterate(niters : int, n : int) : void {
  mutable x = array(n) : array[float];
  mutable x1 = array(n) : array[float];
  def a = array(n, n) : array[float];
  def b = array(n) : array[float];
  // ... initialize
  do(it in niters) {
    matvecmul(x1, a, x, b, n);
    x1 <-> x;
  }
  // ... print x
}  // iterate()
matvecmul(x1 : array[float], a : array[2, float], x : array[float],
  b : array[float], n : int) : void {
  nuwork(128) do(i in n) {
    mutable r = b[i];
    do(j in n) r += a[j, i] * x[j];
    x1[i] = r;
  }
}  // matvecmul()
```

Рис. 2. Код для подходов libgpuvm и полного копирования

Мы используем две тестовые системы, характеристики которых приведены в таблице 1. Там же приведены измеренные нами затраты на выполнение ряда важных для нас операций. Было проведено 5 типов экспериментов: для полного копирования (fullcopy), для автосинхронизации (auto-sync), для libgpuvm (libgpuvm), а также дополнительные тесты libgpuvm-copy и libgpuvm-false. Тест libgpuvm-copy использует libgpuvm, но добавляет между итерациями обращения к данным на хосте, что приводит к полному копированию данных на каждой итерации. Этот тест демонстрирует накладные расходы в (маловероятном) худшем случае использования libgpuvm, когда все данные попеременно используются на ГПУ и на хосте. В тесте libgpuvm-false также используется libgpuvm, но между итерациями происходит обращение к первой странице одного из векторов, что вызывает копирование одной страницы данных обратно на хост. В NUDA (и mono) это может возникнуть в том случае, если вызвать на массиве какой-то метод, например, получение длины. Это связано с тем, что заголовок массива хранится вместе с его данными. По сути, данный тест демонстрирует накладные расходы в типичном случае ложного совместного использования, отсюда его

```
iterate(niters : int, n : int) : void {
  mutable x = nunew array(n) : array[float];
  mutable x1 = nunew array(n) : array[float];
  def a = nunew array(n, n) : array[float];
  def b = nunew array(n) : array[float];
  // ... initialize
  do(it in niters) {
    matvecmul(x1, a, x, b, n);
    x1 <-> x;
  }
  // ... print x
}  // iterate()
matvecmul(nuarr x1 : array[float], nuarr a : array[2, float],
  nuarr x : array[float], nuarr b : array[float], n : int) : void {
  nuwork(128) do(i in n) {
    mutable r = b[i];
    do(j in n) r += a[j, i] * x[j];
    x1[i] = r;
  }
}  // matvecmul()
```

Рис. 3. Код для подхода автосинхронизации

название.

Результаты работы тестов для ГПУ NVidia и AMD и представлены в таблицах 2 и 3, соответственно. Использовался размер вектора 2048 (размер матрицы 2048×2048), и выполнялись 1000 итераций умножения матрицы на вектор. Перед основным циклом выполнялся цикл "разогрева", чтобы скомпилировать байт-код и ядра ГПУ; время его выполнения исключено из приведённых данных. Времена приведены в микросекундах, в расчёте на одну итерацию. Времена исполнения ядра и копирования измерялись при помощи профилировки OpenCL, как разность между временами `COMMAND_START` и `COMMAND_END`; использовались только блокирующие команды OpenCL. Остальные времена измерялись при помощи функции `gettimeofday()`. Более конкретные значения временных параметров:

- *общее время* — время, затраченное на одну итерацию
- *время устройства* — время исполнения ядра на устройстве
- *время копирования* — время копирования данных на устройство и с него
- *время накладных расходов* — время накладных расходов OpenCL (по большей части) и NUDA на вызов ядра и копирование данных; оно измеряется как разность между временем выполнения команд, измеренных вручную, и временем исполнения команд устройством, полученное при помощи профилировки OpenCL
- *время обработки сигнала* — время, затраченное на обработку сигнала, начиная от момента перед остановкой потоков, и заканчивая моментом после возобновления последнего потока
- *прочее время* — время, которое остаётся от общего времени после вычитания остальных факторов

Таблица 1. Системы, используемые для тестирования libgpuvm + NUDA

Название	**fermi**	**amd**
ГПУ	NVidia Fermi C2050	AMD Radeon HD5830
Версия драйвера	280.13	11.9
Реализация OpenCL	CUDA 4.0	AMD APP 2.5
ЦПУ	Intel Core 2 Quad, 2.4 GHz	Intel Core 2 Duo, 2.2 GHz
Версия ОС	Debian 6.0, 2.6.32-amd64	Ubuntu 11.10, 2.6.38-13-server
Время запуска ядра	55 μc	100 μc
Время запуска копирования	150 μc	600 μc
Время обработки сигнала	52 μc	33 μc

Таблица 2. Результаты экспериментов на ГПУ NVidia Fermi C2050

Тест	**full-copy**	**auto-sync**	**libgpuvm**	**libgpuvm copy**	**libgpuvm false**
Общее время, μc	20655	1058	1077	25678	1247
Время устройства, μc	1005	998	998	1005	999
Время копирования, μc	18097	10	10	21542	16
Время накладных расходов, μc	1447	44	44	2322	136
Время обработки сигналов, μc	0	0	0.38	538	52
Прочее время, μc	106	6	25	271	44

Ясно, что время исполнения ядра на устройстве не зависит от используемого подхода. В варианте с полным копированием ясно доминирует копирование, в то время как в вариантах с libgpuvm или автосинхронизацией, накладные расходы “распределяются” на большое число итераций, и в среднем доминирует исполнение на устройстве. Это именно то, к чему стремятся разработчики ГПУ-приложений. В целом, накладные расходы libgpuvm по сравнению с автосинхронизацией составляют 1–2%, в основном из-за большего числа операций при установке параметров ядра, примерно на 6 μc на параметр. Но мы считаем, что упрощение интерфейса передачи данных более чем компенсирует дополнительные накладные расходы.

Тест libgpuvm-copy демонстрирует наихудший случай. Накладные расходы вырастают на 25–30% по сравнению с просто полным копированием. Однако лишь 4% составляет прирост за счёт обработки страничных прерываний. Ещё 2–4% занимают прочие расходы, в

Таблица 3. Результаты экспериментов на ГПУ AMD Radeon HD5830

Test	**full-copy**	**auto-sync**	**libgpuvm**	**libgpuvm copy**	**libgpuvm false**
Общее время, μc	23273	1399	1436	30268	2367
Время устройства, μc	1322	1321	1321	1322	1321
Время копирования, μc	17549	7	8	17877	82
Время накладных расходов, μc	4330	69	82	10181	877
Время обработки сигналов, μc	0	0	0.23	446	33
Прочее время, μc	72	2	24	442	54

частности, выполнение вызова `mprotect()` на матрице размером 64МБ, и связанная с этим очистка кэша памяти и кэша трансляции. Оставшиеся 15–25% — это дополнительные расходы на исполнение команд OpenCL: хотя объём копирования тот же, количество команд составляет 18 с использованием libgpuvm, и 8 просто с полным копированием. В случае AMD это отражается как рост накладных расходов, в случае NVidia — как рост времени копирования. Может быть, это связано с различными реализациями копирования, или с различной трактовкой “времени исполнения копирования на устройстве”. Однако ясно, что в этом случае большинство накладных расходов связано с реализацией OpenCL, а не с libgpuvm напрямую.

Тест libgpuvm-false представляет промежуточный случай, когда накладные расходы связаны только с копированием небольшого массива. Дополнительные расходы для ГПУ NVidia составляют 15% по сравнению с использованием данных только на ГПУ. Половина этого связана с libgpuvm, а половина с реализацией OpenCL. В случае AMD накладные расходы составляют 60%, большая часть которых составляет время инициализации копирования OpenCL. Это совершенно точно является проблемой реализации OpenCL, и мы ожидаем, что в следующих версиях AMD APP SDK проблема будет решена, и накладные расходы упадут до уровня реализации NVidia. Но даже с таким уровнем накладных расходов использование libgpuvm даёт на порядок лучшие результаты, чем полное копирование.

5. Обсуждение и дальнейшая работа

Мы предложили подход к обеспечению абстракции разделяемой памяти между хостом и ГПУ. Наше решение поддерживает динамически выделенную хост-память и может работать с ведущими реализациями OpenCL в многопоточной среде с несколькими ГПУ. Продемонстрирована интеграция libgpuvm в NUDA для обеспечения более удобного обмена данными между хостом и ГПУ. Проведённые эксперименты показали, что даже в случае некоторого ложного совместного использования подход libgpuvm на порядок быстрее, чем полное копирование. А в маловероятном худшем случае большая часть накладных расходов связана с реализацией OpenCL, а не с обработкой сигналов.

Возможно несколько направлений будущего развития. Прежде всего, это реализация поддержки других интерфейсов взаимодействия с ГПУ, прежде всего CUDA. Во-вторых,

это поддержка использования ГПУ-памяти как кэша для хост-данных. Это упростит интеграцию нашей библиотеки в существующие языки. В-третьих, это более эффективная поддержка многопоточных приложений, с копированием данных до снятия защиты. Наконец, это более эффективная поддержка различных паттернов совместного использования данных хостом и ГПУ. Например, это использование данных на ГПУ только на чтение, что достаточно типично, или же использование на хосте только некоторых порций ГПУ-массива.

Разработанная библиотека может быть использована для более удобной и эффективной организации взаимодействия с ГПУ в прикладных библиотеках и системах времени выполнения в реализациях языков программирования. В связи с появлением более высокоуровневых инструментов работы с ГПУ становится актуальным.

Литература

1. NVidia Corporation. NVidia CUDA C Programming Guide, Version 4.0, 2011. — 05.
2. Khronos Group. The OpenCL Specification, version 1.2, document revision 15, 2011. — 11.
3. The Portlang Group. PGI Accelerator Programming Model for Fortran & C, 2010. — 11. http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf.
4. Caps Enterprise. CAPS HMPP Workbench User Guide, Version 2.3.1, 2010. — 06.
5. OpenACC Application Programming Interface, Version 1.0, 2011. — 11. http://www.openacc-standard.org/Downloads/OpenACC.1.0.pdf.
6. Adinetz A. V. libgpuvm project on GitHub. 2011. — 12. http://github.com/canonizer/libgpuvm.
7. Gelado I., Stone J. E., Cabezas J. et al. http://dx.doi.org/http://doi.acm.org/10.1145/1736020.1736059An asymmetric distributed shared memory model for heterogeneous parallel systems // Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems. ASPLOS '10. New York, NY, USA: ACM, 2010. Pp. 347–358. http://doi.acm.org/10.1145/1736020.1736059.
8. Adinetz A. V. Extran and NUDA Programmer's Guide. 2011. — 01. http://sourceforge.net/projects/nuda/files/0.0.5/extran-guide.pdf/download.
9. enable writing to /proc/pid/mem. http://lwn.net/Articles/435018/.
10. Skalski K., Moskal M., Olszta P. Meta-programming in Nemerle.
11. Boehm H., Demers A., Shenker S. Mostly Parallel Garbage Collection // Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation. SIGPLAN Notices 26. New York, NY, USA: ACM, 1991. — 06. Pp. 157–164.

Алгоритмы решения обратных геофизических задач на многопроцессорных вычислительных системах*

Е.Н. Акимова[1], Д.В. Белоусов[1], В.Е. Мисилов[2]

Институт математики и механики УрО РАН[1], Уральский федеральный университет[2]

Для решения обратных задач гравиметрии предложены устойчивые параллельные алгоритмы на основе итерационных методов градиентного типа. Для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки построены параллельные методы матричной прогонки, квадратного корня и метод сопряженных градиентов с предобуславливателем. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel с использованием новых вычислительных технологий. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

1. Введение

Важнейшими задачами исследования структуры земной коры являются обратные задачи гравиметрии: задача гравиметрии о нахождении переменной плотности в слое [1] и структурная задача гравиметрии о восстановлении поверхности раздела между средами [2]. Задачи гравиметрии описываются линейными и нелинейными интегральными уравнениями Фредгольма первого рода, т.е. являются существенно некорректными задачами. При разработке методов решения задач используются идеи итеративной регуляризации [3]. После дискретизации с использованием итерационных процессов задачи сводятся к системам линейных алгебраических уравнений (СЛАУ) с плохо обусловленными заполненными матрицами большой размерности (несколько сотен тысяч). Необходимость повышения точности результатов решения задач, в частности использование более мелких сеток, существенно увеличивает время вычислений.

Важнейшими задачами исследования неоднородности земной коры являются задачи электроразведки. Одним из известных методов электроразведки является метод вертикального электрического зондирования (ВЭЗ). После использования конечно-разностной аппроксимации задача ВЭЗ сводится к решению СЛАУ с блочно-трехдиагональной матрицей [4]. Другой важной задачей электроразведки является задача бокового каротажного зондирования (БКЗ). В результате интерпретации данных каротажа получают значение удельного электрического сопротивления пласта, близкое к истинному. В работе [5] показано, что после использования конечно-разностной аппроксимации задача БКЗ сводится к решению СЛАУ с блочно-трехдиагональной матрицей большой размерности.

Одним из путей уменьшения времени расчетов и повышения эффективности решения геофизических задач является распараллеливание алгоритмов и использование многопроцессорных вычислительных систем (МВС). В научно-исследовательских институтах и университетах России распространены массивно-параллельные суперкомпьютеры кластерного типа с распределенной памятью. В Институте математики и механики УрО РАН (г. Екатеринбург) установлены МВС-1000/17ЕК, МВС-ИММ и суперкомпьютер «Уран» (http://parallel.uran.ru/node/6), которые успешно используются при решении прикладных задач.

В настоящее время в мире для решения прикладных задач наметилась тенденция к использованию в качестве вычислительных систем многоядерных гибридных вычислителей с графическими процессорами (видеокартами). По сравнению с суперкомпьютерами МВС гибридные вычислительные системы на основе графических процессоров представляют собой более деше-

* Работа выполнена при поддержке УрО РАН в рамках программ фундаментальных исследований Президиума РАН № 15 (проект 12-П-1-1023) и № 18 (проект 12-П-15-2019).

вую многопроцессорную технику с низким энергопотреблением. В Институте математики и механики УрО РАН установлен гибридный вычислительный кластер на основе видеоускорителей NVIDIA Tesla (URL: http://parallel.uran.ru/news).

В данной работе для решения линейной обратной задачи гравиметрии о нахождении плотности в слое итерационными методами градиентного типа и нелинейной обратной задачи гравиметрии о восстановлении поверхности раздела между средами с помощью итеративно регуляризованного метода Ньютона, а также для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки построены эффективные параллельные прямые и итерационные алгоритмы. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel. Проведено исследование эффективности и оптимизация параллельных алгоритмов для решения геофизических задач на гибридных вычислительных системах. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

2. Методы решения обратных задач гравиметрии

2.1 Задача о нахождении плотности в слое

Рассматривается обратная задача гравиметрии о нахождении переменной плотности $\sigma = \sigma(x, y)$ в слое $\Pi = \{(x, y, z) \in R^3 : (x, y) \in D,\ H_1(x, y) \le z \le H_2(x, y)\}$ по гравитационным данным, измеренным на площади $D = \{(x, y) \in R^2 : a \le x \le b,\ c \le y \le d\}$ земной поверхности ($H_1, H_2 -$ константы для горизонтального слоя). Используется априорная информация об отсутствии аномалий плотности вне слоя с границами $H_1 = H_1(x, y)$ и $H_2 = H_2(x, y)$ такими, что $H_1 < H_2\ \forall (x, y)$, и выполняется условие $H_i(x, y) \underset{\substack{x\to\pm\infty \\ y\to\pm\infty}}{\to} h_i = const$. При этом предполагается, что распределение плотности $\sigma(x, y)$ внутри слоя не зависит от z (ось z направлена вниз). Задача нахождения неизвестной плотности сводится к решению линейного двумерного интегрального уравнения Фредгольма первого рода [1]

$$A\sigma \equiv f \int_a^b \int_c^d \left\{ \frac{1}{\left[(x-x')^2 + (y-y')^2 + H_1^2(x', y') \right]^{1/2}} - \frac{1}{\left[(x-x')^2 + (y-y')^2 + H_2^2(x', y') \right]^{1/2}} \right\} \sigma(x', y')\, dx'dy' = \Delta g(x, y), \tag{1}$$

где $f -$ гравитационная постоянная, $\Delta g(x, y) -$ гравитационный эффект, порождаемый источниками в горизонтальном или криволинейном слое.

Предварительная обработка гравитационных данных, связанная с выделением аномального поля, выполняется по методике, предложенной П.С. Мартышко и И.Л. Пруткиным [6].

После дискретизации уравнения на сетке, где задана $\Delta g(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам задача (1) сводится к решению системы линейных алгебраических уравнений (СЛАУ) либо с симметричной положительно определенной матрицей (горизонтальный слой), либо с несимметричной матрицей (криволинейный слой). Так как уравнение (1) относится к классу некорректно поставленных задач, то СЛАУ, возникающее в результате дискретизации уравнения, является плохо обусловленной и преобразуется к виду (схема Лаврентьева)

$$(A + \alpha E)z = b, \tag{2}$$

где $\alpha -$ параметр регуляризации.

В случае криволинейного слоя исходная матрица СЛАУ несимметрична, поэтому система предварительно преобразуется к виду (схема Тихонова)

$$(A^T A + \alpha' E)z = A^T b. \tag{3}$$

где A^T – транспонированная матрица, α' – параметр регуляризации.

Для решения систем уравнений (2) и (3) используются итерационные методы градиентного типа: метод простой итерации (МПИ)

$$z^{k+1} = z^k - \frac{1}{\lambda_{\max}}[(A + \alpha E)z^k - b], \tag{4}$$

где $\lambda_{\max}$ – максимальное собственное значение матрицы $A + \alpha E$ (симметричный случай);
метод минимальных невязок (ММН)

$$z^{k+1} = z^k - \frac{(A(Az^k - b), Az^k - b)}{\left\|A(Az^k - b)\right\|^2}(Az^k - b); \tag{5}$$

метод минимальной ошибки (ММО)

$$z^{k+1} = z^k - \frac{\left\|Az^k - b\right\|^2}{\left\|A^T(Az^k - b)\right\|^2} A^T(Az^k - b); \tag{6}$$

и метод наискорейшего спуска (МНС)

$$z^{k+1} = z^k - \frac{\left\|A^T Az^k - A^T b\right\|^2}{\left\|A(A^T Az^k - A^T b)\right\|^2} A^T(Az^k - b). \tag{7}$$

Для методов (5) – (7) в регуляризованном варианте матрица A заменяется на $A + \alpha E$. Условием останова итерационных процессов является «останов по невязке»: $\left\|Az^k - b\right\| / \left\|b\right\| < \varepsilon$.

Для решения линейной обратной задачи гравиметрии о восстановлении плотности в слое устойчивые итерационные методы градиентного типа численно реализованы на многопроцессорном комплексе МВС-ИММ с помощью библиотеки MPI [7] и графических процессорах NVIDIA с помощью технологии CUDA [8].

Распараллеливание итерационных методов градиентного типа основано на разбиении матрицы A горизонтальными полосами на m блоков, а вектора решения z и вектора правой части b СЛАУ на m частей так, что $n = m \times L$, где n – размерность системы уравнений, m – число процессоров, L – число строк матрицы в блоке (рис.1). На текущей итерации каждый из m процессоров вычисляет свою часть вектора решения. В случае умножения матрицы A на вектор z каждый из m процессоров умножает свою часть строк матрицы A на вектор z. В случае матричного умножения $A^T A$ каждый из m процессоров умножает свою часть строк транспонированной матрицы A^T на всю матрицу A. Host-процессор (ведущий) отвечает за пересылки данных и также вычисляет свою часть вектора решения.

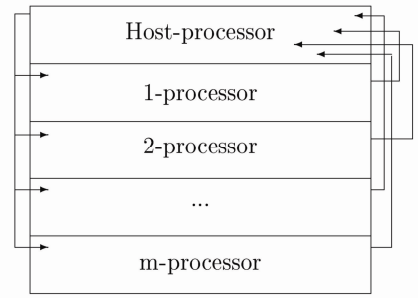


Рис. 1. Схема распределения данных по процессорам

Распараллеливание и численная реализация итеративно регуляризованного МПИ на видеоускорителях NVIDIA при решении линейной задачи гравиметрии описаны в работе [9]. В данной работе эти принципы распространяются на распараллеливание и реализацию на видеоускорителях NVIDIA итерационных методов ММН, ММО и МНС.

Для оптимизации работы с памятью при вычислениях используются два приема.

1. Для сеток не очень большой размерности (размер СЛАУ 12100×12100), когда данные входят в память видеокарты, матрица A порядка n и вектор z размерности n расширяются до размерности M и дополняются нулями таким образом, чтобы M было кратно числу блоков. Размер блока BLOCK_SIZE (threads) выбирается кратным 16, поскольку в одном блоке группируются до 512 потоков. Тогда количество блоков вычисляется по формуле: blocks=M/BLOCK_SIZE. Вычисления производятся без выгрузки данных в память Host-процессора. Данные находятся только в памяти видеокарты.

2. Для сеток довольно большой размерности (размер СЛАУ 40000×40000), когда данные не входят в память видеокарты, наилучшим по быстродействию оказывается метод вычисления элементов матрицы А «на лету», т.е. вычисление значения элемента матрицы происходит в момент обращения к этому элементу без сохранения его в память видеокарты. Это позволяет существенно снизить количество обращений к памяти видеокарты и заметно ускорить процесс вычислений по сравнению с хранением матрицы А в памяти Host-процессора и порционной загрузкой в видеоускоритель для вычислений.

2.2 Задача о нахождении поверхности раздела между средами

Рассматривается трехмерная структурная обратная задача гравиметрии о восстановлении поверхности раздела между средами по известному скачку плотности и гравитационному полю, измеренному на некоторой площади земной поверхности.

Предполагается, что нижнее полупространство состоит из слоев постоянной плотности, разделенных искомыми поверхностями S_i. В предположении, что гравитационная аномалия создана отклонением искомой поверхности S от горизонтальной плоскости $z = H$ (ось z направлена вниз), в декартовой системе координат функция $z = z(x, y)$, описывающая искомую поверхность раздела, удовлетворяет нелинейному двумерному интегральному уравнению Фредгольма первого рода

$$A[z] \equiv f\Delta\sigma \int_a^b \int_c^d \left\{ \frac{1}{\left[(x-x')^2 + (y-y')^2 + z^2(x',y') \right]^{1/2}} - \frac{1}{\left[(x-x')^2 + (y-y')^2 + H^2 \right]^{1/2}} \right\} dx'dy' = G(x,y), \qquad (8)$$

где f – гравитационная постоянная, $\Delta\sigma$ – скачок плотности на границе раздела сред, $G(x, y)$ – аномальное гравитационное поле, $z = H$ – асимптотическая плоскость для данной границы раздела. Задача гравиметрии (8) является существенно некорректной задачей.

После дискретизации уравнения (8) на сетке $n = M \times N$, где задана $G(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам имеем систему нелинейных уравнений

$$A_n[z] = F_n. \qquad (9)$$

Для решения системы уравнений (9) используется итеративно регуляризованный метод Ньютона [10]

$$z^{k+1} = z^k - \left[A_n'\left(z^k\right) + \alpha_k I \right]^{-1} \left[A_n\left(z^k\right) + \alpha_k z^k - F_n \right]. \qquad (10)$$

Здесь $A_n(z^k)$ и F_n – конечномерные аппроксимации интегрального оператора и правой части в уравнении (8), $A_n'(z^k)$ – производная оператора A в точке z^k, I – единичный оператор, α_k – последовательность положительных параметров регуляризации.

Нахождение очередного приближения метода Ньютона z^{k+1} по найденному z^k сводится к решению СЛАУ

$$A_n{}^k z^{k+1} = F_n{}^k, \tag{11}$$

где $A_n{}^k = A_n'\left(z^k\right) + \alpha_k I$ – плохо обусловленная несимметричная заполненная $n \times n$ матрица, $F_n{}^k = A_n{}^k z^k - \left(A_n\left(z^k\right) + \alpha_k z^k - F_n\right)$ – вектор размерности n.

Предварительно система уравнений (11) приводится к виду

$$B^k z^{k+1} \equiv \left[(A_n^k)^T A_n^k + \alpha'_k I\right] z^{k+1} = (A_n^k)^T F_n^k \equiv b, \tag{12}$$

где $(A_n^k)^T$ – транспонированная матрица, α'_k – параметры регуляризации.

На каждом шаге метода Ньютона для решения СЛАУ (12) с симметричной положительно определенной матрицей используются итерационные методы градиентного типа (4) – (7) и метод сопряженных градиентов (МСГ) в регуляризованном варианте

$$z^{k+1} = z^k - \gamma_k (B^k z^k - b) + \beta_k (z^k - z^{k-1}), \tag{13}$$

где γ_k и β_k вычисляются по известным формулам [11]. Условием останова итерационного процесса МСГ является следующее: $\left\|B^k z^k - b\right\| / \left\|b\right\| < \varepsilon$.

Численная реализация и распараллеливание метода Ньютона с использованием градиентных методов для решения нелинейной обратной задачи гравиметрии о нахождении поверхности раздела между средами выполнены на МВС-ИММ. В ближайшее время предполагается реализация метода Ньютона на графических процессорах NVIDIA.

3. Параллельные алгоритмы решения СЛАУ с блочно-трехдиагональными матрицами

Как упоминалось во введении, задачи электроразведки (ВЭЗ и БКЗ) после конечно-разностной аппроксимации [4,5] сводятся к решению СЛАУ с блочно-трехдиагональными матрицами, представленными на рис.2.

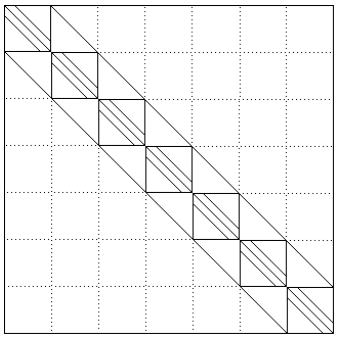


Рис. 2. Вид матрицы СЛАУ

Для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки используются параллельные алгоритмы матричной прогонки (ПАМП), квадратного корня (ПАКК) и метод сопряженных градиентов с предобуславливателем (ПМСГ) в случае решения СЛАУ с симметричной положительно-определенной матрицей.

Параллельные алгоритмы реализованы на графических процессорах NVIDIA с помощью технологии CUDA и многоядерном процессоре Intel с помощью технологии OpenMP [7].

3.1 Параллельный алгоритм матричной прогонки

Рассмотрим систему уравнений с блочно-трехдиагональными матрицами общего вида

$$\begin{cases} C_0\overline{Y}_0 - B_0\overline{Y}_1 = \overline{F}_0, & i = 0 \\ -A_i\overline{Y}_{i-1} + C_i\overline{Y}_i - B_i\overline{Y}_{i+1} = \overline{F}_i, & i = 1,...,N-1 \\ -A_N\overline{Y}_{N-1} + C_N\overline{Y}_N = \overline{F}_N, & i = N, \end{cases} \tag{14}$$

где $\overline{Y}_i$ – искомые векторы размерности *n*, $\overline{F}_i$ – заданные векторы размерности *n*, A_i, B_i, C_i – квадратные матрицы порядка *n*.

В работе [12] для решения СЛАУ (14) предложен параллельный алгоритм матричной прогонки. Основная идея параллельного алгоритма заключается в следующем. Исходную область *P* (прямоугольник) разобьем на *L* подобластей вертикальными линиями так, что $N = L \times M$.
В качестве параметрических неизвестных выберем векторы $\overline{Y}_K$, $K = 0, M, ..., N$, связывающие неизвестные на сетке по вертикали. Относительно $\overline{Y}_K$ строится редуцированная система уравнений меньшей размерности по сравнению с исходной, которая решается классическим методом матричной прогонки [13]. После нахождения векторов-параметров $\overline{Y}_K$ остальные искомые неизвестные выражаются через параметрические неизвестные и находятся в каждой подобласти *L* независимо.

3.2 Метод сопряженных градиентов с предобуславливателем

Для решения СЛАУ (14) можно использовать эффективный метод сопряженных градиентов с предобуславливателем. Введение предобуславливания применяется с целью ускорения сходимости итерационного процесса и состоит в том, что исходная система уравнений $Ax = b$ заменяется на систему

$$C^{-1}Ax = C^{-1}b, \tag{15}$$

для которой итерационный метод сходится существенно быстрее.

Условием выбора предобуславливателя C является условие

$$cond(\tilde{A}) << cond(A), \quad cond(\tilde{A}) = \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}, \quad cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{16}$$

где $cond(A)$ и $cond(\tilde{A})$ – числа обусловленности матриц A и $\tilde{A}$; $\lambda_{\max}$, $\tilde{\lambda}_{\max}$ и $\lambda_{\min}, \tilde{\lambda}_{\min}$ – наибольшее и наименьшее собственные значения матриц A и $\tilde{A}$, соответственно.

Для системы уравнений (15) метод сопряженных градиентов с предобуславливателем C имеет следующий вид

$$\begin{gathered} r^0 = b - Ax^0, \qquad p^0 = C^{-1}r^0, \qquad z^0 = p^0, \\ x^{k+1} = x^k + \alpha_k p^k, \quad \alpha_k = \frac{(r^k, z^k)}{(Ap^k, p^k)}, \quad r^{k+1} = r^k - \alpha_k Ap^k, \\ z^{k+1} = C^{-1}r^{k+1}, \quad p^{k+1} = z^{k+1} + \beta_k p^k, \quad \beta_k = \frac{(r^{k+1}, z^{k+1})}{(r^k, z^k)}. \end{gathered} \tag{17}$$

Условием останова итерационного процесса МСГ с предобуславливателем является «останов по невязке».

Распараллеливание МСГ аналогично распараллеливанию других итерационных методов градиентного типа (см. рис. 1).

3.3 Метод квадратного корня

Одним из быстрых методов решения СЛАУ с симметричной положительно определенной матрицей является метод квадратного корня [11]. Метод основан на разложении симметричной матрицы A в произведение $A = S^T S$, где S – верхняя треугольная матрица с положительными элементами на главной диагонали, S^T – транспонированная матрица. Метод состоит в последовательном решении двух систем уравнений с треугольными матрицами

$$S^T y = b, \quad Sz = y. \tag{18}$$

Решения систем уравнений (18) находятся по рекуррентным формулам

$$\begin{cases} y_1 = b_1 / s_{11}, \\ y_i = \dfrac{b_i - \sum\limits_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2,3,....,n; \end{cases} \quad \begin{cases} z_n = y_n / s_{nn}, \\ z_i = \dfrac{y_i - \sum\limits_{k=i+1}^{n} s_{ik} z_k}{s_{ii}}, \ i = n-1, n-2, ..., 1. \end{cases} \tag{19}$$

Основная идея распараллеливания метода квадратного корня для многопроцессорного вычислителя с общей памятью основана на параллельном вычислении элементов s_{ij}, $j = i,...,n$, каждой $i-$ой строки матрицы S. Строка с номером i разбивается на m частей так, что $n - i = m \times L_i$, где $i-$номер строки, $n-$размерность системы уравнений, $m-$число процессоров, L_i-число элементов строки, вычисляемых каждым процессором (рис. 3).

1 – процессор	2 – процессор	…	m – процессор

Рис. 3. Разбиение $i-$ой строки по процессорам

4. Специализированный Веб-портал решения задач на многопроцессорных вычислителях

Параллельные алгоритмы решения обратной задачи гравиметрии о восстановления плотности в слое и структурной задачи гравиметрии о восстановлении поверхности раздела между средами, а также параллельные алгоритмы решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях», установленный в Отделе некорректных задач анализа и приложений Института математики и механики УрО РАН (рис.4).

Изначально Веб-портал был предназначен для запуска программ решения задач гравиметрии на многопроцессорном комплексе МВС-1000/17ЕК через Веб-интерфейс [14]. В настоящее время на Веб-портале предусмотрен запуск программ для решения задач на МВС-ИММ, гибридном вычислительном кластере NVIDIA Tesla, установленном в ИММ УрО РАН и гибридной вычислительной системе (ГВС) NVIDIA GeForce, установленной на кафедре ВМ и УМФ ИРИТ-РтФ УрФУ.

МВС-ИММ состоит из 14 2-х процессорных 2-х ядерных модулей AMD Opteron 64 bit (2.6 Ггц), интерфейса GbitEthernet и 112 Гб оперативной памяти. Кластер NVIDIA Tesla включает 20 вычислительных узлов, имеющих 8 GPU Tesla S2050, 50 Гб ОЗУ и 2 шестиядерных CPU. ГВС представляет собой 4-ядерном процессор Intel Core I7-950 с графическими процессорами NVIDIA GeForce GTX 480.

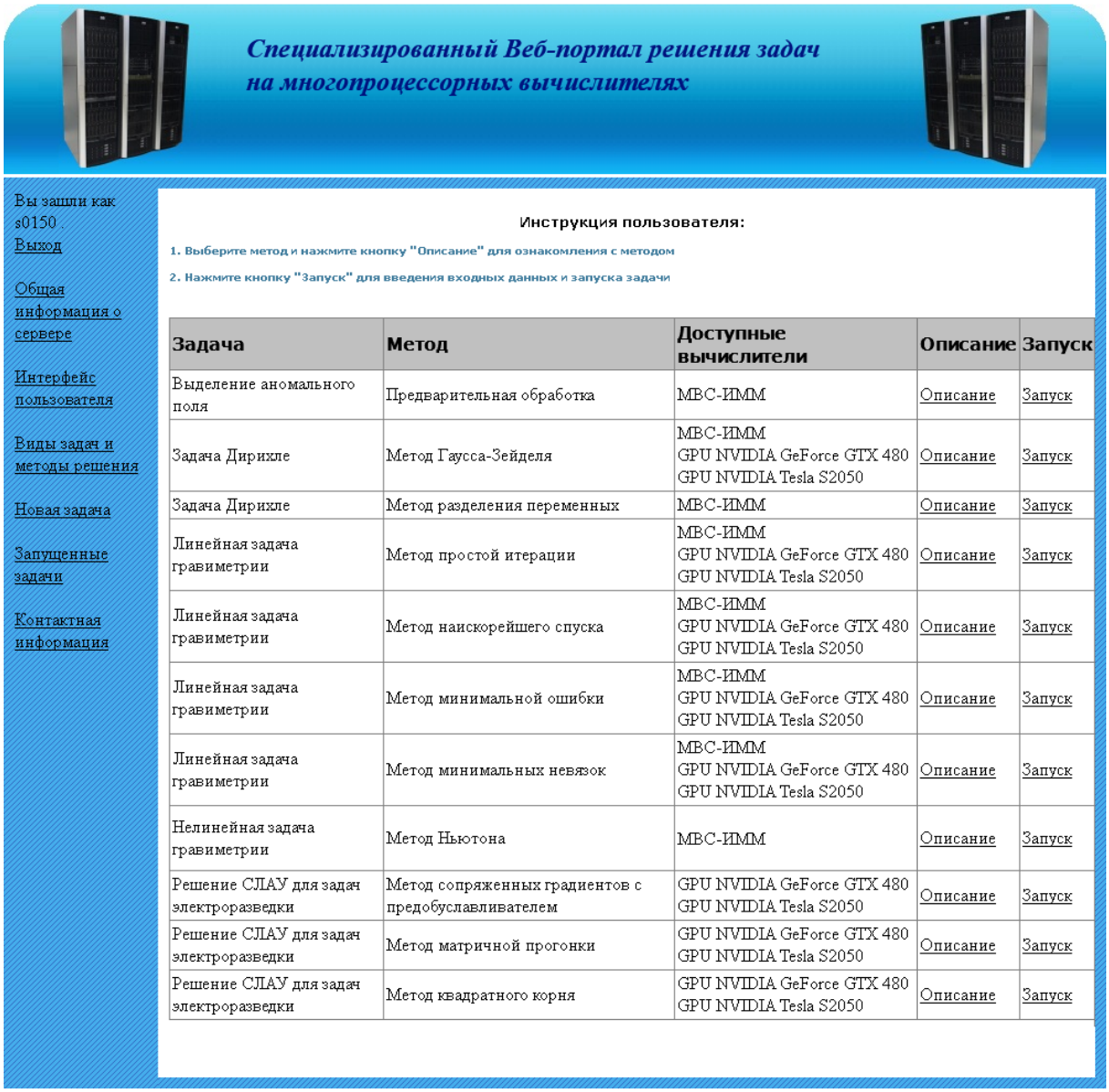


Рис. 4. Специализированный Веб-портал

4.1 Общая характеристика

Специализированный Веб-портал предназначен для запуска программ решения задач гравиметрии (выделение аномального поля, нахождения плотности в слое, восстановления поверхности раздела между средами) и решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки на многопроцессорном комплексе МВС-ИММ, вычислительном кластере NVIDIA Tesla и системе ГВС NVIDIA GeForce. Веб-портал предоставляет возможность пользователю через Веб-интерфейс выбирать тип вычислителя с указанием числа процессорных узлов, вид задачи и метод ее решения, загружать входные данные, получать выходные данные и графическое изображение результатов решения с помощью графических пакетов Surfer и gnuplot. Для каждой задачи выводится время счета.

Веб-портал состоит из трех основных частей (рис. 5): HTTP-сервер IIS (Internet Information Services – информационные службы Интернета), на котором установлено Веб-приложение; база данных SQL Server 2000, в которой хранятся все задачи пользователей с входными и выходными данными; служба, выполняющая загрузку данных, запуск задач на многопроцессорные вычислители различных типов, просмотр состояния задачи и загрузку результатов завершившихся задач на Веб-портал.

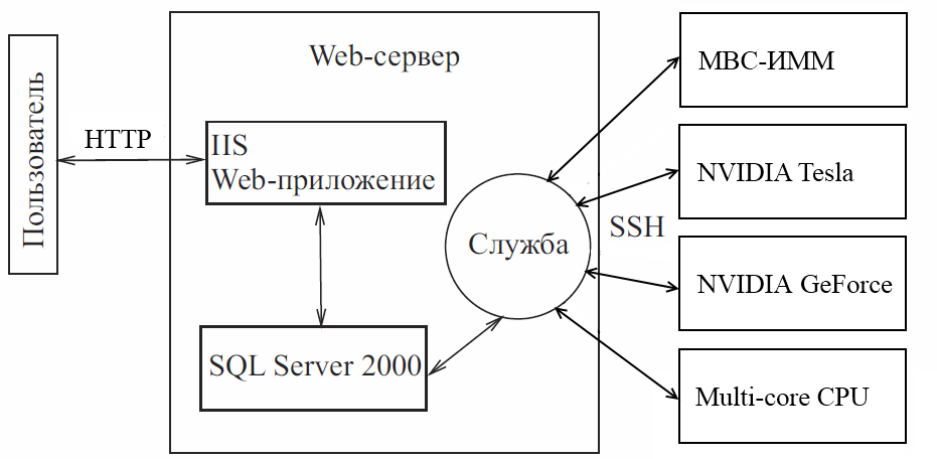


Рис. 5. Архитектура Веб-портала

4.2 Веб-приложение

В настоящее время к исходному Веб-приложению (см. [14]) добавлено следующее.

1. Дополнена система аутентификации. При регистрации на Веб-портале происходит проверка одноименной учетной записи на МВС-ИММ. При ее отсутствии или несовпадении паролей регистрация отменяется. Реализовано SSL-шифрование трафика.

2. Предусмотрена возможность взаимодействия с разными вычислительными устройствами. Веб-портал позволяет запускать методы решения задач на многопроцессорных вычислителях различного типа, поддерживающих связь по **ssh**, передачу файлы по **scp** и запуск программ через планировщик **mpirun** или **sbatch**.

3. Модифицированы службы, отвечающие за прием и передачу файлов и запуск задач на вычислителях различного типа. В интерфейс запуска задачи добавлен выбор вычислителя, на котором эта задача будет запущена. В интерфейсе просмотра запущенных задач добавлено отображение вычислителя, на котором была запущена выбранная задача. Добавлена возможность удаления запущенной задачи из списка задач.

4. Добавлено управление загруженными файлами данных для вычислений, а именно, возможность запуска другой задачи с этими же данными с целью снижения нагрузки на сервер (перекачки файлов). Расширены наборы параметров задачи. Например, для решения линейных задач гравиметрии имеются две возможности – введение двух констант (границ) для горизонтального слоя и загрузка двух файлов (границ) для криволинейного слоя.

5. Подключено управление созданием изображений: пользователь может выбрать, из каких входных и выходных файлов строить изображения, которые будут видны на странице с конкретной решенной задачей. Для задач гравиметрии строится трехмерная поверхность и линии уровня с помощью программы Scripter пакета Surfer.

6. Изменен дизайн страниц, добавлены подробные описания задач и методов. Реализована система управления контентом, а именно, возможность редактирования на сайте описания методов решения задач и добавления новых.

5. Результаты численных экспериментов

5.1 Решение задач гравиметрии с реальными данными

Для восточной части Урала был обработан массив гравитационных данных, измеренный на площади S, имеющей размеры 59.4×144 км^2. Эта площадь пространственно совпадает с зоной Буткинской аномалии векового хода, ограничивающей Урал с востока. Зона представляет собой субмеридиональную аномалию электропроводности земной коры. Аномальное гравитационное поле предоставлено сотрудниками Института геофизики УрО РАН (г. Екатеринбург).

Для изучения природы аномалии по реальным наблюденным данным решены задача 1 о нахождении плотности в горизонтальном слое между глубинами $H_1 = 10$ км и $H_2 = 20$ км для области S и задача 2 о восстановлении поверхности раздела между средами. При этом шаги сетки $\Delta x = 0.594$ км и $\Delta y = 1.44$ км, гравитационная постоянная $f = 6.67 \cdot 10^{-8}$ см^3/г·с^2. Расстояние до асимптотической плоскости составляло $H = 15$ км. Скачок плотности принимался равным $\Delta\sigma = 0.2$ г/см^3. После дискретизации исходных уравнений на сетке задачи свелись к СЛАУ с симметричной (задача 1) и несимметричной (задача 2) матрицами 10000×10000. Для решения задачи 1 использовался параллельный итеративно регуляризованный ММН с параметром регуляризации $\alpha = 0.001$. Для решения задачи 2 использовался итеративно регуляризованный метод Ньютона, на каждом шаге которого применялся параллельный МСГ.

Задача 1 решена на МВС-ИММ, NVIDIA Tesla и ГВС GeForce. Задача 2 решена на МВС-ИММ. На рис. 6 изображено распределение плотности в слое, восстановленной по аномальному полю для области S. На рис. 7 изображена восстановленная поверхность раздела.

Интерпретация результатов проведена сотрудниками ИГФ УрО РАН (г. Екатеринбург). В результате интерпретации выделен протяженный субмеридиональный блок земной коры пониженной плотности (0.2─0.3 г/см^3) (см. [15]).

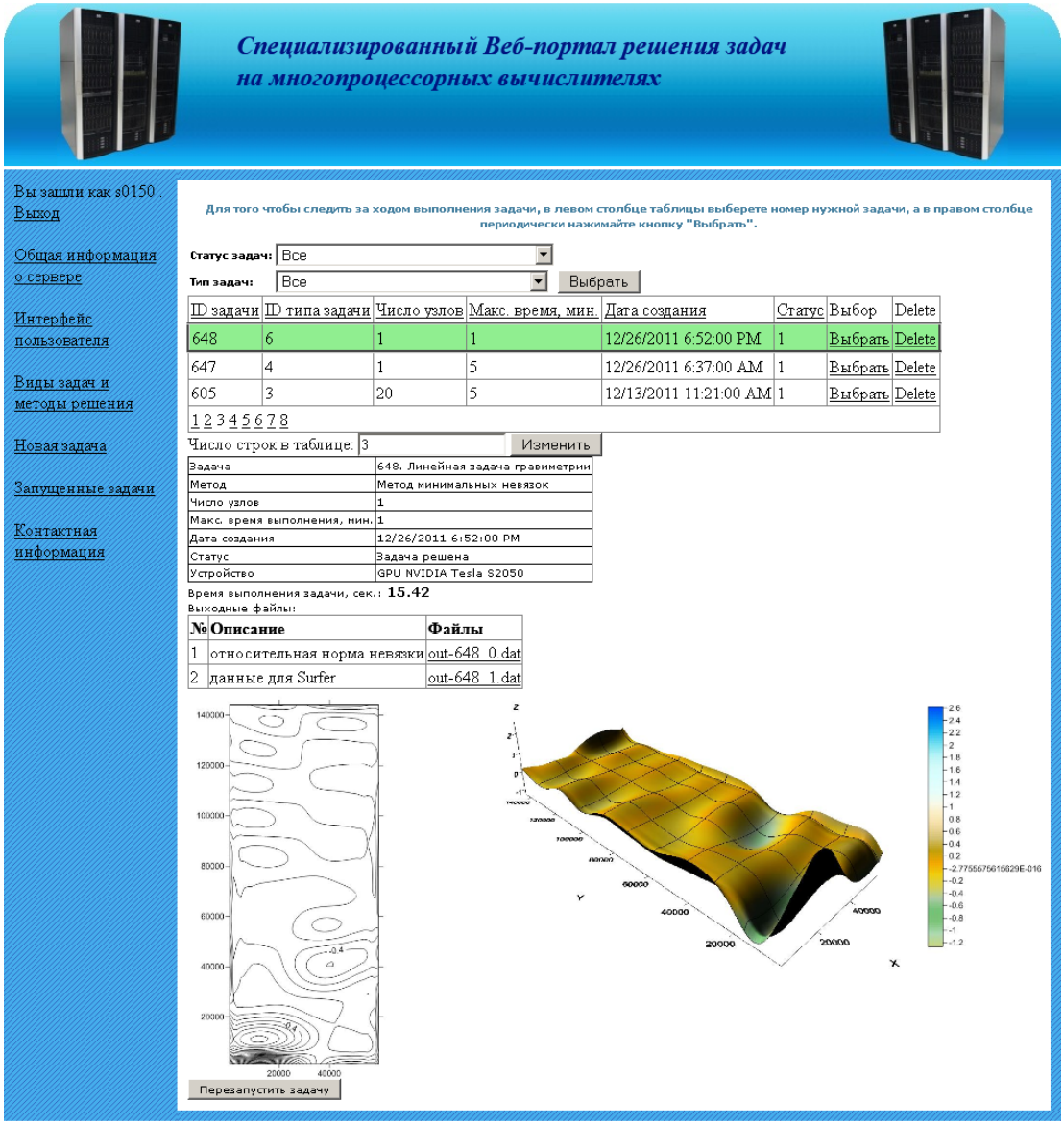


Рис. 6. Решение линейной задачи гравиметрии

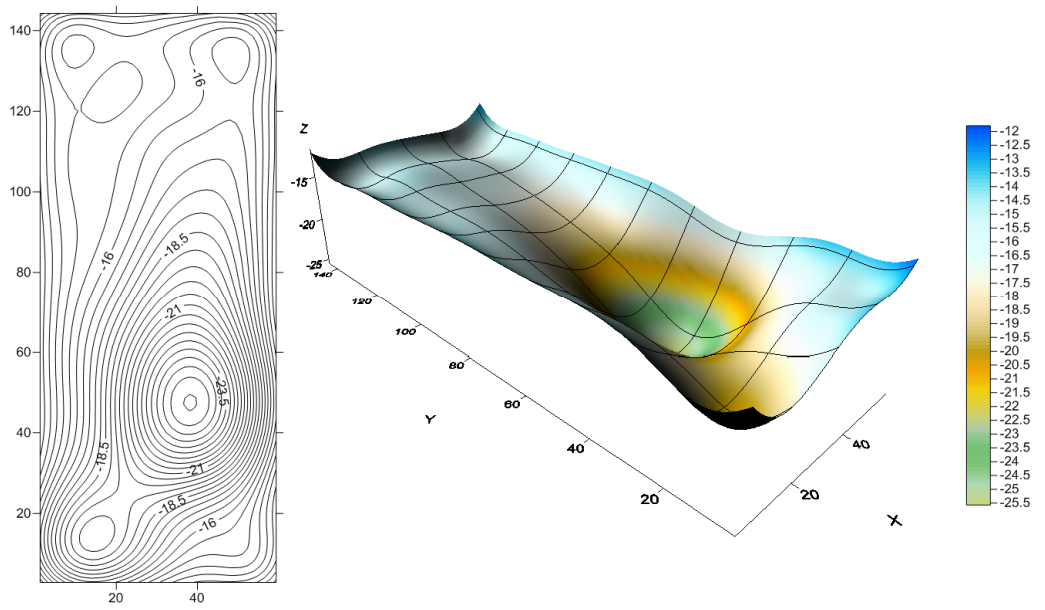


Рис. 7. Решение нелинейной задачи гравиметрии

В табл. 1 приводятся времена решения линейной задачи гравиметрии при $\|Az^k - b\| / \|b\| \approx 0.011$ (749 итераций) на МВС-ИММ и ГВС NVIDIA GeForce GTX 480. Для решения задачи на МВС-ИММ использовалась технологии MPI, для решения задачи на многоядерном процессоре Intel использовалась технология OpenMP, для решения задачи на видеоускорителе GeForce использовалась технология CUDA.

Для сравнения времени счета решения задачи введем коэффициенты ускорения и эффективности параллельных алгоритмов

$$S_m = T_1 / T_m, \quad E_m = S_m / m, \quad S = T_1 / T_2,$$

где T_m – время выполнения параллельного алгоритма на МВС–ИММ либо на многоядерном процессоре с числом процессоров или ядер m $(m > 1)$, T_1 – время выполнения последовательного алгоритма на одном процессоре либо на одном ядре, T_2 – время решения задачи на видеоускорителе.

Таблица 1. Времена решения линейной задачи гравиметрии

Вычислитель	Время T_m, мин.	Ускорение S_m либо S	Эффективность E_m
Intel Core I7 (1 ядро)	7.73	–	–
Intel Core I7 (2 ядра)	4.05	1.91	0.96
Intel Core I7 (4 ядра)	2.1	3.68	0.92
NVIDIA GeForce GTX 480	0.2	38.7	–
МВС-ИММ (1 проц.)	24.33	–	–
МВС-ИММ (2 проц.)	12.18	1.99	0.99
МВС-ИММ (4 проц.)	6.10	3.99	0.99
МВС-ИММ (10 проц.)	2.46	9.89	0.99
МВС-ИММ (20 проц.)	1.24	19.6	0.98
МВС-ИММ (50 проц.)	0.5	48.7	0.97

Результаты вычислений показывают, что использование метода Ньютона и итерационных методов градиентного типа при решении обратных задач гравиметрии позволяют получать корректные решения и определять аномальные плотностные параметры изучаемых глубинных зон земной коры. Применение параллельных алгоритмов при решении задач гравиметрии уменьшает время счета.

5.2 Решение задачи о нахождении распределения потенциала

С помощью параллельного алгоритма матричной прогонки, предобусловленного метода сопряженных градиентов и метода квадратного корня решена задача о нахождении распределения потенциала в проводящей среде с известным квази-модельным решением.

Исходные данные и квази-модельное решение задачи предоставлены лабораторией скважинной геофизики Института нефтегазовой геологии и геофизики СО РАН (г. Новосибирск).

После дискретизации задача сводится к решению СЛАУ с плохо обусловленной симметричной положительно-определенной блочно-трехдиагональной матрицей вида размерности 76136×76136 с квадратными блоками порядка 248.

Приближенное решение задачи сравнивалось с модельным решением с помощью вычисления относительной погрешности

$$\sigma = \left\| \overline{Y}^{M} - \overline{Y}^{\Pi} \right\| / \left\| \overline{Y}^{M} \right\|,$$

где $\overline{Y}^{M}$ – модельное решение задачи, $\overline{Y}^{\Pi}$ – приближенное решение задачи.

Условие $\sigma < \varepsilon$ выбиралось в качестве критерия останова итерационного ПМСГ при решении задачи. Предварительно находилось число обусловленности исходной матрицы A

$$cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \approx 1.3 \cdot 10^{11}, \quad \lambda_{\max} \approx 1.4 \cdot 10^{6}, \quad \lambda_{\min} \approx 1.1 \cdot 10^{-5} > 0,$$

где $\lambda_{\max}$ и $\lambda_{\min}$ – наибольшее и наименьшее собственные значения исходной матрицы.

В случае решения задачи предобусловленным ПМСГ с целью проверки условия (16) находилось число обусловленности матрицы $\tilde{A}$

$$cond(\tilde{A}) = \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}} \approx 4.1 \cdot 10^{9} < cond(A).$$

Задача решена с помощью параллельного метода сопряженных градиентов с предобуславливателем, параллельного алгоритма матричной прогонки и параллельного метода квадратного корня при $\sigma_{ПМСГ} \approx 10^{-7}$, $\sigma_{ПАМП} \approx 2 \cdot 10^{-7}$, $\sigma_{ПМКК} \approx 6 \cdot 10^{-7}$. На рис. 8 представлено численное решение задачи.

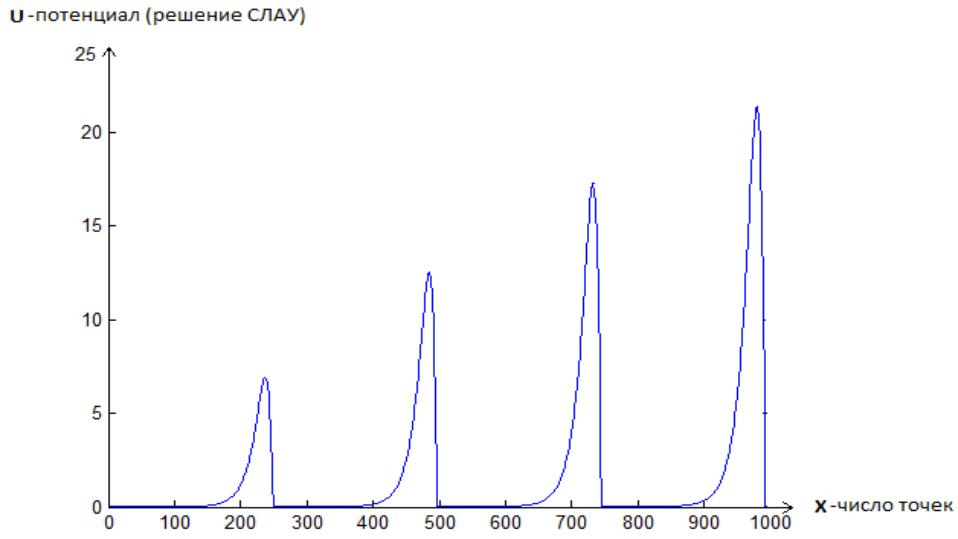


Рис. 8. Численное решение задачи

В табл. 2 приведены времена счета решения задачи на ГВС, установленной в Отделе некорректных задач анализа и приложений ИММ УрО РАН. Система состоит из 4-х ядерного процессора Intel Core I5-750 и видеоускорителя NVIDIA GeForce GTX 285. Предполагается включить данную ГВС в Веб-портал.

Заметим, что время решения задачи с помощью МСГ без предобуславливателя на одном ядре Intel Core I5-750 при $\sigma_{МСГ} = 10^{-3}$ составило 55 мин., что существенно превышает времена решения задачи, представленные в табл. 2.

Таблица 2. Времена решения задачи

Метод	Вычислитель	T_m, сек.(Windows API)	T_m, сек. (OpenMP)
ПМСГ	Intel Core I5 (1 ядро)	57	21
	Intel Core I5 (2 ядра)	46	16
	Intel Core I5 (4 ядра)	36	14
	NVIDIA GeForce GTX 285	28	14
ПАМП	Intel Core I5 (1 ядро)	52	21
	Intel Core I5 (2 ядра)	28	18
	Intel Core I5 (4 ядра)	16	14
	NVIDIA GeForce GTX 285	–	10
ПАКК	Intel Core I5 (1 ядро)	12	7.4
	Intel Core I5 (2 ядра)	9	4.6
	Intel Core I5 (3 ядра)	10	3.8
	Intel Core I5 (4 ядра)	12	4.2
	NVIDIA GeForce GTX 285	–	3

Вначале распараллеливание методов ПМСГ, ПАМП и ПАКК для многоядерного процессора Intel с общей памятью проводилось средствами создания потоков операционной системы (ОС) с использованием средств разработки Windows API [16]. Для параллельного выполнения блока вычислений программы создавались потоки (Threads), каждый из потоков выполнялся на «логическом процессоре» ОС, вычисляя свою порцию данных. В конце каждого блока вычислений производилась барьерная синхронизация потоков. Для оптимизации и уменьшения времени счета была использована технология OpenMP. Средствами библиотеки функций OpenMP с использованием специальных директив компилятора проведено автоматическое распараллеливание циклов. Интервал размера L переменной цикла i разбивался на m частей. Каждый поток процесса вычислял свою p -ую часть данных, где $p = L / m$ (рис. 3).

Для уменьшения времени решения задачи также использовалась технология NVIDIA CUDA. Существенное отличие выполнения программы с использованием CUDA от выполнения программы с помощью OpenMP состоит в том, что программа на CUDA выполняется «в тысячи потоков». Такие программы используют массивно-параллельный принцип программирования. При разработке алгоритмов были перенесены на GPU только «ресурсоемкие» векторно-матричные операции. Принцип работы состоял в следующем. Основной поток программы выполнялся на CPU, данные из оперативной памяти загружались на GPU, где проводились расчеты. Далее результаты расчетов выгружались в оперативную память. Очень часто распараллеливание на GPU не является эффективным ввиду образования «узкого горлышка» пропускной способности памяти.

По времени счета наиболее быстрым является метод ПАКК. Время решения СЛАУ 76136×76136 на 4-ядерном процессоре Intel и видеокарте составляет несколько секунд.

Результаты вычислений показывают, что использование параллельных методов ПАМП, ПАКК и ПМСГ с предобуславливателем позволяет достаточно быстро решать задачи с плохо обусловленными матрицами на многопроцессорных вычислителях и можно рекомендовать данные методы для решения задач электроразведки.

6. Заключение

Для решения линейной обратной задачи гравиметрии о нахождении плотности в слое итерационными методами градиентного типа и нелинейной обратной задачи гравиметрии о восстановлении поверхности раздела между средами с помощью итеративно регуляризованного метода Ньютона, а также для решения СЛАУ с блочно-трехдиагональными матрицами примени-

тельно к задачам электроразведки построены параллельные прямые и итерационные алгоритмы. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

Результаты вычислений показывают, что использование метода Ньютона и параллельных методов градиентного типа при решении обратных задач гравиметрии позволяют получать корректные решения и определять аномальные плотностные параметры изучаемых глубинных зон земной коры. Использование параллельных алгоритмов решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки позволяет достаточно быстро решать задачи с плохо обусловленными матрицами на многопроцессорных вычислителях.

Литература

1. Мартышко П.С., Кокшаров Д.Е. Об определении плотности в слоистой среде по гравитационным данным // Геофизический журнал. 2005. Т. 27. № 4. С. 678–684.
2. Нумеров Б.В. Интерпретация гравитационных наблюдений в случае одной контактной поверхности // ДАН СССР. 1930. № 21. С. 569–574.
3. Васин В.В., Еремин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН. 2005.
4. Тихонов А.Н., Самарский А.А. Уравнения математической физики. М.: Наука, 1966.
5. Дашевский Ю.А., Суродина И.В., Эпов М.И. Квазитрехмерное математическое моделирование диаграмм неосесимметричных зондов постоянного тока в анизотропных разрезах // Сиб. ЖИМ. 2002. Т. 5. № 3 (11). С. 76-91.
6. Мартышко П.С., Пруткин И.Л. Технология разделения источников гравитационного поля по глубине // Геофизический журнал. 2003. Т. 25. № 3. С. 159–68.
7. Воеводин Вл.В. Технологии параллельного программирования. URL: http://parallel.ru/ (дата обращения: 06.02.2012).
8. Берилло А. NVIDIA CUDA – неграфические вычисления на графических процессорах. URL: http://www.ixbt.com/video3/cuda-1.shtml (дата обращения: 06.02.2012).
9. Акимова Е.Н., Белоусов Д.В. Распараллеливание алгоритмов решения линейной обратной задачи гравиметрии на МВС-1000 и графических процессорах // Вестник ННГУ. 2010. № 5. Ч. 1. С. 193–200.
10. Bakushinsky A., Goncharsky A. Ill-Posed Problems: Theory and Applications. London: Kluwer Akad. Publ. 1994.
11. Фаддеев В.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Гос. издат. физ.-мат. литературы. 1963.
12. Акимова Е.Н. Распараллеливание алгоритма матричной прогонки // Математическое моделирование. 1994. Т. 6. № 9. С. 61-67.
13. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978.
14. Акимова Е.Н., Гемайдинов Д.В. Параллельные алгоритмы решения обратной задачи гравиметрии и организация удаленного взаимодействия между МВС-1000 и пользователем // Вычислительные методы и программирование. 2008. Т. 9. № 1. С. 133–144.
15. Мартышко П.С., Васин В.В., Акимова Е.Н., Пьянков В.А. О комплексной интерпретации гравитационных и магнитовариационных данных (на примере Башкирского предуралья) // Геофизика. 2011. № 4. С. 30-36.
16. Методика разработки многопоточных приложений: принципы и практическая реализация. URL: http://www.rsdn.ru/article/baseserv/RUThreadingMethodology.xml (дата обр. 06.02.2012).

Анализ точности численного решения стохастических дифференциальных уравнений на суперкомпьютерах*

С.С. Артемьев, В.Д. Корнеев

Институт вычислительной математики и математической геофизики
Сибирского отделения Российской академии наук,
Новосибирск, Россия

В работе проведены исследования точности оценок функционалов от решений СДУ на суперкомпьютерах. Показано, что для точной оценки математического ожидания и дисперсии решений линейных СДУ с мультипликативным шумом первого и второго порядка, требуется моделирование ансамблей траекторий размера $10^9 \div 10^{12}$, что невозможно осуществить на ПК из-за высокой трудоемкости. Проведены исследования поведения математического ожидания и дисперсии решения нелинейного стохастического уравнения Ван-дер-Поля. Описываются способы распараллеливания статистических алгоритмов на многопроцессорном кластере. Приводятся результаты численных экспериментов, полученных на суперкомпьютере Сибирского суперкомпьютерного центра.

1. Введение

Использование методов Монте-Карло (ММК) для нахождения решений краевых задач математической физики на основе вероятностных представлений часто требует численного решения сопутствующих СДУ и вычисления интегралов вдоль моделируемых траекторий. В настоящей работе исследуется зависимость точности статистических алгоритмов решений простейших СДУ от размера ансамбля моделируемых траекторий и от размера шага интегрирования обобщённого метода Эйлера. Приводятся результаты численных экспериментов по оценке моментов решений СДУ с растущей дисперсией и оценке среднего времени первого выхода траекторий СДУ с мультипликативным шумом на границу заданной области.

Точность оценок функционалов от решений СДУ зависит не только от размеров ансамблей моделируемых траекторий, но и от размера шага интегрирования используемого численного метода решения СДУ. Особую трудность представляет статистическое моделирование стохастических осцилляторов. Математические модели в виде СДУ с осциллирующими решениями возникают в самых разных областях науки [1–3]. Особый интерес представляет анализ возможных переходов от одного типа осцилляций к другому, например, при прогнозировании аварий и катастроф, вызываемых ростом амплитуды колебаний. В связи с этим возникает задача оценки устойчивости заданного режима работы. Как показали ранее проведённые эксперименты [4], при численном решении осциллирующих СДУ зачастую возникает неустойчивость численного решения, т. е. сильный рост дисперсии осцилляций. В связи с этим возникает необходимость уменьшения размера шага интегрирования на несколько порядков. Кроме того, малые объёмы моделируемых траекторий дают совершенно неверные оценки моментов решений в случае сильной асимметрии их плотностей распределения.

Оценки математического ожидания функционалов от решений СДУ, требующие вычисления интегралов вдоль моделируемых траекторий, сверхмалые размеры шага интегрирования обобщённого метода Эйлера и огромные объёмы моделируемых траекторий ведут к необходимости использования суперкомпьютеров с большим количеством процессоров, чтобы получить удовлетворительную точность численного анализа за приемлемое время

*Работа выполнена при финансовой поддержке гранта РФФИ (проект № 11-01-00252).

счета. Первые исследования авторов по численному решению СДУ с растущей дисперсией на суперкомпьютерах были описаны в работе [5].

В настоящей работе помимо численных экспериментов приводятся описания параллельных программ и оценок зависимости времени счета от числа используемых процессоров. Численные эксперименты проводятся на кластере НКС-30Т Сибирского суперкомпьютерного центра при Институте вычислительной математики и математической геофизики СО РАН.

2. Решаемые задачи

Для численного решения задачи Коши для общих СДУ в смысле Ито (японский математик)

$$dy = f(y)\,dt + \sigma(y)\,dw(t), \qquad y(0) = y_0, \quad 0 \le t \le T \tag{1}$$

обычно используется наименее трудоёмкий обобщенный метод Эйлера:

$$y_{n+1} = y_n + h\,f(y_n) + \sigma(y_n)\sqrt{h}\;\xi_{n+1}. \tag{2}$$

Здесь y_{n+1} - численное решение на сетке $t_{n+1} = t_n + h$, $\{\xi_{n+1}\}_0^{N-1}$ - последовательность независимых между собой и с y_n стандартных гауссовских случайных величин. Для моделирования ξ_n используется стандартная формула $\xi = \sqrt{-2\ln\alpha_1}\,\sin 2\pi\alpha_2$, где α_1, α_2 - равномерно распределенные случайные величины в (0,1) [6].

1) Решение СДУ с мультипликативным шумом

ММК часто рекомендуются для нахождения решений краевых задач для эллиптических уравнений с использованием вероятностного представления [7]. Например, для одномерной задачи Дирихле

$$\frac{1}{2}\,\sigma^2(y)\frac{d^2u}{dy^2} + f(y)\frac{du}{dy} + 1 = 0 \tag{3}$$

в интервале $[a,b]$ с граничными условиями $u(a) = u(b) = 0$ решение $u(y_0)$ может быть представлено в вероятностном виде

$$u(y_0) = E\tau(y_0), \tag{4}$$

где τ - время первого выхода решения СДУ (1) из $[a,b]$.

2) Стохастические осцилляторы

а) Мультипликативные шумы зачастую связаны с "шумящими" коэффициентами в ОДУ. Рассматривается случай "шумящих" коэффициентов в линейном колебательном контуре, который задается СДУ второго порядка вида

$$\frac{d^2y}{dt^2} + (\lambda + \sigma_1\,\frac{dw_1}{dt})\,\frac{dy}{dt} + (\beta^2 + \sigma_2\,\frac{dw_2}{dt})\,y = 0 \tag{5}$$

с постоянными λ, β, σ_1, σ_2. В частном случае СДУ (5) вида

$$\frac{d^2y}{dt^2} + (\beta^2 + \frac{dw}{dt})\,y = 0, \qquad y(0) \in N(1,1), \qquad \frac{dy}{dt}(0) = 0, \tag{6}$$

когда "шумит" только частота колебаний решения и отсутствует декремент затухания, математическое ожидание точного решения задаётся простой формулой $m(t) = cos(\beta t)$.

б) Стохастическое нелинейное уравнение Ван-дер-Поля с одним "шумящим" коэффициентом, записанное в виде системы СДУ в смысле Ито вида

$$\begin{aligned} dy_1 &= y_2\,dt, \\ dy_2 &= (\lambda\,y_2\,(1 - d\,y_1^2) - \beta^2 y_1)\,dt + \sigma y_1\,d\omega(t), \end{aligned} \tag{7}$$

описывает колебания нелинейного контура (см., например, [1]). В (7) постоянные λ, d, β определяют скорость переходных участков в решении. Для математического ожидания точного решения СДУ (7) не существует ни явного формульного представления, ни замкнутой системы ОДУ для его численного расчета. Единственным конструктивным способом анализа нелинейных СДУ с большим шумом является ММК.

3. Описание параллельных программ

Описываются два способа распараллеливания алгоритмов на многопроцессорном кластере. Распараллеливание реализуется в системе параллельного программирования MPI [8]. Первый способ распараллеливания связан с оценкой математического ожидания решения СДУ на всем интервале интегрирования и отличается одинаковой продолжительностью моделирования всех траекторий из ансамбля. Поскольку в ММК моделируются независимые реализации решения СДУ, это допускает эффективную организацию их параллельного выполнения на многопроцессорном кластере. Схема распараллеливания в данном случае проста: различные процессоры вычислительной системы осуществляют полностью независимое решение, т.е. вычисляют последовательности, полученные на основе разных (в каждом процессоре) псевдослучайных чисел.

Пусть K - количество процессоров в вычислительной системе, реализующей алгоритм и M - размер ансамбля моделируемых траекторий. Положим $M_k = M/K$ - размер ансамбля на одном процессоре. Тогда формула для оценки математического ожидания решения в узле сетки t_n в параллельном исполнении имеет вид

$$m_n = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{M_k}\sum_{m=1}^{M_k} y_n^{(m,k)}. \tag{8}$$

Здесь $y_n^{(m,k)}$ - значение m -ой реализации решения СДУ в n -ом узле сетки, полученное на k -ом процессоре.

При распараллеливании данного алгоритма временные затраты сведены к минимуму: здесь время, затраченное на финальное осреднение независимых результатов, практически играет небольшую роль [9, 10].

Второй способ распараллеливания связан с моделированием траекторий до их первого выхода на границу заданной области и отличается различной продолжительностью моделирования каждой траектории из ансамбля. При этом фиксируется первый выход моделируемой последовательности $\{y_n\}$ за границу заданной области и фиксируется количество итерационных шагов, осуществленных до этого выхода. Под итерационным шагом здесь понимается вычисление следующего значения y_{n+1}. После чего процесс моделирования начинается заново из точки y_0.

Как и в предыдущем алгоритме, здесь моделируются независимые реализации решения СДУ и это позволяет эффективно организовать их параллельное выполнение. Пусть M^{min} - задаваемое минимальное количество выходов на границу области, определяющее размер ансамбля. Положим $M_k^{min} = M^{min}/K$. Тогда в параллельном исполнении формулу оценки среднего времени до первого выхода траекторий, стартующих из y_0, можно записать в виде

$$\hat{\tau}(y_0) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{M_k^{min}}\sum_{m=1}^{M_k^{min}} n_m^{(k)} h. \tag{9}$$

Здесь $n_m^{(k)}$ - количество итерационных шагов, осуществленных до выхода на границу m -ой реализации на k -ом процессоре. Отметим, что по окончании моделирования число выходов реализаций на границу области может сильно разниться от процессора к процессору. Главное, что бы суммарное число выходов было не меньше M^{min}. Заметим, что в отличие

от первого алгоритма, здесь шаг h должен быть достаточно малым, что бы при фиксации момента выхода траектории за границу области не было большой потери точности оценки.

Схема распараллеливания здесь иная, чем в первом алгоритме, так как необходимо постоянно, через некоторое заданное количество итерационных шагов, сканировать по всем процессорам величины M_k^{min}, вычислять их сумму и проверять на достижение заданного общего минимума M^{min}. Количество итерационных шагов в каждом процессоре до выхода реализации за границы области сильно зависит от параметров в (3), причем приблизительное среднее число таких шагов неизвестно. Частота обменов данными между компьютерами оказывает сильное влияние на время выполнения параллельного алгоритма. Поэтому необходимо оптимизировать взаимодействия процессоров и тем самым минимизировать время решения задач при проведении большого количества численных экспериментов.

В алгоритмах 2 и 3 параллельные вычисления оптимизируются по двум параметрам: вначале а) во всех процессорах вычисляется размер временного интервала (рис.1), через который потом осуществляются межпроцессорные обмены данными и, затем, б) выбираются типы и схемы межпроцессорных обменов.

Интервал *Is* измеряется в количествах итерационных шагов и устанавливается одинаковым на всех процессорах. При вычислении интервала *Is* его размер динамически подстраивается к параметрам решаемой задачи. Поскольку алгоритм решения задачи один и тот же на всех процессорах, то и время выполнения одного итерационного шага будет одинаковым на всех процессорах. На рис. 1 PPi - логические MPI-номера процессоров, фигурными скобками обозначен диапазон для процессов, кривыми линиями обозначены моделируемые траектории, $n_m^{(k)}$ - количество итераций (шагов) до соответствующего выхода на границу, *Is* - интервал, через который процессоры взаимодействуют друг с другом. При вычислении *Is* используется синхронная, редуцированная MPI-функция ($MPI\_Allreduce(\ldots)$) [8].

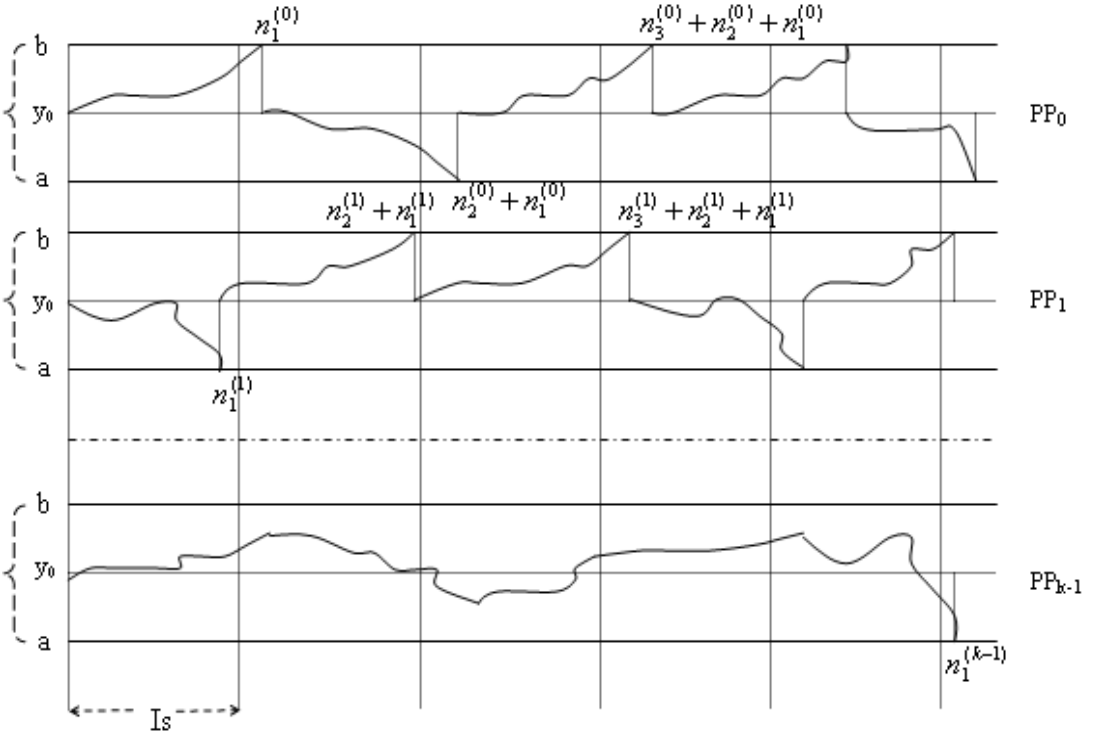


Рис. 1. Схема организации процесса вычислений для второго алгоритма

В данной работе реализуется следующая стратегия динамического выбора размера интервала *Is*. Пусть GSM_k^{min} - текущее суммарное (глобальное) по всем процессорам количество выходов реализаций, а через GSn_k - текущее суммарное (глобальное) по всем процессорам количество итерационных шагов, связанных с достигнутыми выходами.

1. В начальный момент счета устанавливается некоторый размер интервала $Is = N_{Is}$.

2. Если $GSM_k^{min} = 0$, то все процессоры увеличивают размер интервала Is в два раза. Таким образом, все процессоры в следующий раз выйдут на взаимодействие уже через интервал в два раза больший предыдущего.

3. Если $GSM_k^{min} \neq 0$ и $GSn_k \neq 0$, то каждый процессор присваивает своей переменной Is значение $C \cdot (GSn_k)/GSM_k^{min}$. После этого Is больше не меняется.

Здесь специально отметим, что суммируются только те итерационные шаги, которые связаны с фактом выхода за границы области. Отметим так же, что в результате редуцированной (MPI-) операции суммирования, значения счетчиков GSM_k^{min} и GSn_k будут находиться в каждом процессоре.

После вычисления интервала Is следует выбор типа и схемы взаимодействий процессоров. Вычисления продолжаются в двух вариантах: 1) с синхронными межпроцессорными взаимодействиями (2-й) алгоритм или 2) с асинхронными взаимодействиями (3-й алгоритм).

В решаемой задаче на каждом интервале Is суммируются по всем процессорам три величины, накапливаемые в течении текущего интервала: LM_k^{min} - счетчик количества выходов реализаций на границы области, LS_{nk} - счетчик суммарного количества шагов, связанных с этими выходами и LS_{vk} - счетчик вторых моментов.

Во втором алгоритме тип обменов между процессорами - синхронный, схема обменов - "взаимодействующие равные". Суммирование указанных счетчиков осуществляется синхронной редуцированной MPI-функцией ($MPI\_Allreduce(\ldots)$).

В третьем алгоритме тип обменов - асинхронный, схема обменов "управляющий-рабочие". Нулевой процессор - управляющий, все остальные - рабочие. Управляющий принимает от рабочих счетчики, суммирует их и передает рабочим флаг завершения вычислений. Рабочие процессоры взаимодействуют с управляющим процессором асинхронно.

Вычисления векторизованы и программы решаемых задач за счет векторизации вычислений ускоряются на $50\% - 60\%$ (на каждом процессоре) по сравнению с этими же программами, но без векторизации вычислений.

В решаемых задачах необходимый объем выборки базовых случайных чисел очень велик ($\approx 2 \cdot 10^{14}$), поэтому целесообразно использовать "длиннопериодные" псевдослучайные последовательности. В описываемых ниже расчетах используется генератор псевдослучайных чисел - MT2203. Это фактически набор из 1024-х генераторов псевдослучайных чисел, которые предназначены для использования в параллельных моделированиях ММК. Каждый из них генерирует последовательность с периодичностью, равной 2 в степени 2203. Параметры генераторов обеспечивают взаимную независимость соответствующих последовательностей псевдослучайных чисел. Одним из таких параметров является идентификационный логический номер процессора, назначаемый системой параллельного программирования MPI [8].

4. Численные эксперименты

Вычисления производились на кластере НКС-30Т ССКЦ ИВМиМГ СО РАН. Далее под словом процессор имеется ввиду процессорное ядро.

Тест 1.

Для решения СДУ в смысле Ито вида

$$\begin{aligned} dy &= y\, dw(t), \quad 0 \leq t \leq 10, \\ y(0) &= 1 \end{aligned} \tag{10}$$

моделируемого по формуле

$$y_n = y_{n-1} \exp\left(-\frac{h}{2} + \sqrt{h}\ \xi_n\right), \quad y_0 = 1,$$

оцениваются первый и второй момент случайной величины τ - времени первого выхода реализаций на границу интервала $[0,\ 2]$. В табл. 1 проведены результаты расчетов по формуле (9) при $K = 64, h = 10^{-4}$ для размеров ансамбля моделируемых траекторий $M = 10^2, 10^4, 10^7$. В данном тесте точные значения $E\tau$ и $E\tau^2$ неизвестны, тем не менее

Таблица 1.

M	$\hat{\tau}$	$\hat{\tau}^2$	Время счета (сек)
10^2	171.4	223305.1	1.37
10^4	714.9	1011618.6	104.41
10^7	712.1	1008687.7	110559.60

из расчетов видно большое различие в оценках при $M = 10^2$ и $M = 10^4,\ 10^7$, что говорит о необходимости моделирования ансамблей максимального размера в задачах о достижении границ. Критерием точности может служить число совпадающих значащих цифр оценок для разных M. Малое значение оценки $\hat{\tau}^2$ для $M = 10^2$ связано с отсутствием в выборке редких реализаций, долго не выходящих на границу интервала.

В таблице 2 приведена зависимость времени счета от числа используемых процессоров для $M = 10^4$.

Таблица 2.

K	1	8	16	32	64
Время счета (сек)	3088	528	267	167	104

Тест 2.

Если в (3) положить $f(y) = 0,\ \sigma(y) = \sigma$, то сопутствующее СДУ является СДУ с аддитивным шумом:

$$dy = \sigma\, dw(t),\ y(0) = y_0, \tag{11}$$

решением которого является винеровский процесс

$$y(t) = y_0 + \sigma\, w(t), \tag{12}$$

имеющий математическое ожидание $Ey(t) \equiv y_0$ и второй момент

$$Ey^2 = E(y_0 + \sigma\, w(t))^2 = y_0^2 + \sigma^2 t.$$

Для процесса (12) известно точное среднее время первого выхода траекторий из $[a, b]$ (см., например, [12]):

$$E\tau(y_0) = \frac{(y_0 - a)(b - y_0)}{\sigma^2}. \tag{13}$$

Из (13) видим, что при уменьшении σ математическое ожидание τ растет как σ^{-2}. Для второго момента имеем

$$E\tau^2(y_0) = \frac{1}{3\sigma^4}\,[\,(b-a)^3(y_0-a) - 2\,(b-a)(y_0-a)^3 + (y_0-a)^4\,] \tag{14}$$

и рост уже как σ^{-4}. Такая зависимость от σ связана с продолжительным временем достижения процессом (12) границ интервала $[a, b]$.

Для решения СДУ (11), моделируемого по формуле

$$y_n = y_{n-1} + \sigma\xi_n\sqrt{h},$$

оценивается среднее время и второй момент величины τ - времени первого выхода реализаций на границу интервала $[-1, 1]$. Из (13) и (14) имеем

$$E\tau(y_0) = \frac{1 - y_0^2}{\sigma^2},$$

$$E\tau^2(y_0) = \frac{1}{3\sigma^4}\left[8\,(y_0+1) - 4\,(y_0+1)^3 + (y_0+1)^4\right].$$

В табл. 3 проведены результаты расчетов при $K = 64,\ h = 10^{-4}$ для размеров ансамбля $M = 10^2, 10^3, 10^6, 10^8$. В первом случае, при $y_0 = 0$, траектории СДУ стартуют из центра интервала $[-1, 1]$. Во втором случае, $y_0 = 0.9$, траектории стартуют из точки вблизи правой границы интервала и сильное влияние на точность оценок оказывают редкие реализации, выходящие на левую границу интервала. Расчеты показывают сильное различие в точно-

Таблица 3.

y_0	σ	$E\tau(y_0)$	$E\tau_2(y_0)$	M	$\hat{\tau}$	$\hat{\tau}^2$	Время счета (сек)
0	10^{-1}	10^2	$1.667 \cdot 10^4$	10^2	65.4	5652.9	0.17
				10^3	95.5	15132.7	0.87
				10^6	100.1	16726.8	749.88
				10^8	100.1	16709.3	79026.5
	10^{-2}	10^4	$1.667 \cdot 10^8$	10^2	5644.1	41852534.9	6.05
				10^3	9114.6	135506216.4	52.66
				10^6	10004.2	166840038.2	60655.44
				10^8	–	–	–
0.9	10^{-1}	19	$2.654 \cdot 10^3$	10^2	1.4	3.6	0.06
				10^3	10.4	714.6	0.27
				10^6	19.0	2663.6	177.50
				10^8	19.1	2674.1	18126.60
	10^{-2}	1900	$2.654 \cdot 10^7$	10^2	117.9	23078.5	0.34
				10^3	1034.8	6664365.1	11.07
				10^6	1904.5	26630171.4	13887.96
				10^8	–	–	–

сти оценок для разных M, как для случая y_0 в центре интервала $[-1, 1]$, так и вблизи границы для различных σ. Во всех тестах очень низкая точность оценок вторых моментов для всех $M = 10^2, 10^3$. Большие размеры моделируемого ансамбля для $K = 64$ требуют слишком большого времени счета, более двух суток. С этим связано отсутствие некоторых результатов расчетов при $M = 10^8$.

В таблице 4 приведены временные характеристики счета 2-го (синхронного) и 3-го (асинхронного) алгоритмов для этого же теста при K =16, 32 и 64, размеров ансамбля $M = 10^6$ и для двух точек старта реализаций: $y_0 = 0$ и $y_0 = 0.9$. Из таблицы видно, что при K=16 и 32 времена вычислений обоих алгоритмов близки для малого (2048 шагов) и большого (131072 шагов) размеров интервала *Is*. Но при K =64 для *Is* =2048 асинхронный

алгоритм (254 сек.) уступает синхронному (177 сек.), а для $Is = 131072$ наоборот, асинхронный (540 сек.) уже существенно превосходит синхронный (749 сек.). Отсюда следует, что после нахождения интервала Is, можно минимизировать дальнейшие вычисления. Например, для $Is \leq 32768$ - реализуется синхронный алгоритм, а для $Is > 32768$ - асинхронный алгоритм.

Таблица 4.

y_0	Is	Тип алгоритма	16 проц. (сек)	32 проц. (сек)	64 проц. (сек)
0.0	131072	Синхронный	2072	1050	749
		Асинхронный	2083	1065	540
0.9	2048	Синхронный	474	263	177
		Асинхронный	449	294	254

Тест 3.

Оценка математического ожидания и второго момента решения $y(t)$ линейного СДУ (6). Математическое ожидание при начальных условиях $m(0) = 1$, $\frac{dm}{dt}(0) = 0$ задается формулой $m(t) = cos(\beta t)$. При $\beta = 2\pi$ функция $m(t)$ на интервале $[0, 100]$ имеет 100 периодов колебаний с одинаковой амплитудой 1. СДУ (5) можно переписать в виде линейной системы

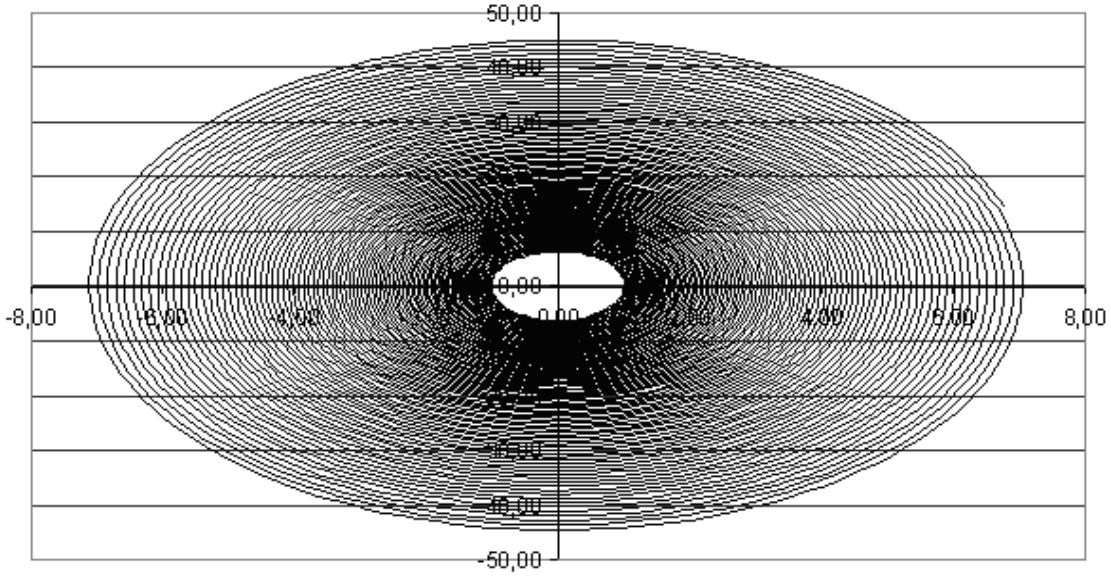


Рис. 2. Фазовая траектория оценки математического ожидания решения СДУ (6)

$$\begin{aligned} dy_1 &= y_2\, dt, \\ dy_2 &= -(\beta^2 y_1 + \lambda\, y_2)\, dt - \sigma_1\, y_2\, d\omega_1(t) - \sigma_2\, y_1\, d\omega_2(t). \end{aligned} \tag{15}$$

Применяя метод Эйлера (2) к СДУ (6), записанному в виде системы первого порядка (15),

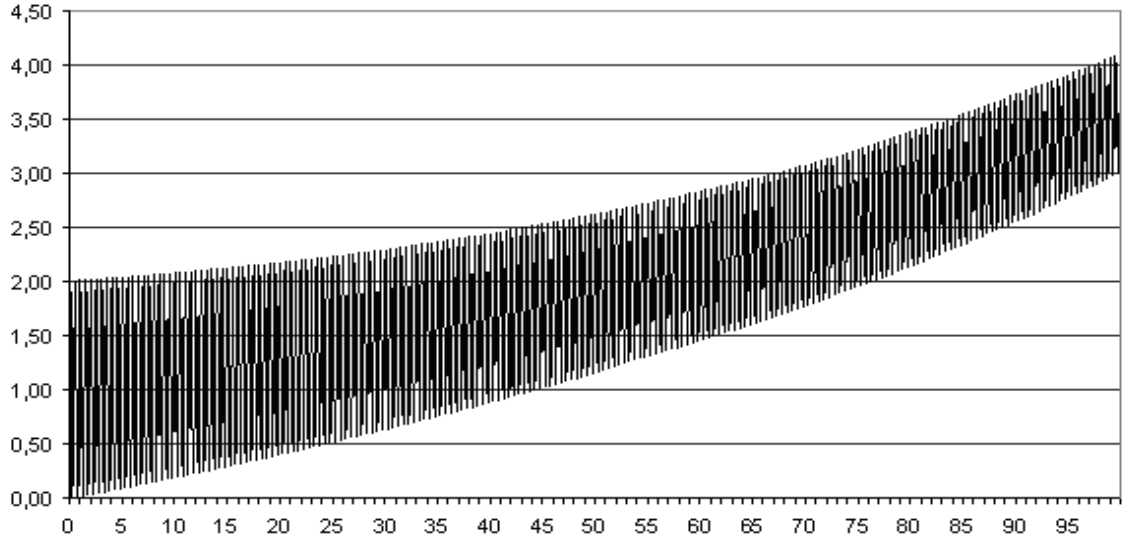


Рис. 3. График оценки второго момента решения СДУ (6)

получим следующую разностную схему:

$$\begin{aligned} y_{n+1}^{(1)} &= y_n^{(1)} + h y_n^{(2)}, \\ y_{n+1}^{(2)} &= y_n^{(2)} - h\beta^2 y_n^{(1)} + \sqrt{h}\, y_n^{(1)}\, \xi_{n+1}\,. \end{aligned} \tag{16}$$

На рис. 2 приведён график фазовой траектории $(Ey_n^{(1)},\, Ey_n^{(2)})$, расчитанной с помощью разностной схемы (16) с шагом $h = 10^{-3}$ и размером ансамбля моделируемых траекторий решения $M = 8^7$.

Расчеты показывают, что при таком размере шага интегрирования наблюдается рост амплитуды колебаний оценки математического ожидания. Устойчивую фазовую траекторию в виде эллипса $(cos\, 2\pi t,\, -2\pi\, sin\, 2\pi t)$ удаётся получить только при шаге интегрирования $h = 10^{-6}$ и меньше.

В оценке второго момента, полученной с шагом $h = 10^{-3}$, имеем максимальное значение равное 205, что говорит о полной потере точности оценки с таким шагом. Удовлетворительная точность получается только при h не более 10^{-6}, когда имеем оценку максимального значения, близкую к точному значению равному 4 (рис.3). Число моделируемых траекторий выбиралось настолько большим, чтобы в данном расчете размер ансамбля моделируемых траекторий не оказывал влияния на точность оценок.

Рост дисперсии решения СДУ (6) со временем делает проблемным точную оценку математического ожидания $Ey(t) = cos\, 2\pi t$ на большом числе периодов колебаний и требует увеличения размера ансамбля моделируемых траекторий при увеличении T, что естественно увеличивает время счёта задачи.

Время счёта на 64 процессорах с шагом $h = 10^{-6}$, $M = 8^6$ составило около 16 часов. Расчёты с более мелким шагом интегрирования требуют многосуточных вычислений.

Тест 4.

Оценка математического ожидания и второго момента решения нелинейного уравнения Ван-дер-Поля (7) с параметрами $\lambda = 20$, $b = 1$, $\beta = 2\pi$, $\sigma = 1$.

На рис. 4 приведён график одной смоделированной траектории решения $y_1(t)$ системы СДУ (7) с шагом $h = 10^{-8}$ При так выбранном параметре $\lambda = 20$ стохастическое уравне-

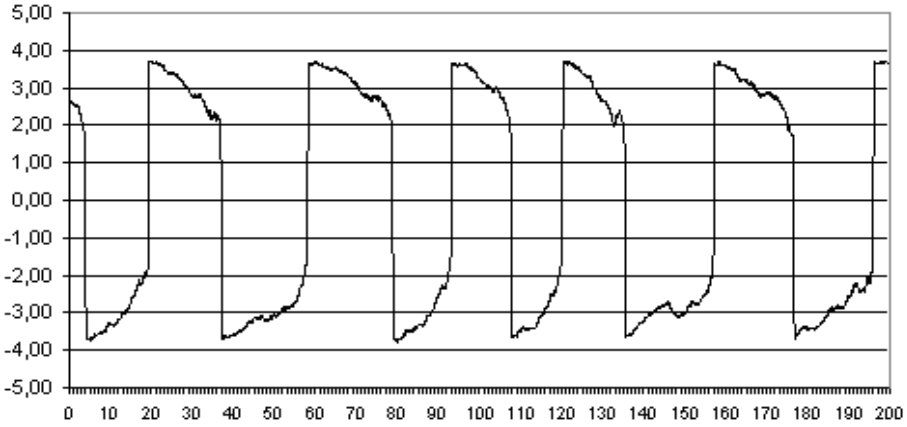


Рис. 4. График одной смоделированной траектории уравнения Ван-дер-Поля (7)

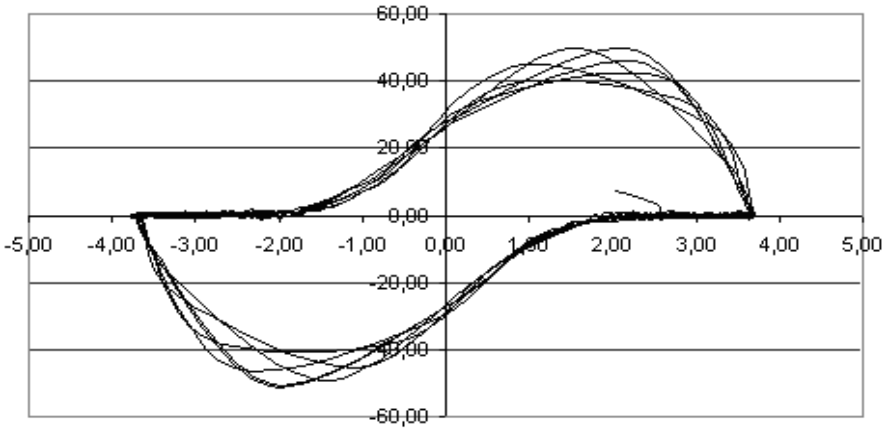


Рис. 5. Фазовая траектория решения уравнения Ван-дер-Поля (7)

ние Ван-дер-Поля можно считать "жестким" [4], имеющим быстрые переходные участки и "полочки" вблизи значений ± 3.

На рис. 5 приведён график смделированной фазовой траектории $(y_1(t), y_2(t))$. Как видно из рис.4 и рис.5, амплитуда колебаний траектории решений не уменьшается. Однако, численные расчёты показывают, что этого нельзя сказать о поведении математического ожидания $Ey_1(t)$.

На рис. 6 приведён график оценки математического ожидания $Ey_1(t)$, полученной с шагом $h = 10^{-8}$ и с размером ансамбля $M = 8^5$. Расчеты показывают, что амплитуда колебаний $Ey_1(t)$ уменьшается от периода к периоду, что находится в резком контрасте с поведением математического ожидания в линейном колебательном контуре (Тест 3). Можно сказать, что математическое ожидание нелинейного СДУ со временем "теряет информацию" о поведении каждой отдельной траектории решения (7). Дополнительные расчёты показали, что и в случае малой "жёсткости" (с $\lambda = 1$) математическое ожидание решения (7) также имеет убывающую со временем амплитуду колебаний. Такое резкое различие в поведении математического ожидания решений линейных и нелинейных осциляторов предупреждает об опасностях решения нелинейных СДУ с помощью их линеаризации.

На рис. 7 приведён график оценки второго момента решения $Ey_1^2(t)$. Как видим, достаточно быстро происходит стабилизация дисперсии решения, что также отлично от пове-

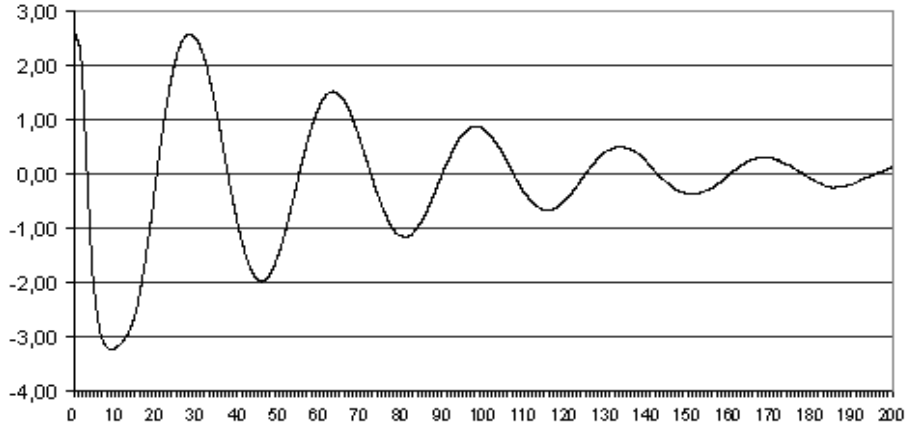


Рис. 6. График оценки математического ожидания решения уравнения Ван-дер-Поля (7)

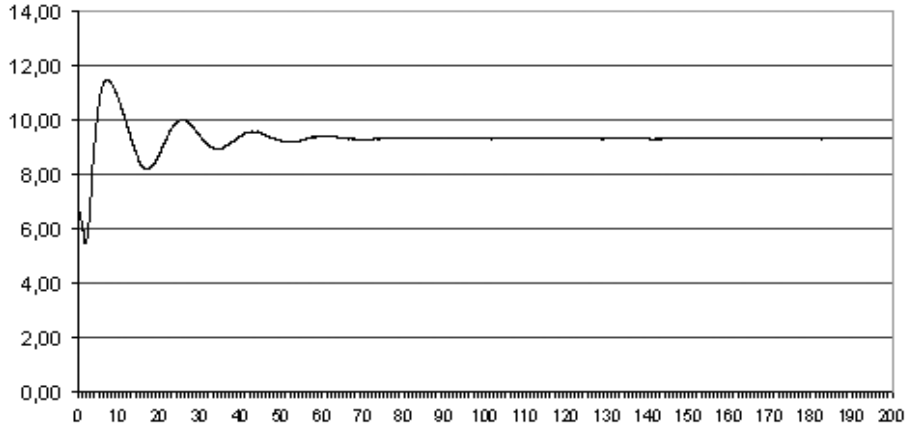


Рис. 7. График оценки второго момента решения уравнения Ван-дер-Поля (7)

дения дисперсии в линейном случае. Время счета этого теста на 128 процессорах с шагом $h = 10^{-8}$ и $M = 8^5$ составило около 73 часов. Отметим, что использование большего размера шага интегрирования ведёт к неустойчивости численного решения, что в конечном итоге приводит к переполнению разрядной сетки арифметического устройства процессоров.

5. Заключение

Расчеты наглядно показывают необходимость использования суперкомпьютеров для численного решения СДУ. Кроме того, проведенные исследования показали, что при численном анализе стохастическимх осцилляторов методом Монте-Карло использование больших размеров шагов интегрирования решений СДУ, что практически всегда делается при проведении численных экспериментов на ПК, могут приводить к совершенно ошибочным выводам.

Так же отметим, что сделанные выводы по поведению моментов решения стохастического уравнения Ван-дер-Поля с большим шумом невозможно получить никаким другим методом, кроме статичтического моделирования. Это касается как методов гаусовской аппроксимации, так и спектрального метода [13].

Литература

1. Диментберг М. Ф. Нелинейные стохастические задачи механических колебаний. —М.: Наука, 1980.
2. Бабицкий В. И. Теория виброударных систем. —М.: Наука, 1978.
3. Артемьев С.С., Якунин М. А. Математическое и статистическое моделирование в финансах. —Новосибирск: Изд. ИВМиМГ СО РАН, 2008.
4. Артемьев С.С. Численные методы решения задачи Коши для систем обыкновенных и стохастических дифференциальных уравнений. —Новосибирск: Изд. ВЦ СО РАН, 1993.
5. Артемьев С.С., Корнеев В.Д. Численное решение стохастических дифференциальных уравнений на суперкомпьютерах //Сиб. ЖВМ / РАН. Сиб. отд-ние. —Новосибирск, 2011. —Т. 14, № 1. —С. 5–17.
6. Ермаков С.М., Михайлов Г.А. Статистическое моделирование. —М.: Наука, 1982.
7. Ермаков С.М., Некруткин В.В., Сипин А.С. Случайные процессы для решения классических уравнений математической физики. —М.: Наука, 1984.
8. Snir M., Otto S. W., Huss–Lederman S., Walker D., and Dongarra J.. MPI: The Complete Reference. MIT Press. Boston, 1996.
9. Корнеев В.Д. Параллельное программирование в MPI. —Москва–Ижевск: Институт компьютерных исследований, 2003, —304 с.
10. Корнеев В.Д. Параллельное программирование кластеров: учеб. пособие/—Новосибирск: Изд-во НГТУ, 2008. —312 с.
11. http://software.intel.com/ru-ru/intel-mkl/
12. Аверина Т.А., Артемьев С.С. Анализ точности методов Монте-Карло при решении краевых задач посредством вероятностного представления//Сиб.ЖВМ/РАН. Сиб. отделение. —Новосибирск, 2008. —Т. 11, № 3. —С. 239–250.
13. Пантелеев А.В., Рыбаков К.А., Сотскова И.А. Спектральный метод анализа нелинейных стохастических систем управления. —М.: Вузовская книга, 2006.

Параллельная реализация метода расщепления для системы из нескольких GPU с применением в задачах аэрогидродинамики*

С.Б. Березин[1], И.С. Каргапольцев[1], Н.Д. Марковский[2], Н.А. Сахарных[1,2]

МГУ им. М.В. Ломоносова[1], NVIDIA Ltd.[2]

Метод прямого численного моделирования турбулентных течений требует сильно детализированных сеток и огромных вычислительных ресурсов. Современные GPU имеют высокую пропускную способность памяти и вычислений с плавающей точкой, поэтому эти устройства очень хорошо подходят для решения задач аэрогидродинамики. Однако, из-за ограничения на доступный объем памяти, для расчета больших сеток необходимо эффективное распределение работы между отдельными GPU в системе. В этой статье будет предложен эффективный параллельный алгоритм метода расщепления для нескольких GPU и рассмотрены варианты балансировки работы для системы с несколькими узлами. Также будет проведен анализ производительности метода на различных входных данных.

1. Введение

Быстрый рост производительности графических процессоров (GPU) в течение последнего десятилетия открыл новые возможности для использования GPU в реальных приложениях, требующих больших вычислительных мощностей. Несколько лет назад к решению подобных задач можно было прийти только путем использования суперкомпьютеров и кластеров с большим числом процессоров. Сейчас в нашем распоряжении находятся гораздо более энерго-эффективные и компактные устройства GPU с массивно-параллельной архитектурой, которые могут использоваться как со-процессоры, значительно ускоряя расчеты.

Относительно недавно появились удобные инструменты для разработки приложений общего назначения для GPU - такие как CUDA, OpenCL, Parallel Nsight [1]. Эти средства значительно упростили разработку сложных приложений, эффективно использующих параллельную архитектуру GPU.

Аэрогидродинамика является, пожалуй, одной из самых сложных областей моделирования явлений природы, требующих огромного числа вычислительных ресурсов. Численное моделирование течений жидкостей и газов требует применения самых современных суперкомпьютеров для получения результата достаточной точности за приемлемое время..

2. Решение уравнений Навье-Стокса на GPU

Течения жидкостей и газов в естественной среде (движение воздуха в земной атмосфере, воды в реках и морях, газа в атмосферах звезд и т.п.) и инженерных сооружениях (в трубах, каналах, струях, пограничных слоях около движущихся в жидкости или газе твердых тел, следах за такими телами и т.п.) могут быть описаны системой уравнений Навье-Стокса. Полная форма уравнений Навье-Стокса в трёхмерной декартовой системе координат для идеального газа с постоянной плотностью имеет вид:

$$\nabla \cdot \vec{\mathbf{u}} = 0, \tag{1}$$

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right] = -\nabla T + \frac{1}{Re} \nabla^2 \mathbf{u}, \tag{2}$$

$$\rho \left[\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T \right] = \frac{1}{Re \cdot Pr} \Delta T + \frac{\gamma - 1}{\gamma \cdot Re} \Phi, \tag{3}$$

*Работа выполнена при финансовой поддержке РФФИ, проекты № 10-01-00288-а, № 09-07-00424-а.

$$p = \rho RT, \tag{4}$$

где $\mathbf{u} = (u, v, w)$ – компоненты вектора скорости, p – давление, ρ – плотность. Здесь уравнение неразрывности 1 выражает постоянство объема, 2 – закон сохранения энергии, 3 – уравнение теплопроводности, и замыкает систему уравнение состояния 4. Число Рейнольса $Re = \frac{UL}{\nu}$ характеризует свойства течения: $\nu = \frac{\mu}{\rho}$ – кинематическая вязкость, μ – вязкость, а L и U – средняя скорость и размер области в конкретной задаче. В уравнениях также присутствуют число Прандтля Pr, газовая постоянная R и удельная теплоемкость γ. Φ – это диссипативная функция, характеризующая работу вязких сил:

$$\Phi = \Phi_x + \Phi_y + \Phi_z$$

$$\begin{aligned}
\Phi_x &= 2(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial x})^2 + (\frac{\partial w}{\partial x})^2 + \frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x}, \\
\Phi_y &= (\frac{\partial u}{\partial y})^2 + 2(\frac{\partial v}{\partial y})^2 + (\frac{\partial w}{\partial y})^2 + \frac{\partial v}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial z}\frac{\partial w}{\partial y}, \\
\Phi_z &= (\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2 + 2(\frac{\partial w}{\partial z})^2 + \frac{\partial w}{\partial x}\frac{\partial u}{\partial z} + \frac{\partial w}{\partial y}\frac{\partial v}{\partial z}.
\end{aligned} \tag{5}$$

Подробный вывод уравнений, их описание и применение можно найти в [2].

Во многих случаях течение становится турбулентным – скорость, температура и давления начинают хаотично изменяться во времени и пространстве. В случае турбулентных течений численное решение системы (1) - (4) требует явного описания всего диапазона пространственных и временных масштабов, что возможно при числе узлов сетки пропорциональном $Re^{9/4}$ [3]. Обычно турбулентность наступает при больших числах Re, порядка нескольких тысяч или десятков тысяч. Таким образом, практическое применение численного моделирования течений на основе системы уравнений Навье-Стокса сильно ограничено объемом памяти и производительностью существующих вычислительных систем. Существуют модели LES (large eddy simulation), основанные на воспроизведении наиболее значимых вихрей и параметризации оставшейся части спектра [3, 4], что позволяет снизить требования к разрешению сетки и вычислительным ресурсам, но требует выдвижения некоторых априорных утверждений о природе моделируемого течения, что не всегда представляеся возможными. В данной работе будет рассмотрен метод прямого численного решения системы уравнений Навье-Стокса без дополнительных упрощений.

3. Метод покоординатного расщепления и соответствующий разностный метод первого порядка

Parallel reduction for 3D Navier-Stokes equation and its 1-st order discretization

Метод покоординатного расщепления применяется для решения параболических или эллиптических уравнений в частных производных. Идея метода заключается в расщеплении уравнений на несколько более простых – по уравнению вдоль каждой из осей координат. Эта операция проводится таким образом, что производные вдоль соответствующего направления берутся неявно, а остальные координаты считаются постоянными.

Например, уравнение для x-компоненты 2

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = -\frac{\partial T}{\partial x} + \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) \tag{6}$$

расщепляется на 3 уравнения 7-9. Применение к ним конечно-разностной схемы первого порядка приводит к набору независимых трехдиагональных систем.

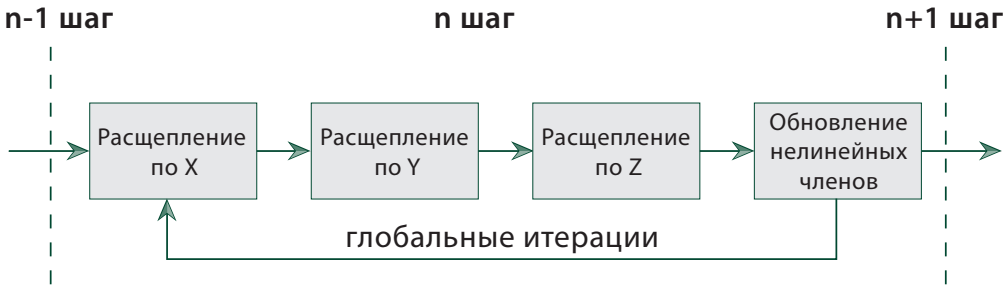


Рис. 1. Целый шаг метода покоординатного расщепления.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = -\frac{\partial T}{\partial x} + \frac{1}{Re}\frac{\partial^2 u}{\partial x^2}, \tag{7}$$

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial y} = \frac{1}{Re}\frac{\partial^2 u}{\partial y^2}, \tag{8}$$

$$\frac{\partial u}{\partial t} + w\frac{\partial u}{\partial z} = \frac{1}{Re}\frac{\partial^2 u}{\partial z^2}. \tag{9}$$

Остальные уравнения могут быть преобразованы аналогичным образом. В общем случае метод расщепления может быть применен к задаче любой размерности.

Поскольку уравнения Навье-Стокса являются нелинейными, необходимо также проводить итерации для обновления нелинейных коэффициентов. В данной реализации проводятся итерации двух типов: глобальные и локальные. Глобальные итерации применяются для целого временного шага для всей системы, локальные – для каждого дробного шага, соответствующего одному из направлений. Общая схема алгоритма приведена на рис. 1. На каждом целом временном шаге система расщепляется по трем направлениям, и соответствующие системы уравнений последовательно решаются в глобальных итерациях. Число глобальных итераций выбирается так, чтобы численная ошибка, рассчитываемая как невязка в уравнении неразрывности, после каждого шага не превышала установленную величину.

На каждом дробном шаге расщепления соответствующие уравнения решаются при использовании конечно-разностной схемы. На рис. 2 показана модель данных и расчетов для отдельного шага расщепления. Основное вычислительное ядро – это решатель наборов трехдиагональных систем. Полученные решения систем используются для обновления нелинейных коэффициентов. После этого проводится несколько локальных итераций для уменьшения численной ошибки.

Подобная схема была использована для решения двумерных уравнений Навье-Стокса в работе [5]. В дальнейшем на основе этой идеи был реализован численный метод для трехмерного случая и динамическая система визуализации [6].

4. Реализация метода прогонок на одном GPU

В CUDA-реализации прогонки каждая нить решает ровно одну трёхдиагональную систему. За счет массивной параллельности нитей на GPU и большого числа систем удается эффективно загрузить устройство. На рис. 3 показано распределение трехмерного массива на дробном шаге расщепления по отдельным нитям. Каждая нить обрабатывает отдельный столбец массива в определенном направлении. Код соответствующего ядра аналогичен

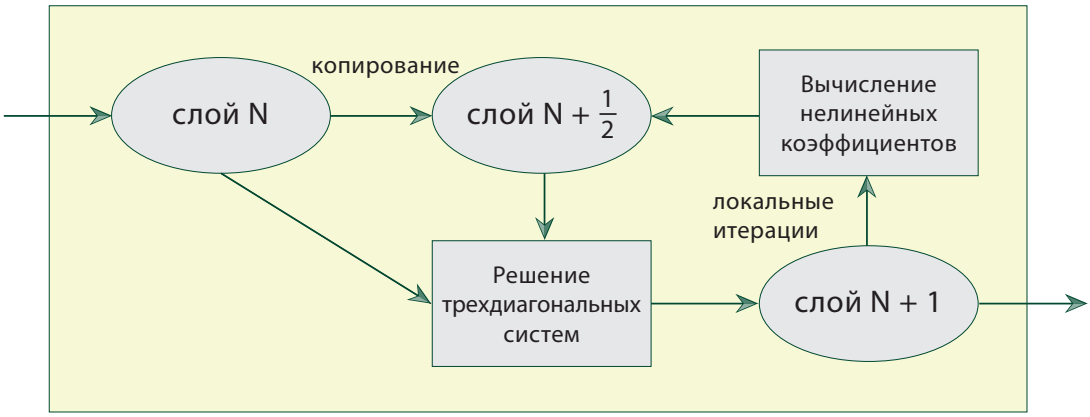


Рис. 2. Дробный шаг расщепления.

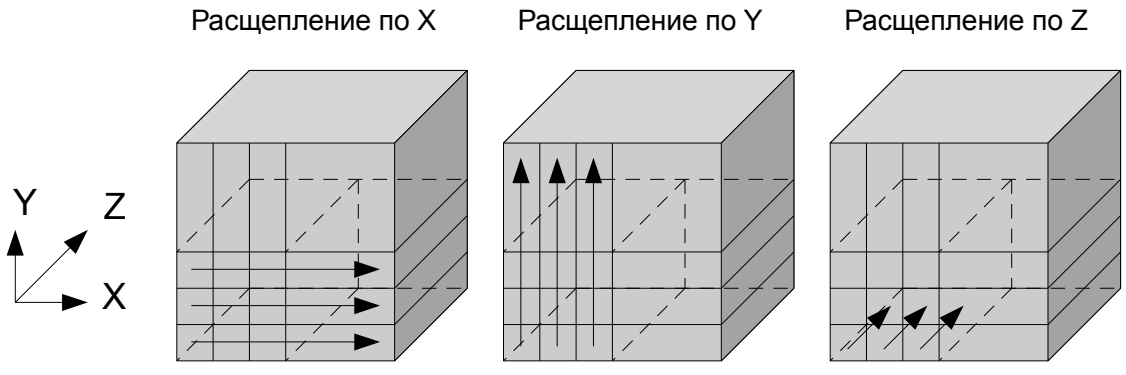


Рис. 3. Прогонки вдоль каждого направления расщепления.

CPU-версии (см. Рис. 4).

```
__device__ void solve_tridiagonal(
        FTYPE *a, FTYPE *b, FTYPE *c, FTYPE *d, FTYPE *x,
        int num, int id, int num_seg, int max_n )
{
    get(c,num-1) = 0.0;

    get(c,0) = get(c,0) / get(b,0);
    get(d,0) = get(d,0) / get(b,0);

    // прямой ход прогонки
    for (int i = 1; i < num; i++)
    {
      get(c,i) = get(c,i) / (get(b,i) - get(a,i) * get(c,i-1));
      get(d,i) = (get(d,i) - get(d,i-1) * get(a,i)) /
                 (get(b,i) - get(a,i) * get(c,i-1));
    }

    get(x,num-1) = get(d,num-1);

    // обратный ход прогонки
    for (int i = num-2; i >= 0; i--)
      get(x,i) = get(d,i) - get(c,i) * get(x,i+1);
}
```

Рис. 4. Реализация алгоритма прогонки на CUDA. Одна нить решает ровно одну систему.

Часто для достижения максимальной эффективности на GPU приходится менять структуру данных, а иногда и сам метод.

Анализ показал, что подобные прогонки вдоль оси Z работают медленнее, чем по X и Y. Это объясняется тем, что прогонки вдоль Z приводят к чтению **каждой нитью** последовательных значений в памяти. В результате, запросы нитей варпа к памяти не объединяются (uncoalesced, см. [1]). Напротив, в прогонках вдоль X и Y, **соседние нити** читают последовательные элементы в памяти, и все запросы объединяются. Оптимизировать прогонки по Z можно одним из двух способов: либо менять метод, либо менять шаблон доступа к данным. В данном случае реализован второй способ: входной массив данных транспонируется, а затем вместо Z прогонки запускается Y-прогонка. Эффективное транспонирование возможно за счет использования разделяемой памяти. В таблице 1 показано насколько быстрее работает улучшенный вариант с переупорядочиванием данных в одинарной и двойной точности.

Реализация метода расщепления тестировалась на двух задачах. В первой задаче моделировалось течение внутри прямоугольного канала со стенками параллельными осям координат (box_pipe). Такой вид границ позволяет равномерно загрузить вычислительное устройство. Во второй задаче рассматривалась геометрия акватории Белого моря (white_sea). В этом случае можно оценить насколько хорошо метод работает на реальных задачах со сложными границами и неравномерной загрузкой устройства. Тестирование производительности проводилось на одном GPU (Tesla C1060 или Tesla C2050), а также на одном многоядерном CPU (Intel Core i7, 4 ядра, 2.8GHz). На CPU метод был распараллен с помощью директив OpenMP и использовал все доступные ядра. Ниже приведены таблицы результатов по каждому направлению расщепления и для всего метода в целом.

5. Реализации метода прогонок на нескольких GPU

Схема работы метода прогонок на нескольких GPU, подключенных к одному хосту может состоять из следующих трёх стадий:

- Разделение данных между GPU
- Распараллеливание кода

Таблица 1. Влияние транспонирования на производительность (среднее время работы ядра в миллисекундах), NVIDIA Tesla C2050.

точность	одинарная			двойная		
ядро	оригинал	с трансп.	ускорение	оригинал	с трансп.	ускорение
X dir	523	528	**1.0x**	717	709	**1.0x**
Y dir	525	531	**1.0x**	694	686	**1.0x**
Z dir	1681	533	**2.4x**	1901	693	**2.7x**
трансп.		164			190	
всего	2729	1756	**1.6x**	3312	2278	**1.5x**

Таблица 2. Результаты тестов производительности на модели box_pipe (систем в секунду)

направление	CPU (4 ядра)	Tesla C1060	Tesla C2050	Tesla C2050 vs CPU
X	1.55	5.61	17.73	**11.4x**
Y	2.17	5.36	18.11	**8.3x**
Z	2.34	5.64	18.12	**7.8x**
все	1.96	5.30	16.09	**8.2x**

Таблица 3. Результаты тестов производительности на модели white_sea (систем в секунду)

направление	CPU (4 ядра)	Tesla C1060	Tesla C2050	Tesla C2050 vs CPU
X	3.65	15.40	37.95	**10.4x**
Y	5.12	16.89	44.91	**8.8x**
Z	2.25	5.65	13.38	**5.9x**
все	3.82	11.88	27.40	**7.2x**

- Синхронизация результатов между GPU

Скорость обмена данными между GPU существенно ниже скорости глобальной памяти, поэтому необходимо минимизировать количество пересылаемой информации. Разделение данных между GPU лучше всего провести вдоль направления X, поскольку все трехмерные массивы хранятся по строкам в виде одномерного вектора $(i, j, k) \rightarrow [dimY \cdot dimZ \cdot i + dimZ \cdot j + k]$, и сетка естественным образом разбивается на блоки размера, кратного $dimY \cdot dimZ$. В этом случае распараллеливание прогонок вдоль Y и Z тривиально и сводится к одновременному запуску ядер на нескольких GPU, работающих над соответствующими часятми сетки (см. Рис. 5).

```
for (int i = 0; i < nGPU; i++)
{
    cudaSetDevice(i); // Переключение устройства
    kernel<<<...>>>(devArray[i], ...); // Расчет своей области сетки
}
```

Рис. 5. Одновременный запуск вычислительных ядер на нескольких GPU.

Прогонка вдоль направления X разделяется между всеми GPU и выполняется последовательно, по цепочке: каждый GPU сначала ожидает данные, затем вычисляет часть результатов и отправляет их следующему GPU, используя cudaMemcpyPeer (рис. 6).

Таким образом, с ростом числа GPU, прироста производительности вдоль оси X не будет, и, согласно закону Амдала, это послужит основным лимитирующим фактором производительности при дальнейшем масштабировании. Тем не менее, другие части шага по X – вычисление производных и диссипативной функции – можно эффективно распределить между GPU, за счёт чего достигается хороший скейлинг на нескольких GPU, подключённых к одному хосту (рис. 7).

В силу того, что вычисление производных у границ требует доступа к граничным значениям частей сетки соседних GPU, необходима их эффективная синхронизация на каждом шаге метода. Синхронизацию такого рода можно реализовать с помощью функции прямого асинхронного копирования *cudaMemcpyPeerAsync* (или *cudaMemcpyAsync* при использовании UVA) (рис. 8). Также было проведено исследование влияния на производительность возможности прямого копирования данных между GPU в устройствах Tesla поколения Fermi минуя оперативную память процессора.

Доступна реализация метода для нескольких GPU с использованием технологии MPI. Код можно использовать на нескольких нодах с различным количеством GPU на каждой из нод. Полный исходный код модели, использующей данную реализацию метода можно найти на сайте проекта [7].

6. Заключение

В данной статье была представлена эффективная GPU-реализация метода покоординатного расщепления для решения полной системы уравнений Навье-Стокса в трехмерной области. Численный метод был протестирован на модельной задаче и продемонстрировал хорошие результаты по производительности.

Один из важных результатов работы - это успешная реализация на системах с разделенной памятью с несколькими GPU. Учитывая возрастающие потребности науки, очень важно иметь параллельную реализацию, использующую все доступные ресурсы. Одно устройство имеет относительно небольшой объем памяти и вычислительных ресурсов. Обобщение программы на несколько GPU устройств позволяет легко масштабировать расчеты и эффективно использовать вычислительные кластеры для моделирования сложных течений.

Среди ближайших планов является имплементация параллельного алгоритма без ограничения скейлинга вдоль направления X. Для этого элементы сетки вдоль направления

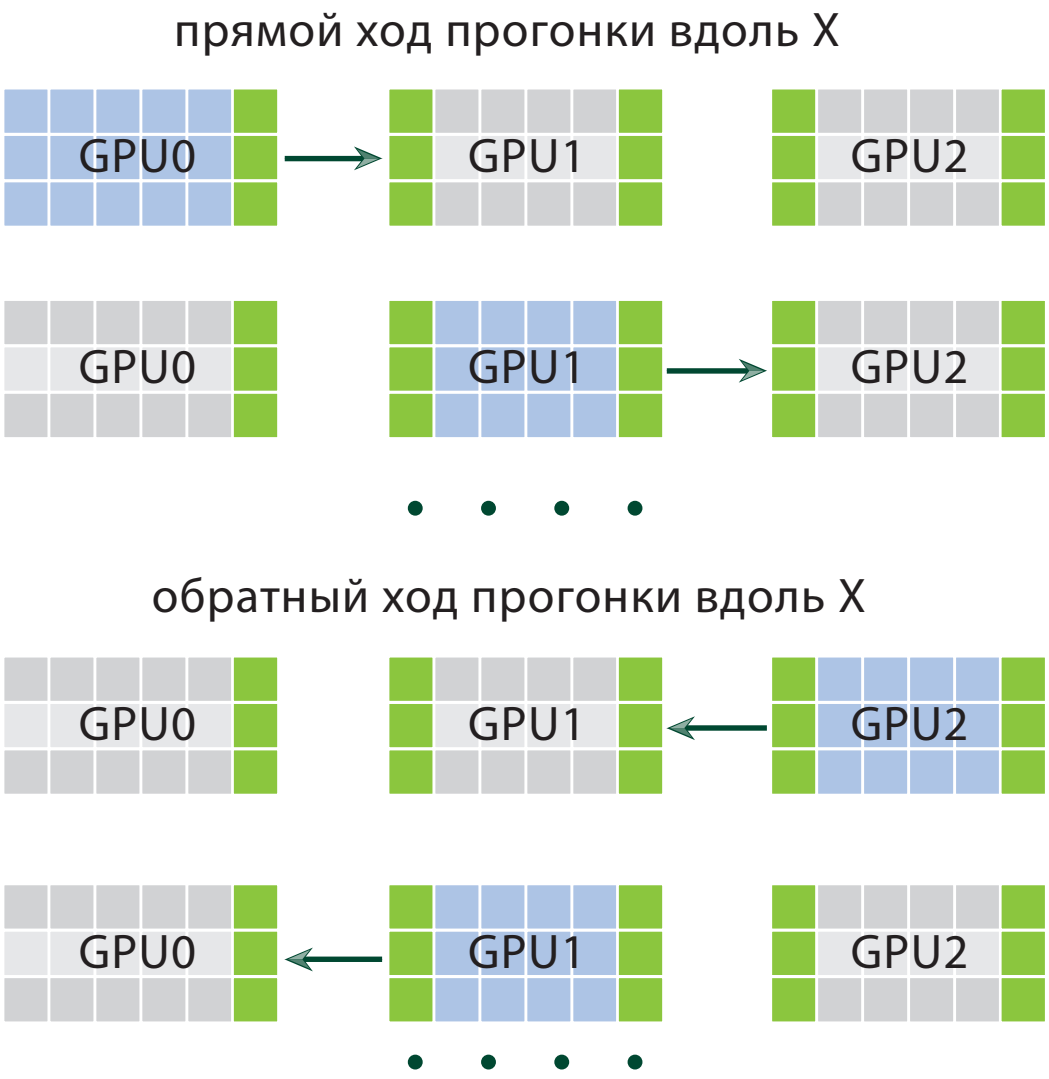
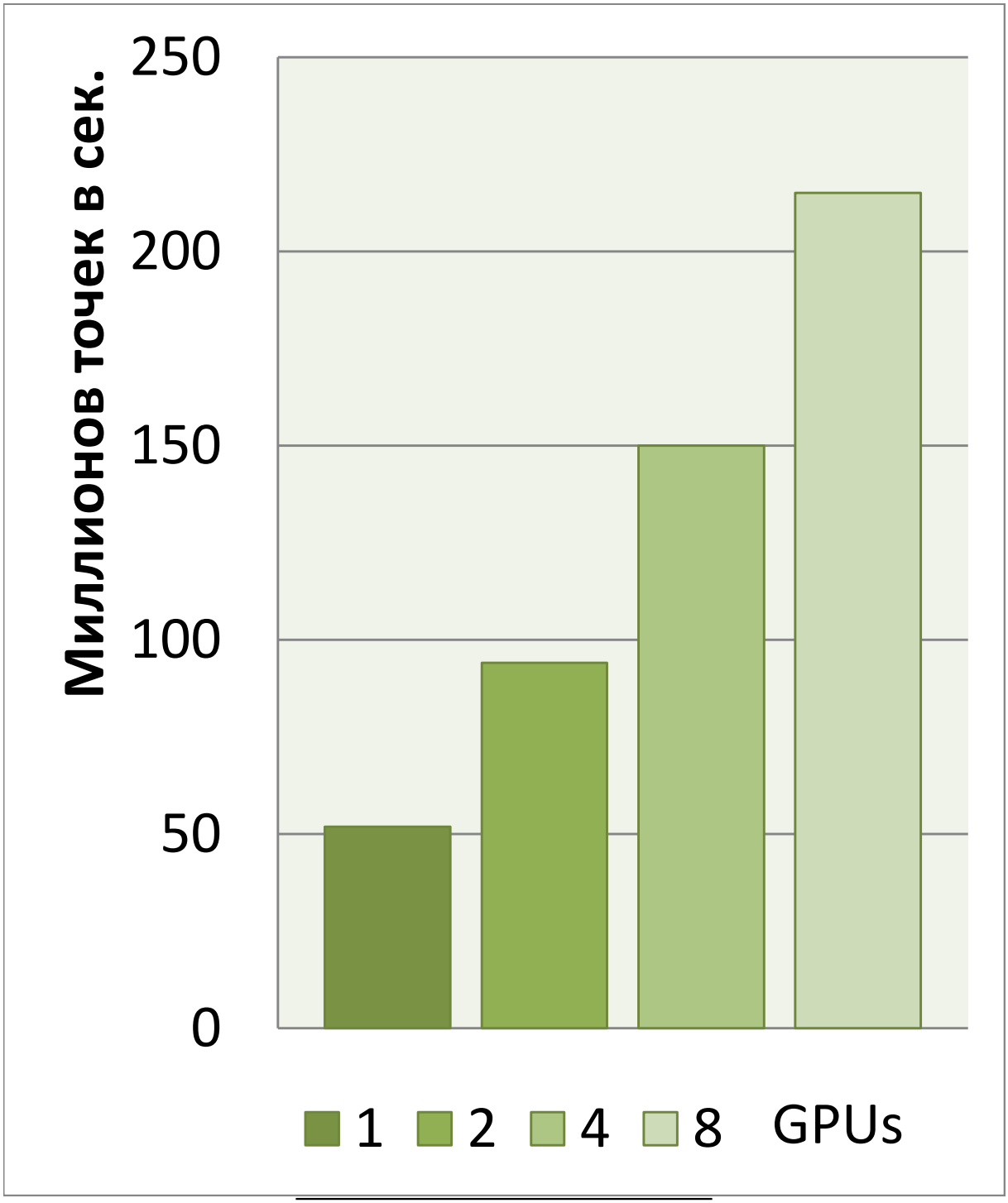


Рис. 6. В каждый момент времени работает только один GPU над своей частью прогонки.



1 GPU	2 GPU	4 GPUs	8 GPUs
51.9	94.1	150.0	215.0

Рис. 7. Производительность системы на базе Tesla M2050, измеряемая в количестве точек обработанного объема в секунду для 1–8 GPUs.

```
// Массивы указателей с граничными полосами на каждом устройстве:
float **dev_halo_right; //  правая синхронизируемая область
float **dev_halo_left; //  левая синхронизируемая область
float **dev_data_right; // правая граничная область
float **dev_data_left; //  левая граничная область
float **dev_data; // область, которую не требуется синхронизировать

cudaStream_t *devStream; // один поток на устройство

// Выделение и инициализация массивов и потоков на каждом устройстве.
...

int haloSize = ...;

// Синхронизация правой области с левой на устройстве i+1.
for( int i = 0; i < numDev - 1; i++)
    cudaMemcpyPeerAsync(dev_halo_left[i+1], i+1, dev_data_right[i], i,
        sizeof(float) * haloSize, devStream[i]);

// Синхронизация левой области с правой на устройстве i-1.
for( int i = 1; i < numDev; i++)
    cudaMemcpyPeerAsync(dev_halo_right[i-1], i-1, dev_data_left[i], i,
        sizeof(float) * haloSize, devStream[i]);

// Ожидание окончания копирования:
for( int i = 0; i < numDev; i++)
{
    cudaSetDevice(i);
    cudaStreamSynchronize(devStream[i]);
}

// Очередной полный пересчёт узлов в отдельных частях сетки.
for( int i = 0; i < numDev; i++)
{
    // Переключение индекса текущего GPU.
    cudaSetDevice(i);
    kernel<<<grid, block, 0, devStream[i]>>>(dev_data[i], dev_data_left[i],
                  dev_data_right[i], dev_halo_left[i], dev_halo_right[i]);
}

...
```

Рис. 8. Пример синхронизации между устройствами путем асинхронного копирования

Z разбиваются на блоки XY, в которых выполнение прогонок вдоль X для одних блоков покрывается прогонками вдоль X и Y для других. (Наличие локальных итераций вдоль каждого направления будет компенсироваться большим количеством блоков). Ожидается, что эффективность данного алгоритма будет улучшаться с размером сетки.

Литература

1. NVIDIA Inc. NVIDIA CUDA programming guide, version 4.1. 2011.
2. Флетчер К. Вычислительные методы в динамике жидкостей. Том 2 // Мир, М., 1991.
3. Лапин Ю.В. Статистическая теория турбулентности // Научно-технические ведомости 2(36)/2004, Сп.Б., Изд-во Политехнического университета.
4. Pierre Sagaut Large eddy simulation for incompressible flows, 3rd edition // Springer, 2006
5. Пасконов В. М., Березин С. Б. Неклассические решения классической задачи о течении вязкой несжимаемой жидкости в плоском канале // Прикладная математика и информатика, №17: МАКС Пресс, Москва, 2004.

6. Пасконов В. М., Березин С. Б., Корухова Е. С. Динамическая система визуализации для многопроцессорных систем с общей памятью и ее применение для численного моделирования турбулентных течений вязких жидкостей // Вестник Московского университета, Серия 15: Вычислительная математика и кибернетика, 2007. С. 7–16.

7. Sakharnykh N., Berezin S., Paskonov V. Fluid solver based on Navier-Stokes equations using finite difference methods in areas with dynamic boundaries:
URL: http://code.google.com/p/cmc-fluid-solver/

УДК 536.42, 542.9, 532.517.2, 519.8

Моделирование процесса роста нанопленок методом химического осаждения из газовой фазы

Ю.Я. Болдырев, К.Ю. Замотин, Е.П. Петухов

Санкт-Петербургский государственный политехнический университет

Большинство задач, которые связаны со многими аспектами развития нанотехнологий по своей природе существенно междисциплинарны. Одним из наиболее характерных примеров этого является проблематика применения газофазного синтеза в нанотехнологиях. По своему существу такие технологии являются реализацией процессов химического осаждения вещества из газообразного состояния, подаваемого в реакционную зону, в твердое состояние. Междисциплинарность, рассматриваемых в газофазном синтезе процессов порождает серьезные трудности при их изучении. При этом, в рамках традиционного физического эксперимента, не удается получить хорошего результата, так как такой эксперимент: не является наглядным, не позволяет изучать зависимость конечного материала от различных физических параметров системы, занимает много времени, дорог. По этому, естественно, искать пути решения задач на базе математического моделирования, которое лежит в основе виртуального эксперимента.

В основе работы лежит разработка и апробация технологий математического моделирования с использованием высокопроизводительных вычислений в области процессов газофазного синтеза наноразмерных структур и наноматериалов с целью изучения и обеспечения визуализации протекающих физико – химических процессов.

Химическое осаждение из газовой фазы – получение твердых веществ с помощью химических реакций, реагенты подаются в реакционную зону в газообразном или плазменном состоянии [1]. Используют для получения текстурированных покрытий, монокристаллов, эпитаксиальных и монокристаллических пленок, нитевидных монокристаллов, порошков, барьерных слоев др. Выражение «химическое осаждение из газовой фазы» является наиболее точным переводом с английского языка термина *chemical vapor deposition* (общепринятая аббревиатура - CVD), который был впервые введен Blocher в 1966 году и с тех пор общепринят во всем мире.

Появление и бурное развитие микроэлектроники придало мощный импульс для разработки разнообразных CVD технологий. Этим методом получают тонкие пленки металлов, диэлектриков и полупроводников, выращивают монокристаллы и эпитаксиальные пленки. Особо следует подчеркнуть, что необходимость получения пленок заданного состава и с требуемым комплексом физических и химических слоев для применения в электронике обусловила проведение тщательных исследований физико-химических закономерностей процессов, что с неизбежностью привело к более глубокому пониманию сущности и механизмов CVD процессов.

К настоящему времени в мировой практике накоплен большой экспериментальный материал по результатам исследования разнообразных процессов химического осаждения из газовой фазы тонких пленок, нанопорошков, нановолокон, наностержней и наноструктур. Однако, несмотря на тот факт, что исследованию некоторых конкретных технологических процессов, посвящены сотни и даже тысячи публикаций, их детерминированные модели, достоверно и однозначно описывающие физико-химические закономерности, отсутствуют. Это обусловлено чрезвычайной сложностью механизма CVD процессов, характеризующихся многомаршрутностью химических реакций, присутствием нескольких гомогенных и гетерогенных стадий, а также многоступенчатостью превращений [2,3]. Используя современные вычислительные технологии удается надежно описывать лишь процессы массо- и теплопереноса, что, в ряде случаев, позволяет успешно оптимизировать и разрабатывать конструкции реакционных камер, при-

меняемых для реализации процессов ХОГФ [4]. Именно на основе таких расчетов, принимая во внимание экспериментально полученные сведения о кинетических закономерностях некоторых конкретных процессов ХОГФ, осуществляется проектирование промышленных реакторов в крупных международных корпорациях (Applied Materials, Samsung Electronics и др.).

Для успешного исследования и моделирования CVD систем необходимы строгие представления о схемах используемых на практике установок, технологических параметрах и их влиянии на условия синтеза, о химии, физике и физико-химии элементарных явлений, сопровождающих синтез, и характер их взаимодействии, иметь экспериментальные данные о процессах и возможности их управления [5]. Схематично CVD процесс может быть проиллюстрирован рисунком 1.

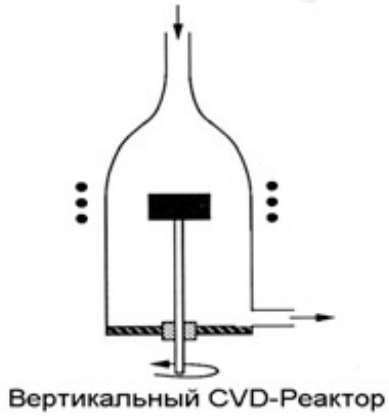


Рис. 1 — Схематическое изображение CVD реактора

Моделирование CVD-процессов базируется на системе уравнений газовой динамики, - уравнениях Навье-Стокса, записанной для многокомпонентной реагирующей среды [6]. Кроме того, требуется построение «подмоделей», описывающих химические реакции, турбулентность и теплообмен за счет излучения.

Задача осложняется требованиями расчета температуры и состава смеси, которые сводятся к решению уравнений для компонентов смеси и температуры. Последние остаются незамкнутыми, поскольку требуют определения величины средней скорости реакции. Таким образом, ее расчет является основной целью химической модели.

Поскольку современные программные комплексы позволяют решать весьма широкий круг междисциплинарных задач, для проведения численного моделирования был выбран программный комплекс ANSYS FLUENT [7]. В случае решения уравнения сохранения для химических веществ, этот комплекс получает значение локальной массовой доля каждого вещества Yi через решение уравнения конвекции-диффузии для i вещества. Запишем уравнение сохранения для i-компоненты в следующем общем виде:

$$\frac{\partial}{\partial t}\left(pY_i\right)+\nabla\cdot\left(p\vec{v}Y_i\right)=-\nabla\cdot\vec{J}_i+R_i+S_i \qquad (1)$$

где Ri является нетто-коэффициентом воспроизводства i - компоненты вещества в результате химической реакции, а Si скорость воспроизводства от добавления из дисперсной фазы с учетом дополнительных источников, задаваемых пользователем. Уравнение такого вида должно решаться для N-1 химического вещества, где N есть общее число химических компонент, представленных в газовой фазе. Поскольку сумма массовых долей всех компонент должна быть тождественно равна единице, то массовая доля N-ого компонента будет равна единице минус сумма массовых долей первых N-1 веществ. Чтобы свести к минимуму численные ошибки, в согласии со стандартными подходами N-м веществом должен быть выбрана компонента с наибольшей массовой долей, например N2 в случае, когда окислителем является воздух.

Отдельно остановимся на проблеме вычислений потока диффузии. В уравнении (1), $\vec{J}_i$ есть поток диффузии компоненты i, который возникает в результате появления градиентов

концентраций. По умолчанию, комплекс ANSYS FLUENT использует так называемую "слабую" аппроксимацию, при которой поток диффузии можно записать в виде:

$$\vec{J}_i = -\rho D_{i,m} \nabla Y_i \tag{2}$$

где $D_{i,m}$ коэффициент диффузии для i-й компоненты смеси. Подобное приближение не всегда может быть приемлемым, когда требуется моделирование полной многокомпонентной диффузии. В таких случаях, к системе может быть добавлено и решено уравнение Максвелла-Стефана.

Скорости реакции, которые появляются в качестве источниковых членов в уравнении (1) вычисляются по следующей модели: эффект турбулентных флуктуаций игнорируется, и скорость реакции определяется формулой Аррениуса [1]. В рамках данной ламинарной модели конечной скорости вычисляются химические источники в терминах выражения Аррениуса и игнорируются эффекты турбулентных флуктуаций. Модель является точной для ламинарных течений, но, как правило, неточна для турбулентных течений из-за высокой нелинейной химической кинетики Аррениуса. Ламинарная модель, однако, может быть приемлемым для процессов с относительно медленно протекающими реакциями и малыми турбулентными флуктуациями.

Чистый источник химического вещества i определяется как Аррениусовская сумма по N_R реакциям, в которую данное вещество входит:

$$R_i = M_{w,i} \sum_{r=1}^{N_r} \hat{R}_{i,r} \tag{3}$$

где $M_{w,i}$ это молекулярная масса вещества I, а $\hat{R}_{i,r}$ Аррениусовская молярная скорость притока / оттока вещества i в реакции r [12]. При этом, реакция может происходить:

- в непрерывной фазе;
- между веществами в непрерывной фазе;
- на поверхности стенки в результате осаждения;
- в результате развития данной фазы.

Рассмотрим некоторую реакцию r, записанную в общем виде:

$$\sum_{r=1}^{N_r} v'_{i,r} M_i \xrightarrow{k_{f,r}} \xleftarrow{k_{b,r}} \sum_{r=1}^{N_r} v''_{i,r} M_i \tag{4}$$

где N - количество химических веществ в системе

$v'_{i,r}$ - стехиометрический коэффициент для i реагента в реакции r

$v''_{i,r}$ - стехиометрический коэффициент для i продукта в реакции r

M - символ, обозначающий i вещество

$k_{f,r}$ - константа скорости прямой реакции r

$k_{b,r}$ - константа скорости обратной реакции r.

Уравнение (4) справедливо как для обратимых, так и необратимых реакций, заметим, что по умолчанию реакции не являются обратимыми. Для необратимых реакций константа обратной скорости $k_{b,r}$ опускается.

Суммирование в уравнении (4) производится для всех химических веществ, участвующих в процессе. Но только вещества, которые появляются в качестве реагентов или продуктов будут иметь ненулевые стехиометрические коэффициенты. Таким образом, элементы, которые не участвуют в реакции, выпадают из уравнения.

Для необратимых реакций, молярная скорость притока / оттока материала i в реакции r ($\hat{R}_{i,r}$ в уравнении (3)) дается следующей формулой:

$$\hat{R}_{i,r} = \Gamma\left(v''_{i,r} - v'_{i,r}\right)\left(k_{f,r}\prod_{j=1}^{N}\left[C_{j,r}\right]^{\left(n''_{j,r}+n'_{j,r}\right)}\right), \quad (5)$$

где $C_{j,r}$ - молярная концентрация j-го элемента в реакции r (кмоль/м3),

$\eta'_{j,r}$ - экспонента скорости для j-го реагента в реакции r ,

$\eta''_{j,r}$ - экспонента скорости для j-го продукта в реакции r ,

Для обратимой реакции молярная скорость создания / уничтожения материала i в реакции r определяется:

$$\hat{R}_{i,r} = \Gamma\left(v''_{i,r} - v'_{i,r}\right)\left(k_{f,r}\prod_{j=1}^{N}\left[C_{j,r}\right]^{\left(n'_{j,r}\right)} - k_{b,r}\prod_{j=1}^{N}\left[C_{j,r}\right]^{\left(n''_{j,r}\right)}\right) \quad (6)$$

Отметим, что показатель скорости для обратной части реакции в уравнении (5) всегда равен стехиометрическому коэффициенту вещества ($v''_{j,r}$). Величина Γ учитывает влияние от присутствия третьих тел в реакции на ее скорость. Третьи тела - вещества, которые не являются реагентами в данной реакции, но влияют на ее протекание. Данный член определяется из следующего соотношения:

$$\Gamma = \sum_{j}^{N}\gamma_{j,r}C_j \quad (7)$$

где $\gamma_{j,r}$ является коэффициентами влияния третьего тела j-го элемента в r-ой реакции. По умолчанию, комплекс ANSYS FLUENT не включает эффекты влияния третьего тела в процессе расчета скоростей реакций. Можно, однако, принудительно включать в учет влияние третьих тел, когда имеются основания их учитывать.

Переходя к скорости прямой реакции r – величине $k_{f,r}$, укажем, что она вычисляется с использованием уравнения Аррениуса:

$$k_{f,r} = A_r T^{\beta_r} e^{-E_r/RT}, \quad (8)$$

где A_r - предэкспоненциальный множитель (размерная единица)

β_r - температурный показатель экспоненты (безразмерная величина)

E_r - энергия активации (Дж/кмоль)

R - универсальная газовая постоянная (Дж/кмоль-К)

Константы реакций уравнения Аррениуса приведены в таблице 1.

Для корректной постановки задачи необходимо знать (или получить из базы данных) значения для $v'_{i,r}$, $v''_{i,r}$, $\eta'_{j,r}$, $\eta''_{j,r}$, β_r, A_r, E_r, $\gamma_{j,r}$. В том случае, если реакция обратима, константа скорости обратной реакции $r, k_{b,r}$ вычисляется через скорость прямой реакции по следующему соотношению [12]:

$$k_{b,r} = \frac{k_{f,r}}{K_r} \quad (9)$$

где K_r константа равновесия r-ой реакции, вычисляемая по формуле:

$$K_r = \exp\left(\frac{\Delta S_r^0}{R} - \frac{\Delta H_r^0}{RT}\right)\left(\frac{p_{atm}}{RT}\right)^{\sum_{i=1}^{N}\left(v''_{i,r}-v'_{i,r}\right)} \quad (10)$$

В этой формуле величина p_{atm} обозначает атмосферное давление (101325 Па). Показатель степени (в круглых скобках) экспоненциальной функции представляет собой изменение свободной энергии Гиббса, и его компоненты рассчитываются следующим образом [7]:

$$\frac{\Delta S_r^0}{R} = \sum_{i=1}^{N} \left(v''_{i,r} - v'_{i,r}\right) \frac{\Delta S_i^0}{R} \tag{11}$$

$$\frac{\Delta H_r^0}{RT} = \sum_{i=1}^{N} \left(v''_{i,r} - v'_{i,r}\right) \frac{h_i^0}{RT} \tag{12}$$

где величины S_i^0 и h_i^0 стандартные состояния энтропии и энтальпии (теплоты образования). Эти значения определяются в комплексе ANSYS FLUENT как свойства смеси материала. Свойства энтропии и энтальпии для использованных в работе материалов задавались в полиномиальном формате CHEMKIN [9] и были получены из термодинамического справочника [10].

Для моделирования процесса металлорганического процесса химического осаждения из газовой фазы (МО ХОГФ) тонких пленок была выбрана система материалов GaAs в реакционном наборе. Выбор материалов обусловливается тем, что данные полупроводниковые материалы широко используются при получении наногетероструктур, на основе которых создаётся множество приборов (полевые транзисторы, лазеры, светодиоды и фотоприемники и др.). Приведенная реакционная система уравнений (11-12) отличается большим количеством реакций протекающих в объеме и на поверхности. Для каждой реакции должны быть указаны следующие (были взяты из общедоступных источников, таблица 1) значения параметров:

- предэкспоненциальный множитель (A),
- показатель температуры (b),
- энергия активации (E) для скорости реакции в Аррениусовом виде (9)

Таблица 1 Параметры скоростей реакций

Объемные реакции	A	b	E
TMG => DMG + CH3	1.6E17	0	30057
DMG => MMG + CH3	2.5E15	0	17883
CH3 + H2 => CH4 + H	1.2E9 0	6300	
ASH3 + CH3 = ASH2 + CH4	9.7E8 0	900	
TMG + H = DMG + CH4	5.0E10	0	5051
DMG + H => MMG + CH4	5.0E10	0	5051
2H + M = H2 + M	1.0E13	0	0
2CH3 = C2H6	2.0E10	0	0
CH3 + H + M => CH4 + M	2.4E19	-1	0
TMG + CH3 => ADDUCT + CH4	2.0E8 0	5051	
MMG => GA + CH3	1.0E16	0	39052
Поверхностные реакции			
H + OPENAS(S) => H_AS(S)	4.95E9	0.5	0
H + OPENG(S) => H_G(S)	4.95E9	0.5	0
CH3 + OPENG(S) = CH3_G(S)	1.27E9	0.5	0
CH3 + OPENAS(S) = CH3_AS(S)	1.27E9	0.5	0
MMG + OPENAS(S) = MMG_AS(S)	5.37E8	0.5	0
DMG + OPENAS(S) => MMG_AS(S) + CH3	4.95E8	0.5	0
ASH + OPENG(S) = ASH(S)	5.68E8	0.5	0
ASH2 + OPENG(S) => ASH(S) + H	5.68E8	0.5	0
ASH3 + OPENG(S) => ASH(S) + H2	5.68E8	0.5	0
CH3 + H_AS(S) => CH4 + OPENAS(S)	1.26E8	0.5	0
CH3 + H_G(S) => CH4 + OPENG(S)	1.26E8	0.5	0
H + CH3_AS(S) => CH4 + OPENAS(S)	4.94E8	0.5	0
H + CH3_G(S) => CH4 + OPENG(S)	4.94E8	0.5	0
H_AS(S) + CH3_G(S) => CH4 + OPENAS(S) + OPENG(S)	1.0E16	0	5051
H_G(S) + CH3_AS(S) => CH4 + OPENAS(S) + OPENG(S)	1.0E16	0	5051

H_G(S) + H_AS(S) => H2 + OPENAS(S) + OPENG(S)	1.2E16	0	10102
CH3_G(S) + CH3_AS(S) => C2H6 + OPENAS(S) + OPENG(S)	1.0E16	0	10102
MMG_AS(S) + ASH(S) => CH4 + OPENG(S) + OPENAS(S) + GAAS(B)	5.0E17	0	14801
MMG_AS(S) + AS(S) => CH3 + OPENG(S) + OPENAS(S) + GAAS(B)	5.0E17	0	10103
2ASH(S) => AS2 + H2 + 2OPENG(S)	1.0E16	0	19681
CH3 + ASH(S) => AS(S) + CH4	1.28E8	0.5	10103
2AS(S) = AS2 + 2OPENG(S)	1.0E17	0	15155
TMG + OPENAS(S) => MMG_AS(S) + 2CH3	4.62E8	0.5	0
GA + OPENAS(S) = GA(S)	5.9E8	0.5	0
GA(S) + AS(S) => OPENAS(S) + OPENG(S) + GAAS(B)	1.1E9	0	505

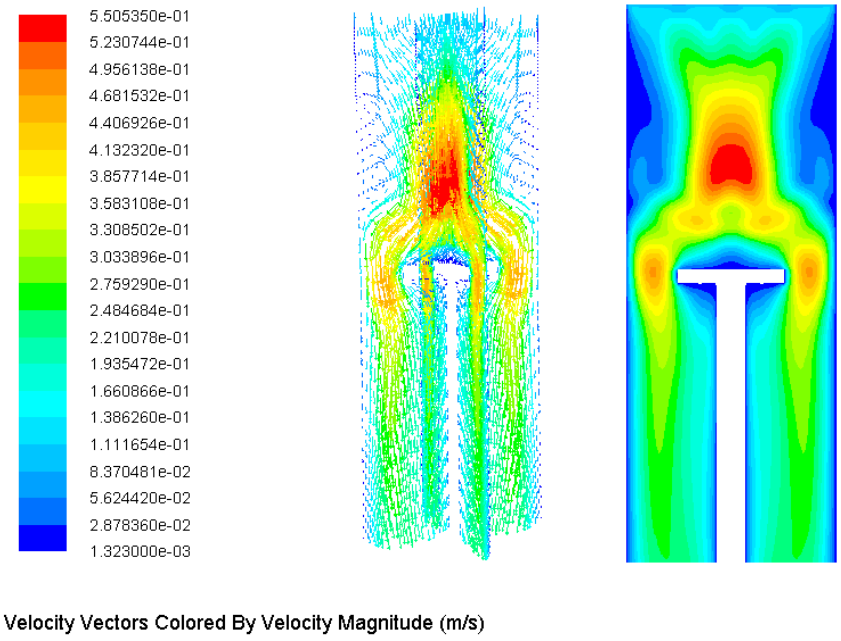


Рис. 2 — Скорость потока внутри реактора

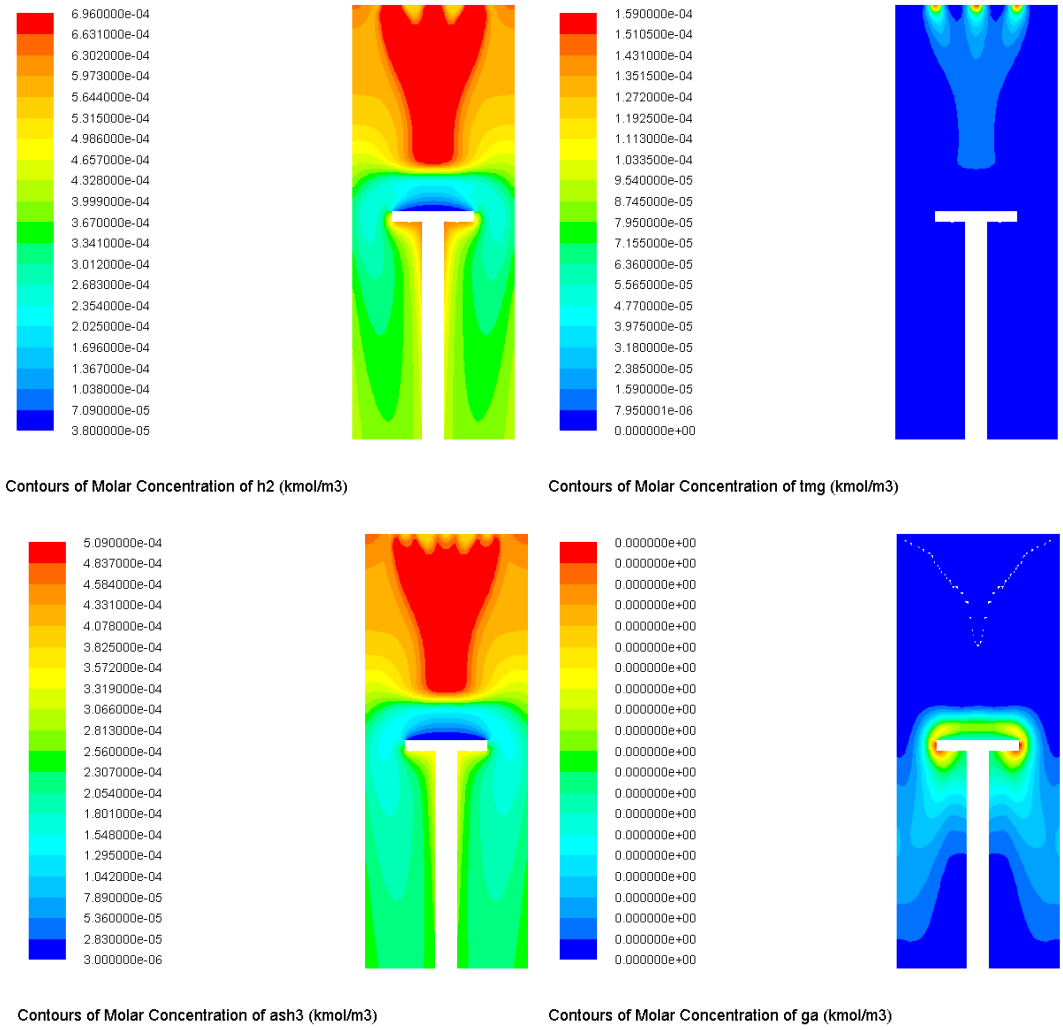


Рис. 3 — Молярные концентрации компонент смеси

В данной работе моделирование осаждения нанопленок осуществляются в реакторе вертикального типа с вращающимся диском (рис.1). Все геометрические размеры реактора (высота, диаметр, высота расположения диска, способ подачи газов в реактор), а также физические (температуры стенок реактора и диска, рабочее давление в реакторе, скорость вращения подложки, расходы газов через входные отверстия) были выбраны в соответствии с данными приведенными в работе [8]. Такой выбор данных позволил обеспечить верификацию полученных результатов, которые представлены на рисунках 2-4, с экспериментальными данными.

Задача решалась с применением программного комплекса Ansys Fluent 12.1 на вычислительном кластере СПбГПУ (64 узла: 2x AMD Opteron 280, 8Гб ОЗУ, IB 4x SDR). В связи с относительно небольшим размером расчетной сетки задачи, для одного расчета использовался только один вычислительный узел. Для получения исчерпывающей информации о физических зависимостях протекающих в реакторе процессов требовалось проведение массовых расчетов. Для этого уже были задействовано до 16 узлов кластера. Всего было решено и исследовано более 1000 расчетных случаев.

Представляется, что главным достижением применяемых технологий математического моделирования является получение всего спектра физических значений (рисунки 2,3) реагирующего газового потока внутри реактора: компоненты скорости, давление, температура, концентрации как исходных, так и результирующих веществ-реагентов. Кроме того, основным результатом моделирования можно считать получение распределения скорости осаждения пленки по

площади подложки, представленные на рисунке 4, находящейся на вращающемся диске, на котором происходит осаждение.

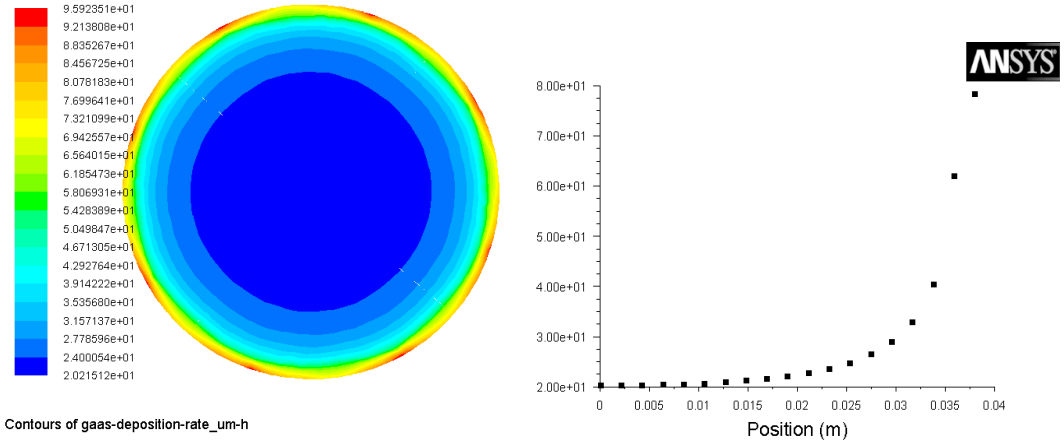
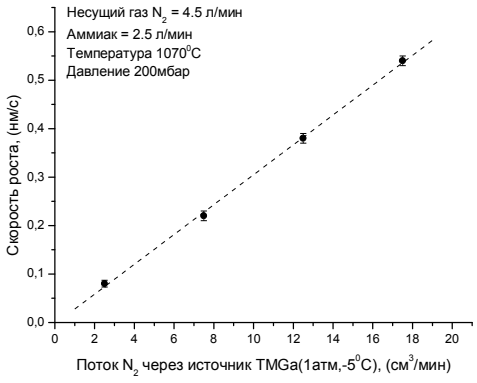
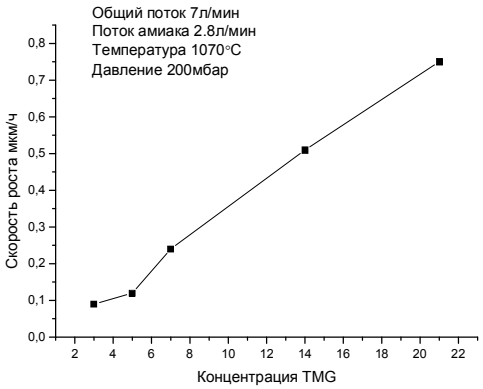


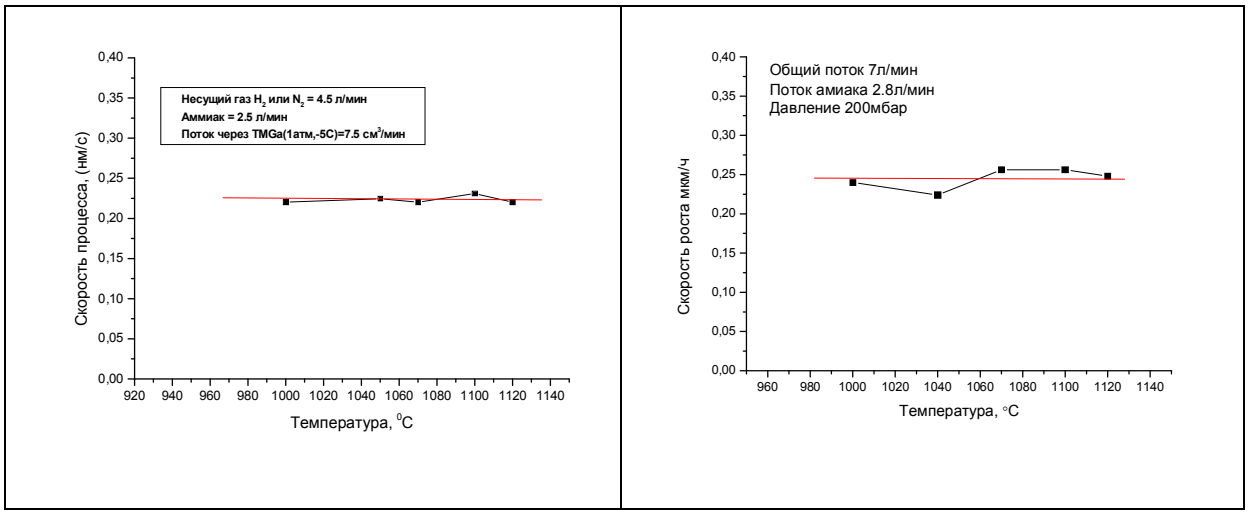
Рис. 4 — Скорость осаждения арсенида галлия

Вместе с тем, несмотря на положительные результаты данного исследования, отсутствие детерминированных моделей, достоверно и однозначно описывающие физико-химические закономерности является существенной проблемой. Как уже отмечалось, данное обстоятельство обусловлено чрезвычайной сложностью механизма CVD процессов, характеризующихся многомаршрутностью химических реакций, присутствием нескольких гомогенных и гетерогенных стадий, а также многоступенчатостью превращений.

Согласованность результатов вычислений с опытным экспериментом демонстрирует таблица 2. Был выполнен ряд расчетов для вертикального реактора с варьируемыми параметрами: концентрацией триметилгаллия и температурой подложки. По полученным массивам данных построена зависимость скорости осаждения. Сравнение результатов расчета с экспериментом показывает полное качественное, а также достаточно хорошее количественное совпадение полученных зависимостей. Что является, несомненно, лучшим подтверждением адекватности выбранных моделей при решении задач химического осаждения тонких пленок из газовой фазы.

Таблица 2 Сравнение результатов вычислений с экспериментом

Результаты опытного эксперимента	**Результаты вычислений**
	



Заключение

По результатам исследования можно утверждать, что выбранные модели физико-химических процессов адекватно описывают реагирующее течение газовой среды и тех инструментов, на основе которых реализуется синтез наноразмерных структур и наноматериалов (подложки). Результаты моделирования осаждения GaAs легко согласуются с экспериментальными данными [8] как количественно, так и качественно.

Использование высокопроизводительных вычислительных технологий позволило расширить круг решаемых задач в области нанотехнологий путем математического моделирования различного технологических инструментов (оборудования) (установки химического осаждения из газовой фазы), технологических параметров (температуры подложки, рабочего давления и др.), а также различных материалов (пленки, порошки и др.).

Использование технологий математического моделирования на базе – высокопроизводительных вычислений позволяет получать информацию о процессе ХОГФ и полученном продукте за достаточно короткое время (в пределах несколько лабораторных работ) и не требует применения дорогостоящего оборудования, которое отсутствует в большинстве учебных заведениях. При этом отметим, что весьма важную роль в образовательном процессе с использованием виртуальных лабораторных практикумов играют технологии удаленного доступа к среде моделирования [11].

Статья подготовлена в рамках выполнения государственного контракта с Министерством образования и науки № 16.647.12.2020 от 25 ноября 2010 г.

Литература

1. Chemical Vapour Deposition. Precursors, Processes and Application / Eds. A.C. Jones, M.L. Hitchman, London: RSC Publishing, 2009, 582 c.
2. В.С. Протопопова, С.Е. Александров Химическое осаждение из газовой фазы слоев ni из бис - (этилциклопентадиенил) никеля/ СПб.: Изд-во Политехн. ун-та, Научно-Технические ведомости СПбГПУ Физико-математические науки.
3. А.А. Уваров, С.Е. Александров. Химическое осаждение из газовой фазы диэлектрических пленок политетрафторэтилена/ СПб.: Изд-во Политехн. ун-та, Научно-Технические ведомости СПбГПУ Физико-математические науки.
4. Александров С.Е. Технология материалов электронной техники. Процессы химического осаждения из газовой фазы: Учеб. пособие. / СПб.: Изд-во Политехн. ун-та, 2005, 92 с.
5. Chemical Vapor Deposition, Principals and Application / Eds. Hitchman M.I. and Jencen K.F. London: Academic Press, 1993, 678 p.
6. Лойцянский Л.Г. Механика жидкости и газа. Учеб. для вузов. Изд. 6-е, перераб. и доп. М.: Наука, 1987, 600 с.
7. FLUENT 6.3 User's Guide

8. S. Mazumder and S. Lowry, The importance of predicting rate-limited growth for accurate modeling of commercial MOCVD reactors, J. Crystal Growth, 224 (2001) 165
9. CHEMKIN/CHEMKIN-PRO Input Manual (August 2010)
10. NIST-JANAF Thermochemical Tables, 4th Edition, M. Chase Monograph No. 9: 1998, 1952 pages, 2 volumes, hardcover, ISBN 1-56396-831-2
11. Д.И. Иванов, Н.В. Захаревич, И.А. Цикин. Визуализация результатов моделирования процессов газофазного синтеза наноразмерных структур при сетевом доступе к кластерному вычислителю/ СПб.: Изд-во Политехн. ун-та, Научно – технические ведомости (в печати).
12. Laidler, K. J. Chemical Kinetics, Third Edition, Benjamin-Cummings,1997.

Алгоритмы решения СЛАУ на системах с распределённой памятью в применении к задачам электромагнетизма *

Д.С. Бутюгин[1,2]

Институт Вычислительной Математики и Математической Геофизики СО РАН[1],
Новосибирский Государственный Университет[2]

В работе рассматриваются различные аспекты решения систем линейных алгебраических уравнений (СЛАУ) на компьютерах с распределённой памятью. Рассмотрены эффективные и экономичные подходы к декомпозиции расчётной области и матрицы системы. Решение распределённых СЛАУ осуществляется методами в подпространствах Крылова с использованием аддитивного метода Шварца в качестве предобуславливателя. Реализованные решатели используют MPI для организации обмена данными. Результаты серии численных экспериментов демонстрируют эффективность предлагаемых алгоритмов.

1. Введение

Многие актуальные задачи вычислительной математики требуют решения разреженных СЛАУ высоких порядков. Задача вычисления трёхмерного электромагнитного поля в частотной области возникает при расчётах различных волновых устройств, решении задач геоэлектроразведки, таких как электромагнитного каротажа, и других. Требования к точности получаемого решения задач с разномасштабными объектами приводят к необходимости построения сеток с числом конечных элементов до $10^5 \div 10^7$ и порядком получаемых после аппроксимации СЛАУ до $10^7 \div 10^9$. Решение таких СЛАУ невозможно без использования вычислительной мощности кластеров. Это требует подходящих алгоритмов решения систем алгебраических уравнений для машин с распределённой памятью.

В данной работе рассматриваются алгоритмы решения СЛАУ, основанные на аддитивном методе Шварца. Рассмотрено два подхода к декомпозиции задачи. Первый из подходов основан на геометрическом разбиении сетки на связанные подобласти и позволяет построить декомпозицию с пересечениями подобластей. Второй подход основан на переупорядочивании графа матрицы системы. Этот способ представляет из себя вариант однонаправленного разбиения графа и приводит к СЛАУ блочно-трёхдиагонального вида. Отличительной особенностью предлагаемых методов является низкая асимптотическая вычислительная сложность алгоритмов, в отличие от вариантов метода вложенных сечений (например, реализованного в библиотеке METIS [1]).

Решение полученных систем уравнений осуществляется методами в подпространствах Крылова [2], такими как обобщённый метод минимальных невязок (GMRES). В качестве предобуславливателя выступает аддитивный метод Шварца. Использование такого подхода, в отличие от использования метода Шварца напрямую, позволяет решить более широкий класс задач и повысить скорость сходимости итерационного процесса.

Структура данной работы следующая. В разделе 2 описывается математическая постановка задачи, в разделе 3 — дискретная постановка. В разделе 4 описаны предлагаемые методы декомпозиции. Раздел 5 содержит описание используемых итерационных алгоритмов. В разделе 6 приведены результаты численных экспериментов. Наконец, в последнем разделе обсуждаются полученные в работе результаты.

*Работа выполнена при поддержке РФФИ (грант 11-01-00205).

2. Математическая постановка задачи

Электромагнитное поле с гармонической зависимостью от времени в случае линейности задачи при отсутствии внешнего заряда и магнитной проводимости может быть описано следующей формой системы уравнений Максвелла:

$$\begin{aligned} \nabla \times \vec{E} &= -i\omega\mu\vec{H}, & \nabla \cdot (\varepsilon_r \vec{E}) &= \rho/\varepsilon_0, \\ \nabla \times \vec{H} &= i\omega\dot{\varepsilon}\vec{E} + \vec{J}, & \nabla \cdot (\mu_r \vec{H}) &= 0, \end{aligned} \tag{1}$$

где электрическая и магнитная проницаемости имеют вид

$$\dot{\varepsilon} = \varepsilon_0\varepsilon_r - i\sigma^e/\omega, \qquad \mu = \mu_0\mu_r - i\sigma^m/\omega.$$

Здесь i — мнимая единица, $\vec{E}$ и $\vec{H}$ — векторы напряжённости электрического и магнитного полей соответственно, ω — круговая частота решения, $\vec{J}$ и ρ — плотности внешнего тока и объёмного заряда соответственно, ε_0 и μ_0 — диэлектрическая и магнитная проницаемости вакуума, ε_r и μ_r — относительная диэлектрическая и магнитная проницаемости среды, а σ^e и σ^m — электрическая и магнитная проводимости.

Используя предположения о линейности задачи (то есть независимости параметров среды от электромагнитного поля), а также отсутствие объёмного внешнего заряда ρ и магнитной проводимости $\sigma^m = 0$, уравнения (1) можно привести к форме комплексного векторного уравнения Гельмгольца

$$\nabla \times \left(\frac{1}{\mu_r} \nabla \times \vec{E} \right) - k_0^2 \dot{\varepsilon}_r \vec{E} = -ik_0 Z_0 \vec{J}. \tag{2}$$

Здесь $k_0 = \omega\sqrt{\varepsilon_0\mu_0}$, $Z_0 = \sqrt{\mu_0/\varepsilon_0}$ и $\dot{\varepsilon}_r = \dot{\varepsilon}/\varepsilon_0$. Параметры среды полагаются кусочно-постоянными. Мы будем предполагать, что k_0 не является Максвелловским собственным числом, то есть рабочая частота ω не является резонансной. Стоит отметить, что в случае, если хотя бы в одной подобласти $\sigma^e > 0$, то рабочая частота может быть любой, поскольку в системе отсутствуют резонансы [3].

Решение ищется в области Ω с границей $S = S_1 \cup S_2$, на каждой из частей которой поставлено одно из следующих граничных условий:

$$\vec{n} \times \vec{E} \Big|_{S_1} = \vec{n} \times \vec{E}_0, \qquad \vec{n} \times \vec{H} \Big|_{S_2} = 0, \tag{3}$$

где $\vec{n}$ — внешняя нормаль к границе. При $\vec{E}_0 = 0$ первое из условий соответствует идеальному электрическому проводнику, при $\vec{E}_0 \neq 0$ — волновому входу, а второе условие соответствует идеальному магнитному проводнику. Полагая область Ω состоящей из подобластей $\Omega = \bigcup \Omega_k$, в каждой из которых физические параметры среды являются константными, введём на каждой из внутренних границ Γ с нормалью $\vec{n}$ условия сопряжения

$$\begin{aligned} \vec{n} \cdot (\dot{\varepsilon}_1 \vec{E}_1 - \dot{\varepsilon}_1 \vec{E}_2) &= 0, & \vec{n} \times (\vec{E}_1 - \vec{E}_2) &= 0, \\ \vec{n} \cdot (\mu_1 \vec{H}_1 - \mu_2 \vec{H}_2) &= 0, & \vec{n} \times (\vec{H}_1 - \vec{H}_2) &= 0. \end{aligned} \tag{4}$$

Вводятся стандартные соболевские пространства и подмножества:

$$H^{\mathrm{rot}} = \left\{ \vec{\psi} \in \left[L^2(\Omega)\right]^3 : \nabla \times \vec{\psi} \in \left[L^2(\Omega)\right]^3 \right\},$$
$$H_0^{\mathrm{rot}} = \left\{ \vec{\psi} \in H^{\mathrm{rot}} : \vec{n} \times \vec{\psi}|_{S_1} = 0 \right\}, \; H_S^{\mathrm{rot}} = \left\{ \vec{\psi} \in H^{\mathrm{rot}} : \vec{n} \times \vec{\psi}|_{S_1} = \vec{n} \times \vec{E}_0 \right\}.$$

Вариационная формулировка уравнения (2) с краевыми условиями (3) в форме Галёркина имеет следующий вид (см. [4]): найти такое $\vec{E} \in H_S^{\mathrm{rot}}$, что для всех $\vec{\psi} \in H_0^{\mathrm{rot}}$ выполнено условие

$$\int\limits_\Omega \frac{1}{\mu_r}\left(\nabla\times\vec{E}\right)\cdot\left(\nabla\times\vec{\psi}\right)\mathrm{d}\Omega - k_0^2\int\limits_\Omega \dot{\varepsilon}_r\left(\vec{E}\cdot\vec{\psi}\right)\mathrm{d}\Omega = -ik_0Z_0\int\limits_\Omega \vec{J}\cdot\vec{\psi}\mathrm{d}\Omega, \tag{5}$$

3. Дискретная постановка задачи

В работе используются неструктурированные тетраэдральные сетки. Рассматривается соответствующее разбиение расчётной области Ω на непересекающиеся тетраэдральные элементы $\Omega = \bigcup\Omega_T$. В каждом из тетраэдров вводятся базисные функции, соответствующие его степеням свободы. Пусть $\mathcal{W}_l$ — конечномерное пространство базисных функций порядка не выше l, конформное H^{rot}. В работе рассматриваются иерархические базисные функции, предложенные в [5]:

$$\begin{aligned}\mathcal{W}_l &= \tilde{\mathcal{W}}_1 \oplus \tilde{\mathcal{W}}_2 \oplus \ldots \oplus \tilde{\mathcal{W}}_l,\\ \tilde{\mathcal{W}}_1 &= \tilde{A}_1, \quad \tilde{\mathcal{W}}_i = \tilde{\mathcal{A}}_i \oplus \nabla\tilde{\mathcal{V}}_i,\end{aligned}$$

где $\tilde{\mathcal{W}}_i$ — инкрементальное подпространство с базисными функциями порядка i, $\tilde{\mathcal{V}}_i$ — инкрементальное подпространство со скалярными базисными функциями, конформными H^1, а $\tilde{\mathcal{A}}_i$ — инкрементальное подпространство с роторными базисными функциями.

Возможные варианты базисных функций (см. [5]) представлены в таблицах 1 и 2, где за $\xi_i(\vec{r})$ обозначены кусочно-линейные функции, каждая из которых равна 1 в одном из узлов сетки и 0 — в остальных.

Таблица 1. Скалярные базисные функции

Пространство	Функции	Ассоциированы с
$\tilde{\mathcal{V}}_1$	ξ_i	узлами {i}
$\tilde{\mathcal{V}}_2$	$\xi_i\xi_j$	ребрами {ij}
$\tilde{\mathcal{V}}_3$	$\xi_i\xi_j(\xi_i-\xi_j)$	ребрами {ij}
	$\xi_i\xi_j\xi_k$	гранями {ijk}

Подпространства $\mathcal{V}_{l,0}$ и $\mathcal{W}_{l,0}$ вводятся естественным образом, а подмножество $\mathcal{W}_{l,S}$ — как множество функций, у которых коэффициенты разложения u^S по базисным функциям с ненулевым следом на S_1 принимают фиксированные значения, соответствующие краевому условию (3).

Приближенное решение $\vec{E}^h$ будем искать в виде

$$\vec{E}^h = \sum_j u_j^S\vec{\psi}_j^S + \sum_i u_i\vec{\psi}_i^0. \tag{6}$$

Вводятся матрицы соответствующих билинейных форм:

$$M_{i,j} = \int\limits_\Omega \dot{\varepsilon}_r\left(\vec{\psi}_j^0\cdot\vec{\psi}_i^0\right)\mathrm{d}\Omega, \quad S_{i,j} = \int\limits_\Omega \frac{1}{\mu_r}\left(\nabla\times\vec{\psi}_j^0\right)\cdot\left(\nabla\times\vec{\psi}_i^0\right)\mathrm{d}\Omega,$$

где базисные функции $\vec{\psi}_i^0, \vec{\psi}_j^0 \in \mathcal{W}_{l,0}$. Вектор правой части определяется как

$$f_i = -ik_0Z_0\int\limits_\Omega\left[\vec{J}\cdot\vec{\psi}_i^0 + \sum_j u_j^S\left(\frac{1}{\mu_r}\left(\nabla\times\vec{\psi}_j^S\right)\cdot\left(\nabla\times\vec{\psi}_i^0\right) - k_0^2\dot{\varepsilon}_r\vec{\psi}_j^S\cdot\vec{\psi}_i^0\right)\right]\mathrm{d}\Omega.$$

Таблица 2. Векторные роторные базисные функции

Пространство	Функции	Ассоциированы с
$\tilde{\mathcal{A}}_1$	$\xi_i\nabla\xi_j - \xi_j\nabla\xi_i$	ребрами {ij}
$\tilde{\mathcal{A}}_2$	$3\xi_j\xi_k\nabla\xi_i - \nabla(\xi_i\xi_j\xi_k)$ $3\xi_k\xi_i\nabla\xi_j - \nabla(\xi_i\xi_j\xi_k)$	гранями {ijk}
$\tilde{\mathcal{A}}_3$	$4\xi_j\xi_k(\xi_j-\xi_k)\nabla\xi_i-\nabla(\xi_i\xi_j\xi_k(\xi_j-\xi_k))$ $4\xi_k\xi_i(\xi_k-\xi_i)\nabla\xi_j-\nabla(\xi_i\xi_j\xi_k(\xi_k-\xi_i))$ $4\xi_i\xi_j(\xi_i-\xi_j)\nabla\xi_k-\nabla(\xi_i\xi_j\xi_k(\xi_i-\xi_j))$	гранями {ijk}
	$4\xi_j\xi_k\xi_l\nabla\xi_i - \nabla(\xi_i\xi_j\xi_k\xi_l)$ $4\xi_k\xi_l\xi_i\nabla\xi_j - \nabla(\xi_i\xi_j\xi_k\xi_l)$ $4\xi_l\xi_i\xi_j\nabla\xi_k - \nabla(\xi_i\xi_j\xi_k\xi_l)$	тетраэдрами {ijkl}

Тогда итоговая система принимает вид

$$\left[S - k_0^2 M\right] u = f. \tag{7}$$

Для вычисления элементов матрицы и вектора правой части можно воспользоваться поэлементной технологией сборки [6], заменив интегрирование по расчётной области Ω суммой интегралов по каждому из тетраэдров и вычислением в каждом из тетраэдров локальных матриц и векторов.

4. Методы декомпозиции на подобласти

Для решения задачи на системах с распределённой памятью требуется распределить данные между узлами системы, по возможности минимизируя объём коммуникационных данных. Основной проблемой здесь является то, что алгоритмы, генерирующие разбиения высокого качества, сами требуют больших вычислительных ресурсов. Поэтому, как правило, приходится идти на компромисс и выбирать алгоритмы, позволяющие получить достаточно хорошее разбиение за приемлемое время.

4.1. Геометрическая декомпозиция расчётной области

Предлагаемый в работе алгоритм геометрической декомпозиции расчётной области является упрощённым вариантом алгоритма построения BSP-дерева (binary space partitioning, двоичное разбиение пространства). Метод двоичного разбиения — это метод рекурсивного разбиения пространства на выпуклые множества гиперплоскостями [7]. Получающаяся в результате структура данных находит широкое применение в различных областях компьютерной графики для проверки пересечений, определения прямой видимости объектов и др. Стоит отметить, что построение оптимального BSP дерева представляет из себя очень трудоёмкую задачу, поэтому, в большинстве случаев, ограничиваются субоптимальными деревьями.

В рамках данной работы предлагается использовать следующее упрощённое двоичное дерево. Упрощение состоит в том, что секущие плоскости предлагается проводить ортогонально осям координат. При этом мы будем каждый раз разделять имеющееся множество

тетраэдров на две приблизительно равные части. Алгоритм построения разбиения будет основан на методе обработки событий. Подготовительная часть алгоритма такая:

- для x, y и z: спроектировать тетраэдры на соответствующую ось; для каждого тетраэдра получаем отрезки, соответствующие его началу и концу;
- для x, y и z: сгенерировать отсортированные по координате события — начало и конец каждого из тетраэдров.

В результате мы получим 3 отсортированных списка событий, для осей Ox, Oy и Oz соответственно.

Далее к этим спискам мы рекурсивно будем применять следующую процедуру. Будем последовательно обрабатывать события для каждой из осей, что соответствует движению секущей плоскости в сторону увеличения соответствующей координаты. При этом мы будем подсчитывать число тетраэдров в каждом из полупространств. Тетраэдры, пересекающиеся и касающиеся плоскости сечения, будем относить к обеим частям одновременно. Схема алгоритма следующая:

- Для x, y и z:
  - left_count $\leftarrow 0$, right_count $\leftarrow T$, где T — число тетраэдров.
  - Для каждой координаты последовательно: обработать события
    - Обработать начала тетраэдров: для каждого left_count $\leftarrow$ left_count $+ 1$.
    - Проверить left_count и right_count на оптимальность.
    - Обработать концы тетраэдров: right_count $\leftarrow$ right_count $- 1$.
  - Запомнить оптимальное разбиение для оси.
- Выбрать наилучшую ось для разбиения.
- Построить разбиение:
  - Для x, y и z:
    - Линейным проходом разбить список событий для соответствующей оси на 2 в соответствии с разбиением. При этом упорядоченность сохранится.
- Рекурсивно запустить алгоритм для двух полученных частей.

Алгоритм останавливает рекурсивное разбиение тогда, когда количество тетраэдров в обрабатываемой части становится достаточно мало, либо когда в результате деление количество тетраэдров в одной из частей оказался равен исходному их числу.

Проверка на оптимальность для каждой из осей может быть следующей: выбирается положение плоскости, при котором размеры левой и правой частей максимально близки друг к другу. Наилучшая из осей выбирается по минимуму $\min(\text{left\_count}, \text{right\_count})$ для оптимального положения плоскости, что позволяет выбрать минимальный из разрезов подпространства и уменьшить коммуникации между подобластями. Отметим, что "толщину" слоя пересечения легко варьировать. Для этого достаточно осуществить проход по событиям двумя указателями вместо одного. Один из указателей при этом будет соответствовать началу пересечения подобластей, второй — концу. При этом легко поддерживать как нужную геометрическую толщину пересечения подобластей, так и минимальное число тетраэдров в пересечении.

Легко видеть, что асимптотическая сложность алгоритма равна $O(T \log T)$, где T — исходное число тетраэдров, при условии, что каждый раз в пересечении оказывается небольшое число тетраэдров и части делятся примерно поровну. Действительно, сложность сортировки событий в начале алгоритма равна $O(T \log T)$. Далее, если число тетраэдров растет

несущественно по сравнению с величиной T (например, конечное число тетраэдров есть $O(T)$), то на каждом уровне дерева требуется линейный проход по всем событиям для каждого из узлов дерева этого уровня, суммарное количество которых есть $O(T)$. При этом число подобластей при переходе на уровень ниже увеличивается в 2 раза, поэтому общее число уровней не превзойдет $O(\log T)$. Отсюда получаем заявленную сложность.

Замечание 1. Алгоритм осуществляет разбиение сетки на подобласти с пересечениями, причём никакие две подобласти не имеют протяжённой общей границы кроме, возможно, части внешней границы расчётной области.

Доказательство этого факта следует из следующего тривиального наблюдения: два тетраэдра с общей гранью невозможно разбить плоскостью так, чтобы один тетраэдр лежал по одну сторону от неё, второй по другую и плоскость не касалась хотя бы одного из тетраэдров. При этом, как следует из алгоритма, тетраэдры, касающиеся либо пересекающиеся с плоскостью сечения, помещаются в пересечение подобластей. Данный факт оказывается очень важным для дальнейшего решения задачи методом декомпозиции на подобласти.

Рассмотрим теперь постановку задачи нахождения поля для подобластей. Пусть имеется разбиение расчётной области Ω на подобласти $\Omega = \bigcup \Omega_p$. Будем искать решение задачи (5) на области Ω путём решения задачи (5) в каждой из подобластей Ω_p. На внутренней границе Γ каждой из подобластей Ω_p будем ставить условия Дирихле из (3). Эти условия будут определяться из значений поля на внутри подобластей, соседних с Ω_p, и, на конечно-элементном уровне, из коэффициентов разложения поля $\vec{E}$ по базисным функциям, соответствующим граничным рёбрам и граням. То есть, требуется найти поля в каждой из подобластей так, чтобы они были согласованы друг с другом по краевым условиям.

Теорема 1. Решение задачи для подобластей для предлагаемого алгоритма декомпозиции существует. Оно единственно, если частота излучения не совпадает с резонансными частотами полостей, образованных подобластями и их пересечениями.

Доказательство. Ясно, что решение исходной задачи на всей области Ω удовлетворяет задаче на подобластях. Единственность следует из следующей леммы.

Лемма 1. Пусть поле $\vec{E}$ является решением задачи для подобластей. Тогда для любых Ω_p и $\Omega_{p'}$, имеющих непустое пересечение, значения $\vec{E}$ в общих точках Ω_p и $\Omega_{p'}$ совпадают.

Доказательство напрямую следует из того, что для пересечения подобластей мы имеем краевую задачу, которая, если частота не резонансная для данной задачи, имеет единственное решение. Поэтому решение задачи для подобластей однозначно определяет электрическое поле в любой точке исходной области Ω.

Доказательство теоремы 1 завершается замечанием того, что

- для любой внутренней точки области Ω существует замкнутая окрестность, что эта точка вместе с окрестностью целиком лежит внутри какой-либо подобласти Ω_p;
- для любой базисной функции из $\mathcal{W}_{l,0}$ её носитель целиком лежит внутри какой-нибудь подобласти Ω_p;

Отсюда следует, что электрическое поле, являющееся решением задачи для подобластей, удовлетворяет и вариационной формулировки (5) в непрерывном случае, и дискретному аналогу этой формулировке в конечно-элементном случае. Однако решение вариационной формулировки единственно при указанных ограничениях на параметры среды и частоту излучения [3], поэтому единственным оказывается и решение задачи для подобластей.

Теорема доказана.

4.2. Алгебраическая декомпозиция

Другим вариантом декомпозиции задачи на подобласти, рассматриваемым в рамках работы, является модифицированный алгоритм одно-направленного разбиения графа матрицы системы [2]. Алгоритм строит разбиение неизвестных системы на множества — алгебраические подобласти Ω_p^A. В дальнейшем индекс A в Ω_p^A мы будем опускать.

Первым шагом алгоритма является нахождение псевдо-периферийной вершины v. Поиск такой вершины осуществляется следующим образом [2].

1. Выбирается произвольная вершина v. Псевдо-диаметр графа D полагается равным 0.
2. Запускается обход в ширину по графу, начиная от вершины v.
3. Находится любая максимально удалённая вершина v'.
4. Если расстояние $d(v, v')$ больше D, то $v \leftarrow v'$, $D \leftarrow d(v, v')$, переход на шаг 2.

Информация с последнего обхода графа в ширину используется для дальнейшего построения разбиения. Для начала все вершины разбиваются во "фронты" F_k — множества вершин, равноудалённых от v на расстояние k. Далее фронты объединяются в алгебраические подобласти по следующему принципу.

С помощью бинарного поиска мы будем максимизировать размер минимальной подобласти. Проверка того, что при данном минимальном размере подобласти можно получить необходимое число подобластей, может быть осуществлена жадным алгоритмом [8]. Пример кода, определяющего максимальное число подобластей, представлен на рисунке 1.

```
int get_max_domains(int k, int front_size[], int min_size)
{
    int domains = 0, domain_size = 0, i;
    for(i = 0; i < k; i++) {
        domain_size += front_size[i]; // Размер в вершинах
        if(domain_size >= min_size) {
                ++domains;
                domain_size = 0;
        }
    }
    // Оставшиеся вершины (domain_size) отходят к последней подобласти
    return domains;
}
```

Рис. 1. Алгоритм определения максимального числа подобластей

Представленный алгоритм генерирует подобласти без пересечений. Однако получить разбиение с пересечениями довольно легко — достаточно при начале генерации следующей подобласти отступить от границы предыдущей внутрь необходимое число фронтов. Графическое представление объединения фронтов в подобласти с пересечениями изображено на рисунке 2, где фронты изображены вертикальными линиями.

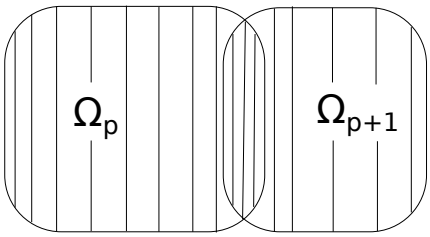


Рис. 2. Объединение фронтов матрицы в подобласти

Обозначив l_p и r_p — границы подобласти Ω_p, мы имеем разбиение на подобласти

$$\Omega_p = \bigcup_{k=l_p}^{r_p} F_k,$$

Необходимо отметить, что ребра в графе матрицы могут соединять вершины либо одного, либо двух соседних фронтов. Это означает, что каждая подобласть Ω_p может граничить

одновременно максимум с двумя подобластями — Ω_{p-1} и Ω_{p+1} (при наличии пересечений всегда можно считать, что два граничных с Ω_p фронта F_{l_p-1} и F_{r_p+1} принадлежат соседним подобластям Ω_{p-1} и Ω_{p+1} соответственно). Тогда последовательно занумеровав переменные в каждой из подобластей, а в каждой из подобластей — последовательно в каждом из фронтов, мы получим следующую блочно-трёхдиагональную СЛАУ:

$$\begin{bmatrix} D_1 & U_1 & & \\ L_1 & D_2 & \ddots & \\ & \ddots & \ddots & U_{p-1} \\ & & L_{p-1} & D_p \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \end{bmatrix}.$$

При этом если разбиение исходной СЛАУ выполнялось с пересечениями, то в итоговой СЛАУ окажется по несколько переменных, соответствующих вершинам в пересечении.

5. Алгоритмы решения СЛАУ

После декомпозиции задачи на подобласти и построения блоков матрицы на соответствующих узлах кластера, мы имеем распределённую СЛАУ

$$Au = f.$$

Для решения таких СЛАУ широко используются методы Шварца. Одна итерация аддитивного метода Шварца в матричном виде может быть записана как (см. [2])

$$u_{i+1} = u_i + \sum_p R_p^T A_p^{-1} R_p (f - Au_i),$$

где R_p — оператор сужения на подобласть Ω_p. При этом, на самом деле, итерировать достаточно только переменные, соответствующие внутренним границам подобластей, поскольку зная граничные значения, легко определить значения внутри подобластей, решив соответствующую задачу в подобласти. То есть

$$u_p = A_p^{-1} R_p (f - A\hat{R}^T u^\Gamma), \tag{8}$$

где u^Γ — вектор граничных переменных, а $\hat{R}$ — оператор сужения на граничные переменные. Подпространство граничных переменных принято называть подпространством следов. Тогда итерация метода Шварца в подпространстве следов переписывается как

$$u_{i+1}^\Gamma = u_i^\Gamma + \hat{R} \sum_p R_p^T A_p^{-1} R_p (f - A\hat{R}^T u_i^\Gamma).$$

Эту итерацию можно представить в виде $u_{i+1}^\Gamma = S(u_i^\Gamma) = Tu_i^\Gamma + g$, где

$$T = I - \hat{R} \sum_p R_p^T A_p^{-1} R_p A \hat{R}^T, \quad g = \hat{R} \sum_p R_p^T A_p^{-1} R_p f.$$

Сходимость данного метода имеется при спектральном радиусе $\rho(T) < 1$. В то время как для эллиптических уравнений это условие выполнено, для уравнения Гельмгольца это не так. Однако в этом случае можно рассмотреть решение системы

$$[I - T]\, u^\Gamma = g. \tag{9}$$

Теорема 2. Система (9) является совместной. В случае геометрической декомпозиции при выполнении условий теоремы 1 оно единственно.

Доказательство. Действительно, система (9) эквивалентна системе

$$\hat{R}\sum_p R_p^T A_p^{-1} R_p A u = \hat{R}\sum_p R_p^T A_p^{-1} R_p f,$$

поэтому решение $u^\Gamma = \hat{R}u$, где u — решение $Au = f$ удовлетворяет ей. С другой стороны любое решение (9) является неподвижной точкой итерации Шварца. В случае геометрической декомпозиции на подобласти у неподвижной точки итерации Шварца есть простая интерпретация. Это вектор, задающий поле в подобластях, согласованное по граничным условиям. Однако при выполнении условий теоремы 1 последняя утверждает, что такое поле единственно. Значит, единственна и неподвижная точка итерации Шварца.

Теорема доказана.

Можно отметить, что решение системы методом Шварца есть решение предобусловленной системы $M^{-1}Au = M^{-1}f$ с M^{-1}, связанным с A и T соотношением $T = I - M^{-1}A$, откуда $M^{-1} = [I - T]A^{-1}$ и предобусловленная система переходит в систему (9). Эту систему можно решать и итерационными методами в подпространствах Крылова, например методом обобщенных минимальных невязок (GMRES) [2]. Для методов в подпространствах Крылова не требуется явный вид матрицы системы, а достаточно только операции умножения матрицы на вектор. Запишем результат умножения $I - T$ на вектор v через результат одной итерации метода аддитивного Шварца $S(v)$:

$$[I - T]\, v = v - Tv = v - S(v) + g, \quad g = S(0).$$

Таким образом, для решения СЛАУ (9) методом GMRES достаточно реализовать обычный метод аддитивного Шварца. Для решения задач в подобластях $A_p x = b$ можно использовать какой-нибудь прямой решатель для разреженных систем. В работе использовался решатель PARDISO из библиотеки Intel MKL [9]. Такой подход оказывается достаточно эффективным, поскольку на каждой итерации требуется решать системы с одними и теми же матрицами, и LU-разложение матриц A_p можно посчитать только один раз. Кроме того, такой подход оказывается существенно эффективнее простого использования решателя PARDISO для исходной системы, поскольку время, требуемое PARDISO для разложения матрицы, растёт практически как $O(N^3)$, где N — порядок системы. Дополнительно эффективность достигается за счёт использования подпространства следов, так как в этом случае существенно уменьшается объём коммуникационных затрат, а также объём памяти, требуемый для хранения базиса подпространства Крылова в методе GMRES, поскольку хранить и пересылать требуется только граничные переменные. После того, как решение системы найдено, искомое поле $\vec{E}$ восстанавливается в каждой из подобластей.

6. Численные эксперименты

6.1. Модельная задача

В качестве одного из тестов рассматривалась модельная задача с расчётной областью в виде волновода с линейными размерами $a = 72$, $b = 34$, $c = 200$ мм. Параметры задачи: $\mu_r = 1$, $\dot{\varepsilon}_r = 1 - 0.1i$, частота $\omega = 6\pi \cdot 10^9$ Гц. Граничные условия: на грани $z = 200$ задавалось условие

$$\vec{E}_0 \times \vec{n} = \vec{e}_y \sin(\pi x/a),$$

на остальных гранях — условие металлической стенки ($\vec{E}_0 = 0$). Аналитическое решение

$$\vec{E} = \vec{e}_y \sin\left(\frac{\pi x}{a}\right) \frac{\sin \gamma z}{\sin \gamma c}.$$

При решении СЛАУ критерием остановки итерационного процесса служило выполнение условия $(r, r) < \varepsilon^2 (f, f)$, где f — вектор правой части системы, $r = f - Au$ — вектор невязки. В проведенных экспериментах ε было выбрано равным 10^{-7}.

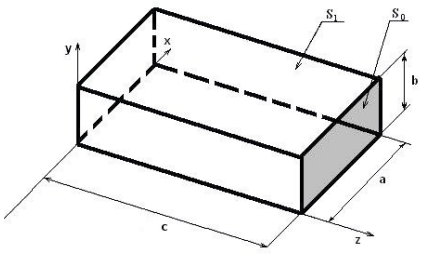


Рис. 3. Расчётная область для модельной задачи

Для построения сетки расчетная область делилась по каждому из измерений на некоторое число шагов, и каждый из получающихся параллелепипедов делился на 6 тетраэдров. В тестах использовалась квазиравномерная сетка с разбиениями по каждой из координат с некоторым шагом h. Были рассмотрены сетки с разбиениями $4 \times 2 \times 10$, $8 \times 4 \times 20$ и $15 \times 7 \times 40$. Для аппроксимации использовались базисные функции второго порядка.

Тесты проводились на двух системах: системе Intel Core2 Duo E6550 @ 2.33 ГГц, 2 ядра, с оперативной памятью объемом 4 ГБ, и на узлах HP BL2x220 G6 кластера НКС-30Т. Каждый узел — Intel Xeon E5540, 2.53 ГГц, 2x4 ядер, 16 ГБ памяти.

Таблица 3. Решение модельной задачи с геометрической декомпозицией

Сетка	N	δE		nproc					
				2	4	8	16	32	64
$4 \times 2 \times 10$	2392	3.1e-02	n	8	11	22	—	—	—
			N_b	136	408	2656	—	—	—
			N_p	1408	916	670	—	—	—
$8 \times 4 \times 20$	21664	7.3e-03	n	9	14	23	40	53	70
			N_b	592	1776	4556	15764	40396	105200
			N_p	11782	6292	4522	2610	1834	1058
$15 \times 7 \times 40$	149874	2.1e-03	n	10	17	27	50	64	80
			N_b	2012	6036	14084	34560	67872	143256
			N_p	78206	40486	21626	12818	8346	5478

В таблице 3 приведены результаты тестирования решателя при использовании геометрической декомпозиции. В таблице nproc — число MPI процессов (и подобластей), N — порядок системы для nproc=1, N_b — размерность пространства следов, N_p — максимальный порядок СЛАУ для подобластей, n — число итераций решателя GMRES, δE — относительная ошибка для поля $\vec{E}$ в барицентрах тетраэдров. Тесты для сетки $4 \times 2 \times 10$ при 16, 32 и 64 подобластях не проводились, так как при этом не хватает тетраэдров сетки для построения нужного числа подобластей. Тесты проводились на двухъядерном Core2 Duo, поэтому время работы решателей не приводится.

В таблице 4 приведены результаты тестирования для случая алгебраической декомпо-

Таблица 4. Решение модельной задачи с алгебраической декомпозицией

Сетка	N	δE		nproc				
				2	4	8	16	32
$4 \times 2 \times 10$	2392	3.1e-02	n	6	12	49	—	—
			N_b	378	980	2261	—	—
			N_p	1542	1334	796	—	—
$8 \times 4 \times 20$	21664	7.3e-03	n	9	13	21	258	—
			N_b	1540	4678	10708	23052	—
			N_p	12420	9232	6306	3200	—
$15 \times 7 \times 40$	149874	2.1e-03	n	13	18	25	38	517
			N_b	5210	16646	37670	78446	162720
			N_p	81132	49618	31168	22678	11282

зиции. Параметр пересечения подобластей был установлен в 4, то есть подобласти пересекались, если это было возможно, по 4 фронтам. Из таблицы видно, что при декомпозиции матрицы на слишком большое число подобластей число итераций начинает быстро расти. В этих случаях на каждую подобласть приходится всего по нескольку фронтов.

6.2. Задача электромагнитного каротажа

Вторая из рассматриваемых задач — задача электромагнитного каротажа. Расчётная область представляет из себя куб с центром в начале координат со стороной 10 метров. Куб заполнен средой с $\sigma = 0.1$ См/м, в среде имеется слой с $\sigma = 0.05$ См/м. Координаты слоя по оси Oz: от -0.425 м до -0.275 м. В среде пробурена вертикальная цилиндрическая скважина радиуса 0.108 м с центром в начале координат в плоскости Oxy. В скважине имеется цилиндрическая каверна внешнего радиуса 0.118 м и положением по оси Oz от -0.0725 до 0.0725 м. Проводимость скважины и каверны $\sigma = 5$ См/м. В скважину вставлен полый цилиндрический прибор радиуса 0.043 м, смещённый по оси x относительно центра скважины на -0.064. Проводимость прибора равна нулю. В приборе имеется одна генераторная петля при $z = 0.5$ м и две приёмные петли при $z = 0.0$ и $z = 0.1$ м. Радиус петель — 0.0365 м. По генераторной петле течёт ток 1 А с частотой 14 МГц. Искомой является разность фаз ЭДС в приёмниках.

Генерация неравномерной сетки проводилась при помощи утилиты NETGEN v4.9.13 [10]. В результате была получена сетка с 71892 узлами и 388836 тетраэдрами.

Результаты решения задачи на кластере с использованием алгебраической декомпозиции представлены в таблице 5. В ней дополнительно указаны времена t_{fact} — максимальное время факторизации матриц D_p в подобластях и t_{tot} — общее время работы решателя. Результаты тестирования для nproc < 4 не приведены, так как в этом случае PARDISO не хватает 16 Гб памяти на узле для хранения LU факторов матрицы, а режим Out-of-Core с хранением факторов на диске является слишком медленным.

Результат решения задачи показал хорошее соответствие с результатом группы НГТУ, которая решала эту задачу другим методом — методом выделения поля источника, при этом

Таблица 5. Решение задачи каротажа с алгебраической декомпозицией матрицы

nproc	N	N_b	t_{fact}, сек	t_{tot}, сек	n
4	2398750	371126	8.52e+02	9.49e+02	21
8	2398750	833744	3.30e+02	4.26e+02	28

искомая разность фаз ЭДС в приёмниках совпала с ошибкой 1.8%. Стоит отметить также, что для данной задачи обычные итерационные алгоритмы в подпространствах Крылова, применённые к СЛАУ для всей подобласти, показывают отсутствие сходимости.

7. Заключение

В работе рассматривается задача моделирования электромагнитного поля в частотной области. Предложено два эффективных алгоритма геометрической и алгебраической декомпозиции задачи, причём последний может применяться и для решения других задач. Предлагаемые итерационные алгоритмы в подпространствах следов с предобуславливателем Шварца требуют невысоких коммуникационных затрат и хорошо подходят для вычислительных систем с распределённой памятью. Реализация алгоритмов на языке C++ с использованием средств MPI продемонстрировала хорошую производительность на методических и практических задачах. Предложенные решатели позволили решить задачу электромагнитного каротажа с высокой точностью.

Литература

1. Karypis G., Kumar V. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, N. 1, P. 359–392.
2. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, 2003.
3. Monk P. Finite Element Methods for Maxwell's Equations. Oxford University Press, 2003.
4. Соловейчик Ю.Г., Рояк М.Э., Персова М.Г. Метод конечных элементов для решения скалярных и векторных задач. Новосибирск: Изд-во НГТУ, 2007.
5. Ingelstrom P. A new set of H(curl)-conforming hierarchical basis functions for tetrahedral meshes // IEEE Transactions on Microwave Theory and Techniques. January 2006. Vol. 54. N. 1. P. 160–114.
6. Ильин В.П. Методы и технологии конечных элементов. Новосибирск: Изд-во ИВМиМГ СО РАН, 2007. 370 с.
7. Fuchs H., Kedem Z.M., Naylor B.F. On visible surface generation by a priori tree structures // ACM Computer Graphics. July 1980. Vol. 14, N. 3. P. 124–133.
8. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М., МЦНМО: БИНОМ. Лаборатория знаний, 2004. 960 с.
9. Intel (R) Math Kernel Library Reference Manual: URL: http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/mklxe/mkl_manual_win_mac/index.htm
10. Schöberl J. NETGEN — An advancing front 2D/3D-mesh generator based on abstract rules // Computing and Visualization in Science. July 1997. Vol. 1, N. 1. P. 41–52.

Методы параллельного решения СЛАУ на системах с распределённой памятью в библиотеке Krylov *

Д.С. Бутюгин[1,2], В.П. Ильин[1], Д.В. Перевозкин[1]

Институт Вычислительной Математики и Математической Геофизики СО РАН[1],
Новосибирский Государственный Университет[2]

В работе рассматривается подход к созданию итерационного black-box ("чёрного ящика") параллельного решателя, использованный в библиотеке Krylov для систем линейных алгебраических уравнений (СЛАУ) с разреженными матрицами, представленными в сжатом строчном формате CSR. Реализованные алгоритмы апробированы на численном решении ряда задач вычислительной математики, таких как задачи гидродинамики, диффузионно-конвективные уравнения, задачи электромагнетизма и др. Приведённые результаты численных экспериментов демонстрируют эффективность предлагаемых решений для многопроцессорных вычислительных систем с распределённой памятью.

1. Введение

Системы линейных алгебраических уравнений с разреженной матрицей возникают при численном решении большого числа задач вычислительной математики. Вычислительная сложность таких задач часто требует кластерных расчётов. В настоящее время существует ряд подходов к декомпозиции задач и, соответственно, матриц систем, а также итерационных алгоритмов, позволяющих свести решение исходной задачи к серии решений задач в подобластях. К методам декомпозиции задачи можно отнести геометрические методы декомпозиции расчётной области и последующую аппроксимацию задачи в каждой из подобластей, а также алгебраические методы декомпозиции матрицы системы, например, реализованные в библиотеке METIS [1].

В рамках данной работы предлагается вариант алгебраической одномерной декомпозиции СЛАУ. Алгоритм основан на обходе в ширину графа матрицы системы и позволяет привести её к блочно-трехдиагональному виду. За основу алгебраического решателя системы взят аддитивный метод Шварца, который естественным образом ложится на архитектуру вычислительных систем с распределённой памятью. К данному методу применяется крыловское ускорение [2], заключающееся в замене стационарного процесса $x_{n+1} = Tx_n + g$ решением системы $(I - T)x = g$, которое осуществляется при помощи итерационных методов в подпространствах Крылова, таких как GMRES. При этом итерации осуществляются в подпространстве следов, образованных переменными на границах подобластей. Решение систем в подобластях осуществляется параллельным прямым решателем разреженных систем PARDISO из библиотеки Intel® MKL [3]. Использование такого подхода позволяет реализовать black-box решатель, не требующий дополнительной информации для решения СЛАУ и имеющий небольшое число конфигурационных параметров. Реализация алгоритма на языке C++ с использованием средств MPI и высокоуровневой библиотеки линейной алгебры Eigen [4] включена в библиотеку Krylov [5].

Тестирование алгоритма проводилось на большом числе методических и практических задач вычислительной физики. Было произведено исследование быстродействия решателя и числа итераций в зависимости от различных параметрах решателя. К варьируемым параметрам относились параметры алгебраической декомпозиции, такие как перекрытие подобластей и их количество, а к параметрам времени исполнения — число потоков в PARDISO при решении задач в подобластях, а также число MPI процессов на узел. Кроме того, была продемонстрирована точность получаемых решений для представленных задач.

*Работа выполнена при поддержке грантов РФФИ №11-01-00205 и Президиума РАН №2.5.

2. Алгоритмы решения СЛАУ

2.1. Метод алгебраической декомпозиции СЛАУ

Аппроксимация различных краевых задач соответствующих дифференциальных уравнений различными методами, такими как методы конечных разностей, конечных объёмов и конечных элементов, приводит к системе линейных алгебраических уравнений вида

$$Ax = b, \quad x, b \in \mathcal{R}^N, \quad A \in \mathcal{R}^{N,N}, \quad \det|A| \neq 0. \tag{1}$$

Решение СЛАУ на кластере требует предварительной её декомпозиции и распределения блоков матрицы, а также векторов неизвестных и правой части между узлами кластера. В работе предлагается использовать следующую модификацию метода одно-направленной алгебраической декомпозиции задачи [2], основными достоинствами которой являются простота, высокая скорость работы алгоритма и возможность получить распределённую блочно-трёхдиагональную матрицу на выходе.

Сначала ищется псевдо-периферийная вершина v графа G, ассоциированного с матрицей системы (1).

Алгоритм 1. Поиск псевдо-периферийной вершины v графа G. Запускается обход в ширину [6] с произвольной вершины графа матрицы и находятся наиболее удалённые от неё вершины и расстояние до них. Далее снова запускается обход в ширину, начиная уже с произвольной вершины среди множества наиболее удалённых вершин, полученных на предыдущем шаге. Обходы в ширину повторяются до тех пор, пока увеличивается максимальное расстояние от стартовой вершины до остальных. Как только расстояние перестанет увеличиваться, алгоритм останавливается и в качестве псевдо-периферийной вершины v возвращает вершину, с которой начинался последний обход в ширину.

Время работы алгоритма 1 равно $O(E \cdot K)$, где E — количество рёбер в графе матрицы. Для большинства конечных схем аппроксимации уравнений число рёбер пропорционально количеству вершин сетки в расчётной области. Этот факт следует из того, что соответствующие конечные базисы финитны и, как правило, используются сетки, у которых имеются ограничения на минимальные величины плоских и телесных внутренних углов. Число K в оценке равно количеству итераций, совершённых алгоритмом 1. Для произвольных графов в худшем случае это число может быть довольно велико, однако для графов практических задач алгоритму обычно требуется всего лишь несколько итераций [2]. За счёт этого на практике алгоритм поиска псевдо-периферийных вершин оказывается весьма эффективным. Более того, во многих случаях оказывается достаточно выбрать произвольную, например, первую вершину графа, и вообще не запускать алгоритм поиска псевдо-периферийных вершин.

После того, как выбрана некоторая вершина v, начиная с неё запускается обход графа G в ширину. Все вершины разделяются на множества — "фронты" — согласно расстоянию от них до вершины v. То есть, во фронт F_k ($k = 0 \dots d$, где d — псевдо-диаметр графа G) попадают все вершины, равноудалённые от вершины v на расстояние k.

Далее рассмотрим способ объединения фронтов в алгебраические подобласти Ω_p. Мы будем объединять только последовательные фронты в подобласти, так что

$$\Omega_p = \bigcup_{k=l_p}^{r_p} F_k,$$

где l_p и r_p — границы подобласти Ω_p. Пусть требуется построить P подобластей (например, по числу используемых узлов кластера) для системы (1) с N неизвестными. Желательно разбить неизвестные в подобласти приблизительно по N/P неизвестных. Однако поскольку мы объединяем в подобласти не отдельные вершины, а фронты, то точное разбиение неизвестных по N/P в каждой подобласти может не получиться. Тем не менее, мы можем при

помощи бинарного поиска минимизировать максимальный размер подобласти либо максимизировать минимальный размер.

Рассмотрим, например, максимизацию минимального размера, которая может использоваться для того, чтобы, по возможности, более равномерно загрузить все узлы кластера. Легко видеть, что функция зависимости максимально возможного числа подобластей при фиксированном минимально допустимом размере монотонна. Действительно, при фиксированном минимальном размере разделение вершин графа разбиение на подобласти можно произвести при помощи следующего жадного [6] алгоритма.

Алгоритм 2. Алгоритм построения максимального числа подобластей. К первой подобласти добавляем последовательно фронты, начиная с первого, пока размер подобласти не станет больше либо равным минимально допустимому размеру. В этом момент можно прекратить увеличение первой подобласти и приступить к построению второй подобласти. Этот процесс продолжается до тех пор, пока все фронты не окажутся в какой-либо подобласти. При этом может оказаться так, что в конце окажется несколько фронтов, которых недостаточно для формирования отдельной подобласти. Такие фронты можно отнести к последней построенной подобласти.

Ясно, что приведённый выше способ строит максимальное число подобластей. Действительно, если бы можно было построить больше подобластей при заданном минимальном размере подобласти, то тогда, начиная с некоторого i, подобласть Ω_i в "жадном" разбиении и подобласть Ω'_i отличались бы. Рассмотрим минимальное такое i. Если в подобласти Ω'_i больше фронтов, чем в Ω_i, то тогда оптимальное разбиение можно перестроить таким образом, чтобы в подобласти i было столько же фронтов, что и в оптимальном. Значит, в подобласти Ω'_i должно быть меньше фронтов, чем в Ω_i. Однако Ω_i строилось таким образом, что в него поместили минимально возможное число фронтов при фиксированном начальном. Получаем противоречие, и значит, построенное алгоритмом разбиение является максимальным.

Из приведённых выше рассуждений становится ясно, что с увеличением минимального размера число подобластей не может увеличиться. Значит, функция зависимости их числа от минимального размера действительно монотонна, и можно использовать двоичный поиск максимального минимального размера подобласти, чтобы получить P подобластей. Кроме того, описанный выше жадный алгоритм позволяет собственно построить эти подобласти уже после определения минимального размера.

Представленный алгоритм генерирует подобласти без пересечений. Однако получить разбиение с пересечениями довольно легко — достаточно просто расширить каждую из подобластей на несколько фронтов слева и справа. Также можно при начале генерации следующей подобласти отступить от границы предыдущей внутрь необходимое число фронтов.

Заметим, что рёбра в графе матрицы могут соединять вершины либо одного, либо двух соседних фронтов. Тогда получается, что граничными для подобласти p будут вершины, принадлежащие двум фронтам F_{l_p-1} и F_{r_p+1}. Можно считать, что эти фронты принадлежат соседним подобластям Ω_{p-1} и Ω_{p+1} соответственно. Получается, что если последовательно занумеровать переменные в каждой из подобластей, а в каждой из подобластей — последовательно в каждом из фронтов, мы получим следующую блочно-трёхдиагональную СЛАУ:

$$\bar{A}\bar{x} = \begin{bmatrix} D_1 & U_1 & & \\ L_1 & D_2 & \ddots & \\ & \ddots & \ddots & U_{P-1} \\ & & L_{P-1} & D_P \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_P \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_P \end{bmatrix} = \bar{f}. \tag{2}$$

Будем полагать, что диагональные блоки D_p, соответствующие СЛАУ в подобластях Ω_p, не вырождены. При этом если разбиение исходной СЛАУ выполнялось с пересечениями, то

в полученной СЛАУ (2) порядка N' окажется по несколько переменных, соответствующих одной и той же вершине графа, но лежащим в разных подобластях; в этом случае $N' > N$.

2.2. Крыловское ускорение аддитивного метода Шварца

После декомпозиции задачи на подобласти и пересылки блоков матрицы в соответствующие узлы кластера мы имеем распределённую блочную СЛАУ (2). Для решения таких СЛАУ широко используются методы Шварца. Одна итерация аддитивного метода Шварца в матричном виде может быть записана как (см. [2])

$$\bar{x}^{i+1} = \bar{x}^i + \sum_p R_p^T D_p^{-1} R_p (f - \bar{A}\bar{x}^i),$$

где $R_p : \mathcal{R}^{N'} \to \mathcal{R}^{N_p}$ — оператор сужения на подобласть Ω_p. Здесь N' — размерность СЛАУ (2), а N_p — количество неизвестных в подобласти Ω_p. С использованием блочного вида СЛАУ итерацию метода Шварца можно переписать следующим образом:

$$x_p^{i+1} = x_p^i + D_p^{-1}(f_p - L_p x_{p-1}^i - U_p x_{p+1}^i),$$

где для $p = 1$ и $p = P$ полагаем, что $L_p x_{p-1}^i = 0$ и $U_p x_{p+1}^i = 0$ соответственно. В общем случае члены $L_p x_{p-1}^i$ и $U_p x_{p+1}^i$ являются аналогами условий Дирихле для краевых задач.

Заметим, что на самом деле итерировать достаточно только переменные, образующие внешние границы подобластей, поскольку, зная граничные значения, легко определить значения внутри подобластей, решив соответствующую подзадачу. Для этого в произведениях $L_p x_{p-1}^i$ и $U_p x_{p+1}^i$ достаточно использовать только части векторов из подобластей Ω_{p-1} и Ω_{p+1}, соответствующие фронтам $l_p - 1$ и $r_p + 1$. Эти переменные и будут образовывать внешнюю границу области Ω_p.

Обозначим за u^b вектор граничных переменных для всех подобластей, и введём оператор сужения на граничные переменные $\hat{R} : \mathcal{R}^{N'} \to \mathcal{R}^{N_b}$, где N_b — размерность пространства следов. Тогда итерация метода Шварца в подпространстве следов выглядит следующим образом:

$$u_{i+1}^b = u_i^b + \hat{R} \sum_p R_p^T D_p^{-1} R_p (f - \bar{A}\hat{R}^T u_i^b).$$

Эту итерацию можно представить в виде $u_{i+1}^b = S(u_i^b) = T u_i^b + g$, где

$$T = I - \hat{R} \sum_p R_p^T D_p^{-1} R_p \bar{A} \hat{R}^T, \quad g = \hat{R} \sum_p R_p^T D_p^{-1} R_p f.$$

Сходимость стационарных итераций имеется при спектральном радиусе $\rho(T) < 1$. В то время как для эллиптических уравнений это условие выполнено, для произвольных уравнений это не так. Однако можно заметить, что решение u_*^b должно удовлетворять $u_*^b = T u_*^b + g$, что равносильно системе

$$[I - T]\, u^b = g. \tag{3}$$

Данную систему можно решать итерационными методами в подпространствах Крылова, например обобщённым методом минимальных невязок (GMRES) [2]. Для этого достаточно, чтобы матрица $[I - T] = \hat{R} \sum R_p^T D_p^{-1} R_p \bar{A} \hat{R}^T$ была невырожденной, что является существенно более слабым условием, по сравнению с условием $\rho(T) < 1$. Заметим также, что матрица $[I - T]$ не является симметричной, поэтому нельзя использовать экономичные методы в подпространствах Крылова, такие как метод сопряженных градиентов, ориентированные на решение симметричных СЛАУ и использующие короткие рекурсии для построения базиса подпространств Крылова.

Метод GMRES рассчитан на решение произвольных невырожденных СЛАУ вида $Ax = b$, при этом он минимизирует невязку $r_j = b - Ax_j$ в подпространстве Крылова

$$K_m(A, r_0) = \mathrm{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}.$$

Псевдокод алгоритма следующий:

- $r_0 \leftarrow b - Ax_0$, $\beta = ||r_0||$, $q_0 \leftarrow r_0/||r_0||$, $\xi \leftarrow (1, 0, \ldots, 0)^T$
- $j \leftarrow 0, 1, \ldots$ while $||r_j|| > \varepsilon\beta$
  - $\tilde{q}_{j+1} \leftarrow Aq_j$
  - For $k \in [0, \ldots, j]$
    - $H_{k,j} \leftarrow (\tilde{q}_{j+1}, q_k)$
    - $\tilde{q}_{j+1} \leftarrow \tilde{q}_{j+1} - H_{k,j}q_k$
  - $H_{j+1,j} \leftarrow ||\tilde{q}_{j+1}||$, $q_{j+1} \leftarrow \tilde{q}_{j+1}/H_{j+1,j}$
  - For $k \in [0, \ldots, j-1]$
    - $$\begin{bmatrix} H_{k,j} \\ H_{k+1,j} \end{bmatrix} \leftarrow \begin{bmatrix} c_k & s_k \\ -\bar{s}_k & c_k \end{bmatrix} \begin{bmatrix} H_{k,j} \\ H_{k+1,j} \end{bmatrix}$$
  - $c_j \leftarrow |H_{j,j}|/\sqrt{|H_{j,j}|^2 + |H_{j+1,j}|^2}$
  - $\bar{s}_j \leftarrow c_j H_{j+1,j}/H_{j,j}$
  - $$\begin{bmatrix} \xi_j \\ \xi_{j+1} \end{bmatrix} \leftarrow \begin{bmatrix} c_j & s_j \\ -\bar{s}_j & c_j \end{bmatrix} \begin{bmatrix} \xi_j \\ \xi_{j+1} \end{bmatrix}$$
  - $H_{j,j} \leftarrow c_j H_{j,j} + s_j H_{j+1,j}$, $H_{j+1,j} \leftarrow 0$
  - $||r_{j+1}|| \leftarrow \beta|\xi_{j+1}|$
- $y_j \leftarrow \beta H^{-1}\xi$
- $x_j \leftarrow x_0 + [q_0 \ldots q_{j-1}]y_j$.

Из псевдокода видно, что для алгоритма требуется только операция умножения матрицы системы на вектор. Запишем результат умножения $I - T$ на вектор v через результат одной итерации метода аддитивного Шварца $S(v)$:

$$[I - T]\, v = v - Tv = v - S(v) + g, \quad g = S(0).$$

Таким образом, для решения СЛАУ (3) методом GMRES достаточно реализовать обычный метод аддитивного Шварца. Для решения задач в подобластях $x_p = D_p^{-1}b_p$ можно использовать какой-нибудь прямой решатель для разреженных систем. При этом LU-разложение матриц D_p достаточно вычислить всего один раз.

Оценим эффективность данного алгоритма для простого случая — решения трёхмерного уравнения Пуассона в параллелепипеде со структурированной сеткой. Пусть имеется сетка с n вершинами по каждой из координатных осей. Тогда матрица системы будет иметь порядок $N = n^3$. При решении такой системы прямыми методами требуется $O(N^{5/3})$ памяти и $O(N^{7/3})$ операций [7], поскольку ширина ленты матрицы k в этом случае оказывается пропорциональной n^2.

Предположим теперь, что декомпозиция задачи на P подобластей выполнена равномерно с небольшими перехлёстами, то есть порядок СЛАУ в каждой подобласти есть $O(N/P)$. Размерность пространства следов будем считать равной $O(P \cdot n^2) = O(P \cdot N^{2/3})$. Пусть для решения СЛАУ (3) требуется K итераций. Считая, что в подобластях сохраняются оценки для прямых решателей, получим, что общие затраты памяти M будут равны

$$M = O(P \cdot (N/P)^{5/3} + K \cdot P \cdot N^{2/3}) = O(N^{5/3}/P^{2/3} + K \cdot P \cdot N^{2/3}),$$

или, в пересчёте на один узел (при условии использования распределённого GMRES),

$$M_p = O((N/P)^{5/3} + K \cdot N^{2/3}).$$

Время работы решателя T будет складываться из максимального времени факторизации t_{fact}, времени работы итерационного метода t_{iter} и времени на коммуникации t_{comm} ($T = t_{fact} + t_{iter} + t_{comm}$). Для них можно получить следующие оценки:

$$\begin{aligned} t_{fact} &= O((N/P)^{7/3}), \\ t_{iter} &= O(K \cdot (N/P)^{5/3} + K^2 \cdot N^{2/3}), \\ t_{comm} &= O(K \cdot P \cdot N^{2/3}). \end{aligned}$$

Данные оценки справедливы для распределённого GMRES. При использовании GMRES на одном узле оценка для t_{iter} увеличиваются до

$$t_{iter} = O(K \cdot (N/P)^{5/3} + K^2 \cdot P \cdot N^{2/3}).$$

Тогда имеем

$$T = O((N/P)^{7/3} + K \cdot (N/P)^{5/3} + K^2 \cdot P \cdot N^{2/3}).$$

Поскольку, как правило, K растёт с ростом P, получаем, что при фиксированном N для T имеется оптимум при некотором значении P. Этот оптимум достигается, с одной стороны, за счёт уменьшения времени факторизации блоков, и, с другой стороны, за счёт увеличения временны́х затрат на ортогонализацию векторов и межпроцессные коммуникации.

3. Технологии реализация решателя

При выборе языка программирования, сторонних библиотек и технологий для реализации решателя учитывалась специфика используемых в решателе алгоритмов. Необходим был высокоуровневый язык и набор библиотек, хорошо подходящий для реализации различных алгоритмов на графах, манипуляций с векторами и разреженными матрицами. Дополнительное требование заключалось в том, чтобы компиляторы языка генерировали высоко-эффективный код. В итоге выбор был сделан в пользу языка C++ и использования стандартной библиотеки STL, из которой использовались различные контейнеры, а также библиотеки линейной алгебры Eigen [4]. Последняя имеет в своём составе классы разреженных матриц и динамических векторов, предоставляет удобный API для манипуляции ими, а также имеет встроенную векторизацию алгоритмов с использованием инструкций SSE.

Для решения задач в подобластях требуется эффективный решатель для СЛАУ с разреженными матрицами невысокого порядка, причём необходимо решать СЛАУ с различными правыми частями. Кроме того, обусловленность диагональных матриц D_p может быть довольно плохой. Для решения таких систем хорошо подходят прямые решатели разреженных систем. В качестве такого решателя был выбран PARDISO (PARallel DIrect SOlver) из библиотеки Intel ® MKL [3]. Данный решатель является многопоточным, что позволяет реализовать гибридный подход и повысить производительность: на каждом используемом узле кластера можно решать задачи для одной подобласти, причём факторизацию СЛАУ, выполняющуюся при инициализации решателя, и решение серии систем с разными правыми частями можно выполнять в многопоточном режиме с использованием всех доступных процессорных ядер.

Для организации взаимодействия процессов на узлах кластера был использован интерфейс MPI, который широко используется при написании распределённых программ. При этом решатель был построен на основе master-slave архитектуры, с выделенным основным

узлом (с рангом MPI, равным 0). На этом узле решателю передаётся матрица, выполняется декомпозиция на подобласти, а также работает решатель GMRES. Остальные MPI процессы работают как ведомые, принимая от мастер-процесса матрицу системы на этапе инициализации и граничные переменные в процессе работы для решения задач в подобластях. Граничные для соседних подобластей переменные после решения отсылаются назад. Использование такого подхода позволяет на этапе итерационного решения ограничиться вызовами трёх функций MPI на одну итерацию: MPI_scatter и MPI_gather для рассылки и сбора граничных переменных и MPI_Bcast для управления ведомыми процессами.

Представленный решатель был включён в состав библиотеки Krylov [5]. Интерфейс решателя, позволяющий вызывать его из процедур, написанных на различных языках программирования, представлен на рис. 1.

```
void cschwarz_solver(MPI_Comm *pcomm, const int *n,
            const double *a, const int *ia, const int *ja,
            double *u, const double *f,
            const double *eps, const int *ovl,
            const int *nmax, const int *nres, int *niter,
            double *time, int *err);
void cschwarz_client(MPI_Comm *pcomm);
```

Рис. 1. Интерфейс решателя

Параметры решателя следующие:

pcomm — MPI коммуникатор, процессы которого участвуют в решении СЛАУ; процедура cschwarz_solver(...) должна вызываться в процессе с рангом 0, в остальных процессах необходимо вызывать cschwarz_client(...);

n — порядок СЛАУ;

a, ia, ja — матрица системы в формате CSR (Compressed Sparse Row format) [3];

u — вектор размера **n**, в который записывается полученное решение системы;

f — правая часть СЛАУ, должна иметь размер **n**;

eps — критерий останова итераций ε (**eps**) для решателя GMRES, решение считается найденным, если

$$||g - [I - T]u^b|| < \varepsilon||g||;$$

ovl — параметр пересечения подобластей; после построения разбиения на подобласти без пересечений каждая подобласть расширяется на **ovl** фронтов влево и вправо, соответственно пересечение соседних подобластей состоит из $2 \cdot$ **ovl** фронтов;

nmax — максимальное число итераций решателя GMRES;

nres — число итераций решателя GMRES до рестарта, позволяет ограничить длину последовательности q_j базисных векторов подпространства Крылова; после выполнения каждых **nres** итераций, даже если не была достигнута необходимая точность решения, текущее приближение пересчитывается по формулам

$$y_j = \beta H^{-1}\xi, \quad u_j^b \leftarrow u_0^b + [q_0 \dots q_{j-1}]y_j,$$

и новое значение u_j^b используется в качестве начального приближения для решателя;

niter — выходной параметр, количество совершённых решателем итераций;

time — время, затраченное на решение системы;

err — код возврата решателя; ненулевое значение свидетельствует об ошибке.

4. Численные эксперименты

В последующих таблицах приняты следующие обозначения: N — порядок системы, N_{nz} — количество ненулевых элементов в исходной матрице, N_b — размерность пространства следов, t_{fac} — время (в секундах) факторизации матриц D_p, t_{tot} — общее время решения и n — количество итераций. Эксперименты проводились на вычислительных узлах HP BL2x220c G6 кластера НКС-30Т [8], каждый из которых оборудован двумя четырёхъядерными процессорами Intel Xeon E5540 и 16-ю гигабайтами памяти. В целях увеличения объёма памяти, доступной PARDISO для хранения факторов матриц D_p, запускалось по одному MPI-процессу на узел, а для ускорения факторизации были задействованы все имеющиеся на узлах ядра. Параметр пересечения подобластей при решении всех задач был установлен таким образом, чтобы подобласти пересекались максимум по 8 фронтам, за исключением второй модельной задачи, где пересечение составляет 4 фронта. Параметр ε в критерии останова итераций был равен 10^{-7}. Рестарты итерационного процесса не производились, то есть значение параметра **nres** было выбрано достаточно большим.

4.1. Модельная задача для диффузионно-конвективного уравнения

В качестве первой модельной задачи была выбрана задача Дирихле для линейного диффузионно-конвективного уравнения

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + p\frac{\partial u}{\partial x} + q\frac{\partial u}{\partial y} + r\frac{\partial u}{\partial z} = f(x,y,z), \quad (x,y,z) \in \Omega,$$

$$u|_\Gamma = g(x,y,z),$$

в единичном кубе $\Omega = [0,1]^3$ на равномерной кубической сетке. Дискретизация задачи проводилась с помощью семиточечной схемы экспоненциального типа [7]. Функции f и g выбирались соответственно точному решению $u(x,y,z) = x^2 + y^2 + z^2$, а в качестве начального приближения была взята функция $u^0(x,y,z) = 0$. В таблицах 1–3 приведены результаты решения данной модельной задачи при различных значениях p, q и r.

Таблица 1. Решение первой модельной задачи при $p = q = r = 0$

Сетка	N	N_{nz}		Количество узлов			
				2	4	8	16
64^3	262144	1810432	N_b	6104	16196	34688	71786
			n	12	16	21	32
			t_{fac}	3.31	0.94	0.46	0.30
			t_{tot}	4.65	1.74	1.18	1.26
128^3	2097152	14581760	N_b	24536	65508	139183	285485
			n	17	23	29	40
			t_{fac}	219	33.3	8.36	2.56
			t_{tot}	245	44.5	15.6	9.25

Максимальная погрешность решения первой модельной задачи в равномерной норме составила $7.7 \cdot 10^{-7}$.

Таблица 2. Решение первой модельной задачи при $p = q = r = 16$

Сетка	N	N_{nz}		Количество узлов			
				2	4	8	16
64^3	262144	1810432	N_b	6104	16196	34688	71786
			n	10	11	14	24
			t_{fac}	3.29	0.95	0.50	0.30
			t_{tot}	4.44	1.67	1.11	1.08
128^3	2097152	14581760	N_b	24536	65508	139183	285485
			n	14	16	20	29
			t_{fac}	233	34.7	8.73	2.60
			t_{tot}	255	43.9	14.8	8.23

Таблица 3. Решение первой модельной задачи при $p = q = r = -16$

Сетка	N	N_{nz}		Количество узлов			
				2	4	8	16
64^3	262144	1810432	N_b	6104	16196	34688	71786
			n	10	11	15	25
			t_{fac}	3.33	1.00	0.47	0.44
			t_{tot}	4.48	1.97	1.14	1.33
128^3	2097152	14581760	N_b	24536	65508	139183	285485
			n	14	17	21	30
			t_{fac}	226	33.9	8.67	3.01
			t_{tot}	248	43.5	15.2	8.77

4.2. Модельная задача с волноводом

Вторым примером является решение трёхмерной краевой задачи для уравнения Гельмгольца

$$\nabla \times \left(\frac{1}{\mu_r} \nabla \times \vec{E} \right) - k_0^2 \dot{\varepsilon}_r \vec{E} = 0, \quad k_0 = \omega \sqrt{\varepsilon_0 \mu_0},$$

дискретизация которой выполнялась с помощью конечно-элементных аппроксимаций с базисными функциями Неделека второго порядка [9, 10]. Расчётной областью является волновод с линейными размерами $a = 72$, $b = 34$, $c = 200$ мм. Параметры задачи: $\mu_r = 1$, $\dot{\varepsilon}_r = 1 - 0.1i$, частота $\omega = 6\pi \cdot 10^9$ Гц. Граничные условия: на грани $z = 200$ мм задавалась

касательная компонента поля

$$\vec{E}_0 \times \vec{n} = \vec{e}_y \sin(\pi x/a) \times \vec{n},$$

а на остальных гранях — условие металлической стенки ($\vec{E}_0 = 0$). Аналитическое решение в данном случае имеет вид

$$\vec{E} = \vec{e}_y \sin\left(\frac{\pi x}{a}\right) \frac{\sin \gamma z}{\sin \gamma c}.$$

В табл. 4 приводятся результаты решения указанной задачи при различных размерностях сетки.

Таблица 4. Решение второй модельной задачи

Сетка	N	N_{nz}		Количество узлов			
				2	4	8	16
$8 \times 4 \times 20$	21664	423392	N_b	1540	4678	10708	23052
			n	9	13	21	258
			t_{fac}	0.19	0.17	0.15	0.36
			t_{tot}	0.51	0.47	0.67	6.19
$15 \times 7 \times 40$	149874	3117779	N_b	5210	16646	37670	78446
			n	13	18	25	38
			t_{fac}	2.56	1.26	0.63	0.40
			t_{tot}	5.88	4.00	3.12	3.15
$29 \times 14 \times 80$	1196026	25795767	N_b	20562	67084	151032	318108
			n	18	26	34	49
			t_{fac}	108	39.9	14.9	5.29
			t_{tot}	173	85.6	43.3	32.94

Наличие точного решения позволяет проверить сходимость итерационного процесса. Данные тесты продемонстрировали ошибку порядка $O(h^2)$, что согласуется с теорией.

4.3. Примеры решения практических задач

В заключение данного пункта мы приведём результаты решения трёх СЛАУ (обозначаемых ниже в табл. 5 как Г1, Г2, Г3), возникающих из актуальных практических краевых задач, описываемых трёхмерными уравнениями гидродинамики, которые аппроксимируются с помощью неявных конечно-объёмных схем на сложных неструктурированных сетках.

Следует отметить, что приведенные результаты параллельных вычислений дают значительный выигрыш в сравнении с однопроцессорными расчётами, в которых некоторые из рассмотренного типа СЛАУ с помощью “стандартных” предобусловленных крыловских алгоритмов решаются часами.

Таблица 5. Решение практических задач гидродинамики

Задача	N	N_{nz}		Количество узлов			
				2	4	8	16
Г1	4875172	26455278	N_b	180487	439384	895041	1831917
			n	40	58	85	162
			t_{fac}	101	64	49.8	19.7
			t_{tot}	166	117	119	201
Г2	1726272	11984539	N_b	6912	18533	42476	90000
			n	52	109	207	334
			t_{fac}	54.4	26.0	9.55	3.26
			t_{tot}	79.0	116	83.0	64.0
Г3	475000	13672497	N_b	2500	7500	17500	37110
			n	24	25	36	53
			t_{fac}	6.12	2.62	1.08	0.53
			t_{tot}	10.1	5.45	3.31	2.88

5. Заключение

Рассмотренные результаты позволяют сделать следующие выводы.

- С увеличением количества подобластей число итераций увеличивается, но (за одним исключением) не более чем линейно. В этом плане улучшение сходимости можно ожидать за счёт включения других типов граничных условий на смежных границах, вместо используемого условия Дирихле.
- Применение прямого решателя PARDISO для подобластей является достаточно экономичным. В частности, с увеличением числа MPI-процессов относительные временные расходы на факторизацию вспомогательных подсистем значительно уменьшаются.
- Размерности СЛАУ в подпространствах следов (N_b) значительно меньше исходных порядков N. Это позволяет экономично реализовать с помощью GMRES итерации по подобластям.
- Описанная программная реализация позволяет проводить оптимизацию алгоритмов за счёт выбора различных счётных параметров (величина перехлёстов, варьирования числа ядер на подобласти, количества MPI-процессов на вычислительный узел и т.д.).
- При увеличении числа подобластей и используемых процессоров до нескольких тысяч целесообразен, по-видимому, переход от используемой 1-D декомпозиции к 2-D или даже 3-D декомпозиции.

Литература

1. Karypis G., Kumar V. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, N. 1, P. 359–392.
2. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, 2003.
3. Intel (R) Math Kernel Library Reference Manual:
   URL: http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/mklxe/mkl_manual_win_mac/index.htm
4. Eigen: URL: http://eigen.tuxfamily.org/index.php?title=Main_Page
5. Бутюгин Д.С., Ильин В.П., Ицкович Е.А и др. Krylov: библиотека алгоритмов и программ для решения СЛАУ // Современные проблемы математического моделирования. Математическое моделирование, численные методы и комплексы программ. Сборник трудов Всероссийских научных молодёжных школ.
   Ростов-на-Дону: Изд-во Южного федерального университета, 2009. С. 110–128.
6. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М., МЦНМО: БИНОМ. Лаборатория знаний, 2004. 960 с.
7. Ильин В.П. Методы конечных разностей и конечных объёмов для эллиптических уравнений. Новосибирск: Изд-во ИМ СО РАН, 2001. 318 с.
8. Кластер НКС-30Т. URL: http://www2.sscc.ru/HKC-30T/HKC-30T.htm
9. Monk P. Finite Element Methods for Maxwell's Equations. Oxford University Press, 2003.
10. Ingelstrom P. A new set of H(curl)-conforming hierarchical basis functions for tetrahedral meshes // IEEE Transactions on Microwave Theory and Techniques. January 2006. Vol. 54. N. 1. P. 160–114.

Моделирование астрофизических струйных выбросов на гибридных вычислительных системах с общей памятью*

М.П. Галанин[1], В.В. Лукин[1], В.М. Чечеткин[1]

ИПМ им. М.В. Келдыша РАН[1]

В работе рассмотрена модель образования, коллимации и радиационного ускорения плазменного струйного выброса в окрестности компактного объекта. Модель включает в себя систему уравнений радиационной магнитной гидродинамики в двумерном цилиндрически симметричном приближении. Разработан параллельный программный комплекс для систем с общей памятью, использующий технологии программирования OpenMP и nVidia CUDA, реализующий разностную схему на треугольных неструктурированных сетках для решения уравнений МГД и метод дискретных направлений решения уравнения переноса излучения. Показано, что ускоряемое радиационным давлением вещество достигает высоких скоростей, соответствующих наблюдаемым в астрофизических системах.

1. Введение

Одним из наиболее интересных классов астрофизических процессов является образование струйных выбросов, джетов (от англ. jet — струя), формирующихся в ядрах активных галактик (например, эллиптической галактике M87), микроквазарах (например, двойной звездной системе SS433 [2]), протозвездных объектах и ряде других систем. Тонкие (поперечный размер обычно не превышает нескольких парсек) биполярные струи вещества около таких объектов тянутся на сотни и тысячи световых лет, заканчиваясь гигантскими облаками газа.

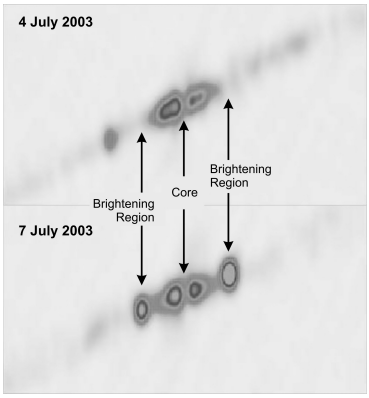


Рис. 1. Результаты наблюдений объекта SS433 на телескопе VLBA [2]: смена максимумов и минимумов яркости в указанных точках

На настоящий момент известно достаточно большое количество джетов, большинство из них наблюдается только в радиодиапазоне. Размер джета, истекающего из ядра галактики M87, достигает 5000 световых лет. Струя состоит из быстро движущихся заряженных частиц, сконцентрированных в узлы размером до 10 световых лет (см. рис. 1), и имеет вид конуса с углом раствора около 6°. Скорость течения вещества в джете галактики M87 достигает $0.8c$, где c — скорость света, скорость вещества в джете SS433 — около $0.26c$.

*Работа выполнена при финансовой поддержке РФФИ (проекты № 12-01-00109-а, 12-02-00687-а).

В данной работе мы строим и исследуем математическую модель ускорения вещества джета в канале над горячим гравитирующим объектом под действием давления излучения тонкого диска, окружающего компактный объект. Модель включает систему уравнений идеальной радиационной магнитной гидродинамики (МГД) в двумерном осесимметричном приближении и основана на МГД-модели образования ускоряющего канала [5].

Моделирование излучения с помощью уравнения переноса с учетом интеграла рассеяния представляет вычислительно тяжелую задачу. В двумерном случае для адекватного расчета поля излучения применены параллельные алгоритмы для высокопроизводительных вычислительных систем с общей памятью, включающих графические ускорители.

2. Постановка задачи

Предлагается использовать различные эффекты для объяснения различных наблюдаемых свойств джетов. Так, для объяснения высокой степени коллимации джета принято использовать магнитогидродинамические (МГД) модели, в которых коллимация потока происходит под действием осевой и частично азимутальной компонент магнитного поля [4, 5].

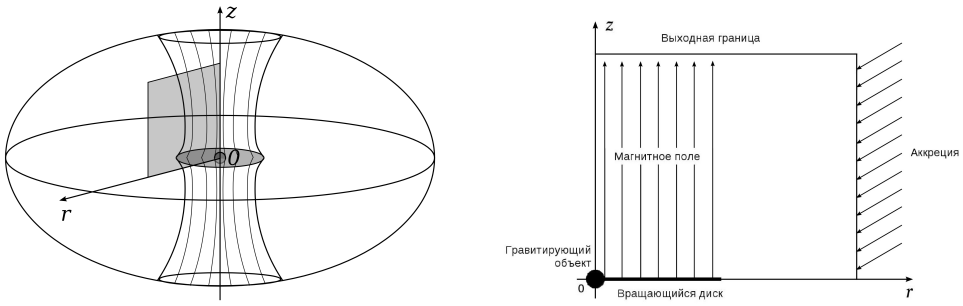


Рис. 2. Схема модели системы, порождающей джет и расчетная область

В работах [5] показано, что над гравитирующим объектом, окруженным тонким вращающимся диском и погруженным в облако аккрецирующей плазмы (см. рис. 2), образуется замагниченный канал, внутри которого развивается струйный выброс плазмы, источником которой является тонкий диск (см. рис. 3). Полученный канализированный выброс оказывается устойчивым во времени, канал имеет плотные, оптически толстые стенки, в то время как вещество внутри джета сильно разрежено, его оптическая толща (для случая томпсоновского рассеяния) составляет

$$\tau = \sigma_T n L_0 \approx 6.7 \cdot 10^{-4}, \tag{1}$$

где L_0 — пространственный масштаб задачи, n — концентрация вещества в канале, σ_T — томпсоновское сечение рассеяния.

В работе [3] рассмотрена нульмерная динамическая модель ускорения твердотельного сгустка в пустотелом канале давлением излучения горячего „дна“ канала, моделирующего окрестности компактного объекта. Показано, что в подобной модели сгусток достигает субсветовых скоростей, вплоть до $0.9c$.

Мы будем использовать схему модели, изображенную на рис. 2, и результаты работы [5] в качестве начальных условий. Будем рассматривать квазистационарную моноэнергетическую модель распространения излучения и модель магнитной гидродинамики с идеальной проводимостью плазмы в двумерном цилиндрически симметричном приближении. Пусть вещество системы представляет собой невязкий совершенный идеально ионизованный газ, электрическое сопротивление среды отсутствует. Магнитное поле вморожено в тонкий идеально проводящий диск и вращается вместе с ним, приобретая коллимирующую осевую

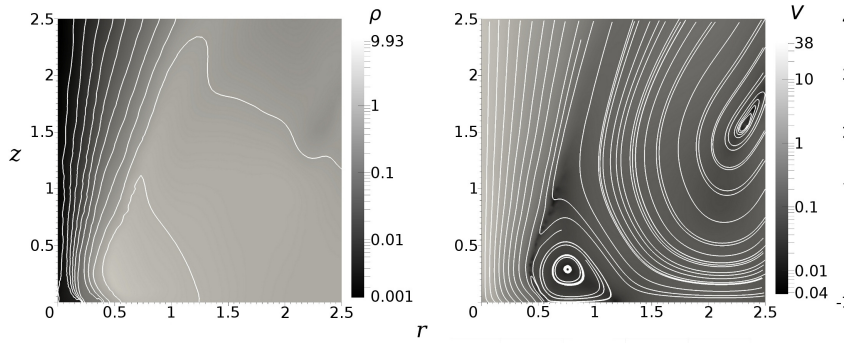


Рис. 3. Ускоряющий канал в модели [5]: распределения плотности и скорости в момент времени $t = 15$ (установившийся режим выброса)

и ускоряющую азимутальную компоненту. Гравитационное поле определяется гравитацией центрального тела (звезды), самогравитация газа не учитывается. Фотоны испытывают однократное рассеяние на электронах (томпсоновское рассеяние), источником излучения в модели служит только тонкий диск, излучение сфокусировано в ускоряющий канал.

3. Математическая модель

3.1. Система уравнений

Полная система уравнений радиационной магнитной гидродинамки в квазистационарном моноэнергетическом приближении имеет вид [7]:

$$\frac{\partial \rho}{\partial t} + \nabla \rho \mathbf{v} = 0, \tag{2}$$

$$\frac{\partial}{\partial t}\left(\rho \mathbf{v} + \mathbf{G}\right) + \nabla \cdot \left(\hat{\mathbf{\Pi}} + \hat{\mathbf{T}}\right) = \frac{1}{4\pi}\left(\nabla \times \mathbf{B}\right) \times \mathbf{B} + \mathbf{F}_g, \tag{3}$$

$$\frac{\partial}{\partial t}\left(e + U\right) + \nabla \cdot \left(\mathbf{v}\left(e + p\right) + \mathbf{W}\right) = \frac{1}{4\pi}\left(\left(\nabla \times \mathbf{B}\right) \times \mathbf{B}\right) \cdot \mathbf{v} + \mathbf{F}_g \cdot \mathbf{v}, \tag{4}$$

$$\boldsymbol{\omega} \cdot \nabla I(t, \mathbf{x}, \boldsymbol{\omega}) + k(t, \mathbf{x}) I(t, \mathbf{x}, \boldsymbol{\omega}) = \beta(t, \mathbf{x}) \int\limits_{\Omega} \Gamma(t, \mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}') I(t, \mathbf{x}, \boldsymbol{\omega}')\, d\boldsymbol{\omega}', \tag{5}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times \left(\mathbf{v} \times \mathbf{B}\right), \tag{6}$$

где ρ — плотность плазмы, $\mathbf{v}$ — вектор скорости вещества, $\hat{\mathbf{\Pi}}$ — тензор плотности потока импульса вещества, $\Pi_{ij} = p\delta_{ij} + \rho v_i v_j$, δ_{ij} — символ Кронекера, p — газовое давление, e — плотность энергии вещества, $\mathbf{B}$ — вектор магнитной индукции, $\mathbf{F}_g$ — объемная плотность гравитационной силы, I — интенсивность излучения, $\Gamma(t, \mathbf{x}, \boldsymbol{\omega} \cdot \boldsymbol{\omega}')$ — индикатриса рассеяния, $k(t, \mathbf{x})$ — коэффициент ослабления, $k = \alpha + \beta$, $\alpha(t, \mathbf{x})$ — коэффициент поглощения, $\beta(t, \mathbf{x})$ — коэффициент рассеяния излучения в веществе, $\mathbf{W} = \int\limits_{\Omega} \boldsymbol{\omega} I\, d\boldsymbol{\omega}$ — поток энергии излучения, $\mathbf{G} = \mathbf{W}/c^2$ — плотность имульса излучения, $U = \frac{1}{c}\int\limits_{\Omega} I\, d\boldsymbol{\omega}$ — плотность энергии излучения, $T_{ik} = \frac{1}{c}\int\limits_{\Omega} \omega_i \omega_k I\, d\boldsymbol{\omega}$ — тензор плотности потока импульса излучения. Считается, что газ совершенный и имеет уравнение состояния $p = \rho\varepsilon(\gamma - 1)$, где ε — удельная внутренняя

энергия вещества, откуда

$$e = \frac{\rho|\mathbf{v}|^2}{2} + \frac{p}{\gamma - 1}.$$

Основную трудность при исследовании этой модели представляет уравнение (5) — уравнение переноса излучения (УПИ). Это связано с принципиальной многомерностью уравнения: вообще говоря, интенсивность I зависит от трех пространственных координат (радиус-вектор $\mathbf{x}$), двух угловых координат (единичный вектор ω можно задать с помощью двух координат $\eta = \cos\theta$ и φ, где θ и φ — соответственно полярный и азимутальный углы) и времени t. Последняя зависимость является в квазистационарной модели переноса излучения неявной и выражается только в зависимости радиационных коэффициентов вещества от термодинамических параметров (плотность, температура) плазмы.

Отметим, что влияние поля излучения на движение вещества описывается в уравнениях (3) и (4) с помощью градиентов различных моментов интенсивности излучения. Данное обстоятельство при численном решении системы (2)–(6) приводит к необходимости получать насколько возможно гладкие распределения интенсивности, минимизируя эффекты численной немонотонности, либо получать сразу указанные градиенты моментов. Подобное обстоятельство становится критическим, например, для методов класса Монте-Карло, поскольку эти методы обладают высокой статистической дисперсией.

В качестве индикатрисы рассеяния будем использовать рэлеевскую индикатрису

$$\Gamma\left(t, \mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}'\right) = \frac{3}{4}\left(1 + \left(\boldsymbol{\omega} \cdot \boldsymbol{\omega}'\right)^2\right),$$

а сечение рассеяния равно $\sigma_T = 6.652 \times 10^{-29}\text{см}^2$ (коэффициент рассеяния, соответственно, равен $\beta = n\sigma_T$, где n — концентрация вещества).

Мы предполагаем, что в начале координат находится тело массы M, являющееся источником гравитационного поля. Чтобы избежать сингулярности в начале координат, будем задавать гравитационную силу следующим образом:

$$F_z = -\mathfrak{G}\frac{M\rho}{R^2}\frac{z}{R}, \quad F_r = -\mathfrak{G}\frac{M\rho}{R^2}\frac{r}{R}, \quad R \geqslant r_*,$$
$$F_z = -\mathfrak{G}\frac{M\rho}{r_*^3}z, \quad F_r = -\mathfrak{G}\frac{M\rho}{r_*^3}r, \quad R \leqslant r_*,$$

где $\mathfrak{G}$ — гравитационная постоянная, $R = \sqrt{z^2 + r^2}$.

Система уравнений стандартным образом приводится к безразмерному виду, в качестве основных масштабов выбираются масштабы расстояния L_0, плотности ρ_0 и времени t_0. Безразмерный параметр $g = \mathfrak{G}Mt_0^2/L_0^3 = 0.5$.

3.2. Граничные условия

Итак, предполагаем, что в пространстве, заполненном идеально проводящей плазмой, имеются диск и гравитирующее тело в центре. Расчетная область $[0, r_M] \times [0, z_M]$ представлена на рис. 2.

- На внешней цилиндрической границе области будем задавать условие сверхзвукового сферического втекания незамагниченной незакрученной межзвездной плазмы.
- Верхняя граница расчетной области моделирует переход потока к режиму течения на бесконечности [8].
- На оси вращения системы $r = 0$ ставится условие ограниченности решения.
- Нижняя граница разбивается на две части. На части границы $z = 0$, $r_d < r < r_M$, где r_d — радиус тонкого диска, ставятся условия, соответствующие экваториальной

симметрии, а на части $z = 0$, $0 < r < r_d$ моделируется тонкий диск. Диск вращается со скоростью $\omega(r) = \omega \cdot (1 - (r/r_d)^2)$. В вещество диска вморожено магнитное поле, имеющее только осевую компоненту $B_{z0}(r)$. Диск считается идеально проводящим. Он является источником вещества джета — вещество поступает в расчетную область со скоростями, определяемыми параметрами течения над диском.

Кроме того, необходимо поставить граничные условия для поля излучения, смоделировав источник в окрестности центрального гравитирующего объекта. Будем предполагать, что эти окрестности излучают с интенсивностью абсолютно черного тела, имеющего характерную температуру $7 \cdot 10^4$ K, что отвечает наблюдательным данным об объекте SS433 [1]. При этом излучение сфокусировано внутрь ускоряющего канала: излучающей является граница $z = 0$, $0 \leqslant r \leqslant 0.2$, излучение распространяется вдоль направлений, для которых $\cos\theta > 0.9$, где θ — полярный угол луча.

4. Численный метод и программная реализация

4.1. Конечно-разностный алгоритм

Для решения системы уравнений идеальной радиационной магнитной гидродинамики в двумерной цилиндрически симметричной постановке использованы треугольные неструктурированные сетки и разностный метод, основанный на расщеплении по физическим процессам. Пересчет неизвестных величин в разностных ячейках на каждом временном шаге состоит из четырех этапов.

1. Решение газодинамической системы уравнений методом типа Годунова (HLLC), а также учет геометрии задачи и действия гравитационного поля в виде слагаемых в правой части [9].

2. Аппроксимация уравнения Фарадея на разностной ячейке в цилиндрической системе координат, согласованная с геометрией задачи.

3. Интегрирование УПИ методом дискретных направлений (МДН) с дискретизацией сферы направлений „от границы“ [6].

4. Восполнение газовых переменных за счет действия электродинамических и радиационных сил.

Реализация данного алгоритма на треугольных неструктурированных сетках в целом изложена в [6].

4.2. Особенности реализации МДН в случае цилиндрической симметрии

Как уже отмечалось, с вычислительной точки зрения наиболее требовательной к ресурсам является процедура решения уравнения переноса излучения (5). Для решения этого уравнения выбран метод дискретных направлений. В соответствии с данным методом необходимо провести интегрирование УПИ вдоль каждого из выбранных дискретных направлений, сведя тем самым многомерное интегродифференциальное уравнение (5) к множеству одномерных обыкновенных дифференциальных уравнений (в случае, если интеграл рассеяния учитывается итерационно):

$$\frac{dI(\eta, \boldsymbol{\omega})}{d\eta} + k(\eta) I(\eta, \boldsymbol{\omega}) = S(\eta, \boldsymbol{\omega}),$$

где η — параметр, изменяющийся вдоль луча, проходящего через данный узел пространственной разностной сетки, и имеющий смысл расстояния, $S(\eta, \boldsymbol{\omega})$ — источник рассеянного излучения.

Таким образом, для каждой точки пространственной сетки $\mathbf{x}_i$ необходимо провести трассировку лучей, приходящих в нее вдоль векторов $\boldsymbol{\omega}_j$ выбранной с учетом удаленности точки $\mathbf{x}_i$ от границы сферы направлений.

Отметим, что аккуратный учет граничных условий для поля излучения приводит к выбору своего набора дискретных направлений для каждой пространственной точки, что серьезно увеличивает требования к вычислительным ресурсам, прежде всего к памяти.

Отметим, что в отличие от чисто двумерной задачи, наличие цилиндрической симметрии приводит к необходимости трассировать направления распространения излучения в трехмерной области (рис. 4). При этом в силу симметрии достаточно вычислить интенсивность излучения $I(\mathbf{x}, \boldsymbol{\omega}) = I_{zOr}(z, r, \boldsymbol{\omega})$ в полуплоскости zOr.

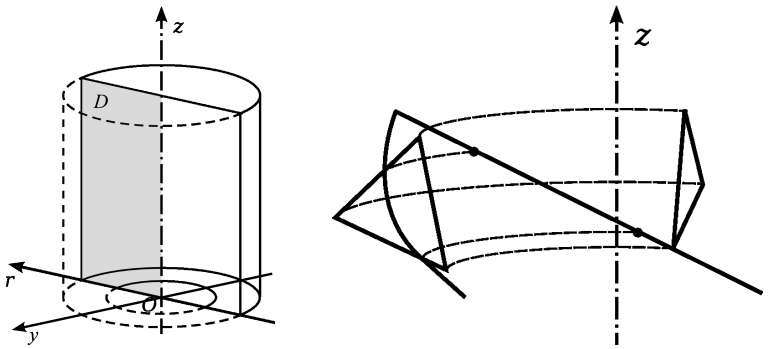


Рис. 4. Цилиндрическая симметрия и трассировка луча в задаче о переносе излучения

Для вычисления интенсивности $I_{zOr}(z, r, \boldsymbol{\omega})$ в соответствии с методом дискретных направлений необходимо протрассировать луч, проходящий через точку $[r, 0, z]^{\mathrm{T}}$ в направлении $\boldsymbol{\omega}$, по полуцилиндрической трехмерной области $\mathcal{D}$, причем треугольные ячейки плоской разностной сетки в области D (залита серым цветом на рис. 4 слева) соответствуют пространственным кольцам треугольного сечения, которые заполняют $\mathcal{D}$. Таким образом, получаем задачу трассировки прямолинейного луча в области $\mathcal{D}$, состоящей из кольцевых подобластей с постоянными оптическими свойствами вещества (пример кольцевой подобласти изображен на рис. 4, справа). Эту задачу легко свести к трассировке кривой второго порядка (гиперболы) в плоской области D.

Отметим, что трассировка гиперболы на треугольной сетке является существенно более затратной процедурой (затраты машинного времени возрастают на порядок), чем трассировка прямой, поэтому предпочтительным является хранение всей информации о трассировке лучей в оперативной памяти в процессе динамического расчета с интегрированием квазистационарного УПИ на каждом временном слое. Подобный объем памяти на данный момент не доступен персональным компьютерам, но вполне достижим на суперкомпьютерной вычислительной технике.

4.3. Учет интеграла рассеяния

Учет интеграла рассеяния

$$S(t, \mathbf{x}, \boldsymbol{\omega}) = \int\limits_{\Omega} \Gamma(t, \mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}') I(t, \mathbf{x}, \boldsymbol{\omega}')\, d\boldsymbol{\omega}' \tag{7}$$

в примененной численной схеме производится итерационно в соответствии с известными схемами для моделирования однократного рассеяния. На каждом шаге по времени для расчета интенсивности излучения производится две итерации. На первой из них решается уравнение переноса излучения в виде

$$\boldsymbol{\omega} \cdot \nabla I^{(1)}(t, \mathbf{x}, \boldsymbol{\omega}) + k(t, \mathbf{x}) I^{(1)}(t, \mathbf{x}, \boldsymbol{\omega}) = 0,$$

т.е. в предположении нулевого вклада рассеянных фотонов в общую интенсивность. Далее вычисляется функция источника рассеяного излучения

$$S^{(1)}(t,\mathbf{x},\boldsymbol{\omega}) = \int\limits_{\Omega} \Gamma(t,\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')I^{(1)}(t,\mathbf{x},\boldsymbol{\omega}')\,d\boldsymbol{\omega}'$$

и производится пересчет поля излучения в расчетной области путем интегрирования уравнения

$$\boldsymbol{\omega}\cdot\nabla I^{(2)}(t,\mathbf{x},\boldsymbol{\omega}) + k(t,\mathbf{x})I^{(2)}(t,\mathbf{x},\boldsymbol{\omega}) = \beta\,(t,\mathbf{x})\,S^{(1)}(t,\mathbf{x},\boldsymbol{\omega})$$

с известной правой частью.

Подобный подход требует на каждом шаге по времени в каждом узле пространственной сетки производить интегрирование функции интенсивности $I^{(1)}(t,\mathbf{x},\boldsymbol{\omega})$ по всей сфере направлений. Взятие таких интегралов является наиболее затратной в вычислительном плане процедурой, причем тем более тяжелой, чем более подробно разбиваются сферы направлений в узлах сетки. В то же время такие интегралы берутся независимо в разных пространственных точках, алгоритм их взятия не содержит управляющих конструкций ветвления, что позволяет широко использовать параллельные вычислительные технологии, в частности технологию CUDA для вычислений на графических ускорителях.

4.4. Параллельные вычислительные технологии

Для решения ресурсоемких задач, подобных рассматриваемым в данной работе моделям радиационной МГД, в настоящее время широко применяется суперкомпьютерная техника, реализующая параллельные вычислительные технологии для систем с общей и распределенной памятью [10]. Мы уже указали на высокие требования метода дискретных направлений в целом и процедуры трассировки лучей и интегрирования вдоль них УПИ в частности к ресурсам памяти вычислительного модуля. В данном случае наиболее целесообразным оказывается именно использование SMP-модулей с максимальным доступным на данном узле кластера числом процессорных ядер и максимальной доступной памятью. В случае кластера К-100 ИПМ им. М.В. Келдыша РАН имеется возможность использования узла кластера целиком в качестве одного вычислительного модуля, содержащего 12 процессорных ядер, 96 Гб оперативной памяти и 3 графических ускорителей nVidia Tesla.

Численный метод реализован в виде программного комплекса на языках программирования Фортран-90 и C++. Программный комплекс разработан для применения на высокопроизводительных кластерах с графическими ускорителями и реализует следующий подход к распараллеливанию на основе технологий для систем с общей памятью.

Применение технологии OpenMP. Интегрирование УПИ вдоль протрассированных и сохраненных в ОЗУ лучей, соответствующих данному узлу пространственной сетки, осуществляется независимо для каждого узла. При этом информация о трассе луча (номер проходимой ячейки, длина отрезка луча) используется только однажды. В данном случае удобно производить параллельные вычисления над общей оперативной памятью, используя возможности многоядерных процессоров общего назначения. В этом случае узел кластера используется как SMP вычислительный модуль с 12 вычислительными ядрами (каждое из которых включает одно процессорное ядро) и 96 Гб общей для всех ядер памяти.

Применение технологии nVidia CUDA. Будучи более тяжелой операцией, вычисление интеграла рассеяния одновременно является более удобной для распараллеливания. В рамках данной операции для каждого узла пространственной сетки вычисляется N_{scat} интегралов вида (7), где N_{scat} — количество дискретных направлений, в которые производится рассеяние излучения центральной аккреционной системы модели. При этом для вычисления каждого из этих N_{scat} интегралов используются одни и те же данные об интенсивности приходящего излучения и телесном угле, откуда это излучение приходит в

рассматриваемый пространственный узел. Именно в таких задачах наиболее эффективными оказываются вычисления на графических ускорителях благодаря относительно низкой доле времени обменов между памятью ускорителя и основной памятью узла в общем времени вычислений. В этом случае узел кластера используется как модуль с одним вычислительным ядром, включающим одно из процессорных ядер, один графический ускоритель с интегрированной памятью и 96 Гб памяти общего назначения. Вычисление всей совокупности интегралов вида (7) в данном пространственном узле производится в рамках блока тредов одного графического мультипроцессора, а каждый отдельный интеграл вычисляется в рамках одного треда в указанном блоке. При этом загрузка распределения интенсивности и телесных углов для данного пространственного узла и данной сферы направлений производится однократно, обеспечивая данные для всего блока тредов.

Использование указанных подходов позволило существенно ускорить работу программного комплекса. В частности, процедура трассировки и интегрирования УПИ вдоль луча с использованием технологии OpenMP на 12-ядерном узле кластера К-100 выполняется в 10.8 раз быстрее, чем на одном процессорном ядре, а процедура вычисления источников рассеяного излучения во всех пространственных узлах выполняется на графическом процессоре nVidia Tesla в 82.3 быстрее, чем на одном процессорном ядре (все измерения производились на одном узле кластера К-100, процессор Intel Xeon X5670 2.93ГГц).

5. Результаты расчетов

Вычисления производились с использованием кластера К-100 ИПМ им. М.В. Келдыша РАН. Использованы треугольные сетки, состоящие из 8000 ячеек, со сгущением у диска. Проведена серия вычислительных экспериментов для разных вариантов граничных условий на тонком диске. Подробно рассмотрим случай, когда плотность ветра с тонкого диска ограничена снизу величиной ρ_w. Шаг по времени в расчетах выбирался автоматически в зависимости от скорости течения плазмы в расчетной области. Во избежание развития неустойчивостей число Куранта принималось равным $CFL = 0.05$.

5.1. Ускорение выброса в канале

Начальными условиями для расчетов служат распределения величин в момент времени $t = 18.05$, полученные в [5]. В рассматриваемой области существует замагниченный канал, внутри которого движется разреженное вещество струйного выброса, причем плотность вещества падает с удалением от центральной области (рис. 3).

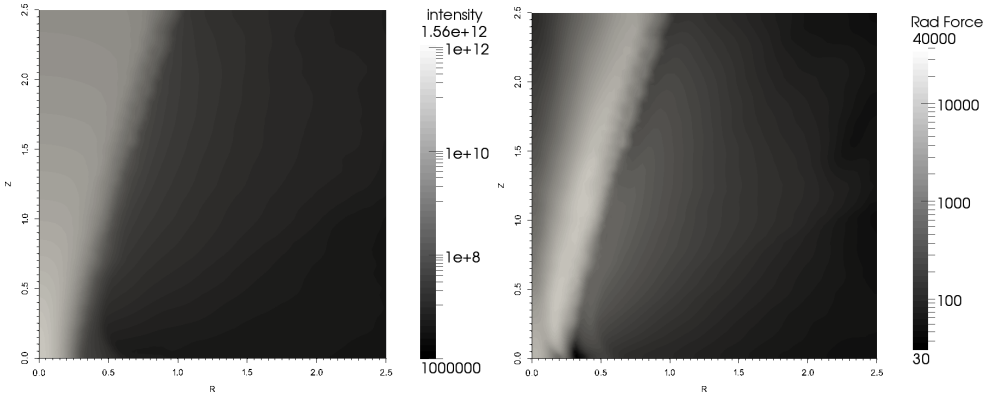


Рис. 5. Начальное распределение интенсивности I и модуля силы давления излучения

„Включение“ поля излучения приводит к возникновению дополнительной ускоряющей силы, действующей на разреженное вещество, причем величина этой силы тем больше, чем

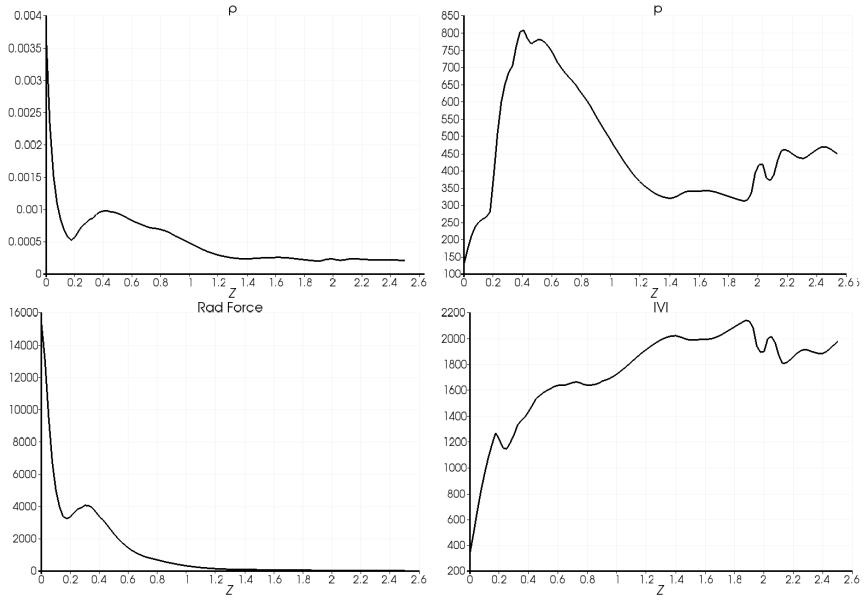


Рис. 6. Распределения параметров течения вещества вдоль оси $r = 0$: плотности ρ, давления p, объемной силы радиационного давления, модуля скорости $|\mathbf{v}|$

ближе к источнику излучения. На рис. 5 приведено начальное распределение интенсивности излучения и модуля силы радиационного давления (под силой в данном случае понимается член $\nabla \cdot \hat{\mathbf{T}}$, поскольку член $\partial G/\partial t$ в (3) мал по модулю и на характер течения не влияет), действующей на вещество выброса. Действие указанной силы в обоих расчетах приводит к тому, что скорость течения выброса быстро достигает значений порядка 10^3, а ускорение происходит в ближайшей окрестности излучающего центрального объекта.

Сила радиационного давления убывает с удалением от источника излучения, так что нижние (относительно оси Oz) слои ускоряются быстрее верхних, что приводит к сжатию и повышению давления в потоке выброса. Наконец, к моменту времени $t = 18.055$ наибольшая скорость течения достигает максимума за все время расчета и дальше не увеличивается, начальные эффекты считаются пройденными, устанавливается стабильный режим выброса. На рис. 6 изображены распределения основных величин течения вещества выброса вдоль оси Oz в момент времени $t = 18.075$. На рис. 7 изображены распределения плотности, модуля скорости а также полоидальных мгновенных траекторий и полоидальных силовых линий магнитного поля в расчетной области в тот же момент времени.

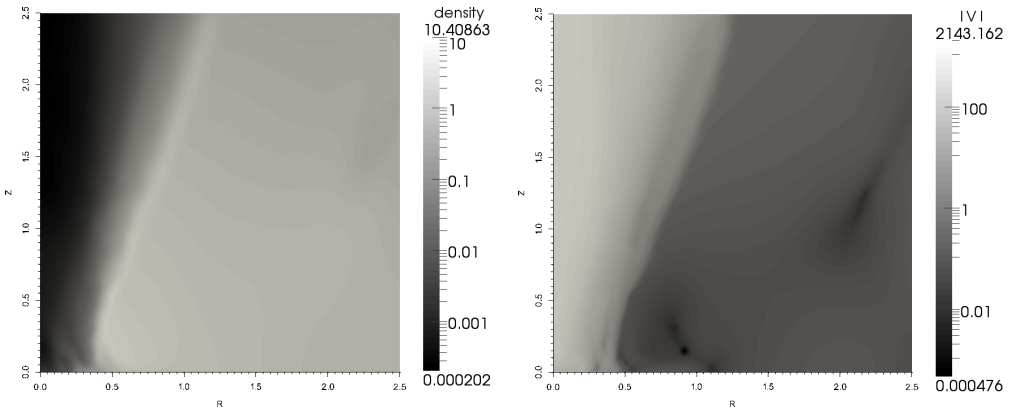


Рис. 7. Распределения плотности и модуля скорости$t = 18.075$

Полученные скорости достигают $1/5\,c_d$, где c_d — безразмерная скорость света. При этом выброс остается хорошо коллимированным, магнитное поле в канале сохраняет свою структуру, вокруг наиболее высокоскоростной части выброса сохраняются мягкие стенки из закрученного магнитного поля. Давление и плотность газа начиная с некоторой высоты z падают при удалении от центрального объекта, поэтому выброс не будет замедляться вне расчетной области и, достигнув предельного значения скорости, будет распространяться в разреженную среду, сохраняя высокую энергетику. В то же время максимум давления приходится на окрестности плоскости $z = 0.4$, т.е. на верхнюю часть сужения канала. Подобное распределение градиента давления вдоль оси Oz может приводить к возникновению эффекта „запирания“ вещества в окрестности центрального объекта в случае, если силы радиационного давления будет недостаточно, чтобы преодолеть этот своеобразный „барьер“.

В расчетах устанавливается периодический режим выброса, при котором сохраняется высокий темп течения вещества, плазма постоянно поступает в канал с тонкого диска в экваториальной плоскости, не давая джету стать более разреженным. Отметим, что для устойчивого функционирования механизма формирования и ускорения струйного выброса необходимо постоянное поступление вещества внутрь ускоряющего канала с плотностью потока не меньше некоторого предельного значения, например, $\rho_w = 0.01$.

5.2. Периодический режим выброса

Особый интерес вызывает установившийся в расчетах периодический режим выброса вещества. В подобном режиме сохраняется средняя высокая скорость истечения вещества, при этом наблюдаются периодические всплески скорости потока. Период всплесков $T_b = 0.01$ совпадает со временем, необходимым плазме, чтобы уйти от центрального объекта и преодолеть сужение ускоряющего канала, выйдя на стабильный уровень скорости.

Система переходит в колебательный режим сразу после установления скорости потока, когда первая порция плазмы, ускоренной излучением, покидает расчетную область. После ухода этой порции над тонким диском образуется область разреженности вещества. В сужении канала над излучающим центральным объектом образуется барьер из плазмы высокого давления, который частично „запирает“ вещество в получившейся „каверне“. Поток массы с диска ограничен снизу путем поддержания плотности плазмы не менее значения ρ_w. Цикл действия полученного механизма выглядит следующим образом.

1. Над тонким диском существует разреженная каверна, запертая сверху барьером давления горячего газа шлейфа предыдущего всплеска. За счет преобладания потока массы с диска над потоком массы через барьер каверна начинает заполняться плазмой, становится более непрозрачной для излучения центрального объекта.

2. Повышение плотности газа в каверне приводит к увеличению силы радиационного давления на порядок. Воздействие давления излучения тем больше, чем ближе вещество находится к центральному объекту — вещество в каверне начинает сжиматься давлением излучения снизу, давление газа в каверне растет.

3. Давление газа в каверне превышает давление газа барьера в сужении канала, накопленное в каверне плотное вещество образует быстро движущийся вверх по оси Oz горячий пузырь. Образуется всплеск скорости, проходящий сужение канала и растущий далее за счет расширения пузыря.

4. Пузырь выходит в расширяющуюся часть канала, плотность его падает вместе с расширением канала, скорость продолжает нарастать, в распределении скорости образуется фронт всплеска, причем нижние слои газа движутся быстрее, накатывая на верхние и заостряя фронт (см. рис. 8). За пузырем образуется область разреженного газа низкого давления, барьер давления в сужении канала закрывается, начинается накопление новой порции выбрасываемого вещества.

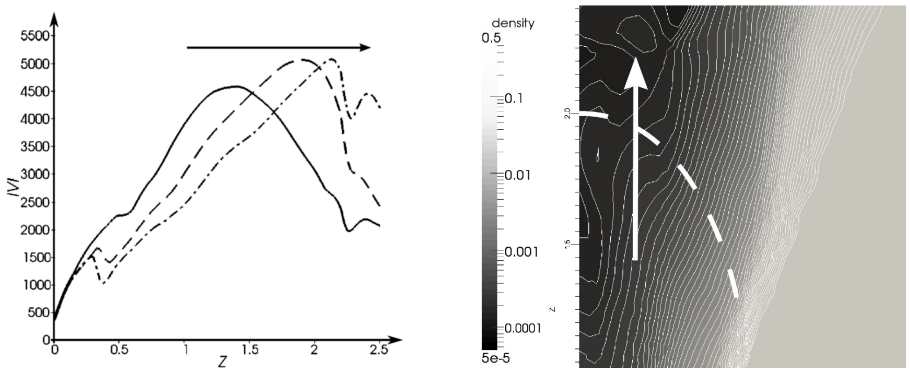


Рис. 8. Обострение волны скорости в ходе ускорения всплеска и образование фронта ударной волны. Распределение модуля скорости вдоль оси Oz в последовательные моменты времени (слева) и волна плотности в районе фронта волны (справа)

5. На выходе из расчетной области фронт всплеска формирует ударную волну, нагребающую на себя газ из фонового течения в канале. Формируется сгусток вещества, движущегося со скоростями порядка $1/5\, c_d$, где c_d — безразмерная скорость света. Сгусток продолжает расширяться и покидает расчетную область. Каверна над центральным объектом заполняется плазмой, истекающей с тонкого диска.

5.3. Обсуждение результатов

Перейдем от результатов расчетов, представленных в безразмерном виде, к оценкам значений размерных переменных задачи. Примем следующие масштабы величин: концентрация $n_0 = 10^8$ см$^{-3}$; вещество выброса — молекулярный водород, т.е. плотность $\rho_0 = 3.34 \times 10^{-16} \frac{\text{г}}{\text{см}^3}$; линейный размер задачи $L = 10^{15}$ см; напряженность магнитного поля $B_0 = 0.06$ Э; масса центрального тела $M = 3M_\odot = 6 \times 10^{33}$ г; временной масштаб задачи $t_0 = 1.12 \times 10^9$ с; масштаб скорости $V_0 = 0.9 \times 10^6$ см/с.

Расчет моделирует истечение от формирующейся протозвезды с массой $M = 3M_\odot$, окруженной околозвездным диском радиусом $r_d = 0.6L_0 \approx 40$ а.е. Диск пронизан полоидальным магнитным полем, напряженность которого составляет ~ 0.06 Э. На систему аккрецирует сверхзвуковой поток вещества с темпом аккреции $\sim 5 \times 10^{-5}\ M_\odot$/год. Центральные области системы излучают, интенсивность излучения соответствует излучению абсолютно черного тела температурой 7×10^4 K. Над звездой с диском сформирован замагниченный канал, содержащий разреженное вещество, источником которого является диск. Излучение центрального объекта сфокусировано внутрь канала, претерпевает лишь однократное рассеяние на электронах плазмы (томпсоновское рассеяние). Перпендикулярно экваториальной плоскости диска формируется коллимированное истечение вещества (джет), ускоряемое давлением излучения. Скорость потока в среднем составляет 2×10^4 км/с, угол раствора джета $\sim 10^\circ$. Поток состоит из отдельных сгустков вещества, движущихся со скоростью большей, чем скорость фонового течения. Максимальная скорость сгустков достигает 5×10^4 км/с. Период выброса сгустков составляет 13 дней.

На образование сгустков принципиальное влияние оказывают три фактора: радиационное давление излучения центрального объекта, эффективно ускоряющего плотное вещество и мало действующего на разреженную плазму; образование на начальном этапе радиационного ускорения выброса в сужении канала „барьера“ газового давления, приводящего к запиранию плазмы в каверне над излучающим диском; ограниченность снизу потока вещества с диска, что обеспечивает постоянное пополнение каверны над излучающим диском веществом и воспроизводство рассмотренного механизма.

Полученный струйный выброс является устойчивым при условии ограничения снизу потока вещества с диска, которое препятствует „выдуванию“ плазмы из канала и прекращению действия силы радиационного давления. Коллимация джета обеспечивается осевым магнитным полем во внутренней части канала и наличием у канала мягких спиральных стенок из закрученного магнитного поля.

6. Заключение

Построена и исследована модель радиационного ускорения вещества в замагниченном канале над тонким идеально проводящим диском. Математическая модель состоит из уравнения переноса излучения и системы уравнений идеальной магнитной гидродинамики с учетом гравитационной силы, создаваемой центральным объектом, и давления излучения. Для численного решения системы уравнений применена модификация алгоритмов, разработанных и рассмотренных в [5]. Рассмотренные методы (разностная схема для системы уравнений идеальной МГД и метод дискретных направлений для УПИ) адаптированы для расчетов в цилиндрической системе координат и реализованы в виде программного комплекса для высокопроизводительных SMP-машин с графическими ускорителями. Построенный программный комплекс показал высокую эффективность параллельного расчета поля интенсивности излучения методом дискретных направлений, причем применение технологии nVidia CUDA дает ускорение процедуры вычисления интеграла рассеяния в 82.3.

Список литературы

1. Черепащук А.М. SS 433: Новые результаты, новые проблемы // Земля и Вселенная. 1986. Vol. 1. Pp. 21–29.
2. Mioduszewski A.J., Rupen M.P., Walker R.C. et al. A Summer of SS433: Forty Days of VLBA Imaging // Bulletin of the American Astronomical Society. 2004. Vol. 36. P. 967.
3. Галанин М.П., Торопин Ю.М., Чечеткин В.М. Радиационное ускорение порций вещества в аккреционных воронках около астрофизических объектов // Астрономический журнал. 1999. Т. 76, № 2. С. 143–160.
4. Savel'ev V.V., Toropin Yu.M., Chechetkin V.M. A Possible Mechanism for the Formation of Molecular Flows // Astronomy Reports. 1996. Vol. 40. Pp. 494–508.
5. Галанин М.П., Лукин В.В., Чечеткин В.М. Ускорение джетов при различных вариантах моделирования источника вещества // Матем. моделирование. 2011. Т. 23, № 10. С. 65–81.
6. Галанин М.П., Лукин В.В., Чечеткин В.М. Радиационное ускорение астрофизического канализированного струйного выброса // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. Спец. выпуск “Прикладная математика”. 2011. С. 11–33.
7. Четверушкин Б.Н. Математическое моделирование задач динамики излучающего газа. М.: Наука, 1985. 304 с.
8. Pogorelov N.V., Semenov A.Yu. Solar wind interaction with the magnetized interstellar medium // Astron. Astrophys. 1997. Vol. 321. Pp. 330–337.
9. Toro E.F. Riemann solvers and numerical methods for fluid dynamics. A practical introduction. Berlin: Springer, 2009. 724 pp.
10. Лукин В.В., Марчевский И.К. Учебно-экспериментальный вычислительный кластер. Ч.1. Инструментарий и возможности // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2011. №4. С. 28–43.

Построение коллизии для 75-шаговой версии хэш-функции SHA-1 с использованием ГПУ-кластеров

Е.А. Гречников[1], А.В. Адинец[2,3]

МГУ им. М. В. Ломоносова, механико-математический факультет[1],
Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова[2],
Объединённый институт ядерных исследований[3]

Криптографические хэш-функции, в частности, SHA-1, в настоящее время широко используются в современных информационных технологиях. Важным свойством таких функций является устойчивость к коллизиям, т.е. сложность построения двух различных входных сообщений, значения хэш-функции от которых совпадают. Мы развиваем метод разностных (дифференциальных) атак для поиска коллизий хеш-функции SHA-1 и её сокращённых вариантов. В данной статье описывается реализация метода характеристик для хэш-функции SHA-1 на кластере из графических процессоров. Использование различных оптимизаций позволило достичь эффективности ГПУ-реализации 50%, и ускорения в 39 раз по сравнению с реализацией на одном ядре ЦПУ. Оптимизации также включали модификацию метода поиска характеристик. На текущий момент наши улучшения метода и использование ГПУ-раздела суперкомпьютера «Ломоносов» позволили найти коллизию для SHA-1, сокращённой до 75 шагов (полная функция состоит из 80), что является мировым рекордом.

1. Введение

В современной криптографии широко используются различные хеш-функции. Хеш-функция есть функция, отображающая сообщения (строки из 0 и 1) произвольной длины в последовательности из 0 и 1 ограниченной длины — значения хеш-функции или хеш-значения. Значение хеш-функции есть как бы отпечаток всего сообщения, а роль его подобна отпечаткам пальцев в процессе идентификации.

Для любой хеш-функции существуют сообщения, имеющие одинаковые хеш-значения, ведь множество сообщений бесконечно, а множество хеш-значений конечно. Если удаётся явно построить два различных сообщения, имеющие совпадающее хеш-значение, иначе говоря, построить коллизию, то хеш-функция считается скомпрометированной, что имеет разрушительные последствия для некоторых криптографических приложений.

Хеш-функция, обозначаемая SHA-1 (Secure Hash Algorithm), преобразует сколь угодно длинные сообщения (в стандарте максимальная длина равна $2^{64} - 1$ бит) в 160-битные хеш-значения. Эта хеш-функция была опубликована NIST (National Institute of Standards and Technology) в США в 1995 г. и в настоящее время используется как важная составная часть различных государственных и индустриальных стандартов безопасности, таких как электронная цифровая подпись, аутентификация пользователей, обмен ключами и построение псевдослучайных последовательностей. SHA-1 внедрена почти во все коммерческие системы безопасности.

В течение ряда лет в различных странах предпринимаются активные попытки компрометации функции SHA-1, и, хотя коллизия для этой функции не построена, исследователи продвинулись в этой деятельности достаточно далеко. В настоящее время NIST проводит конкурс на построение новой хеш-функции, которая смогла бы заменить SHA-1. Предполагается ввести новую функцию в действие в 2012 году.

Криптоаналитические задачи, как правило, достаточно легко распараллеливаются на любое доступное количество вычислительных ресурсов. Поэтому тем более логично ис-

пользовать ГПУ для решения таких задач. И хотя ГПУ активно используются для задачи подбора пароля [1], нам не встречались случаи использования ГПУ для поиска коллизий для хэш-функций.

Наша цель — построить коллизию хеш-функции SHA-1. На настоящий момент с использованием ГПУ-раздела суперкомпьютера «Ломоносов» мы установили мировой рекорд [3], построив коллизию для хеш-функции, устроенной аналогично SHA-1, но с меньшим значением одного из параметров схемы: 75 шагов вместо 80 шагов в SHA-1.

Вклад данной статьи можно кратко описать следующим образом:

- Нами впервые предложена реализация поиска коллизий для хэш-функций, использующая графические процессоры
- При помощи нашей реализации построена коллизия для 75-раундовой версии SHA-1, что на данный момент является мировым рекордом

Данная статья организована следующим образом. В разделе 2 описывается устройство хэш-функции SHA-1. В разделе 3 описывается устройство разностных атак, в том числе применительно к хэш-функции SHA-1, а в разделе 4 описывается используемый алгоритм поиска характеристики. Особенности ГПУ-реализации приводятся в разделе 5, а в разделе 6 описываются проведённые расчёты и приводится полученная коллизия. Наконец, раздел 7 содержит подведение итогов и благодарности.

2. Хэш-функция SHA-1

Используемые в настоящей статье в формулах обозначения приведены в таблице 1 Хеш-функция SHA-1 [2] устроена следующим образом. В конец входного сообщения определённым образом добавляется несколько бит, зависящих от длины сообщения, так, чтобы результат состоял из нескольких 512-битных блоков $M_1, \ldots, M_k$. 160-битное хеш-значение представляет из себя набор из 5 32-битных переменных, для вычисления которых к начальному набору H_0 последовательно «подмешиваются» блоки M_i следующим образом: для $i = 1, \ldots, k$ вычисляется $H_i = H_{i-1} + g(M_i, H_{i-1})$. Константа H_0 является частью стандарта. Хеш-значением входного сообщения является H_k.

Таблица 1. Обозначения, используемые в данной статье

Обозначение	Описание
X	32-битное число, связанное с первым сообщением
X^*	32-битное число, связанное со вторым сообщением
X^2	пара 32-битных чисел (X, X^*)
$X \oplus Y$	побитовое исключающее ИЛИ (XOR)
$X + Y$	сложение по модулю 2^{32}
$[X]_i$	i-й бит числа X ($i = 0$ — младший бит)
$X \lll i$	циклический сдвиг влево на i бит
$X \ggg i$	циклический сдвиг вправо на i бит

Для нахождения коллизии достаточно построить два набора $(M_1, \ldots, M_k)$ и $(M_1^*, \ldots, M_k^*)$ одной и той же длины, для которых $H_k = H_k^*$. Поскольку длины одинаковы, то биты, добавляемые в конец сообщений при вычислении SHA-1, также будут одинаковы, так что последующие значения H_i и H_i^* совпадут.

Сжимающая функция $g(M, H)$ строит новое 160-битное значение по старому 160-битному значению H и 512-битному блоку сообщения M. Входные и выходные данные представляются как 32-битные переменные, $H = (A_0, B_0, C_0, D_0, E_0)$, $M = (M_0, \ldots, M_{15})$, $g(M, H) =$

$(A_{80}, B_{80}, C_{80}, D_{80}, E_{80})$. Вычисление сжимающей функции состоит из двух частей: расширения сообщения и обновления состояния.

Расширение сообщения линейно дополняет 16 переменных M_i до 80 переменных W_i, которые будут использоваться при обновлении состояния:

$$\begin{aligned} W_i &= M_i, \quad 0 \leqslant i < 16; \\ W_i &= (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1, \quad i \geqslant 16. \end{aligned}$$

Обновление состояния состоит из 80 шагов следующего вида, изображённого на рис. 1.

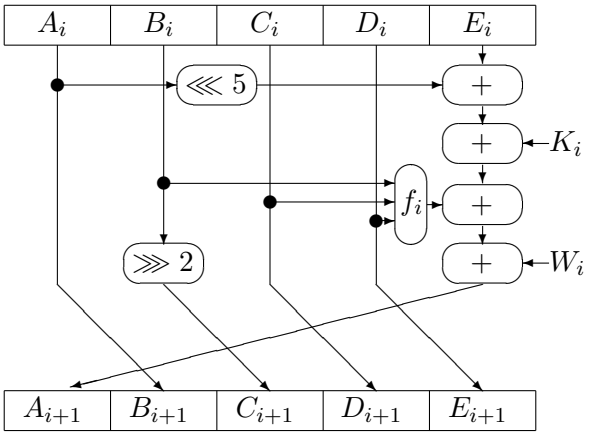


Рис. 1. Вид одного раунда хэш-функции SHA-1

Здесь K_i — константы, а f_i — следующие функции:

- на первых 20 шагах $K_i =$ `0x5A827999`,

$$f_i(b, c, d) = f_{IF}(b, c, d) = (b \wedge c) \vee (\overline{b} \wedge d);$$

- на вторых 20 шагах $K_i =$ `0x6ED9EBA1`,

$$f_i(b, c, d) = f_{XOR}(b, c, d) = b \oplus c \oplus d;$$

- на третьих 20 шагах $K_i =$ `0x8F1BBCDC`,

$$f_i(b, c, d) = f_{MAJ}(b, c, d) = (b \wedge c) \vee (b \wedge d) \vee (c \wedge d);$$

- на последних 20 шагах $K_i =$ `0xCA62C1D6`,

$$f_i(b, c, d) = f_{XOR}(b, c, d) = b \oplus c \oplus d.$$

Легко видеть, что $B_i = A_{i-1}$, $C_i = A_{i-2} \ggg 2$, $D_i = A_{i-3} \ggg 2$, $E_i = A_{i-4} \ggg 2$. Таким образом, достаточно рассматривать только переменные A_i; начальные значения $(A_0, \ldots, E_0)$ задают $(A_{-4}, \ldots, A_0)$, конечные значения $(A_{80}, \ldots, E_{80})$ получаются из $(A_{76}, \ldots, A_{80})$.

Поскольку вычисления для функции SHA-1 достаточно трудоёмки, для демонстрации новых и оптимизации существующих методов принято использовать так называемые сокращённые варианты, которые отличаются от «полной» функции меньшим количеством шагов в сжимающей функции, сами шаги при этом остаются теми же.

3. Разностные атаки

Если взять в качестве блока сообщения случайный набор бит, то хеш-функция SHA-1 выдаст случайное 160-битное хеш-значение с приблизительно равномерным распределением среди всех возможных 160-битных строк; таким образом, если взять два случайных блока, то вероятность того, что они образуют коллизию, составляет примерно 2^{-160}, что, разумеется, слишком мало для практических целей.

Разностные атаки эволюционировали со временем, различные этапы (в том числе атаки на другие хеш-функции, построенные по той же схеме) отмечены в работах [4] (MD4), [7] (35 шагов SHA-0), [8] (полный SHA-0), [5] (MD5), [6] (58 шагов SHA-1), [9] (64 шагов SHA-1), [10] (70 шагов SHA-1). Мы усовершенствуем метод характеристик, описанный в двух последних работах.

Ключевая идея разностных атак заключается в том, чтобы вести перебор не по всем возможным парам сообщений, а только по парам сообщений с фиксированными разностями по модулю 2 $\delta M_i = M_i \oplus M_i^*$, за что атаки и получили такое название. Помимо фиксирования δM_i, полезным также оказывается фиксировать отдельные биты в M_i, а также δA_i и A_i. Более точно, назовём *характеристикой* набор из $(80+85)\cdot 32$ элементарных условий на пары бит $([W_i]_j, [W_i^*]_j)$ и $([A_i]_j, [A_i^*]_j)$. Каждое элементарное условие какие-то из 4 возможных вариантов разрешает, остальные запрещает; все 2^4 возможных элементарных условий собраны в следующей таблице вместе с условными обозначениями для них.

Таблица 2. Обозначения условий на биты, применяемые при записи характеристик

∇_i	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
`*`	✓	✓	✓	✓
`-`	✓	–	–	✓
`x`	–	✓	✓	–
`0`	✓	–	–	–
`u`	–	✓	–	–
`n`	–	–	✓	–
`1`	–	–	–	✓
`#`	–	–	–	–
`3`	✓	✓	–	–
`5`	✓	–	✓	–
`7`	✓	✓	✓	–
`A`	–	✓	–	✓
`B`	✓	✓	–	✓
`C`	–	–	✓	✓
`D`	✓	–	✓	✓
`E`	–	✓	✓	✓

Множество пар (X, X^*), удовлетворяющих всем 32 условиям на соответствующую переменную, будем обозначать ∇X. Пусть известна некоторая характеристика и мы хотим организовать перебор по этой характеристике для нахождения коллизии. Будем последовательно подбирать $M_0^2 = (M_0, M_0^*)$, затем M_1^2 и так далее. На каждом новом шаге мы выбираем очередной вариант для пары M_i^2 с учётом условий из характеристики, вычисляем пару A_{i+1}^2, проверяем условия из характеристики, пока не исчерпаем все возможности или не найдём нужной пары. Если пара найдена, продвигаемся на шаг вперёд, если возможности исчерпаны, отступаем на шаг назад. После того, как мы подобрали M_{15}^2, сообщение становится полностью определено, так что последующие шаги состоят только из проверок условий.

Число вариантов пары A^2_{i+1}, удовлетворяющих условиям характеристики, может быть меньше, чем у соответствующей пары M^2_i. В таком случае мы перебираем A^2_{i+1} и вычисляем M^2_i; при фиксированных предыдущих значениях A_i, A^*_i соответствие взаимно-однозначно.

Перебор будем вести до нахождения первой коллизии или до исчерпания пространства перебора. Предположим, что перебор завершится успехом, и попробуем оценить его сложность. Для этого введём несколько определений.

Набор $(W^2_0, \ldots, W^2_i)$ назовём непротиворечивым, если его можно продолжить до полной пары расширенных сообщений, удовлетворяющих характеристике. *Степень свободы сообщения на шаге i (обозначается $\tilde{F}_W(i)$)* – это число непротиворечивых наборов $(W_0, \ldots, W_i)$, продолжающих непротиворечивый набор $(W_0, \ldots, W_{i-1})$. При $i \geqslant 16$, очевидно, $\tilde{F}_W(i) = 1$. Если в характеристике условия на $W_{16}, \ldots, W_{79}$ тривиальны, то при $i < 16$ степень свободы сообщения на шаге i равна просто $|\nabla W_i|$. В общем случае условия характеристики на $W_{16}, \ldots, W_{79}$ влекут за собой какие-то линейные соотношения на отдельные биты $[M_i]_j$; если есть m таких независимых соотношений, то $\tilde{F}_W(i)$ равно $\frac{|\nabla W_i|}{2^m}$.

Можно также посчитать $\tilde{F}_A(i) = |\nabla A^2_{i+1}|$. Если $\tilde{F}_A(i) \geqslant \tilde{F}_W(i)$, то мы перебираем M^2_i и вычисляем A^2_{i+1}; иначе мы перебираем A^2_{i+1} и вычисляем M^2_i. В первом случае положим $F_W(i) = \tilde{F}_W(i)$, во втором положим $F_W(i) = \frac{\tilde{F}_A(i)}{2^m}$; тогда $F_W(i)$ — число потомков вершины дерева перебора на шаге i с поправкой на неявные линейные ограничения.

В определении хеш-функции значение A_{i+1} вычисляется на каждом шаге по значениям $A_{i-j}, 0 \leqslant j \leqslant 4$, и W_i. Для оценок введём величины, которые характеризуют шаг хеш-функции; временно забудем, что A получаются по сообщениям, и рассмотрим вероятностное пространство, в котором $A_{i-j}, 0 \leqslant j \leqslant 4$, и W_i — независимые величины, удовлетворяющие соответствующим ограничениям из характеристики.

Неконтролируемая вероятность на шаге i (обозначается $P_u(i)$) — это вероятность того, что результат шага i будет удовлетворять характеристике при условии, что на всех предыдущих шагах состояния и расширенные сообщения удовлетворяют характеристике. То есть при $\tilde{F}_A(i) \geqslant \tilde{F}_W(i)$

$$P_u(i) := Pr(A^2_{i+1} \in \nabla A_{i+1} | A^2_{i-j} \in \nabla A_{i-j}, 0 \leqslant j \leqslant 4, W^2_i \in \nabla W_i),$$

а при $\tilde{F}_A(i) < \tilde{F}_W(i)$

$$P_u(i) := Pr(W^2_i \in \nabla W_i | A^2_{i-j} \in \nabla A_{i-j}, 0 \leqslant j \leqslant 4, A^2_{i+1} \in \nabla A_{i+1}).$$

Контролируемая вероятность на шаге i (обозначается $P_c(i)$) – это вероятность того, что найдется хотя бы одна пара W^2_i, удовлетворяющая характеристике, такая что результат шага i будет удовлетворять характеристике при условии, что на всех предыдущих шагах состояния удовлетворяют характеристике. То есть

$$P_u(i) := Pr(\exists W^2_i \in \nabla W_i : A^2_{i+1} \in \nabla A_{i+1} | A^2_{i-j} \in \nabla A_{i-j}, 0 \leqslant j \leqslant 4).$$

(Контролируемая вероятность не меняется при перемене ролей A и W.)

Можно с некоторой погрешностью оценить сложность успешного перебора, считая приближённо, что все шаги независимы. А именно, на i-м шаге в среднем придётся обойти $N_S(i)$ вершин дерева перебора, где:

- $N_S(80) = 1$ (достаточно найти одну коллизию),
- $N_S(i) = \max\left\{\frac{N_S(i+1)}{F_W(i)P_u(i)}, \frac{1}{P_c(i)}\right\}$ (с одной стороны, в среднем у вершины дерева перебора будет $F_W(i)$ потомков, среди которых доля $P_u(i)$ будет давать вершину следующего уровня; с другой стороны, с вероятностью $P_c(i)$ вершина дерева перебора безнадёжна для получения вершин следующего уровня).

Величину

$$\sum_{i=0}^{80} N_S(i),$$

зависящую только от характеристики, будем называть *фактором объёма перебора* характеристики. Чем меньше фактор объёма перебора, тем характеристика лучше.

4. Подбор характеристики

Подбор характеристики состоит из трёх этапов.На первом этапе выбирается *линейная* характеристика, состоящая только из условий `-` и `x` (что эквивалентно, фиксирует все δW_i, δA_i, но не отдельные биты). Для этого рассматривается *линеаризация* хеш-функции, при которой структура функции сохраняется, но все нелинейные операции (включая сложение по модулю 2^{32}) заменяются на подходящие линейные. Идея заключается в том, чтобы найти характеристику с возможно меньшим количеством условий `x`; поскольку любая операция на совпадающих входах даёт совпадающие выходы, то хеш-функция может «разойтись» с линеаризацией только из-за отличающихся бит, соответствующих `x` в линеаризации. Линейные операции достаточно просты. Задачу о нахождении линейной характеристики с малым числом условий `x` можно сформулировать как поиск вектора малого веса в некотором линейном коде, теория кодов даёт решение этой задачи.

Мы строим коллизию, состоящую из двух блоков. Характеристика для каждого блока своя, но строится по одной и той же линейной характеристике. Хеш-значение вычисляется как $H_2 = H_1 + g(M_2, H_1) = H_0 + g(M_1, H_0) + g(M_2, H_1)$, $H_2^* = H_0 + g(M_1^*, H_0) + g(M_2^*, H_1^*)$. Линейная характеристика задаёт $g(M_1, H_0) \oplus g(M_1^*, H_0)$ и $g(M_2, H_1) \oplus g(M_2^*, H_1^*)$; поскольку она одна и та же для обоих блоков, то фиксацией значений различающихся бит можно добиться того, чтобы разность выходов сжимающей функции на первом блоке была противоположна по знаку разности выходов сжимающей функции на втором блоке. Тогда будет $H_2 = H_2^*$.

Второй этап начинается с отбрасывания всех условий линейной характеристики на первых 12 шагах на A_i и подстановки начальных условий для $A_{-4}^2, \ldots, A_0^2$. Для первого блока начальным условием является константа H_0, для второго блока — результат вычислений по первому блоку (таким образом, строить характеристику для второго блока можно только после того, как подобран первый блок коллизии). Кроме того, условия вида `xx` (на две подряд идущих пары бит) можно заменить на `-x`, если различие в старшем из этих бит может быть обусловлено переносом из младшего (это не всегда так из-за циклических сдвигов). На втором этапе следует найти какой-нибудь «путь» (согласованный набор условий) от начальных условий до линейной характеристики. Для этого будем выбирать случайные позиции в A_i^2, на которых ещё нет ограничений, накладывать ограничение `-` и вычислять, какие дополнительные ограничения будут выполнены как следствие всех текущих. Кроме того, при появлении ограничений `x` в A_i^2 (два соответствующих друг другу битов различаются) оказывается полезным дополнительно фиксировать значения различающихся битов (к примеру, требовать, чтобы бит первого сообщения был нулевым, а соответствующий бит второго сообщения был единичным). При обнаружении противоречивых условий возвращаемся к последней фиксации условия `x` и используем вторую возможную фиксацию значения бит. Второй этап заканчивается, когда все условия имеют вид одного из `-xun01`.

На третьем этапе последовательно улучшается фактор объёма перебора найденной характеристики. Для этого перебираем все позиции в характеристике, рассматриваем возможные усиления условия на позиции, вычисляем дополнительные ограничения, появляющиеся как следствие усиления, вычисляем фактор объёма перебора для новой характеристики. После окончания перебора фиксируем то из условий, для которого фактор объёма перебора был минимален.

В подборе характеристики отметим только несколько важных моментов:

- Величины F_W, P_u, P_c вычисляются последовательно от младших битов к старшим при помощи перебора элементарных условий и возможных переносов.

- Следствия от введения нового условия для одного шага вычисляются в два прохода, вначале от младших бит к старшим запоминаются возможные переносы, затем с учётом переносов вычисляются возможные следствия ограничений. Эта процедура быстрая, но находит не все возможные дополнительные ограничения. Поэтому для поиска следствий мы используем цикл по позициям бит, в котором временно фиксируем возможные значения бит и смотрим, не находит ли быстрая процедура противоречий. На третьем этапе цикл включает все биты, на втором этапе — только биты, «соседние» с изменёнными.

- Для повышения эффективности перебота по характеристике на ГПУ важна *когерентность*, т.е. одинаковость путей исполнения ядра в соседних потоках. Когерентность оказывается выше, если сильные ограничения сконцентрированы в начале характеристики. Поэтому на втором этапе мы выбираем позиции в начале с несколько меньшей вероятностью (чтобы условия - в среднем тяготели к шагам с большим номером). Подобная оптимизация даёт повышение производительности поиска на ГПУ более чем на 80%.

5. Реализация поиска на ГПУ

Перейдём к вопросам перебора по готовой характеристике. Именно он является наиболее вычислительно сложной частью задачи. Сначала отметим некоторые особенности, которые не зависят от выбора платформы реализации:

- Итоговые характеристики всегда состоят только из условий, входящих в список `-xun01`. Следовательно, набор условий на каждую пару 32-битных переменных эквивалентен $X \oplus X' = a$, $X \wedge b = c$, где a, b, c — некоторые константы. Предвычисление a, b, c позволяет быстро проверять пару переменных.

- Величины A_i на двух последних шагах не участвуют в вычислениях f_i, а потому вместо двух условий из предыдущего пункта достаточно проверять, что $X - X' = a$. Кроме того, при вычислении первого блока условия на двух последних шагах можно вообще игнорировать, поскольку любую возможную разность можно будет сократить на втором блоке без увеличения количества необходимых условий. Значения $N_S(i)$ в приводимых нами таблицах скорректированы с учётом этого.

- Эффективный перебор всех пар X таких, что $X \wedge b = c$: первый элемент множества есть $X = c$, следующий после x задаётся формулой $(((x \vee b) + 1) \wedge \bar{b}) + c$, перебор заканчивается, когда формула из-за переполнения даёт c.

- Линейные соотношения на M_i, возникающие из условий на W_k при больших k, могут либо выражать какой-то бит $[M_i]_j$ через какие-то биты предыдущих сообщений, либо давать какую-то связь между двумя или более битами $[M_i]_j$. Первый случай меняет только то, что числа a, b, c зависят не только от характеристики, но и от предыдущих сообщений. Соотношений, соответствующих второму случаю, мало, от них можно избавиться, просто наложив дополнительные искусственные ограничения, которые не меняют существенно фактор объёма перебора.

Перебор естественным образом разделяется на *генерацию*, где выполняется собственно перебор и генерация пар сообщений, и *проверку*, где характеристика проверяется для оставшихся раундов для построенной пары сообщений. Генерацию можно рассматривать как поиск с возвратом, и проверка просто добавляется как вызов функции в последний раунд

генерации. Генерация разделяется на часть, выполняемую на хосте, и часть, выполняемую на устройстве. На хосте дерево перебора раскрывается до определённой глубины, чтобы сгенерировать достаточное количество *стеков поиска* для задействования ресурсов ГПУ-параллелизма. Эта глубина сейчас задаётся вручную для каждой характеристики. Слишком маленькая глубина приведёт к недостаточному ресурсу параллелизма, а при слишком большой глубине стеки поиска просто не поместятся в память ГПУ. В результате экспериментов мы установили, что примерно 100 тыс. стеков на ГПУ достаточно для эффективного использования его ресурсов.

Во время ГПУ-части поиск по всем стекам идёт параллельно на большом количестве ГПУ. Если поиск для стека завершён, после завершения ГПУ-ядра стек удаляется. Новые стеки во время ГПУ-части не генерируются. Главное ГПУ-ядро реализует поиск с возвратом и проверку построенного сообщения. В этом ядре каждый ГПУ-поток обрабатывает свой стек поиска. Во время вызова ядра каждый поток выполняет фиксированное число итераций поиска, после чего завершает свою работу. Главное ядро также собирает статистические данные по количеству перебранных вершин, общему количеству раундов проверки, а также максимальной достигнутой при проверке глубине. Между вызовами ядра выполняется проверка на достижение необходимой глубины и сжатие стека поиска.

Вычисление на распределённом кластере реализовано при помощи MPI. Каждый MPI-процесс работает только с одним ГПУ. Используется блочно-циклическое распределение стеков поиска между процессами. Во время хост-части каждый процессор генерирует все стеки, после чего оставляет себе только те, что ему принадлежат. В первой реализации после каждого вызова ГПУ-ядра использовался глобальный барьер, в том числе для обмена статистическими данными. Однако эксперименты показали, что ввиду разбалансировки загрузки, возникающей вследствие неравномерности завершения работы над стеками, для некоторых характеристик ГПУ простаивают 50% времени. Поэтому в текущей реализации от глобального барьера мы отказались. Вместо этого мастер-процесс (с рангом 0) порождает 2 дополнительных потока, один для сбора статистики, и один для показа статистики через заданный промежуток времени (обычно 1 минута). Все процессы отсылают собственную статистику потоку сбора статистики в процессе с номером 0.

Для реализации на ГПУ использовался язык Nemerle и система NUDA (Nemerle Unified Device Architecture) [11], набор расширений этого языка для программирования ГПУ. Система NUDA была выбрана вследствие её свободного распространения и реализации высокоуровневой поддержки ГПУ. При этом низкоуровневые детали, такие как передача данных и параметров ядра, скрыты от программиста и обрабатываются самой системой. Для генерации ГПУ-кода для циклов и взаимодействия с ГПУ NUDA использует технологию OpenCL.

В нашей первой реализации поиск с возвратом был реализован как один цикл, в котором этапы, требующие специальной обработки, были реализованы в виде условий. Таких этапа было 2: переключение между раундами поиска, где требуется предвычислить большое количество значений, и проверка сгенерированного сообщения. Тестирование было реализовано в отдельной функции, а цикл по раундам тестирования полностью развёрнут при помощи аннотации `inline`, доступной в NUDA.

Однако производительность первой реализации была хороша только для некоторых характеристик, и очень плоха для других. Например, на 73-1 была достигнута эффективность 60% (от пиковой производительности ГПУ), в то время как на 72-2 достигнутая эффективность составляла лишь 15%. Низкая эффективность была вызвана низкой когерентностью между потоками одного варпа. Для повышения когерентности мы использовали две оптимизации.

Первая оптимизация состояла в сортировке стеков после каждого вызова основного ядра. Использование устойчивого алгоритма сортировки давало лучшие результаты по сравнению с неустойчивыми алгоритмами (например, быстрой сортировкой), поскольку она со-

храняла порядок значений с одинаковыми ключами, а значит, сохраняла когерентность. Мы пробовали сортировку по разным ключам, однако в конце концов остановились на значении поиска (состоянии или сообщении) на текущем раунде. На характеристике 72-2, просто устойчивая сортировка давала 45%-ый прирост производительности по сравнению с изначальной реализацией. Для реализации устойчивой сортировки на ГПУ использовался алгоритм побитовой сортировки [12], и эксперименты показали, что время сортировки стеков достаточно мало по сравнению с временем выполнения основного ядра. Сортировка сочеталась с модификацией цикла поиска сообщения: выход из цикла разрешался только при переключении раунда. Сортировка по значению поиска вместе с модификацией цикла даёт прирост производительности 87%.

Второй оптимизацией была замена одинарного цикла в поиске возвратов на тройной цикл. Самый вложенный цикл выполняет перебор на одном раунде и проверяет соответствие характеристике, пока либо не будет найдена хорошая вершина, либо не будут перебраны все вершины в этой ветке дерева на этом раунде. Второй по вложенности цикл реализует проверку сообщения, и имеет смысл только на последнем раунде перебора. Так как более 75% времени затрачивается на проверку сообщения, это повышает производительность. Наконец, внешний цикл выполняет переходы между раундами, и проверяет условие завершения ядра. Конструкция с тройным циклом, хотя и может понижать когерентность за счёт разного количества итераций внутреннего цикла у соседних потоков, ещё и препятствует расхождению соседних потоков. В целом мы обнаружили, что использование тройного цикла совместно с устойчивой сортировкой по значению поиска даёт более чем 2-кратное увеличение производительности на характеристике 75-1 по сравнению с изначальной реализацией.

Выполнялись и другие оптимизации. Это включало использование константной памяти для хранения данных характеристик, а также использование разделяемой памяти на чипе для хранения стеков поиска. К нашему удивлению, использование разделяемой памяти улучшило производительность только на 2.5% по сравнению с хранением стеков в глобальной памяти. Это показывает, что используемый алгоритм работает на каждом мультипроцессоре с небольшим множеством адресов, которое хорошо помещается в кэш 1-го уровня на ГПУ NVidia Fermi. Финальная версия, с помощью которой проводился расчёт, использует все описанные оптимизации, включая описанную ранее модификацию алгоритма нахождения характеристики. Эффект от влияния различных оптимизаций на производительность отражён в таблице 3.

Таблица 3. Влияние различных оптимизаций на произовдительность поиска на ГПУ

Оптимизация	**Ускорение**
Модификация поиска характеристики	1.8×
Устойчивая сортировка по значению поиска	1.87×
Тройной цикл	1.25×
Прочее	1.12×
Итого	**4.2**×

Поскольку по предварительным оценкам поиск второй пары блоков должен был длиться очень долго, в приложение была добавлена поддержка контрольных точек. А поскольку ожидалось, что количество доступных узлов будет меняться, реализация контрольных точек поддерживала сохранение с одним числом процессов и последующее восстановление в другое число процессов. Поскольку во время ГПУ-части нет глобальной синхронизации, каждый процесс писал файл контрольной точки независимо от остальных, через фиксированные промежутки времени, по умолчанию каждый час.

6. Результаты вычислений

Первые эксперименты и настройка параметров выполнялись на ГПУ-кластере «ГрафИТ!», установленном в НИВЦ МГУ. Эта система состоит из 16 узлов, на каждом из которых стоит по 3 ГПУ NVidia Fermi M2050 с 3 ГБ памяти на каждом. Итого это даёт 48 ГПУ, но мы проводили расчёты не более чем на 30 из них.

Финальные расчёты для 1-го и 2-го блока коллизии выполнялись на ГПУ-разделе кластера «Ломоносов», также установленного в НИВЦ МГУ. На каждом узле ГПУ-раздела установлено 2 ГПУ NVidia Fermi X2070, с 6 ГБ памяти на каждом ГПУ. Поскольку во время расчётов система находилась в режиме бета-тестирования, не все узлы были доступны для вычислений. Использованные для расчётов характеристики в данной статье для краткости опущены, однако они доступны в [3].

Объём перебора для нахождения первого блока оценивался как 2^{58} вершин. Ввиду ограничения на сообщения, характеристика была разделена на 4 части, и только одна из них выбрана для расчёта. Расчёт выполнялся на 264 ГПУ, и занял 11000 секунд. Количество обработанных вершин дерева перебора было $2^{54.06}$. Здесь в качестве «вершин» учитываются как собственно вершины дерева, так и раунды проверки сообщения, хотя один раунд проверки требует в 2.5 раз больше вычислений, чем вершина собственно перебора. Для первого блока раундов проверки было примерно 40% от общего числа «вершин».

Объём перебора для 2-го блока оценивался как $2^{63.01}$ вершин. Перебор начался на 320 ГПУ, и несколько раз перезапускался с контрольных точек ввиду сбоев на узлах или ввода в строй дополнительных узлов с ГПУ. В конце расчёт проходил на 512 ГПУ, а в среднем число использованных ГПУ составило 455. Весь расчёт длился 1904252 секунды, или примерно 22 дня 45 минут. Было обработано $2^{61.92}$ вершин, примерно 58.8% из которых составили раунды проверки. Эффективность составила 52% от пиковой производительности всех ГПУ. Если бы нам был доступен весь кластер с 1554 ГПУ, для завершения расчёта всё равно потребовалась бы целая неделя.

Для каджого блока нам в определённом смысле «повезло», т.к. поиск занял меньше времени, чем ожидалось. Для 1-го блока «повезло» в 16 раз, а для 2-го — в 2 раза. Если бы нам не «повезло», весь расчёт занял бы полтора месяца.

Если сравнивать ГПУ-реализацию с предыдущей реализацией на кластере из обычных процессоров [13], 1 ГПУ обрабатывает данные в 39 раз быстрее, чем 1 ЦПУ. Если исходить из этого, то эквивалентное количество обычных ядер для нашего расчёта было бы 17745, что не сильно превышает количество ядер, использовавшихся для получения предыдущего результата. Однако получить столько ядер на такой промежуток времени на «Ломоносове» было бы достаточно проблематично. Спрос на ГПУ был намного меньше, и их получить в требуемом объёме было сравнительно легко.

В таблице 4 приводятся данные построенной коллизии.

7. Заключение

В данной статье предложена реализация поиска коллизий для хэш-функции SHA-1 при помощи метода характеристик на кластерах из ГПУ. Основываясь на результатах предыдущей работы, а также с помощью предложенных оптимизаций удалось достигнуть эффективности расчёта 50%. При помощи разработанной реализации удалось построить коллизию для 75-раундовой версии хэш-функции SHA-1, что в настоящее время является мировым рекордом для данной хэш-функции.

Поскольку сложность расчёта с каждым следующим раундом возрастает примерно в 8 раз, дальнейшее увеличение количества используемых раундов потребует хотя бы частичного пересмотра используемого метода. Предварительные работы в этом направлении ведутся, но о результатах говорить пока рано.

Мы выражаем благодарность НИВЦ МГУ им. М.В.Ломоносова за предоставленные

Таблица 4. Пример 75-шаговой коллизии SHA-1

i	Сообщение 1, первый блок				Сообщение 1, второй блок			
1–4	F01EE8EE	BDDFF313	B2F59EE4	BB37F2BB	F072633F	0D32226A	DFF74459	98507743
5–8	2F472A36	1C052F6A	96403EF0	F144298B	EEFE63DD	FE10D5C5	AFE33902	EF74984E
9–12	DAF5519C	7A90DD71	2BF3718E	A7E3DE6D	350272F7	DB382ABC	155B0414	B800179D
13–16	EFFA975E	9B00AA95	6056E3EE	2BA4483A	18ECD4BC	15497213	1505284C	60C4F869
i	Сообщение 2, первый блок				Сообщение 2, второй блок			
1–4	001EE884	3DDFF353	22F59E94	0B37F2E8	00726355	8D32222A	4FF74429	28507710
5–8	1F472A3E	1C052F29	46403E82	4144299B	DEFE63D5	FE10D586	7FE33970	5F74985E
9–12	2AF551FE	BA90DD33	2BF371BE	47E3DE2F	C5027295	1B382AFE	155B0424	580017DF
13–16	CFFA973E	7B00AAD4	4056E3BE	EBA4487B	38ECD4DC	F5497252	3505281C	A0C4F828
i	XOR-разности в двух блоках совпадают							
1–4	F000006A	80000040	90000070	B0000053	F000006A	80000040	90000070	B0000053
5–8	30000008	00000043	D0000072	B0000010	30000008	00000043	D0000072	B0000010
9–12	F0000062	C0000042	00000030	E0000042	F0000062	C0000042	00000030	E0000042
13–16	20000060	E0000041	20000050	C0000041	20000060	E0000041	20000050	C0000041
i	Совпадающие хеш-значения							
1–5	3DF7F21E	130079F3	C2E6EFFF	FD9C4141	9AA8723A			

вычислительные ресурсы. Кроме того, мы выражаем благодарность группе поддержки суперкомпьютера «Ломоносов» и лично Антону Коржу за оперативное решение вопросов, возникающих во время использования кластера.

Литература

1. Teat C., Peltsverger S. The security of cryptographic hashes. // Proceedings of the 49th Annual Southeast Regional Conference, ACM, 2011, pp. 103–108.

2. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

3. Grechnikov E.A., Adinetz A.V. Collision for 75-step SHA-1: Intensive Parallelization with GPU // Cryptology ePrint Archive: Report 2011/641. Available online at `http://eprint.iacr.org/2011/641`.

4. Hans Dobbertin, *Cryptanalysis of MD4*, Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.

5. Xiaoyun Wang and Hongbo Yu, *How to Break MD5 and Other Hash Functions*, Eurocrypt 2005.

6. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu, *Finding Collisions in the Full SHA-1*, in Proceedings of CRYPTO, LNCS 3621, Springer, 2005, pp. 17–36.

7. Florent Chabaud, Antoine Joux, *Differential Collisions in SHA-0*, CRYPTO 1998.

8. Eli Biham, Rafi Chem, Antoine Joux, Patrick Carribault, Christophe Lemuet, William Jalby, *Collisions of SHA-0 and Reduced SHA-1*, Eurocrypt 2005.

9. Christophe De Cannière, Christian Rechberger, *Finding SHA-1 Characteristics: General Results and Applications*, In Proceedings of ASIACRYPT, LNCS 4284, pp. 1–20, Springer, 2006.

10. Christophe De Cannière, Florian Mendel, Christian Rechberger, *Collisions for 70-Step SHA-1: On the Full Cost of Collision Search*, In Proceedings of Selected Areas in Cryptography, LNCS 4876, pp. 56–73, Springer, 2007.

11. Andrew V. Adinetz. *NUDA Programmer's Guide*. URL: `http://nuda.sf.net`.

12. Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. *Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort.* In Proceedings of the 2010 international conference on Management of data (SIGMOD '10), pages 351-362. ACM, New York, NY, USA, 2010.

13. E. A. Grechnikov. *Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics.* Cryptology ePrint Archive: Report 2010/413, available at `http://eprint.iacr.org/2010/413`.

Методология распараллеливания при решении многопараметрических обратных задач химической кинетики

И.М. Губайдуллин[1], Ю.Б. Линд[2], К.Ф. Коледина[1]

Башкирский государственный университет [1], ООО «БашНИПИнефть»[2]

Разработана единая информационно-аналитическая система решения обратных задач химической кинетики (ИАС) на основе технологии параллельных вычислений. ИАС реализована для построения кинетических моделей промышленно значимых реакций металлокомплексного катализа, и с ее использованием получены следующие результаты: количественно оценена зависимость индукционного периода от количества катализатора для реакции гидроалюминирования алкенов, а также реакционная способность олефиновых и ацетиленовых соединений в реакциях циклоалюминирования олефинов; определены оптимальная температура проведения реакции циклоалюминирования олефинов и соотношения исходных реагентов для максимального выхода целевого продукта.

1. Введение

В Учреждении Российской академии наук Институте нефтехимии и катализа (ИНК) РАН работает известная научная школа по металлокомплексному катализу и металлоорганическому синтезу, созданная членом-корреспондентом РАН У.М. Джемилевым. В рамках этой школы открыты перспективы для создания нетрадиционных химических технологий получения новых материалов, катализаторов, сокатализаторов, светочувствительных покрытий для космической и радиоэлектронной промышленности. Особое внимание уделяется идентификации механизмов реакций металлокомплексного катализа [1,2].

Участвующие в реакциях металлокомплексного катализа вещества часто имеют сложную структуру и представляют собой большие макромолекулярные комплексы. Натурные эксперименты для таких процессов проводятся в несколько взаимосвязанных этапов с расщеплением на независимые частные реакции. Для полного понимания природы взаимодействия веществ, участвующих в реакциях металлокомплексного катализа, необходимо рассмотреть большое количество олефинов, ацетиленов, алленов, спиртов и т.д. Спектр температур при этом меняется от минус 65^{o}C до 150^{o}C с интервалом в 3^{o}C. Кроме того при моделировании процессов металлокомплексного катализа приходится рассматривать многочисленные варианты предполагаемых механизмов, которые включают в себя большое количество параллельных стадий, как в виде итоговых уравнений, так и в виде уравнений элементарных стадий. Параллельное изучение подобных сложных механизмов на основе натурных и вычислительных экспериментов требует обработки большого количества информации. Определение параметров кинетических моделей, варьирование входных данных при проведении вычислительных экспериментов на основе этих моделей относится к классу многопараметрических задач. На современном этапе такие задачи целесообразно решать с использованием технологии параллельных вычислений, позволяющей обрабатывать большие объемы данных и вести параллельный расчёт при решении обратных задач химической кинетики на многопроцессорных вычислительных системах для независимых между собой реакций.

2. Единая информационно-аналитическая система (ИАС) решения многопараметрических обратных задач химической кинетики (ОЗХК)

При изучении механизмов сложных химических реакций методами математического моделирования можно выделить четыре взаимосвязанных процесса (подсистемы):

1) Постоянное накопление экспериментальных и расчетных данных разного характера, требующих упорядочения, структурирования, классификации и грамотного хранения, которое приводит к необходимости разработки базы данных кинетических исследований.

2) Повышение сложности решаемых задач, влекущее за собой необходимость разработки новых способов обработки информации, таких как принципы математического моделирования, численные методы, способы алгоритмизации и непосредственно программирование.

3) Обновление технических средств обработки информации, увеличение скорости вычислений и объемов хранения данных, чему в полной мере соответствуют технологии параллельных вычислений.

4) Внедрение автоматических систем научного исследования непосредственно в химические лаборатории, в производство и обучение химика использованию современных компьютерных информационных технологий.

Все вышеперечисленные процессы являются компонентами единой информационно-аналитической системы.

Таким образом, для решения многопараметрических обратных задач химической кинетики необходимо создать единую систему, которая включала бы базу данных кинетической информации, расчет изменения концентраций компонентов реакции во времени, минимизацию функционала отклонения расчетных значений от экспериментальных данных, определение энергий активации отдельных стадий. Совокупность этих подсистем и структура связей между ними для построения кинетических моделей порождает информационно-аналитическую систему обратных задач химической кинетики [3].

Информационно-аналитическая система обратных задач химической кинетики состоит из четырех основных блоков: 1) входные информационные потоки кинетических измерений; 2) выходные информационные потоки кинетических исследований; 3) методы обработки кинетической информации; 4) технические средства обработки кинетических данных (персональный компьютер или суперкомпьютер) (рис. 1).

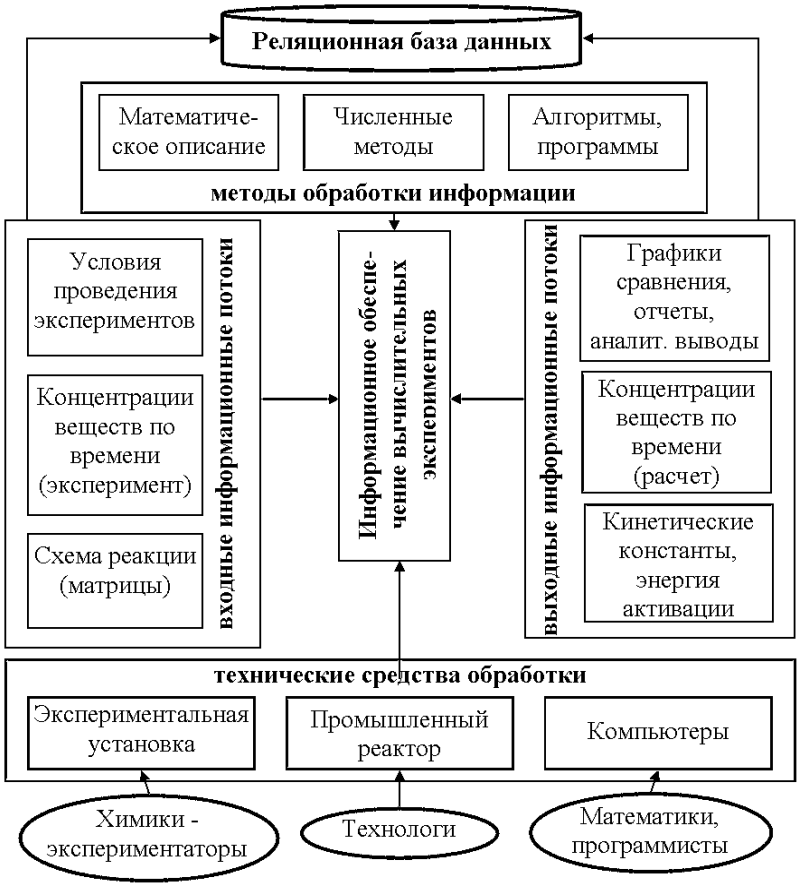


Рис. 1. Структура информационно-аналитической системы обратных задач химической кинетики

Математические методы ИАС ОЗХК делятся на две основные группы: 1) методы решения прямой кинетической задачи, т.е. расчет зависимости концентраций участвующих в реакции веществ от времени – это решение системы обыкновенных нелинейных дифференциальных и алгебраических уравнений; 2) методы решения многопараметрических обратных задач химической кинетики, т.е. нахождение глобального минимума функционала разности расчетных и экспериментальных значений концентраций веществ, наблюдаемых в ходе реакции.

Информационно-аналитическая система, в зависимости от сложности химических реакций и количества определяемых параметров, осуществляет автоматический выбор метода, как при решении прямой задачи, так и при решении обратной задачи химической кинетики (рис. 2).

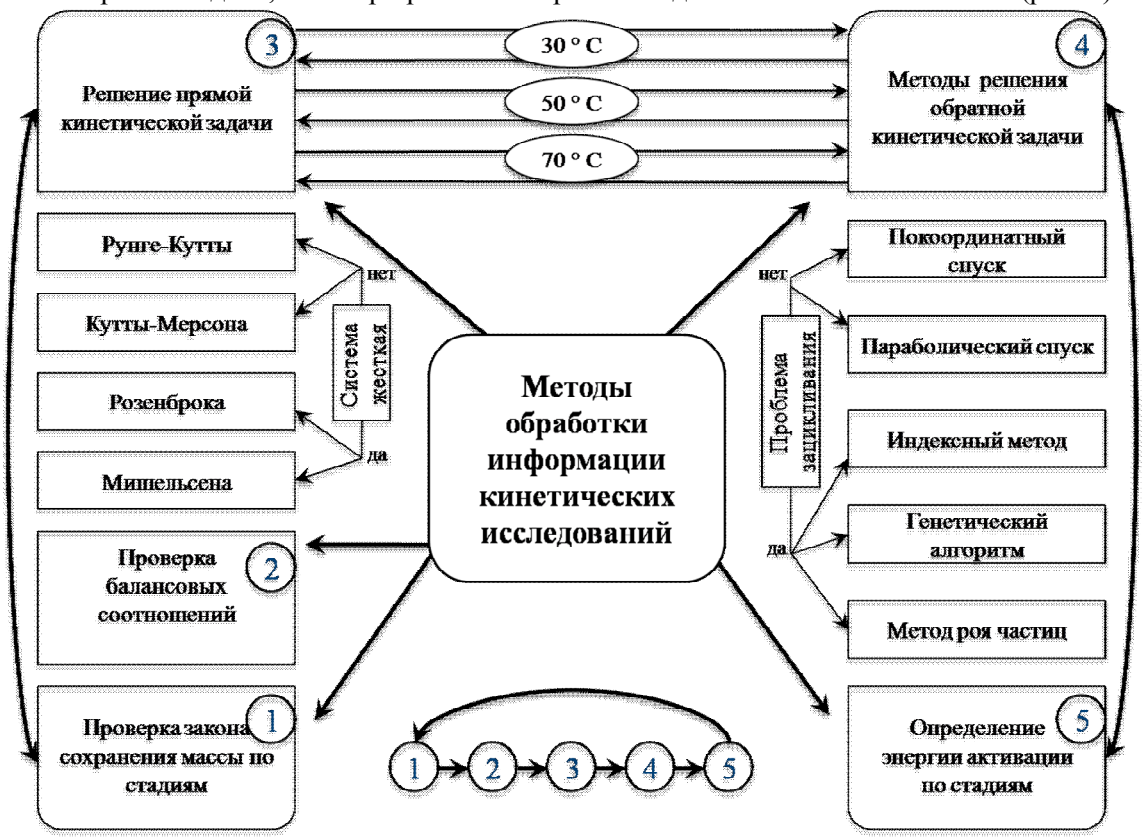


Рис. 2. Методы решения прямой и обратной кинетической задачи в зависимости от сложности химической системы

Для решения прямой задачи, в зависимости от степени ее жесткости, используются как явные, так и неявные численные методы решения систем нелинейных обыкновенных дифференциальных уравнений (ОДУ). При использовании метода Розенброка балансные соотношения для реакции выполняются с более высокой точностью, в то время как при использовании метода Мишельсена получаются более гладкие зависимости скоростей отдельных стадий от времени. При этом оба метода дают более точное решение системы дифференциальных уравнений, чем явный метод Кутты-Мерсона.

Универсального метода решения обратной задачи не существует. Ее решение чаще всего находят, перебирая по определенному алгоритму серию прямых задач и минимизируя выбранный критерий отклонения расчетных и экспериментальных данных. На современном этапе большой популярностью пользуется генетический алгоритм, основу которого составляет заимствованная из биологии идея селекции. Несмотря на то, что сходимость генетического алгоритма теоретически не обоснована, его практическое применение во всех известных случаях приводило к положительным результатам [4].

Набор используемых методов обработки взаимодействует с базой данных кинетических измерений при помощи разработанной многопользовательской системы управления базой

данных кинетических исследований (рис. 3). При этом предусмотрена параллельная организация вычислительного процесса при вводе новых данных кинетических измерений.

Рис. 3. Система управления базой данных кинетических исследований

3. Технологии параллельных вычислений при решении многопараметрической обратной задачи химической кинетики

Решение многопараметрических задач состоит в циклическом чередовании натурного и вычислительного эксперимента. Натурный эксперимент требует больших материальных и энергетических затрат, особенно для сложных реакций, и его информативность напрямую зависит от качества и скорости проведения вычислительного эксперимента. Объем данных вычислительного эксперимента зависит от следующих факторов:

– для каждой реакции необходимо обработать несколько предполагаемых схем химических превращений, чтобы выбрать лучшую схему;

– для каждой реакции проводится несколько экспериментов (обычно более 5) при разных условиях; учитывая, что эксперименты имеют погрешности, обрабатываются все и выбираются 2 (или 3) лучших, по которым расчетные значения наиболее близки к экспериментальным данным;

– каждый из кинетических параметров определяется неоднозначно вследствие недостаточной информативности эксперимента.

Если учесть, что размерность системы ОДУ (в том числе жестких) достигает 30 уравнений, то при решении обратной задачи необходимо решать более 3 миллионов системы нелинейных ОДУ (рис. 4), что определяет целесообразность использования при решении этой задачи технологии параллельных вычислений.

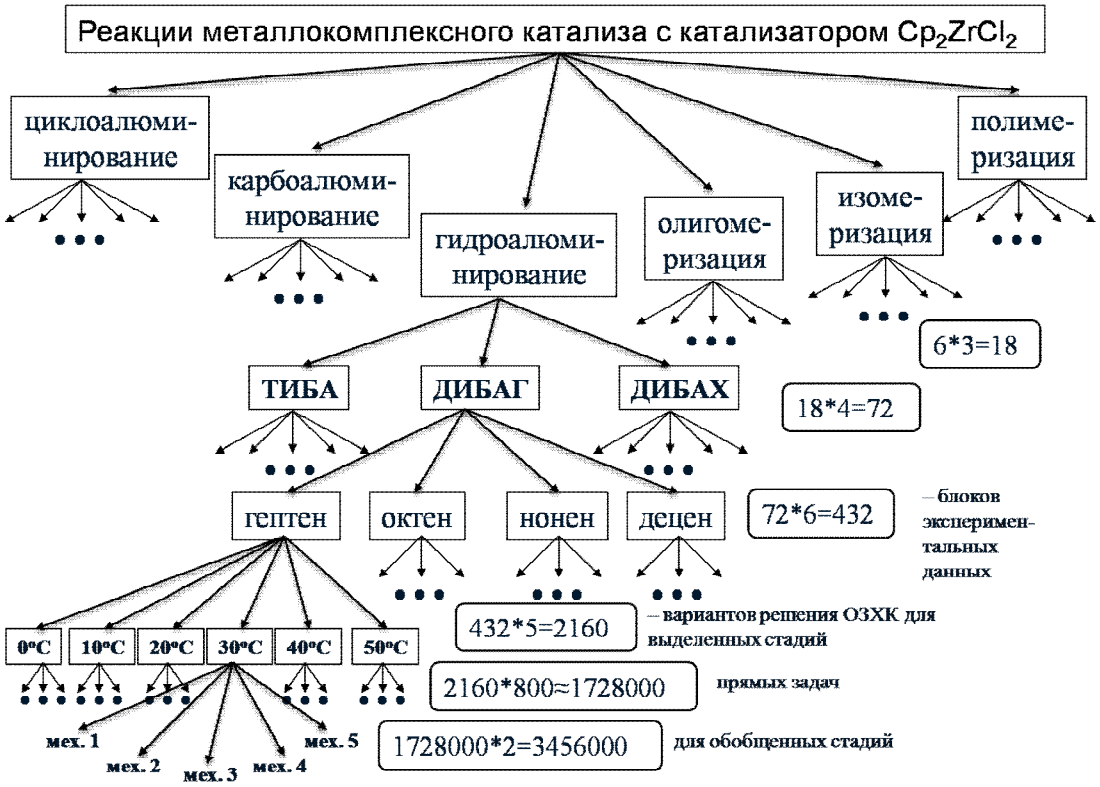


Рис. 4. Число решаемых прямых задач при построении кинетической модели реакции гидроалюминирования олефинов

Для организации вычислительного процесса предложена трехуровневая модель распараллеливания, объединяющая распараллеливание по экспериментальной базе, в соответствии с внутренним параллелизмом задачи и на основе декомпозиции метода решения обратной задачи (рис. 5) [5].

Рис. 5. Трехуровневая модель распараллеливания вычислительного процесса при решении обратной задачи

На первом уровне всё множество процессоров многопроцессорной вычислительной системы разбивается на подмножества для решения обратной задачи при конкретном наборе начальных данных. При этом организация взаимодействия с базой данных осуществляется по принципу master-slave, при котором выбирается один управляющий процессор, имеющий доступ к базе данных и выполняющий распределение данных между всеми рабочими процессорами.

Многим реальным многопараметрическим задачам, для решения которых необходимо использовать ЭВМ, свойственен естественный внутренний параллелизм, то есть возможность в

той или иной форме распараллелить действия, связанные с решением задачи на качественном уровне. Поэтому на втором уровне распараллеливания каждое подмножество процессоров относят к различным коммуникаторам (областям связи) в соответствии с внутренним параллелизмом задачи, который для рассматриваемой задачи заключается в возможности независимого решения задачи для выделенных, частных, детализированных и общих реакций. При этом некоторые кинетические параметры частных реакций используются и в обобщенных механизмах реакции. Распараллеливание на основе использования внутреннего параллелизма задачи при построении кинетической модели реакции гидроалюминирования олефинов (гексен-1, гептен-1, октен-1, нонен-1, децен-1) с алюминийорганическими соединениями изобутилаланами (HAlBui2, ClAlBui2, AlBui3) иллюстрирует рис. 6.

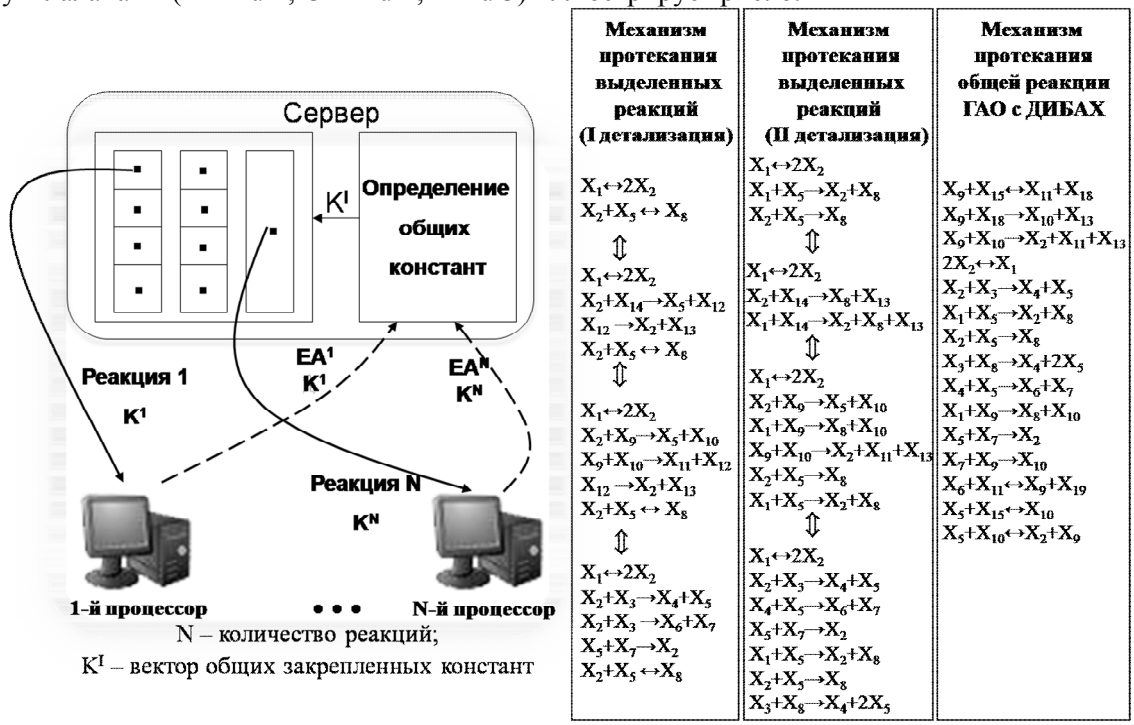


Рис. 6. Второй уровень распараллеливания решения обратной задачи химической кинетики

Третий уровень распараллеливания включает декомпозицию алгоритма решения обратной задачи по числу процессоров, входящих в созданные процессорные коммуникаторы. Для параллельного решения обратной задачи химической кинетики использован показавший наибольшую эффективность генетический алгоритм (рис. 7). Распараллеливание осуществляется на этапе начального заполнения, когда распределенные случайным образом в пространстве параметров точки, координатами которых являются значения кинетических констант, равномерно распределяются по процессорам, каждый из которых реализует дальнейшие шаги для всех доставшихся ему точек.

Рис. 7. Параллельная реализация генетического алгоритма

Обмен данными осуществляется на этапе селекции; при этом время автономной работы процессоров значительно превосходит время межпроцессорного взаимодействия, что обусловливает эффективность распараллеливания.

4. Вычислительный эксперимент: решение двухуровневых многопараметрических задач для сложных реакций металлокомплексного катализа

Разработанная ИАС ОЗХК реализована в виде программного комплекса, осуществляющего расчет кинетических параметров сложных химических реакций. ПК написан на языке С с использованием интерфейса передачи сообщений MPI. При решении обратной задачи для реакции гидроалюминирования олефинов по всем экспериментальным данным общее время расчета составило: на персональном компьютере – 360 часов, на суперкомпьютере МВС-100К Межведомственного суперкомпьютерного центра РАН (с использованием 120 процессоров) – 15 минут.

С использованием разработанного ПК построено свыше 15 кинетических моделей. Основное внимание уделялось рассмотрению двух сложных реакции металлокомплексного катализа в присутствии катализатора Cp_2ZrCl_2: реакции гидроалюминирования олефинов алюминийорганическими соединениями и реакции циклоалюминирования олефинов и ацетиленов триэтилалюминием в алюминациклопентаны и алюминациклопентены.

Последовательно-параллельное проведение циклических натурных и вычислительных экспериментов с помощью разработанного ПК позволило для общей реакции гидроалюминирования олефинов с ДИБАХ определить численные значения кинетических констант скоростей стадий для всех рассматриваемых экспериментов. На рис. 8 представлена кинетическая модель обобщенной реакции каталитического ГА олефинов с помощью ДИБАХ.

Из рис. 8 видно, что первоначально Cp_2ZrCl_2 взаимодействует с молекулой $ClAlBui_2$ с образованием $Cp_2ZrClBu$ (E=31 ккал/моль). Эта стадия обладает наибольшим активационным барьером. В начальный момент времени наблюдается низкая скорость межлигандного обмена между исходными веществами (катализатор и ДИБАХ), благодаря чему в реакции ГА олефинов с ДИБАХ наблюдается индукционный период – время, необходимое на образование комплекса (2), который является ключевым и отвечает за гидрометаллирование олефинов. При увеличении начального количества катализатора увеличивается скорость межлигандного

обмена, что ускоряет образование комплекса (2), для которого включается каталитический цикл, что приводит к уменьшению индукционного периода реакции ГА олефинов с ДИБАХ.

Реакция димеризации комплекса (2) (Е=7,02 ккал/моль) и взаимодействие его с молекулой ДИБАГ (Е=7 ккал/моль) практически равновероятны. Но для ГА олефинов с ДИБАХ ДИБАГ не является исходным веществом, а образуется в ходе реакции в малых количествах. Поэтому более вероятна димеризация комплекса (2), в результате чего образуется комплекс (1). Соединение (2) находится в мономер-димерном равновесии с циклопентадиенильным комплексом (1). Комплексы (2) и (1) могут взаимодействовать с молекулой ДИБАГ с образованием тригидридного комплекса (8).

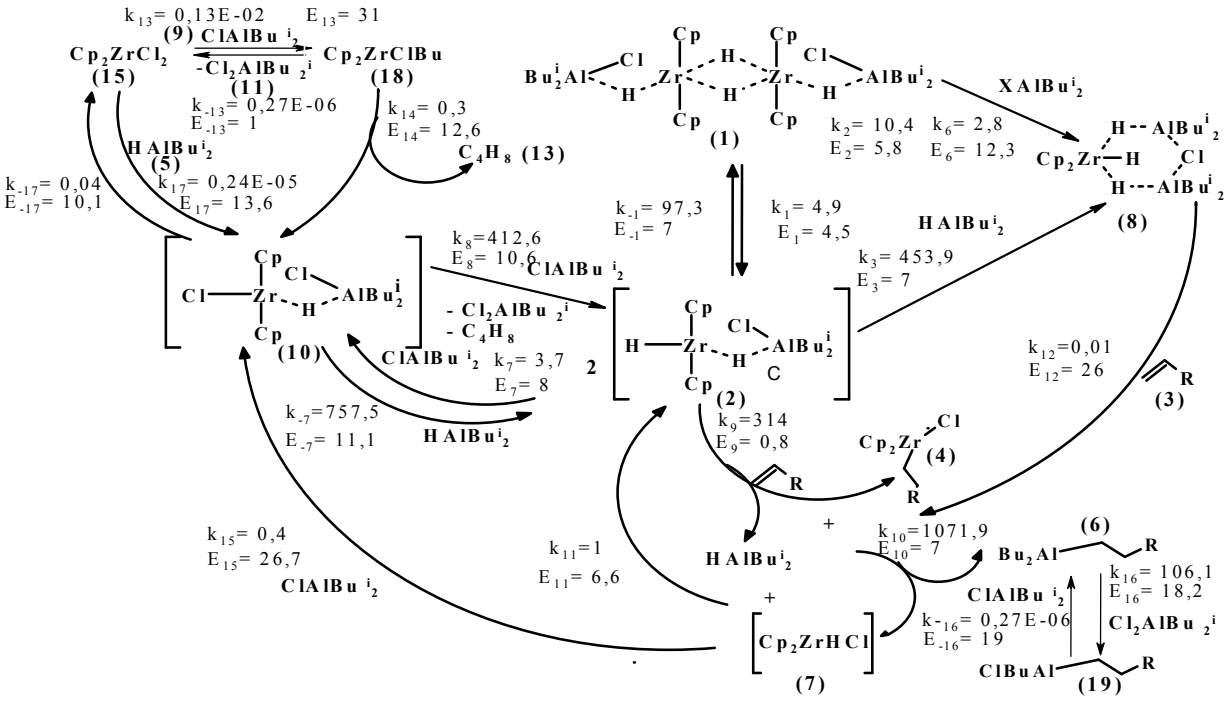


Рис. 8. Кинетическая модель обобщенной реакции каталитического гидроалюминирования алкенов ДИБАХ (октен-1, Т=20°С, Хкт = 0,18 ммоль)

Принимая во внимание активационные барьеры реакций взаимодействия комплекса (7) с ДИБАГ (Е = 6,62 ккал/моль) и ДИБАХ (Е = 26,7 ккал/моль), можно сделать вывод, что более вероятно протекание реакции Cp_2ZrHCl с $HAlBui_2$, что приводит к образованию мономера $[Cp_2ZrH_2{\cdot}ClAlBui_2]$.

Гидрометаллирующая активность комплексов уменьшается в ряду (7) (Е = 6,6 ккал/моль) > (2) (Е = 7 ккал/моль) > (8) (Е = 26 ккал/моль). Тогда в каталитическом процессе скорость будут определять эффективная концентрация и взаимное отношение компонентов (7), (2), (8).

В обобщенной реакции гидроалюминирования олефинов алкилаланами, катализируемой Cp_2ZrCl_2, кинетическая кривая расходования и накопления наблюдаемых веществ имеет s-образный вид. В начале процесса наблюдается очень медленное развитие реакции (индукционный период), которое сменяется затем периодом ускоренного развития. Контролировать процесс протекания индукционного периода важно для предупреждения взрыва в реакторе, для получения максимального выхода полезного продукта или для образования минимального количества побочного продукта. С этой целью были проведены вычислительные эксперименты для определения зависимости индукционного периода реакции от количества катализатора (рис.9).

Рис. 9. Зависимость индукционного периода (Tind)
от количества катализатора (для $0{,}09 \leq Хкт \leq 2$ ммоль)

Как видно из рис. 9, имеет место высокая параметрическая чувствительность индукционного периода реакции к начальному количеству катализатора. С увеличением количества катализатора индукционный период уменьшается, что является закономерным и подтверждает адекватность построенной кинетической модели. Количество катализатора, превышающее 2 ммоль, не оказывает существенного влияния на индукционный период реакции.

На основе построенной кинетической модели для реакции циклоалюминирования олефинов и ацетиленов триэтилалюминием в присутствие катализатора Cp_2ZrCl_2 поставлен многопараметрический вычислительный эксперимент. Путем варьирования условий проведения реакций, а также мольных соотношений исходных продуктов и катализатора исследовались реакционные способности олефинов и ацетиленов. На рис. 10 представлена зависимость времени полупревращения олефина октен-1 от температуры, из которой видно, что температура 40ºС является оптимальной для проведения реакции циклоалюминирования, так как дальнейшее повышение температуры приводит к незначительному сокращению времени протекания реакции.

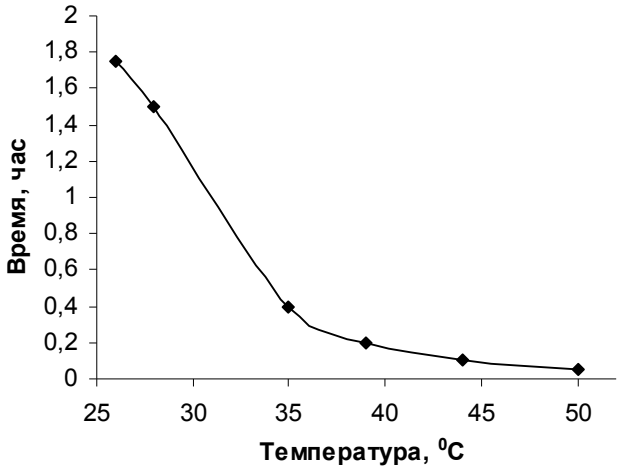


Рис. 10. Зависимость времени полупревращения олефина октен-1 от температуры

Построенная кинетическая модель была использована для изучения зависимости времени полупревращения олефина от начального количества Et_3Al и Cp_2ZrCl_2. Установлено, что оптимальное соотношение количества олефина и Et_3Al составляет 1:1, а уменьшение

количества Cp_2ZrCl_2 с 0.1 до 0.05 ммоль приводит к резкому уменьшению времени полупревращения олефина в 4 раза.

Проведенные эксперименты в лаборатории каталитического синтеза ИНК РАН подтвердили численно установленные закономерности для реакции циклоалюминирования по взаимодействию октена-1 с Et_3Al и Cp_2ZrCl_2, взятых в количествах 2.2 и 0.1 ммоль соответственно, при температуре 40°С. Полученное экспериментально время полупревращения олефина (11 минут) примерно соответствует рассчитанному значению (9 минут).

Таким образом, построенная кинетическая модель реакции циклоалюминирования олефинов и ацетиленов адекватно описывает реакционную способность октена-1 и позволяет прогнозировать выход продукта в зависимости от температуры и начального количества реагентов. На основании вычислительного эксперимента сделан вывод о том, что реакцию оптимально проводить при температуре 40°С и начальном количестве исходных веществ: Et_3Al – 2.0 ммоль; Cp_2ZrCl_2 – 0.1 ммоль; олефин – 2.0 ммоль, что было впоследствии подтверждено экспериментально.

С помощью построенной кинетической модели реакции циклоалюминирования для исследования реакционной способности различных олефинов и ацетиленов была рассмотрена зависимость скорости стадии внедрения олефина или ацетилена от времени (рис. 11).

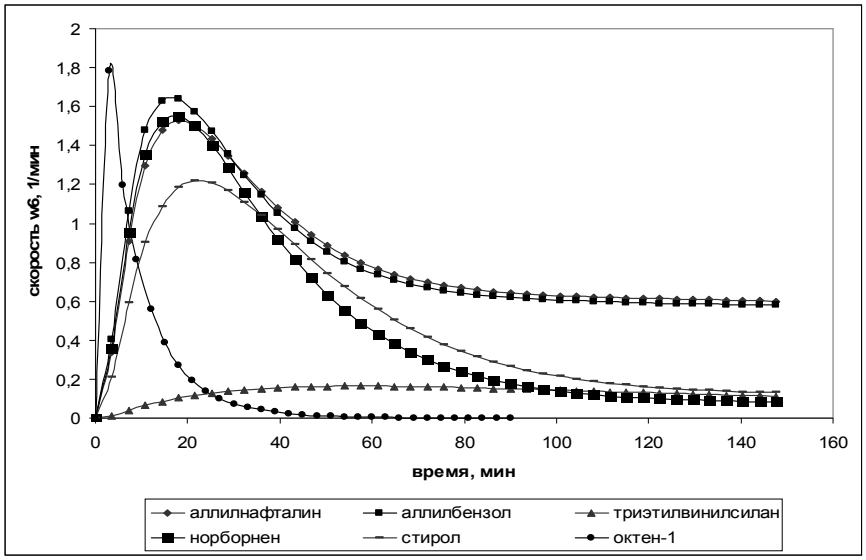


Рис. 11. Зависимость скорости стадии внедрения олефинов от времени, Т=40°С

Установлен следующий ряд активности олефинов и ацетиленовых соединений (по убыванию). Олефины: октен-1 < аллилбензол < норбонен < алилнафталин < стирол < триэтилвинилсилан. Ацетилены: октин-1 < фенилацетилен < октин-4 < триэтил(1-децинил)силан.

5. Заключение

Таким образом, для решения многопараметрических обратных задач химической кинетики разработана единая информационно-аналитическая система, которая включает в себя базу данных кинетических исследований, последовательные и параллельные алгоритмы решения прямой и обратной кинетических задач, реализованные на однопроцессорных и многопроцессорных вычислительных системах.

Решены две конкретные многопараметрические задачи химической кинетики:

1) Идентифицированы сложные механизмы металлокомплексного катализа. Построены кинетические модели, позволяющие постадийно объяснить основные пути прохождения реакций циклоалюминирования и гидроалюминирования олефинов и ацетиленов в присутствие катализатора Cp_2ZrCl_2.

2) Исследованы промышленно значимые физико-химические параметры (индукционные периоды реакции, реакционные способности отдельных соединений), определены оптимальные условия проведения реакции циклоалюминирования.

Разработанная информационно-аналитическая система позволяет разным пользователям формировать новые блоки в базе данных экспериментального материала, выбирать или добавлять новые методы обработки данных, строить математические модели исследуемых объектов разной сложности. Созданная методология позволяет формировать новые динамичные информационно-аналитические системы для смежных областей науки и техники и решать сложные многопараметрические задачи.

Литература

1. Балаев А.В., Парфенова Л.В., Халилов Л.М., Спивак С.И., Губайдуллин И.М., Джемилев У.М. Механизм реакции циклоалюминирования алкенов триэтилалюминием в алюмациклопентаны, катализируемой Cp_2ZrCl_2 // ДАН. – 2001. – Т. 381, №3. – С. 364-367.
2. Parfenova L.V., Balaev A.V., Gubaidullin I.M., Abzalilova L.R., Pechatkina S.V., Khalilov L.M., Spivak S.I., Dzhemilev U.M. Kinetic Model of Olefins Hydrometallation by HAlBui2 and Al Bui3 in the Presence Cp_2ZrCl_2 Catalyst // Int. J. Chem. Kinet. – 2007. – V. 39, № 6. – P. 333-339.
3. Губайдуллин И.М., Спивак С.И. Информационно-аналитическая система обратных задач химической кинетики // Системы управления и информационные технологии. – 2008. – №1.1(31). – С. 150-153.
4. Губайдуллин И.М., Рябов В.В., Тихонова М.В. Применение индексного метода глобальной оптимизации при решении обратных задач химической кинетики // Вычислительные методы и программирование. – 2011. – Т. 12. – С. 137-145.
5. Линд Ю.Б., Губайдуллин И.М., Мулюков Р.А. Методология параллельных вычислений для решения задач химической кинетики и буровой технологии // Системы управления и информационные технологии. – 2009. – №2(36). – С. 44-50.

Экспертная методология анализа производительности взаимодействия процессов в MPI приложениях

А.В. Дергунов[1]

Нижегородский государственный университет им. Н.И. Лобачевского[1]

Для анализа MPI программ часто используют программные системы, которые осуществляют сбор и визуализацию трассы выполнения программы на кластере. Но при использовании таких инструментов пользователь сталкивается с проблемой анализа больших объемов информации. Поэтому возникает потребность в средствах для автоматизации анализа трассы, которые подсказали бы пользователю, как повысить производительность его программы. Задача повышения производительности является достаточно сложной и вряд ли может быть решена формальными методами. В данной работе предлагается экспертная методология решения поставленной задачи.

1. Введение

Для анализа MPI программ наиболее часто используют программные системы, которые осуществляют сбор и визуализацию трассы выполнения программы. Примером такого средства является Jumpshot [1]. На рис. 1 показано окно временной диаграммы системы Jumpshot, показывающее трассу работы MPI программы, реализующей сеточные вычисления при работе на 32 процессорах в течение 9 секунд (см. описание этой программы далее в разделе «Эксперимент по повышению производительности одной MPI программы»).

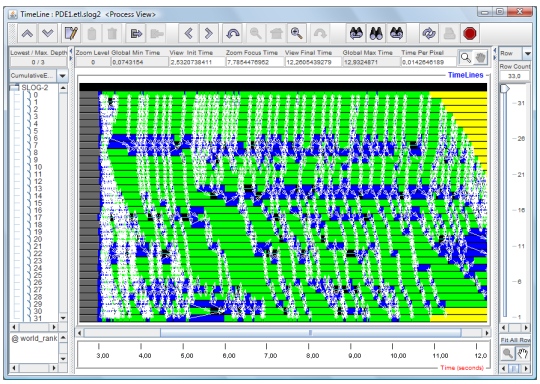


Рис. 1. Окно временной диаграммы системы Jumpshot для трассы MPI программы, реализующей сеточные вычисления при работе на 32 процессорах в течение 9 секунд

Но при использовании таких инструментов пользователь сталкивается с проблемой анализа больших объемов информации. Пользователь должен использовать инструменты зуммирования и фильтрации, чтобы исследовать события трассы. Эта проблема особенно актуальная при анализе трассы с большим количеством процессов, собранных на больших суперкомпьютерах.

Другой проблемой при использовании средств визуализации трассы является то, что часто встречающиеся ситуации, приводящие к потерям производительности MPI программ, явно не визуализируются. Таким образом, пользователь должен опираться на свой опыт работы с MPI и детально исследовать трассу программы для выявления причин недостаточной производительности.

Одной из причин недостаточной производительности является плохая синхронизация приемов и передач данных в MPI программе. В результате процедура приема сообщения может простаивать, дожидаясь посылки сообщения (см. рис. 2). Для решения этой проблемы нужно обеспечить, чтобы сообщения посылались как можно раньше, и т.о. повысить вероятность того, что сообщение прибудет до момента, когда оно потребуется другому процессу.

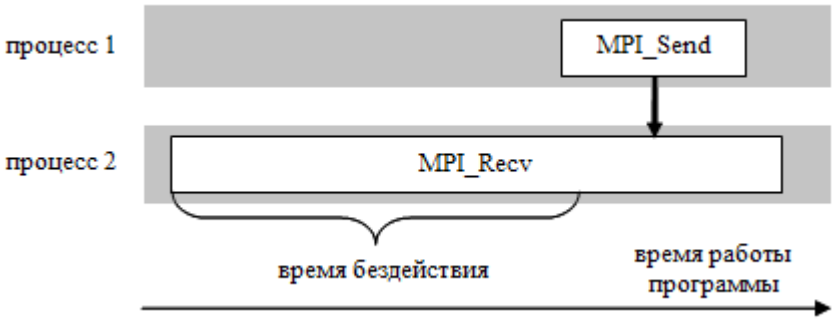


Рис. 2. Поздняя посылка сообщения

Но даже если пользователь обнаружил причину падения производительности, то оказывается невозможным оценить, насколько она влияет на общее время работы программы. Например, чтобы оценить, в какой степени поздняя посылка данных замедлила работу программу, не достаточно просто обратиться к суммарному времени, затраченному на вызов процедур MPI_Recv, т. к. простой при ожидании данных составляет лишь долю времени работы этой процедуры. Поэтому нет возможности оценить, какой выигрыш производительности получит пользователь, если соответствующим образом изменит свою программу.

В этой работе рассматривается созданная автором система Performance Expert, основанная на экспертной методологии, которая позволяет автоматизировать анализ трасс MPI программ.

2. Модели представления знаний для анализа производительности взаимодействия процессов в MPI приложениях

2.1 Модель MPI приложения

MPI приложение рассматривается как множество взаимодействующих процессов $PR = \{PR_1, PR_2, ...PR_N\}$. Взаимодействие осуществляется с помощью действий, инициируемых процессами в определенной последовательности:

$$PR_i \Rightarrow A_i = \langle a_{ij} \rangle; i = 1,2,...,N; j = 1,2,...,M_i$$

Каждое действие процесса состоит в вызове функций библиотеки MPI $F = \{f_k\}, k = 1,2,...,K$. Функция характеризуется вектором входных и выходных аргументов:

$$f_k \Rightarrow FA_k^{IN} = (fa_{k1}^{IN},...,fa_{kL_k^{IN}}^{IN}); k = 1,...,K$$

$$f_k \Rightarrow FA_k^{OUT} = (fa_{k1}^{OUT},...,fa_{kL_k^{OUT}}^{OUT}); k = 1,...,K$$

Таким образом, каждое *действие процессов* характеризуется функцией библиотеки MPI и значениями ее входных и выходных аргументов:

$$a_{ij} = \langle f_k, FAval_k^{IN}, FAval_k^{OUT} \rangle; i = 1,2,...,N; j = 1,2,...,M_i; f_k \in F$$

$$FAval_k^{IN} = \left(av_{k1}^{IN},...,av_{kL_k^{IN}}^{IN}\right); FAval_k^{OUT} = \left(av_{k1}^{OUT},...,av_{kL_k^{OUT}}^{OUT}\right);$$

Существует несколько моделей обмена сообщениями, которые определяются синтаксисом и семантикой функций F.

2.2 Повышение производительности MPI приложения

Каждому действию a_{ij} процесса соответствует время начала его выполнения t_{ij}^a и длительность d_{ij}^a. *Потеря производительности взаимодействия процессов* определяется как сумма длительности выполнения действий всех процессов:

$$D^a = \sum_{i=1}^{N} \sum_{j=1}^{M_i} d_{ij}^a$$

Длительность выполнения действий состоит из следующих компонентов:

$$d_{ij}^{a} = (d_{ij}^{a})_{prep} + (d_{ij}^{a})_{wait} + (d_{ij}^{a})_{comm}$$

где:

$(d_{ij}^{a})_{prep}$ – длительность подготовки к передаче/приему сообщений;

$(d_{ij}^{a})_{wait}$ – длительность ожидания готовности других процессов к передаче/приему сообщений;

$(d_{ij}^{a})_{comm}$ – длительность передачи/приема сообщений по сети.

Способами сокращения длительности выполнений действий являются:

- Сокращение длительности $(d_{ij}^{a})_{prep}$: группировка отдельных сообщений в одно большое для сокращения подготовки данных для отправки.
- Сокращение времени $(d_{ij}^{a})_{wait}$: улучшение синхронизации приемов/передач сообщений, балансировка нагрузки процессов, совмещение обменов сообщениями и вычислений.
- Сокращение времени $(d_{ij}^{a})_{comm}$: сокращение количества пересылаемых сообщений, учет топологии сети при передаче сообщений.

Приведенные способы сокращения времен выполнения действий можно представить в виде *рекомендаций* $REC = \{rec_i\}, i = 1,2,...,R$. Повышение производительности взаимодействия процессов определяется как:

$$D^{a} \Rightarrow \min_{REC}$$

Сформулированная задача повышения производительности является достаточно сложной и вряд ли может быть решена формальными методами. Потому в работе предлагается экспертная методология решения поставленной задачи, включающая выполнение следующих основных этапов:

1. Выявление и систематизация проблем производительности – типовых (повторяющихся) фактов потери производительности.
2. Установление причин потери производительности для выявленных проблем.
3. Разработка рекомендаций по устранению этих причин.
4. Описание выявленных проблем производительности, правил установления причин их возникновения и рекомендаций по их устранению.

Первые три этапа выполняются экспертом на основе своего опыта повышения производительности и знаний о принципах работы библиотеки MPI. Четвертый – с помощью разработанной системы.

2.3 Модель проблемы производительности

В работе применяется следующая модель *проблемы производительности* MPI приложения:

$$pb = \langle pd, dur, TRRULES, ANRULES, REC(A^{INFO}) \rangle$$

где:

pd – вербальное описание проблемы;

dur – длительность проявления данной проблемы;

$TRRULES$ – правила трассировки действий проблемы;

$ANRULES$ – правила распознавания проблемы;

REC – рекомендации по ее устранению;

$AINFO = \{\langle f_i, t_i, d_i, pr_i, cs_i \rangle\}$ – описание действий, приведших к проблеме (где f_i – функция действия, t_i – время вызова функции, d_i – длительность работы функции, pr_i – процесс, который инициировал вызов функции, cs_i – место в исходном коде, из которого осуществлялся вызов).

2.4 Схема работы системы

Схема работы системы Performance Expert представлена на рис. 3 и состоит из двух этапов: подготовительного (этапа обучения) и этапа применения.

Подготовительный этап выполняется экспертом после выявления очередной проблемы производительности. На этом этапе:

1. Эксперт описывает правила трассировки действий процессов, которые (или сочетания которых) могут привести к возникновению выявленной проблемы.
2. Эксперт описывает правила распознавания выявленной проблемы.

На этапе применения:

1. Трассировщик запускается вместе с MPI приложением и формирует трассы, в которой в виде последовательности событий регистрируются все необходимые для последующего анализа действия процессов приложения.
2. После выполнения приложения анализатор считывает события трассы и выполняет правила распознавания проблемы. Результатом анализа является список проблем и рекомендаций.
3. При необходимости пользователь вносит изменения в MPI приложение и повторяет этап применения.

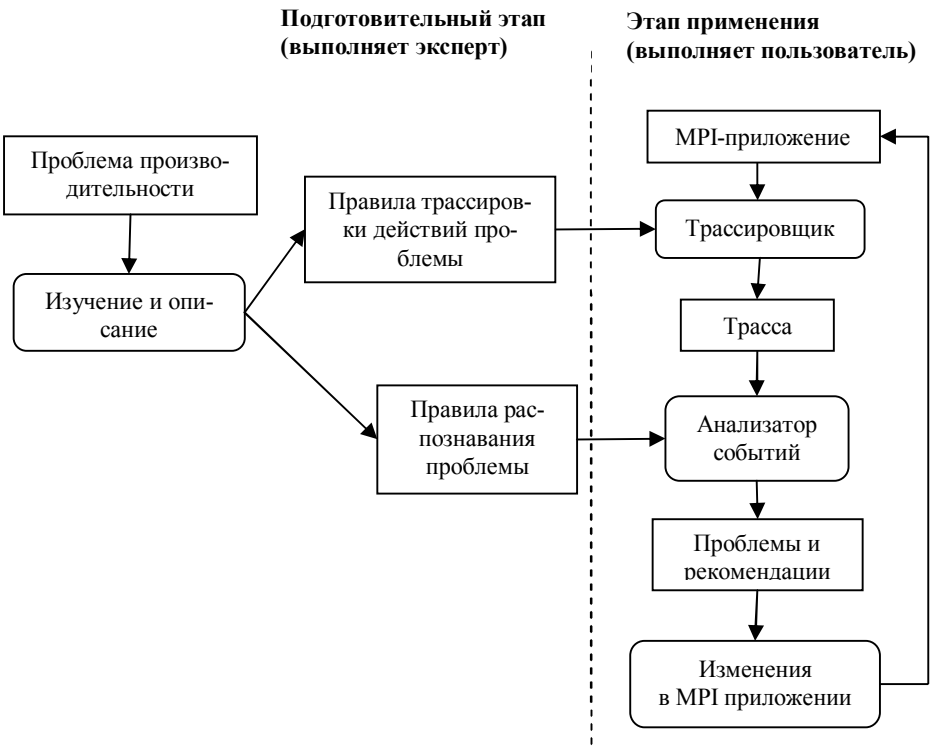


Рис. 3. Схема работы системы

2.5 Трассировка выполнения MPI приложения

Обучение трассировщика выполняется на основе правил трассировки действий проблемы производительности, выполнение которых может привести к ее возникновению. Результатом работы трассировщика должна быть трасса работы процесса приложения, в которой зарегистрированы простые события, вызванные действиями процессов.

Простое событие представляется как:

$$e = \langle f, et, EPval, t, d, pr, cs \rangle$$

где:

f – функция действия;

et – тип события, который задает набор параметров события $EP = (ep_1,...,ep_K)$;

$EPval = (v_1,...,v_K)$ – значения параметров события;

t – время наступления события;

d – длительность события;

pr – процесс, на котором произошло событие;

cs – место в исходном коде, из которого осуществлялся вызов функции.

Правило трассировки представляется как:

$$a = \langle f, FAval^{IN}, FAval^{OUT} \rangle \xrightarrow{trrule} e = \langle f, et, EPval, t, d, pr, cs \rangle$$

$$trrule = \langle f, et, EPtemp \rangle$$

где:

$f \in F$ – функция, при выполнении которой трассировщиком создается событие, определяемое правилом;

et – тип события, задаваемый правилом трассировки;

$EPtemp = \{\langle ep_i, Expr_i, kind_i \rangle\}; i = 1,...,N$ – описание параметров события и способа получения их значений, где:

ep_i – параметр события;

$Expr_i : FAval^{IN} \times FAval^{OUT} \to v_i$ – функция для вычисления значения v_i параметра на основе аргументов вызванной функции;

$kind_i \in \{in, out\}$ – характер параметра (*in* – входной, т.е. вычисляемый только на основе входных аргументов функции; *out* – выходной, т.е. формируемый на основе выходных аргументов функции или входных и выходных).

Значения параметров t, d, pr и cs события формируются трассировщиком.

В рамках системы разработан язык для описания правил трассировки. На основе правил генерируются функции-обертки для трассируемых функций. Трассировщик основан на принципе динамического инструментирования [2]. При работе программы он перехватывает вызовы трассируемых функций и вместо них вызывает функции-обертки. Функции-обертки регистрируют соответствующее событие в трассе и вызывают оригинальную функцию.

2.6 Анализ событий трассы MPI приложения

После выполнения трассировки выполняется анализ полученной трассы на предмет выявления проблем производительности. Исходными данными для анализа является трасса MPI приложения, сформированная в результате трассировки и содержащая простые события, а результатом анализа – множество выявленных проблем производительности. Анализ осуществляется на основе правил распознавания проблем производительности *ANRULES* и осуществляется в два этапа:

1. Конструирование составных событий как претендентов на отдельные типы проблем производительности.
2. Выявление проблем производительности из множества сконструированных составных событий.

Составное событие представляется как:

$$ce = \langle cet, CEPval, ME \rangle$$

где:

cet – тип составного события, который задает набор параметров события $CEP = (cep_1,...,cep_K)$;

$CEPval = (cev_1,...,cev_K)$ – значения параметров составного события;

$ME = \{e_i\} \cup \{ce_j\}; i = 1,...,N; j = 1,...,M$ – множество простых и составных событий-членов.

Правило конструирования составного события представляется как:

$$\left.\begin{array}{l} E \xrightarrow{ET} E^R \\ CE \xrightarrow{CET} CE^R \end{array}\right\} \xrightarrow{cerule} ce = \langle cet, CEPval, ME \rangle$$

$$cerule = \langle ET, CET, \sigma, cet, CEPtemp, ET^S \rangle$$

где:

$ET = \{et_i\}; i = 1,...,N$ – множество типов простых событий для выборки из множества простых событий трассы E подмножества событий $E^R = \{\langle f_i, et_i, EPval_i, t_i, d_i, pr_i, cs_i \rangle\} \subseteq E; i = 1,...,N$, релевантных анализируемой проблеме производительности;

$CET = \{cet_j\}; j = 1,...,M$ – множество типов составных событий для выборки из множества составных событий CE подмножества релевантных событий $CE^R = \{\langle cet_j, CEPval_j, ME_j \rangle\} \subseteq CE; j = 1,...,M$ указанных типов;

Введем обозначения:

$$EPV_i = \langle f_i, EPval_i, t_i, d_i, pr_i, cs_i \rangle; i = 1,...,N$$

$$P = EPV_1 \times ... \times EPV_N \times CEPval_1 \times ... \times CEPval_M$$

$\sigma : P \to \{0,1\}$ – условие конструирования составного события (если значение функции $\sigma = 1$, то составное событие может быть сконструировано для множества релевантных событий E^R и CE^R и нет в противном случае);

cet – тип составного события, конструируемый правилом;

$CEPtemp = \{\langle cep_k, Expr_k \rangle\}; k = 1,...,K$ – описание параметров конструируемого события и способов получения их значений, где:

cep_k – параметр составного события;

$Expr_k : P \to cev_k$ – функция для вычисления значения cev_i параметра составного события;

При выполнении условий правила (т.е. существовании подмножества релевантных событий E^R и CE^R и выполнении условия σ) конструируется событие ce указанного типа cet, где:

$CEPval = (cev_1,...,cev_K)$ – значения параметров событий, вычисляемые с помощью описания параметров $CEPtemp$;

$ME = E^R \cup CE^R$ – множество событий-членов;

Параметр правила $ET^S \subseteq ET$ указывает типы событий, которые являются общими для конструируемого события и других составных событий. Остальные простые события, релевантные данному правилу, принадлежат только конструируемому составному событию.

Правила конструирования составных событий задаются в виде последовательности деклараций, имеющих синтаксис:

```
declare composite event <имя типа составного события>
  member(
    [<shared>] <тип события-члена> as <имя события>,
    ...
  )
  when
    <логическое выражение>
  parameters(
    <параметр события> = <выражение для вычисления значения>,
```

```
    ...
);
```

Рис. 4. Синтаксис правил конструирования составных событий

где:

`declare composite event` описание правила конструирования составного события типа cet;

`member` список типов простых ET и составных CET событий для выборки подмножества релевантных событий; флаг *shared* указывается подмножества типов $ET^S \subseteq ET$; имя события используется для ссылки на его параметры в логическом выражении и выражениях для вычисления значений параметров события;

`when` условие конструирования составного события σ;

`parameters` описание параметров конструируемого события и способов получения их значений $CEPtemp$ (с указанием параметров cep_k и выражений для вычисления их значений $Expr_k$);

Правило выявления проблемы производительности представляется как:

$$ce = \langle cet, CEPval, ME \rangle \xrightarrow{pbrule} pb = \langle pd, dur, REC(A^{INFO}) \rangle$$

$$pbrule = \langle cet, \varphi, pd, RECtemp, L_{dur} \rangle$$

где:

cet – тип составного события как претендента на определяемую проблему производительности;

$\varphi : CEPval \to \{1,0\}$ – условие возникновения проблемы (если значение функции $\varphi = 1$, то проблема существует и нет в противном случае);

pd – вербальное описание проблемы, определяемой правилом;

$RECtemp : CEPval \to REC$ – шаблон рекомендации по устранению проблемы;

$L_{dur} : CEPval \to dur$ – функция вычисления длительности проблемы производительности;

При выполнении условий правила (т.е. выполнении для составного события ce типа cet условия φ) делается вывод о наличии проблемы производительности pb. Описание действий приложения, приведших к этой проблеме, формируется на основе функции интерпретатора $Frec$, т.е. $A^{INFO} = Frec(ME)$. Функция $Frec$ осуществляет рекурсивное извлечение простых событий из ME и формирования на их основе информации о множестве действий.

Правила выявления проблемы производительности задаются в виде последовательности деклараций, имеющих синтаксис:

```
declare problem for <имя типа составного события>
  when
    <логическое выражение>
  parameters(
    name = "<название проблемы>",
    description = "<описание проблемы>",
    advice = <шаблон рекомендации>,
    duration = <выражение для вычисления длительности>);
```

Рис. 5. Синтаксис правил выявления проблемы производительности

где:

`declare problem for` описание правила выявления проблемы производительности для составного события-претендента типа cet;

`when` условие возникновения проблемы φ;

`name` название проблемы;

`description` вербальное описание проблемы pd;

`advice` шаблон рекомендации $RECtemp$;

`duration` функция вычисления длительности L_{dur};

3. Состав базы знаний проблем производительности MPI приложений

В состав базы знаний системы Performance Expert были включены описания следующих типовых проблем производительности MPI приложений:

- Проблемы двухточечных операций обмена:
  - поздняя посылка сообщения;
  - поздний прием сообщения;
- Проблемы коллективных операций обмена:
  - задержка перед барьерной синхронизацией;
  - ранний прием данных при операции «от многих к одному»;
  - поздняя посылка данных при операции «от одного ко многим»;
  - задержка перед операцией «от многих ко многим»;
- Проблемы операций удаленного доступа к памяти:
  - задержка при создании «окна» для удаленного доступа к памяти;
  - конкуренция за блокировку «окна» для удаленного доступа к памяти;
  - позднее начало периода предоставления доступа к «окну»;
  - раннее завершение периода предоставления доступа к «окну».

На рис. 6 продемонстрировано правило для выявления поздней посылки сообщения, о которой говорилось в введении (рис. 2). Это правило срабатывает для составного события двухточечного обмена (типа point_to_point), для которого выполнено условие, указанное в when. При выполнении правила анализатор делает вывод об указанной проблеме производительности.

```
declare problem for point_to_point
  when
    send_start_time > recv_wait_start_time
  parameters(
    name = "Поздняя посылка сообщения",
    description = "Функция посылки сообщения вызывается
      позднее функции приема. Из-за этого блокирующая
      функция приема вынуждена простаивать.",
    advice = "Улучшить синхронизацию приемов и передач
      сообщений, оправляя сообщения по
      возможности раньше, или
      использовать неблокирующие функции приема.",
    duration = send_start_time - recv_wait_start_time);
```

Рис. 6. Правило для выявления поздней посылки сообщения

4. Эксперимент по повышению производительности одного MPI приложения

Для проверки разработанной системы было осуществлено повышение производительности MPI программы, реализующей сеточные вычисления, а именно метод итераций Якоби для численного решения задачи Дирихле для уравнения Лапласа [4]. Уравнение Лапласа представлено ниже:

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0$$

В задаче заданы значения на граничных точках двумерной сетки. Необходимо вычислить значения во внутренних точках. Метод итераций Якоби [4] предусматривает несколько итераций, в которых новые значения в точках вычисляются как среднее из значений четырех ее соседних точек, вычисленных на предыдущей итерации:

```
for (int iter = 0; iter < iterations; iter++)
{
  for (int i = 0; i < height; i++)
    for (int j = 0; j < width; j++)
      new[i][j] = (grid[i-1][j] + grid[i][j-1] +
                   grid[i+1][j] + grid[i][j+1]) / 4;
  copy(grid, new);
}
```

Рис. 7. Метод итераций Якоби

Для реализации метода итераций Якоби на компьютере с распределен ной памятью удобно избавиться от операции копирования матрицы new в grid, продублировав предыдущий цикл и поменяв в нем new и grid местами, а затем каждому процессу необходимо назначить свою прямоугольную полосу. Для вычисления значений на краях нужно использовать данные, размещенные на других процессорах. Поэтому необходимо использовать соответствующие операции обмена MPI. Таким образом, каждый процесс выполняет следующую последовательность действий:

1. вычислить значения во внутренних точках своей полосы;
2. отправить соседям вычисленные значения на краях своей полосы;
3. получить от соседей значения на краях их полос.

В рамках эксперимента MPI программа, реализующая данный алгоритм, была запущена на кластере с 64 процессами и собрана трасса ее выполнения. В результате анализа производительности MPI программы, реализующей описанный алгоритм, с помощью системы Performance Expert были выявлены две проблемы производительности, по которым получены следующие рекомендации по их устранению:

```
Название проблемы: Поздняя посылка сообщения
Описание проблемы: Функция посылки сообщения вызывается позднее
функции приема. Из-за этого блокирующая функция приема вынужде-
на простаивать.
Длительность: 57,96%
Совет: Улучшить синхронизацию приемов и передач сообщений, оп-
равляя сообщения по возможности раньше, или использовать небло-
кирующие функции приема.
Действия:
•     вызов MPI_Send из calculate
•     вызов MPI_Recv из calculate

Название проблемы: Поздний прием сообщения
Описание проблемы: Функция приема сообщения вызывается позднее
функции посылки. Из-за этого блокирующая функция посылки вынуж-
дена простаивать, т.к. ожидает начало приема.
Длительность: 2,83%
Совет: Улучшить синхронизацию приемов и передач сообщений или
использовать неблокирующие функции приема.
Действия:
•     вызов MPI_Send из calculate
•     вызов MPI_Recv из calculate
```

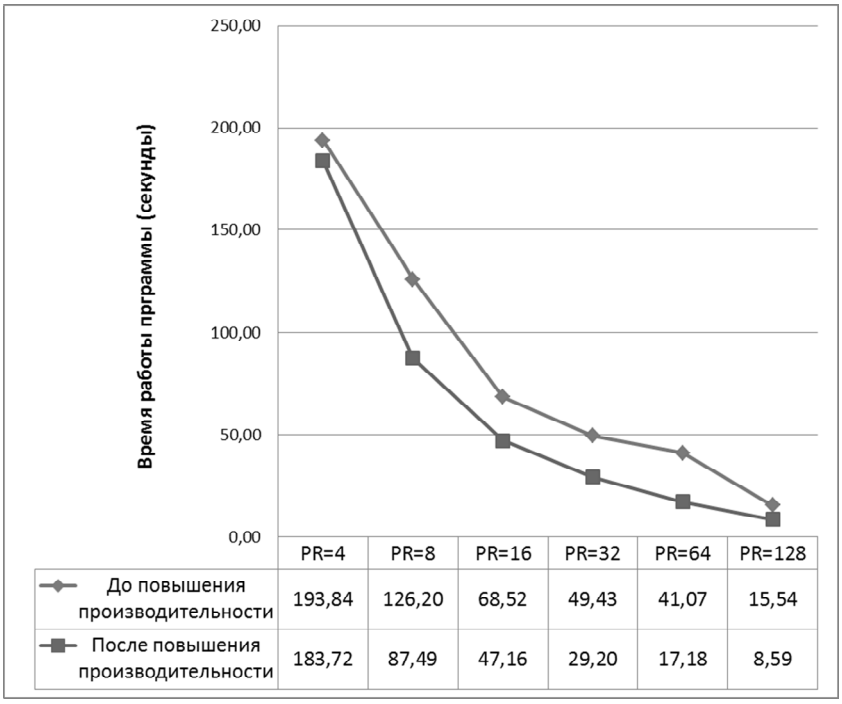
Рис. 8. Рекомендации по устранению проблем производительности

Наибольшую длительность составляет проблема поздней посылки сообщения, поэтому было принято решение о ее устранении. Одним из предложенных советов по устранению этой проблемы было «улучшить синхронизацию приемов и передач данных, отправляя данные по возможности раньше». В этом случае между отправкой и получением сообщений будут выполняться локальные вычисления. Принимая во внимание предложенный совет, исходная программа была изменена так, чтобы каждый процесс выполнял следующую последовательность действий:

1. отправить соседям значения на краях своей полосы;
2. вычислить значения во внутренних точках своей полосы;

3. получить от соседей значения на краях их полос;
4. вычислить значения на краях своей полосы.

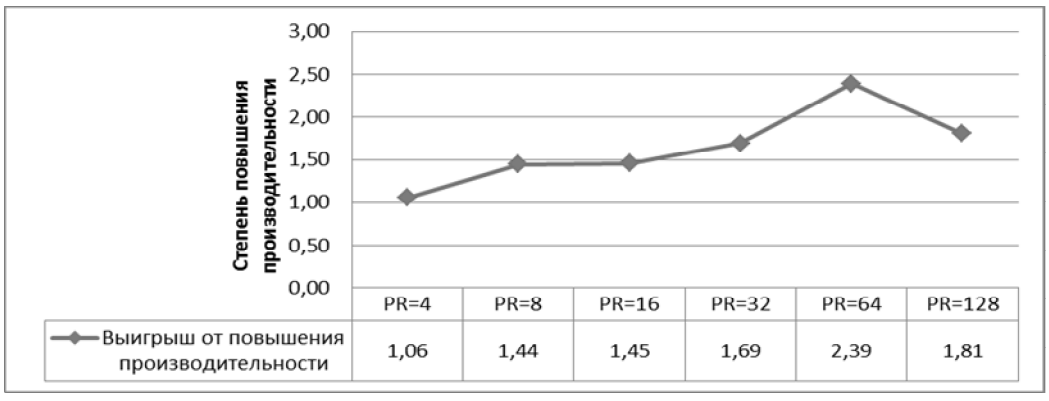
На рис. 9 представлено сравнение времени работы двух версий программы при использовании различного количества процессов *PR*. В расчетах использовалась сетка размером 6400x6400, было выполнено 1000 итераций. Для каждого случая время работы программы улучшилось.



	PR=4	PR=8	PR=16	PR=32	PR=64	PR=128
До повышения производительности	193,84	126,20	68,52	49,43	41,07	15,54
После повышения производительности	183,72	87,49	47,16	29,20	17,18	8,59

Рис. 9. Результат повышения производительности MPI программы

На рис. 10 представлен выигрыш от повышения производительности программы. Наибольший выигрыш достигнут при использовании 64 процессов: время работы программы сократилось в 2,39 раза.



	PR=4	PR=8	PR=16	PR=32	PR=64	PR=128
Выигрыш от повышения производительности	1,06	1,44	1,45	1,69	2,39	1,81

Рис. 10. Выигрыш от повышения производительности MPI программы

Измерения производительности проводились на кластере Нижегородского государственного университета им. Н.И. Лобачевского. Параметры кластера следующие: узлы с двумя двухъядерными процессорами Intel Xeon 5150 2.66 GHz, 4 GB оперативной памяти, сеть Gigabit Ethernet, ОС Windows Server 2008 x64. Запуски программ производились с разным количеством узлов.

5. Заключение

Рассмотрена программная система, которая значительно облегчает задачу анализа производительности взаимодействия процессов в MPI приложений. Для выполнения анализа производительности предложена экспертная методология, базирующаяся на модели проблемы производительности MPI приложения как совокупности действий MPI процессов, определенные сочетания которых при определенных условиях могут привести к возникновению идентифицируемой проблемы производительности, а также на модели правил трассировки действий проблемы и модели правил распознавания проблемы и соответствующих языках.

Разработанная система основана на ином подходе к анализу производительности по сравнению системами для визуализации трассы работы MPI программы (например, Jumpshot [1]). Наиболее близким аналогом является система KOJAK [5]. Но в отличии от разработанной системы Performance Expert, в системе KOJAK описания проблем производительности и способов их устранения фиксированы. В результате эксперту для описания проблем производительности необходимо изменять исходный код системы KOJAK.

В работе приведен список типовых проблем производительности MPI приложений, описанных с использованием разработанной системы. Описан эксперимент по повышению производительности MPI программы с использованием системы Performance Expert. Использование полученных рекомендаций позволило в среднем повысить производительность в 1,64 раза. Максимальное повышение производительности (в 2,4 раза) было достигнуто при 64 процессах.

Литература

1. Toward Scalable Performance Visualization with Jumpshot / O. Zaki [et al.] // High-Performance Computing Applications. – 1999. – Vol. 13. – No. 2. – P. 277-288.
2. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation / C.-K. Luk [et al.] // Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, Chicago, IL, 2005. – P. 190-200.
3. Forgy, C. L. Rete: A Fast Algorithm for the Many Pattern: Many Object Pattern Match Problem / C. L. Forgy // Artificial Intelligence. – 1982. – Vol. 19. – P. 17-37.
4. Эндрюс, Г. Р. Основы многопоточного, параллельного и распределенного программирования / Г. Р. Эндрюс; пер. с англ. А. С. Подосельника, Г. И. Сингаевской, А. Б. Ставровского. – М.: Вильямс, 2003. – 512 с.
5. Wolf, F. Automatic performance analysis of hybrid MPI/OpenMP applications / F. Wolf, B. Mohr // Journal of Systems Architecture: the EUROMICRO Journal. – 2003. – Vol. 49, № 10/11. – P. 421-439.

Ab initio молекулярная динамика: перспективы использования многопроцессорных и гибридных супер-ЭВМ*

П.А. Жиляев[1,2], В.В. Стегайлов[1,2]

Объединенный институт высоких температур РАН [1],
Московский физико-технический институт [2]

В работе представлен обзор распараллеливания алгоритмов ab initio молекулярной динамики на основе теории функционала электронной плотности в базисе плоских волн. Проанализированы требования к балансу вычислительной мощности узлов и коммутационной сети супер-ЭВМ с точки зрения достижения максимальной эффективности для примеров особенно требовательных в вычислительном отношении задач физики разогретого плотного вещества. Описана альтернативная стратегия параллелизации в вейвлетном базисе и выигрыш в производительности при использовании гибридных вычислительных систем в этом случае.

1. Введение

Продолжающееся непрерывное развитие теоретических и вычислительных методов атомистического моделирования на протяжении последних десятилетий обеспечивает основу средств анализа и прогноза для физики конденсированного состояния, материаловедения, химии, молекулярной биологии и нанотехнологий. Свойства материалов определяются откликом многоатомной системы на изменение внешних условий. Вообще говоря, для теоретического описания данного отклика не может быть достаточно методов кинетики и теории сплошных сред, а нужен выход на атомистический уровень описания.

Развитие атомистических моделей вещества изначально шло несколькими практически независимыми путями. Развитие методов квантовой механики позволило с 30-х гг. XX века проводить расчеты электронной структуры молекулярных систем небольшого размера. Совершенствование данных методов привело к созданию чрезвычайно развитой к настоящему времени области под названием квантовая химия. Когда в результате создания компьютеров 50-х годах XX века появился способ проведения больших объемов математических расчетов, одним из первых их применений стали задачи статистической механики, решение которых предполагалось получать путем непосредственного расчета динамики многочастичной системы на основе классических уравнений движения с простейшими модельными межатомными потенциалами. Подобный метод решения задач равновесной и неравновесной статистической механики получил название метода молекулярной динамики (МД).

Метод МД в настоящее время является незаменимым методом исследования конденсированного состояния [1]. Принципиальным условием его успешного использования является наличие адекватных моделей потенциалов межатомного взаимодействия. Межатомное взаимодействие определяется электронной структурой вещества, поэтому современные методы создания потенциалов основаны на квантово-механических расчетах [2,3]. Наряду с методами квантовой химии (применимыми для изолированных молекул и кластеров), важную роль при этом сыграли методы, развитые в квантовой физике твердого тела для периодических систем. По мере роста вычислительных возможностей все чаще рассматриваются ab initio МД модели, не использующие эмпирические потенциалы.

Общей теоретической основой квантовых методов являются методы, основанные на приближение Хартри-Фока, и методы теории функционала электронной плотности (ТФП). Для представления электронной структуры в квантово-химических расчетах используются локализованные базисы [4,5], в то время как для периодических систем, естественным образом, в первую очередь, используется базис плоских волн [6,7], что приводит к совершенно разным вычислительным алгоритмам, в том числе и к разным стратегиям их распараллеливания. Пре-

* Работа выполнена при поддержке гранта РФФИ 11-01-12131-офи-м-2011.

имуществом базиса плоских волн является возможность систематического повышения точности расчетов путем плавного увеличения размера базиса. Использование быстрого преобразования Фурье обеспечивает высокую скорость расчетов в этом базисе. С использованием базиса плоских волн для ab initio МД на настоящее время доступны системы до тысяч атомов и времена до десятков пикосекунд [6-10]. Данная статья посвящена параллельным алгоритмам в рамках ТФП в базисе плоских волн. Заметим, что локализованные базисы также эффективно используются для ab initio МД и обеспечивают точность, сопоставимую с базисом плоских волн [11]. Перспективным направлением является использование смешанных базисов [12].

2. Теория функционала электронной плотности

В рамках теории функционала электронной плотности (ТФП) полная энергия системы взаимодействующих электронов в поле классических ядер $\{\mathbf{R}_I\}$ представляется в виде

$$\min_{\Psi_0}\left\{\left\langle \Psi_0 \left| He \right| \Psi_0 \right\rangle\right\} = \min_{\Psi_0} E^{KS}\left[\{\phi_i\}\right],$$

как минимум функционала Кона-Шэма

$$E^{KS}\left[\{\phi_i\}\right] = T_s\left[\{\phi_i\}\right] + \int d\mathbf{r}\, V_{ext}(\mathbf{r})\; n(\mathbf{r}) + \frac{1}{2}\int d\mathbf{r}\, V_H(\mathbf{r})\; n(\mathbf{r}) + E_{xc}[n] + E_{ions}(\mathbf{R}^N), \qquad (1)$$

который явным образом зависит от набора вспомогательных функций (т.н. орбиталей Кона-Шэма), которые удовлетворяют условию ортонормированности

$$\left\langle \phi_i \left| \phi_j \right\rangle = \delta_{ij}\right..$$

В рамках данного подхода решение многоэлектронной квантово-механической задачи упрощается путем перехода от описания электронов в виде волновых функций к описанию в терминах одночастичной электронной плотности. Метод ТФП отличается от метода Томаса-Ферми тем, что представление электронов в виде волновых функций ϕ_i все-таки сохраняется. Это позволяет учесть обменно-корреляционные эффекты, которые не могут быть описаны в модели Томаса-Ферми.

Распределение электронной плотности рассчитывается на основе одного детерминанта Слэтера, построенного по заполненным орбиталям с учетом чисел заполнения

$$n(\mathbf{r}) = \sum_i^{occ} f_i \left|\phi_i(\mathbf{r})\right|^2 .$$

Первое слагаемое в функционале Кона-Шэма (1.1) соответствует кинетической энергии воображаемой системы невзаимодействующих электронов

$$T_s\left[\{\phi_i\}\right] = \sum_i^{occ} f_i \left\langle \phi_i \left| -\frac{1}{2}\nabla^2 \right| \phi_i \right\rangle,$$

состоящей из того же числа электронов и находящейся в поле того же внешнего потенциала, что и исходная система с полным взаимодействием. Второе слагаемое соответствует внешнему потенциалу $V_{ext}(\mathbf{r})$ - в большинстве случаев он соответствует сумме потенциалов классических ядер. Третье слагаемое соответствует классической электростатической энергии и рассчитывается на основе потенциала Хартри

$$V_H(\mathbf{r}) = \int d\mathbf{r}\,' \frac{n(\mathbf{r}')}{\left|\mathbf{r}-\mathbf{r}'\right|},$$

который в свою очередь связан с распределением электронной плотности уравнением Пуассона

$$\nabla^2 V_H(\mathbf{r}) = -4\pi n(\mathbf{r}) .$$

Предпоследнее слагаемое в функционале Кона-Шэма представляет собой обменно-корреляционный функционал $E_{xc}[n]$, который одновременно включает в себя обменные и корреляционные эффекты. По сути, функционал $E_{xc}[n]$ представляет собой разность между точной энергией многоэлектронной системы и ее разложением по методу Кона-Шэма на три предшествующих слагаемых. Последнее слагаемое в (1) соответствует энергии взаимодействия зарядов ядер.

Путем минимизации функционала Кона-Шэма (1) при условии фиксированного полного числа электронов с учетом условия ортонормированности орбиталей получается система уравнений Кона-Шэма

$$\left\{-\frac{1}{2}\nabla^2 + V_{ext}(\mathbf{r}) + V_H(\mathbf{r}) + \frac{\delta E_{xc}[n]}{n(\mathbf{r})}\right\}\phi_i(\mathbf{r}) = \sum_j \Lambda_{ij}\phi_j(\mathbf{r}),$$

которая может быть записана в виде

$$\left\{-\frac{1}{2}\nabla^2 + V^{KS}(\mathbf{r})\right\}\phi_i(\mathbf{r}) = \sum_j \Lambda_{ij}\phi_j(\mathbf{r})$$

или

$$H^{KS}\phi_i(\mathbf{r}) = \sum_j \Lambda_{ij}\phi_j(\mathbf{r}),$$

если ввести одноэлектронный Гамильтониан H^{KS} с локальным потенциалом $V^{KS}(\mathbf{r})$. Заметим, что данный одноэлектронный Гамильтониан H^{KS} эффективным образом учитывает многочастичные эффекты за счет обменно-корреляционного потенциала

$$\frac{\delta E_{xc}[n]}{n(\mathbf{r})} = V_{xc}(\mathbf{r}).$$

Унитарное преобразование в пространстве заполненных орбиталей дает каноническую форму системы уравнений Кона-Шэма

$$H^{KS}\phi_i(\mathbf{r}) = \varepsilon_i\phi_i(\mathbf{r}). \quad (2)$$

Для того, чтобы получить орбитали и распределение одноэлектронной плотности для основного состояния многоэлектронной системы система, уравнений (2) должна решаться самосогласованным образом.

Для применения ТФП в расчетах реальных систем критическим является используемое приближение для аппроксимации неизвестного обменно-корреляционного функционала (приближение локальной плотности, градиентное приближение, комбинированные подходы).

С вычислительной точки зрения вычисления электронной структуры в рамках теории функционала электронной плотности могут быть существенно облегчены путем введения в модель псевдопотенциалов. Данные псевдопотенциалы учитывают тот факт, что электроны, находящиеся на сильно связанных заполненных электронных оболочках атомов, практически не изменяют свое состояние при взаимодействии с другими атомами (при образовании и разрыве химических связей и т.п.) Таким образом, при построении моделей возможно разделение электронной на валентные, описываемые явно, и электроны сердцевины атома, описываемые эффективным псевдопотенциалом, заменяющим чисто кулоновский потенциал ядра. При конструировании псевдопотенциалов обычно накладывается условие соответствия волновых функций валентных электронов атома в полноэлектронном и псевдопотенциальном представлении (см. рис.1).

3. Расчеты в базисе плоских волн

Основа численного решения задачи о вычислении электронной структуры в рамках ТФП состоит в выборе базиса для представления волновых функций. Одно из ключевых преиму-

ществ базиса плоских волн заключается в том, что его реализация основана на использовании преобразования Фурье, для которого существуют высокоэффективные численные алгоритмы (т.н. быстрое преобразование Фурье).

Входными параметрами задачи являются геометрия системы (положения атомов, размеры расчетной ячейки), модели псевдопотенциала для атомов в модели, обрезка по энергии базиса плоских волн, параметры итерационного алгоритма нахождения самосогласованного решения системы уравнений Кона-Шэма.

Точность решения задачи о вычислении электронной структуры в рамках ТФП в первую очередь зависит от выбора модели псевдопотенциала и модели обменно-корреляционного функционала. Границы применимости существующих моделей определяются сравнением с экспериментальными данными для определенных классов веществ и соединений. Описанные в данной статье алгоритмы реализованы в кодах CPMD [6] и ABINIT [7].

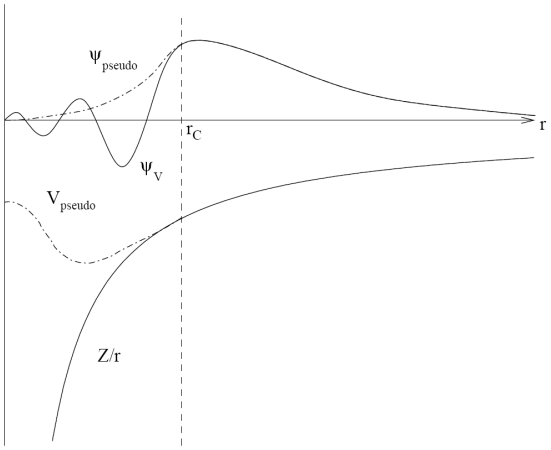


Рис. 1. Качественная схема, иллюстрирующая принцип введения псевдопотенциала для описания эффективного влияния внутренних электронов атома на валентные электроны

2.1 Укрупненная блок-схема последовательного алгоритма

Введем следующие обозначения:

N_{at}	число атомов
N_P	число проекторов
N_b	число электронных состояний
N_{PW}	число плоских волн
N_D	число плоских волн для представления плотности и потенциалов
N_x, N_y, N_z	число точек сетки в направлениях x, y и z
$N= N_xN_yN_z$	полное число точек сетки

В таблице 1 приведены характерные значения указанных переменных для двух систем. Пример модели кристалла кремния с обрезкой по энергии на 13Ry и нелокальностью псевдопотенциала *s*-типа. И пример водной системы (значения приводятся из расчета на одну молекулу) с обрезкой по энергии на 70Ry, с нелокальностью *s*-типа псевдопотенциала кислорода и локальным псевдопотенциалом для водорода.

Таблица 1. Характерные параметры моделей в рамках ТФП в базисе плоских волн

	Кремний	Вода
N_{at}	1	3
N_P	1	1
N_b	2	4
N_{PW}	53	1000
N_D	429	8000
N	1728	31250

2.2 Структура представления данных

Основные величины в расчете с использованием псевдопотенциалов в базисе плоских волн либо не зависят от размера системы, либо увеличиваются линейно, либо квадратично. Примерами величин первого типа являются матрица элементарной ячейки **h** и значение энергии обрезки E_{cut}. Следующие переменные, которые растут линейно с увеличением размера системы:

r(3, N_{at})	координаты атомов
v(3, N_{at})	скорости атомов
f(3, N_{at})	силы, действующие на атомы
g(3, N_{PW})	индексы плоских волн
ipg(3, N_{PW})	отображение G-векторов (положительная часть)
img(3, N_{PW})	отображение G-векторов (отрицательная часть
rhog(N_{PW})	плотности (n, n_c, n_{tot}) в пространстве Фурье
vpot(N_{PW})	потенциалы (V_{loc}, V_{xc}, V_H) в пространстве Фурье
n(N_x,N_y,N)	плотности (n, n_c, n_{tot}) в реальном пространстве
v(N_x,N_y,N_z)	потенциалы (V_{loc}, V_{xc}, V_H) в реальном пространстве
vps(N_D)	локальный псевдопотенциал
rpc(N_D)	заряды кора атомов
pro(N_{PW})	проекторы для нелокальной части псевдопотенциала.

Величины `vps`, `rpc` и `pro`, относящиеся к псевдопотенциалу, зависят линейно от размера системы, но в то же время зависят от числа типов атомов в модели. В дальнейшем описании без потери общности будем предполагать, что в системе один тип атомов. Большая часть оперативной памяти уходит на хранение величин, зависящих квадратично от размера системы:

eigr(N_D, N_{at})	структурные факторы
fnl(N_P, N_b)	перекрытия проекторов и состояний
dfnl(N_P, N_b,3)	производная от fnl
smat(N_b, N_b)	матрицы перекрытия между состояниями
cr(N_{PW}, N_b)	состояния в Фурье-пространстве
cv(N_{PW}, N_b)	скорости состояний в Фурье-пространстве
cf(N_{PW}, N_b)	силы состояний в Фурье-пространстве.

Структурные факторы `eigr` и величины `cr`, `cv`, `cf`, относящиеся к волновым функциям, представляют собой комплексные числа. Другие величны могут храниться в виде действительных чисел, если учитывать фактор $(-i)^l$.

2.3 Вычислительные алгоритмы

Большая часть вычислений в коде, основанном на представлении плоских волн, производится в нескольких базовых алгоритмах. Эти алгоритмы могут быть описаны на псевдокоде. Где возможно, применяются стандартные процедуры линейной алгебры (библиотека BLAS). Первый алгоритм – расчет структурных факторов.

```
MODULE StructureFactor
    FOR i=1:Nat
        s(1:3) = 2 * PI * MATMUL[htm1(1:3,1:3),r(1:3,i)]
        dp(1:3) = CMPLX[COS[s(1:3)],SIN[s(1:3)]]
        dm(1:3) = CONJG[dp(1:3)]
        e(0,1:3,i) = 1
    FOR k=1:gmax
        e(k,1:3,i) = e(k-1,1:3,i) * dp
        e(-k,1:3,i) = e(-k+1,1:3,i) * dm
    END
    FOR j=0:ND
```

```
        eigr(j,i) = e(g(1,j),1,i) * e(g(2,j),2,i) *
        e(g(3,j),3,i)
    END
END
```

Рис. 2. Алгоритм расчета структурных факторов

Алгоритм расчета перекрытий проекторов нелокальной части псевдопотенциала и волновых функций в Фурье-пространстве масштабируется по времени исполнения как $N_P N_b N_{PW}$.

Два следующих алгоритма проводят преобразование зарядовой плотности из Фурье-пространства в прямое пространство и обратно.

```
MODULE INVFFT
scr(1:Nx,1:Ny,1:Nz) = 0
FOR i=1:ND
    scr(ipg(1,i),ipg(2,i),ipg(3,i)) = rhog(i)
    scr(img(1,i),img(2,i),img(3,i)) = CONJG[rhog(i)]
END
CALL FFT3D("INV",scr)
n(1:Nx,1:Ny,1:Nz) = REAL[scr(1:Nx,1:Ny,1:Nz)]

MODULE FWFFT
scr(1:Nx,1:Ny,1:Nz) = n(1:Nx,1:Ny,1:Nz)
CALL FFT3D("FW",scr)
FOR i=1:ND
    rhog(i) = scr(ipg(1,i),ipg(2,i),ipg(3,i))
END
```

Рис. 3. Преобразование зарядовой плотности из Фурье-пространства в прямое пространство и обратно

На рис.3 проиллюстрированы алгоритмы расчета плотности и применение локального потенциала. Число операций в этих алгоритмах пропорцинально $N_b\, N \log N$. В большинстве приложений эти алгоритмы берут на себя больше всего вычислительного времени.

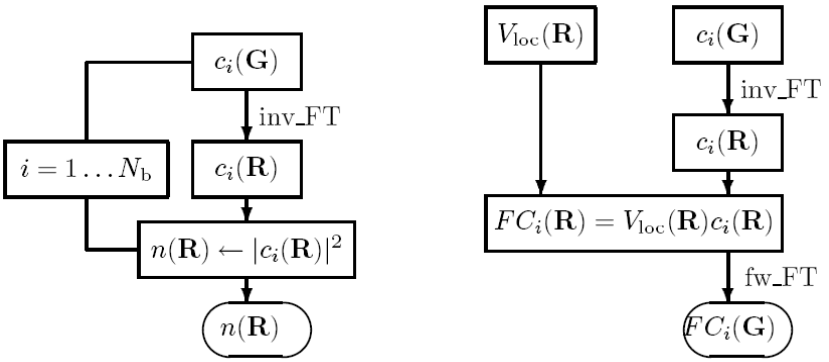


Рис. 4. Схема алгоритмов расчета плотности (слева) и применения локального потенциала для расчета сил, действующих на волновую функцию (справа). Расчет плотности требует N_b трехмерных преобразований Фурье. Для применения локального потенциала необходимо одно Фурье-преобразование на каждое состояние.

2.4 Принципы работы параллельного алгоритма

При распараллеливании алгоритмов для систем с распределенной памятью в рамках представленного метода расчета электронной структуры, основанного на базисе плоских волн, основное внимание уделяется минимизации обменов между вычислительными узлами. В этой связи крайне важно распределение данных по вычислительным узлам. В первую очередь это

относится у информации о волновых функциях. Распределение данных является статическим, т.е. оно не изменяется в процессе расчета. Во всех частях программы, где невозможна параллелизация по плоским волнам, используется параллелизация по числу атомов и состояниям.

На каждый вычислительный узел p приписывается определенное число плоских волн, атомов, электронных состояний и точек сетки в реальном пространстве.

N^p_{at}	число атомов
N^p_P	число проекторов
N^p_b	число электронных состояний
N^p_{PW}	число плоских волн
N^p_D	число плоских волн для представления плотности и потенциалов
N^p_x, N_y, N_z	число точек сетки в направлениях x, y и z
$N^p = N^p_x N_y N_z$	полное число точек сетки

Сетка по пространству распределяется по процессорам только в направлении x. Это связано с производительностью алгоритма Фурье-преобразования. Конкретные алгоритмы, основанные на распределении указанных данных, незначительно влияют по общую эффективность распараллеливания кода.

Структуры данных, которые реплицируются на всех вычислительных узлах:

r(3, N_{at})	координаты атомов
v(3, N_{at})	скорости атомов
f(3, N_{at})	силы, действующие на атомы
fnl(N_P, N_b)	перекрытия проекторов и состояний
smat(N_b, N_b)	матрицы перекрытия между состояниями.

Структуры данных, которые распределяются по процессорам:

g(3, N^p_{PW})	индексы плоских волн
ipg(3, N^p_{PW})	отображение G-векторов (положительная часть)
img(3, N^p_{PW})	отображение G-векторов (отрицательная часть
rhog(N^p_{PW})	плотности (n, n_c, n_{tot}) в пространстве Фурье
vpot(N^p_{PW})	потенциалы (V_{loc}, V_{xc}, V_H) в пространстве Фурье
n(N^p_x,N_y,N)	плотности (n, n_c, n_{tot}) в реальном пространстве
v(N^p_x,N_y,N_z)	потенциалы (V_{loc}, V_{xc}, V_H) в реальном пространстве
vps(N^p_D)	локальный псевдопотенциал
rpc(N^p_D)	заряды кора атомов
pro(N^p_{PW})	проекторы для нелокальной части псевдопотенциала.
eigr(N^p_D, N_{at})	структурные факторы
dfnl(N_P, N^p_b,3)	производная от fnl
cr(N^p_{PW}, N_b)	состояния в Фурье-пространстве
cv(N^p_{PW}, N_b)	скорости состояний в Фурье-пространстве
cf(N^p_{PW}, N_b)	силы состояний в Фурье-пространстве.

При распределении плоских волн по вычислительным узлам должны выполняться несколько условий. Все узлы должны иметь приблизительно равное число плоских волн. Если плоская волна, соответствующая обрезке волной функции, находится на некотором узле, то таже плоская волна должна быть на том же процессоре для обрезки плотности. Распределение плоских волн должно быть организовано таким образом, чтобы в начале и конце трехмерного преобразования Фурье не требовалось дополнительных обменов данными. Специально внимание должно уделяться вычислительному узлу ($p0$), на котором находится компонента $\mathbf{G} = \mathbf{0}$, в связи с особенностями расчета перекрытий с ее участием.

Главными для распараллеливания алгоритмов являются три коммуникационные процедуры. Все они – коллективные, т.е. задействуют все вычислительные узлы, что подразумевает проведение синхронизации при выполнении этих процедур. Этими процедурами являются Broadcast, GlobalSum и MatrixTranspose. Процедура Broadcast пересылает данные с одного узла (px) на все остальные

$x^p \leftarrow x^{px}$.

Процедура GlobalSum заменяет элемент данных на каждом узле суммой этой величины по всем узлам

$$x^p \leftarrow \sum_p x^p$$

Процедура MatrixTranspose производит транспонирование матрицы

$x(p,:) \leftarrow x(:,p)$.

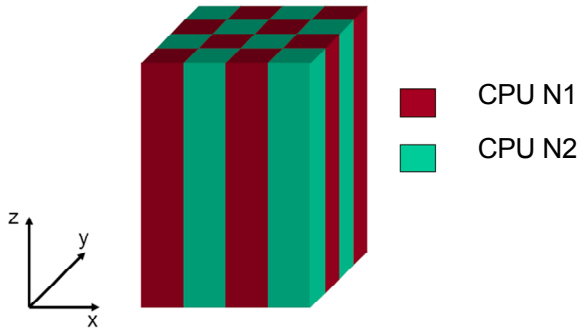


Рис. 5. Распределение данных для проведения 3D FFT. Для каждой волновой функции: ее коэффициенты распределяются по G-векторам в направлении z (на рисунке пример для 2 процессоров)

На параллельной вычислительной системе, состоящей из P процессоров, с характерным временем задержки при коммуникации (латентностью) t_L и пропускной способностью B, на выполнение одного запуска указанных процедур уходит следующее время

Broadcast $\log_2[P]\{t_L + N/B\}$

GlobalSum $\log_2[P]\{t_L + N/B\}$

MatrixTranspose $Pt_L + N/(PB)$.

С учетом описанных правил распределения данных параллелизация большинства алгоритмов достаточно проста.

Процедуры, для распараллеливания которых требуется большое число изменений, содержат как составную часть преобразование Фурье. Вводится процедура `mapxy`, которая обеспечивает непрерывное распределение данных в памяти на каждом вычислительном узле.

```
MODULE INVFFT
scr1(1:Nx,1:NPpencil) = 0
FOR i=1:NPD
    scr1(ipg(1,i),mapxy(ipg(2,i),ipg(3,i))) = rhog(i)
    scr1(img(1,i),mapxy(img(2,i),img(3,i))) =
    CONJG[rhog(i)]
END
CALL ParallelFFT3D("INV",scr1,scr2)
n(1:Np
x ,1:Ny,1:Nz) = REAL[scr2(1:NPx,1:Ny,1:Nz)]

MODULE FWFFT
scr2(1:NPx,1:Ny,1:Nz) = n(1:NPx,1:Ny,1:Nz)
CALL ParallelFFT3D("FW",scr1,scr2)
FOR i=1:NPD
    rhog(i) = scr1(ipg(1,i),mapxy(ipg(2,i),ipg(3,i)))
END
```

Рис. 6. Параллельная версия преобразования трехмерного Фурье

Из-за отображения направлений *y* и *z* на одно направление при преобразовании Фурье массивы ввода и вывода параллельного преобразования Фурье должны иметь различный вид. Процедура трехмерного параллельного Фурье преобразования может быть сконструирована из нескольких одномерных Фурье преобразований и параллельных транспонирований матрицы.

```
MODULE ParallelFFT3D(tag,a,b)
IF (tag == "INV") THEN
   CALL MLTFFT1D(a)
   CALL ParallelTranspose("INV",b,a)
   CALL MLTFFT2D(b)
ELSE
   CALL MLTFFT2D(b)
   CALL ParallelTranspose("FW",b,a)
   CALL MLTFFT1D(a)
END IF
```

Рис. 7. Конструирование трехмерного преобразования Фурье из одномерных

3. Тесты параллельной производительности

На рис.8-11 представлены результаты тестов на суперкомпьютерах МФТИ60, BlueGene ВМК МГУ и "Ломоносов" НИВЦ МГУ. На рис.8 показаны данные по ускорению расчетов молекулы N-метилформамида. Результаты тестов показывают, что даже расчеты такой небольшой системы могут эффективно распараллеливаться на десятки процессоров.

Другим стандартным тестом параллельной эффективности является расчет системы 32 молекул воды (рис.9). Результаты тестов демонстрируют существенное преимущество наличия интерконнекта с высокой связностью для ускорения коллективных операций. Наличие подобного интерконнекта на суперкомпьютере BlueGene ВМК МГУ обеспечивает эффективное масштабирование задачи до 512 вычислительных ядер (кроме того, значение имеет баланс скорости интерконнекта и вычислительной производительнотсти процессоров). В тоже время падение параллельной эффективности на кластере МФТИ-60 (интерконнект Myrinet) происходит уже на 100 ядрах.

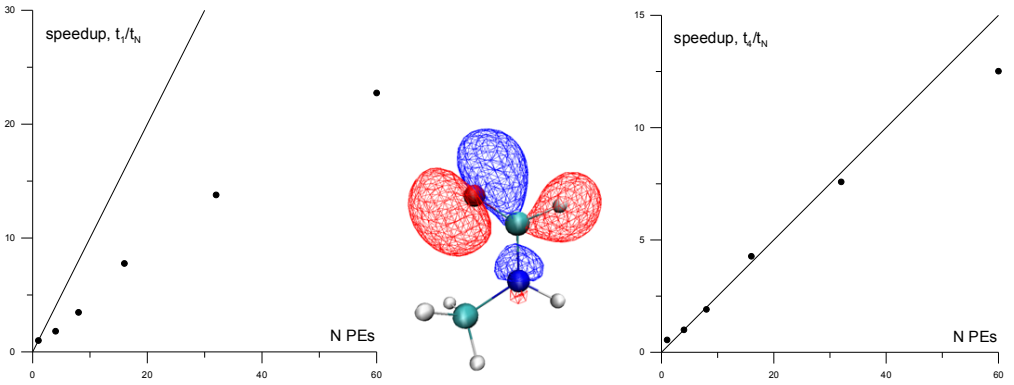


Рис. 8. Относительное ускорение расчетов на параллельной вычислительной системе для расчета электронной структуры 8-ми атомной молекулы N-метилформамида (пакет CPMD). Левый график - рост эффективность по отношению к однопроцессорному варианту расчета. Правый график - рост эффективность по отношению к одному полному узлу вычислительного кластера, имеющему 4 вычислительных ядра. Расчеты проведены на кластере МФТИ-60. Сплошная линия показывает случай стопроцентного распараллеливания. В центре показана визуализация высшей заполненной орбитали (HOMO) основного состояния S_0 N-метилформамида C2H5NO

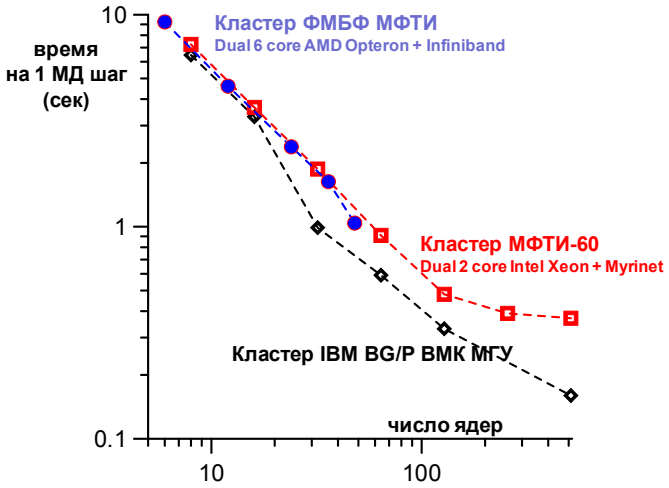


Рис. 9. Результаты тестирования производительности кода CPMD на различных параллельных системах на стандартном примере системы с 32-я молекулами воды

В качестве еще одного примера рассмотрим задачу расчета коэффициента электронной теплопроводности жидкого алюминия в двух температурном случае, когда температура электронной подсистемы (T_e) много больше температуры решетки (T_i). Мотивацией данной проблемы является описание свойств металла в состоянии разогретого плотного состояния для моделирования лазерной абляции. Для этого необходимы численные значения кинетических коэффициентов металла с горячими электронами [13]. На рис. 10 представлены результаты расчет коэффициента электронной теплопроводности на основе формулы Кубо-Гринвуда для проводимости алюминия. Полученная зависимость коэффициента теплопроводности от электронной температуры хорошо согласуется с результатами кинетической модели [14]. Точность МД расчетов (погрешности на графике) определяется усреднением по независимым конфигурациям, которые получаются путем расчета МД траектории достаточной длины. Специфика данного типа расчетов состоит в том, что сходимость по числу атомов и по числу k-точек, покрывающих зону Бриллюэна, является достаточно медленной и для получения надежных данных необходимо использовать системы нескольких сотен атомов с числом k-точек вплоть до 10^3, что предъявляет требования к необходимому суммарному объему оперативной памяти. Возможность расчетов электронной теплопроводности на основе ab initio МД и их точность существенным образом зависит от наличия эффективных параллельных алгоритмов и высокопроизводительных систем.

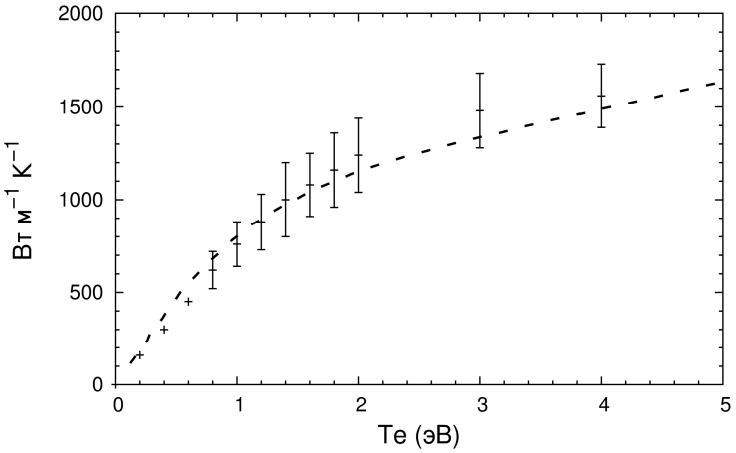


Рис. 10. Теплопроводность жидкого алюминия (Ti = 2000 К) в двухтемпературном случае. Точки – данная работа, пунктир – расчет по кинетической модели [14]

На рис. 11 показаны результаты тестирования масштабируемости расчетов электронной теплопроводности на основе первопринципной молекулярной динамики с использованием ТФП в базисе плоских волн (пакет ABINIT). Результаты демонстрируют очень хорошее ускорение расчетов вплоть до 100-200 ядер на кластере МФТИ60 и кластере "Ломоносов" НИВЦ МГУ.

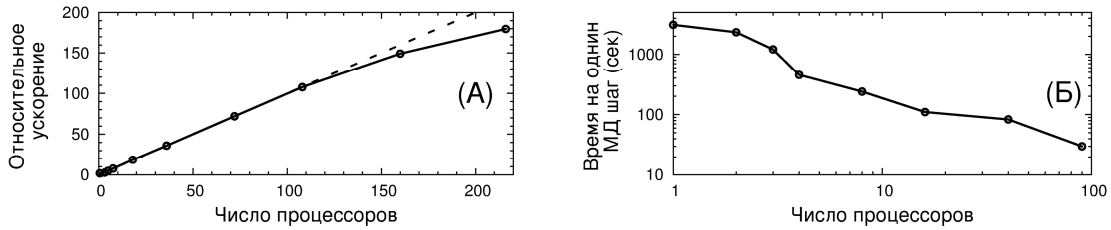


Рис. 11. (а) Относительное ускорение расчетов на параллельной вычислительной системе для расчета электронной структуры жидкого алюминия 108 атомов (пакет ABINIT). Расчеты проведены на кластере МФТИ-60. Сплошная линия показывает случай стопроцентного распараллеливания.
(б) Результаты тестирования производительности кода ABINIT на суперкомпьютере "Ломоносов" НИВЦ МГУ на примере жидкого алюминия 108 атомов

4. Вейвлетный базис и выигрыш в производительности при использовании гибридных вычислительных систем

Среди современных высокопроизводительных систем в последнее время все чаще используются гибридные системы, включающие графические ускорители (ГПУ – графическое процессорное устройство). Этому во многом способствовало появление новых сред программирования, таких как CUDA и OpenCL, предназначенных для создания и выполнения на ГПУ программ, не связанных с обработкой изображений. Гибридные системы на основе ГПУ зачастую показывают быстродействие, в десятки или сотни раз превышающее быстродействие систем на традиционных процессорах (ЦПУ – центральной процессорное устройство). В то же время создание программ, эффективно использующих вычислительные возможности ГПУ, - это достаточно трудоемкий процесс, требующий учета специфики аппаратной архитектуры, без которого производительность на конкретной задаче может оказаться существенно меньше теоретического предела для данного устройства.

В работе [15] представлена реализация полного кода для ТФП расчетов электронной структуры на гибридных параллельных системах. Эта реализация осуществлена на основе свободного программного кода на основе вейвлетов Добеши. Код показывает очень хорошую параллельную эффективность и обладает устойчивой сходимостью. В частности, код может работать на многих узлах, которые могут иметь или не иметь GPU. Показано, что с двойной точностью расчетов можно добиться значительного ускорения: до 20 раз для некоторых операций и до 6 раз для полного набора операции типичного расчета в рамках ТФП.

4. Заключение

Изложена математическая формулировка теории функционала электронной плотности в базисе плоских волн для расчетов электронной структуры и первопринципной молекулярной динамики. Описаны последовательные версии алгоритмов и их параллельные версии. Приведены результаты тестовых расчетов различных систем на кластерах МФТИ60, BlueGene ВМК МГУ и «Ломоносов» НИВЦ МГУ. Кратко обсуждается альтернативная стратегия распараллеливания в вейвлетном базисе и ускорение на гибридных архитектурах.

Приведенные примеры модельных задач (изолированная молекула, система молекул воды, металл в разогретом плотном состоянии) демонстрируют эффективное масштабирование данных алгоритмов на параллельных вычислительных системах без существенного снижения эффективности распараллеливания на 100-500 ядрах.

Литература

1. Янилкин А.В., Жиляев П.А., Куксин А.Ю., Норман Г.Э., Писарев В.В., Стегайлов В.В. Применение суперкомпьютеров для молекулярно-динамического моделирования процессов в конденсированных средах // Вычислительные методы и программирование. 2010. Т. 11. С. 111-116.
2. van Duin A.C.T. et al. ReaxFF: A Reactive Force Field for Hydrocarbons // J. Phys. Chem. A. 2001. V. 105. P. 9396-9409.
3. Smirnova D.E., Starikov S.V., Stegailov V.V. Interatomic potential for uranium in a wide range of pressures and temperatures // J. Phys.: Cond. Mat. 2012. V. 24. P. 015702.
4. Laikov D.N., Ustynyuk Yu.A. PRIRODA-04: a quantum chemical program suite. New possibilities in the study of molecular systems with the application of parallel computing // Russian Chemical Bulletin (International Edition). 2005. V. 54. No. 3. P. 820-826.
5. Granovsky A.A. Extended multi-configuration quasi-degenerate perturbation theory: The new approach to multi-state multi-reference perturbation theory // J. Chem. Phys. 2011. V. 134. P. 214113.
6. Hutter J., Curioni A. Car-Parrinello molecular dynamics on massively parallel computers // ChemPhysChem. 2005. V. 6. P. 1788-1793.
7. Gonze X. et al. ABINIT: First-principles approach to material and nanosystem properties // Computer Physics Communications. 2009. V. 180. P. 2582-2615.
8. Gygi F. et al. Practical algorithms to facilitate large-scale first-principles molecular dynamics // J. Phys.: Conf. Ser. 2009. V. 180. P. 012074.
9. Goedecker S. et al. An efficient 3-dim FFT for plane wave electronic structure calculations on massively parallel machines composed of multiprocessor nodes // Computer Physics Communacations. 2003. V. 154. P. 105-110.
10. Bottin F. et al. Large-scale ab initio calculations based on three levels of parallelization // Computational Material Science. 2008. V. 42. P. 329-336.
11. Gusso M. Study on the maximum accuracy of the pseudopotential density functional method with localized atomic orbitals versus plane-wave basis sets // J.Chem. Phys. 2008. V. 128. P. 044102.
12. Lippert G., Hutter J., Parrinello M. A hybrid Gaussian and plane wave density functional scheme. // Mol. Phys. 1997. V. 92 P. 477-487.
13. Стариков С.В., Стегайлов В.В., Норман Г.Э., Фортов В.Е. и др. Лазерная абляция золота: эксперимент и атомистическое моделирование // Письма в ЖЭТФ. 2011. Т. 93. С. 719-725.
14. Иногамов Н. А., Петров В.Ю. Теплопроводность металлов с горячими электронами // ЖЭТФ 2010. Т. 134. N. 3. С. 505-529.
15. Genovese L. et al. Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures // J. Chem. Physics. 2009. V. 131. P. 034103.

Опыт организации добровольных вычислений на примере проектов OPTIMA@home и SAT@home*

О.С. Заикин[1], М.А. Посыпкин[2], А.А. Семенов[1], Н.П. Храпов[2]

Институт динамики систем и теории управления СО РАН[1],
Институт системного анализа РАН[2]

Описывается процесс построения проектов добровольных вычислений, базирующихся на платформе BOINC и предназначенных для решения задач, предполагающих массовый крупноблочный параллелизм. Детали и особенности этого процесса иллюстрируются на примере реально функционирующих проектов добровольных вычислений OPTIMA@home и SAT@home. Первый проект предназначен для решения задач глобальной оптимизации, второй – для решения комбинаторных задач, сведенных к задачам о булевой выполнимости (SAT).

1. Введение

Добровольные вычисления – относительно новое направление в распределенных вычислениях, идеология которого предполагает предоставление вычислительных ресурсов для организации масштабных расчетов т.н. «волонтерами» (volunteers). Волонтеры или добровольцы – это, как правило, пользователи, располагающие собственными персональными компьютерами (ПК), ресурсы которых они согласны предоставить для решения разнообразных научных задач. Следует отметить, что используются только свободные ресурсы (когда ПК не используется для других целей), поэтому участие в добровольных вычислениях не мешает работе пользователей. ПК добровольцев, которые могут находиться в географически удаленных друг от друга точках, объединяются в грид под управлением некоторой среды. Обычно такие грид-системы создаются под конкретные задачи, на решение которых могут уходить месяцы и даже годы, и называются проектами добровольных вычислений.

За последние несколько лет были организованы проекты добровольных вычислений, предполагающие решение широкого круга задач – от поиска новых космических объектов (Einstein@home) до проверки некоторых гипотез теории чисел (ABC@home) и криптографии (distributed.net).

Наиболее популярная открытая программная платформа для организации добровольных вычислений – это BOINC (Berkeley Open Infrastructure for Network Computing [1]). Она изначально разрабатывалась для проекта SETI@home в U.C. Berkeley Spaces Sciences Laboratory (США). С 2002 года платформа BOINC была сделана открытой и с 2004 года на ее основе стали создаваться другие проекты. На данный момент суммарная мощность всех проектов на платформе BOINC составляет около 6 петафлопс, что сопоставимо с мощностью самого высокопроизводительного суперкомпьютера из списка топ-500 [2].

В настоящей статье описывается технология организации проектов добровольных вычислений, созданных с использованием платформы BOINC. Характерные особенности технологии поясняются на примере двух реально функционирующих проектов – OPTIMA@home и SAT@home. Первый проект предназначен для решения разнообразных оптимизационных задач, второй – для решения комбинаторных задач, сведенных к задачам о выполнимости булевых формул (SAT-задачам).

*Работа выполнена при поддержке РФФИ (гранты № 11-07-00377-а и 10-07-00301-а) и Седьмой Рамочной программы Европейского Союза (FP7/2007-2013), грант № 261561 (DEGISCO).

2. Основные принципы организации проектов добровольных вычислений

Изначально для создания проектов распределенных добровольных вычислений требовалась работа больших коллективов разработчиков. Например, в 1996 году стартовал проект GIMPS по поиску простых чисел Мерсенна, а в 1997 году – distributed.net направленный на решение задачи криптоанализа шифра RC5. Изначально проект предполагал поиск 56-битного секретного ключа. В июле 1996 года на пятой международной конференции по биоастрономии была представлена концепция проекта SETI@home, предназначенного для обработки интенсивных потоков данных, поступающих от мощных радиотелескопов. Данный проект был запущен в 1999 году, а в 2002 году на его основе была разработана открытая платформа BOINC. Использование этой платформы значительно упростило создание новых проектов и на данный момент большинство крупных проектов используют именно BOINC.

Проект добровольных вычислений состоит из трех основных частей: сервера, веб-сайта и прикладного программного обеспечения (ПО). Процессом вычислений управляет сервер, именно на нем создаются задания для обработки. Для того чтобы подключиться к проекту, пользователь (доброволец) должен скачать и установить на свой ПК стандартный «BOINC-клиент». Это специальная программа, позволяющая подключать ПК пользователя к любым BOINC-проектам, осуществляющая обмен данными с сервером выбранного проекта и выделяющая ресурсы ПК пользователя для работы прикладного ПО. Подключение к проекту происходит путем указания URL сайта проекта, после чего BOINC-клиент автоматически скачивает с сервера прикладное ПО (ориентируясь на тип ОС и процессора ПК пользователя) и задания для обработки. Затем весь процесс вычислений на стороне пользователя происходит автоматически. В BOINC-клиенте существует гибкая система настроек, позволяющая эффективно задействовать свободные ресурсы ПК и распределять их между BOINC-проектами.

При организации проекта следует учитывать ряд факторов, способствующих его активному развитию. Специалистами по организации добровольных вычислений обычно выделяются следующие основные причины, мотивирующие пользователей на участие в проекте:

- за каждое выполненное задание пропорционально затраченным вычислительным ресурсам участникам начисляются т.н. «кредиты». Количество кредитов является характеристикой, по которой участники соревнуются между собой. Также участники объединяются в команды по разным признакам (национальному, региональному, пр.), которые также соревнуются между собой;
- при получении результатов обычно на сайте проекта выкладывается информация об участнике, на ПК которого был получен данный результат;
- ощущение причастности к важным научным исследованиям, именно поэтому большой популярностью пользуются медицинские проекты, направленные на поиск новых лекарств (Folding@home, WCG, Rosetta@home).

Поскольку в проектах добровольных вычислений используются ПК частных лиц, всегда имеется возможность того, что результаты вычислений будут сфальсифицированы пользователем, либо будут некорректными в результате аппаратного или программного сбоя. Для учета этого обычно используются избыточные вычисления. А именно, для каждого задания формируются несколько копий, которые отправляются на ПК разных пользователей. Полученные результаты сравниваются, если они совпадают, то задание считается решенным. Если есть расхождения, то результаты аннулируются, формируются новые копии задания, которые отправляются на ПК других пользователей. За различные копии одного задания могут быть запрошены различные количества кредитов. В зависимости от числа копий можно применять различные схемы расчета кредитов.

Существуют три основных статуса проектов добровольных вычислений: «альфа», «бета» и «релиз». Можно выделить следующие критерии, по которым проектам присваивают соответствующие статусы:

- стабильность (доступность сайта и сервера проекта в режиме 24/7, постоянное наличие готовых к отправке заданий);
- степень завершенности серверного и прикладного ПО;

- наличие версий прикладного ПО для основных операционных систем (windows, linux, mac);
- полнота и наглядность описания целей проекта, запланированных и полученных результатов.

Статусы проектов, использующих платформу BOINC, отображены на сайте Formula BOINC [3]. На начало февраля 2012 года из 46 активных проектов 25 имеют статус альфа, 10 – бета и 11 – релиз. Даже для получения статуса альфа проект проходит специальную экспертизу. Этот статус означает, что проект функционирует и находится на начальном этапе разработки. Следует отметить, что на территории СНГ на данный момент активно действуют четыре проекта добровольных вычислений на основе BOINC: три в России (включая OPTIMA@home и SAT@home, которые будут описаны ниже) и один на Украине (SLinCA@home). Все эти проекты имеют статус «альфа».

3. Проект добровольных распределенных вычислений OPTIMA@home

20 июня 2011 года в Институте системного анализа РАН был запущен проект добровольных распределенных вычислений OPTIMA@home [4], предназначенный для исследования эффективной реализации методов оптимизации. Проект OPTIMA@home реализован с использованием платформы BOINC. На **Рис. 1** показана общая схема работы распределенного приложения проекта, включающего в себя две основные части: управляющую и расчетную.

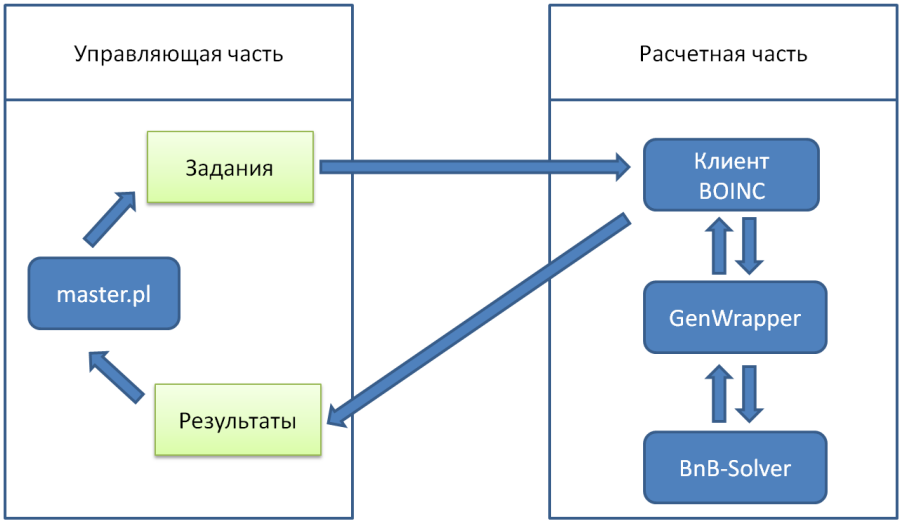


Рис. 1. Общая схема работы распределенного приложения OPTIMA@home

Управляющая часть выполняется на сервере проекта. Ее основная задача – генерировать новые задания и обрабатывать полученные с клиентских машин результаты. В проекте OPTIMA@home серверная часть представляет собой программу master.pl на языке Perl. Эта программа запускается периодически каждые полчаса, просматривает папку с полученными результатами, обрабатывает их и на их основе генерирует новые задания. Сгенерированные задания помещаются в базу данных заданий проекта.

Расчетная часть представляет собой архив выполняемых файлов, который загружается на ПК пользователей. Получив очередное задание, BOINC-клиент запускает расчетную часть, которая обрабатывает полученное задание. Результат возвращается на сервер. Основой расчетной части в проекте OPTIMA@home является выполняемый модуль библиотеки BNB-Solver [5]. Входной файл приложения содержит исходные данные задачи и параметры алгоритма, выходной файл содержит найденные решения. Оба файла используют формат XML.

Для запуска приложения BNB-Solver на пользовательских машинах используется инструмент GenWrapper [6, 7], предназначенный для портирования в грид существующих приложений. При этом не требуется переписывать приложение, вместо этого используется

выполняемый файл программы. На стороне пользователя программа GenWrapper распаковывает архив с выполняемыми файлами приложения и файлами данных и запускает командный файл, написанный на shell-подобном языке. Текст командного файла для проекта OPTIMA@home приведен на **Рис. 2.** Этот файл сначала получает глобальные имена входных/выходных файлов, необходимых для работы приложения, а затем выполняет запуск самого приложения, передавая эти файлы в качестве аргументов командной строки. Перед запуском командного файла программа `GenWrapper` получает исходные файлы от BOINC-клиента, а после выполнения приложения передает ему файлы с результатами.

```
IN=`boinc resolve_filename in`
OUT=`boinc resolve_filename out`
BNBAPP=`boinc resolve_filename bnbss`
./"${BNBAPP}" "${IN}" "${OUT}"
```

Рис. 2. Командный файл, выполняемый на стороне клиента

На данный момент в рамках проекта OPTIMA@home реализован метод Basin-Hopping [8]. Это – эвристический метод, который в результате последовательных возмущений некоторого решения улучшает его. Перед началом работы на сервер загружается набор случайных точек. Каждая из точек служит начальной для работы алгоритма Basin-Hopping на стороне пользователя. Улучшенное решение пересылается на сервер и опять заносится в список заданий, чтобы послужить новой начальной точкой. Таким образом, происходит постоянное улучшение получаемых решений.

В результате удалось создать устойчиво работающий проект. В качестве тестовой задачи была выбрана задача поиска минимума энергии молекулярного кластера Морзе [9]. Полученные решения сопоставимы с получаемыми параллельными методами на суперкомпьютере [10]. На 14 февраля 2011 года проект OPTIMA@home имеет более 2200 подключенных компьютеров. Реальная производительность проекта приближается к одному терафлопсу.

4. Проект добровольных распределенных вычислений SAT@home

4.1 SAT-задачи и сводимые к ним комбинаторные проблемы

SAT-задачи (или «задачи о булевой выполнимости») – это задачи поиска наборов, выполняющих булевы формулы, как правило, в виде конъюнктивных нормальных форм (КНФ). В своей общей постановке задача о булевой выполнимости NP-трудна, однако данная задача настолько важна для современной прикладной кибернетики, что алгоритмы, позволяющие быстро находить решения SAT-задач в различных частных случаях, являются чрезвычайно востребованными. Прогресс исследований в области алгоритмики SAT, наблюдающийся в последние 10 лет, во многом обусловлен тем фактом, что к данной задаче могут быть эффективно (за полиномиальное время) сведены обширные классы комбинаторных задач из, казалось бы, совершенно различных областей. Это и упоминавшиеся выше задачи из теории дискретных управляющих систем, и разнообразные комбинаторные задачи на графах, и задачи поиска различных экстремальных комбинаторных структур и, конечно же, криптографические задачи.

Техника сводимости различных задач к SAT представляет отдельное важное и интересное направление, хотя с чисто теоретической точки зрения эффективность соответствующих процедур есть прямое следствие теоремы С. Кука ([11]). На практике можно каждую конкретную задачу сводить к SAT особым способом либо использовать системы логического проектирования (например, [12]), либо использовать специализированные системы трансляции алгоритмических описаний в булевы уравнения (и в конечном счете в SAT) [13].

Когда SAT-задача, кодирующая некоторую комбинаторную проблему, построена, необходимо определиться с алгоритмом ее решения. На подавляющем множестве SAT-задач, имеющих реальные «прототипы», лучшие результаты показывают SAT-решатели, базирующиеся на алгоритме DPLL [14].

Параллельные SAT-решатели стали массово появляться совсем недавно, несмотря на то, что первые теоретические работы по распараллеливанию соответствующих алгоритмов были опубликованы в 90-х года прошлого века ([15]). Конкурсы параллельных SAT-решателей регулярно проводятся с 2008 года [16]. В своем подавляющем большинстве современные параллельные SAT-решатели предполагают интенсивный обмен данными между вычислительными узлами – конкретно, происходит обмен параллельно накапливаемыми булевыми ограничениями (т.н. «конфликтными дизъюнктами). Конечно же, использование таких решателей в проектах добровольных вычислений, да и вообще в грид, весьма проблематично. Тем не менее, отметим очень интересную работу [17], где описывается подход к построению распределенного SAT-решателя, обмен ограничениями в котором реализован в peer-to-peer сети, состоящей из ПК.

Другой подход к построению распределенных SAT-решателей в рамках проектов добровольных вычислений предполагает крупноблочный параллелизм [18]. В рамках данного подхода основной упор делается на исследование исходной комбинаторной задачи и некоторый (возможно, довольно трудоемкий) препроцессинг, результатом которого является построение параллельного списка заданий. Для обработки данного списка используются независимые друг от друга узлы распределенной вычислительной среды. Управление обработкой списка (рассылка заданий, получение и анализ ответов) происходит на специально выделенном для этой цели сервере.

Анализ исходной задачи и осуществление декомпозиции также может потребовать ресурсов параллельного вычислителя (как правило, для этой цели достаточно ресурсов маломощного кластера). Однако суммарные вычислительные затраты на этот этап существенно меньше тех, которые потом будут использованы для обработки построенного параллельного списка.

Весьма детально процесс декомпозиции SAT-задач, кодирующих проблемы обращения дискретных функций, описан в [19] (см. также [20]). Здесь мы приведем совсем краткое описание этой техники.

Рассматривается некоторая дискретная функция, преобразующая двоичные последовательности в двоичные последовательности, которая задается программой для детерминированной машины Тьюринга (ДМТ), имеющей полиномиальную сложность. Иными словами, речь идет о семействе функций вида

$$f_n : \{0,1\}^n \to \{0,1\}^*,$$

вычислимых некоторым полиномиальным (от n) алгоритмом A, причем для каждого $n \in \mathrm{N}$ $dom\, f_n = \{0,1\}^n$. Здесь $\{0,1\}^n$ – множество всех двоичных слов длины n, а $\{0,1\}^*$ – множество всевозможных двоичных слов произвольной конечной длины (включая, вообще говоря, пустое слово). Задача обращения функции f_n состоит в следующем: зная текст программы A и некоторое $y \in range\, f_n$, найти произвольное $x \in \{0,1\}^n$, такое, что $f_n(x) = y$.

Используя идеи С. Кука о пропозициональном кодировании алгоритмов, можно свести данную проблему к проблеме поиска выполняющего набора выполнимой КНФ. Для этой цели можно описать действие алгоритма A на всевозможных словах из $\{0,1\}^n$ в форме схемы из функциональных элементов над любым полным базисом (например, над $\{\&, \neg\}$), а затем при помощи стандартных процедур (см., например, [21]) построить по такой схеме и вектору $y \in range\, f_n$ КНФ $C(f_n)$, из произвольного выполняющего набора которой можно эффективно выделить такой $x \in \{0,1\}^n$, что $f_n(x) = y$. Таким образом, рассматриваемая задача обращения функции эффективно сведена к задаче поиска выполняющего набора КНФ $C(f_n)$, то есть к SAT-задаче.

Базовая схема крупноблочного распараллеливания полученной SAT-задачи заключается в следующем. Среди переменных КНФ $C(f_n)$ выбирается некоторое подмножество, состоящее из d переменных, после чего варьируются всевозможные их значения. Подстановка каждого

такого вектора значений истинности $\alpha \in \{0,1\}^d$ в $C(f_n)$ дает новую КНФ $C_\alpha(f_n)$, которая и является элементарным заданием параллельного списка. Всего, таким образом, в получаемом списке имеется 2^d элементарных заданий.

Если при переходе от текста алгоритма A к КНФ $C(f_n)$ использовать ряд простых правил (см. [19]), то весьма просто гарантировать, что мощность списка, каждое задание в котором является вычислительно простым, заведомо не превосходит 2^n. Как правило, эту мощность можно весьма существенно уменьшить, используя довольно естественные вычислительные процедуры, которые в совокупности составляют этап препроцессинга (см. [18]). После того как параллельный список сформирован, он загружается в грид, общие принципы архитектуры которой (клиент-сервер) были описаны во втором разделе.

Описанный подход был с успехом применен в криптоанализе ряда генераторов поточного шифрования. В 2009 году этот подход был реализован в вычислительной системе BNB-Grid (см. [22]) в применении к криптоанализу широко известного генератора ключевого потока A5/1 (описание генератора взято из работы [23]).

Сразу оговоримся, что задачи криптоанализа не являются самоцелью, а выступают в роли аргументированно трудных тестов. При этом мы исходим из того, что успешная апробация вычислительной технологии на криптографических тестах означает ее принципиальную применимость для решения задач, вычислительная трудность в которые не заложена искусственно. Успешность решения задач криптоанализа в грид-системе BNB-Grid стимулировала наши исследования в направлении построения проекта добровольных вычислений, ориентированного на решение SAT-задач. Таким проектом стал SAT@home.

4.2 Описание проекта добровольных вычислений SAT@home

29 сентября 2011 года был запущен SAT@home [24] – совместный проект Института системного анализа РАН и Института динамики систем и теории управления СО РАН. При создании проекта была использована открытая платформа BOINC [1] и пакет SZTAKI Desktop Grid [25]. С применением библиотеки DC-API [26] было создано распределенное приложение, состоящее из серверной и клиентской части. Схема работы приложения представлена на **Рис. 3**. Серверная часть отвечает за создание заданий в базе данных проекта, а также за обработку результатов выполнения заданий, присылаемых с ПК пользователей. Отправкой заданий на ПК пользователей и получением результатов занимаются стандартные службы BOINC. Для формирования заданий в серверную часть приложения были перенесены процедуры из решателя PD-SAT [27], отвечающие за декомпозицию SAT-задач.

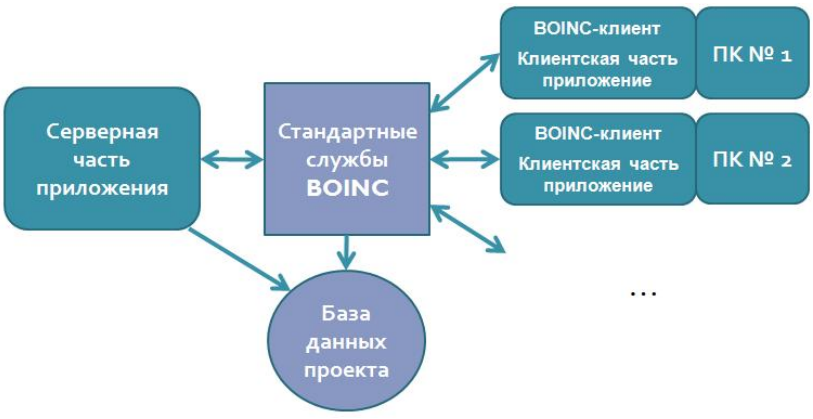


Рис. 3. Схема работы распределенного приложения в проекте SAT@home

В настоящий момент в проекте используется схема, представленная на **Рис. 4**. В соответствии с этой схемой решатель PD-SAT (запущенный на кластере Blackford ИДСТУ СО РАН [28]) получает исходную SAT-задачу (КНФ в формате DIMACS [16]) и вычисляет для нее «хорошие» параметры декомпозиции. Под «хорошими» имеются в виду параметры, найденные

в результате применения специальной техники прогнозных функций, описанной в [18]. Параметры декомпозиции включают в себя способ выбора переменных, включаемых в декомпозиционное множество, и число этих переменных. Данный этап требует относительно небольшого времени (от нескольких минут до нескольких часов). Найденные параметры передаются серверной части приложения проекта, которая осуществляет декомпозицию исходной SAT-задачи на подзадачи. Клиентскую часть приложения в виде исполняемых файлов для конкретной операционной системы запускает на ПК пользователей стандартный BOINC-клиент. Основу клиентской части приложения составляют SAT-решатели minisat-1.14.1 и minisat-2.0 [29], модифицированные с учетом особенностей КНФ, кодирующих задачи обращения дискретных функций [19]. В клиентскую часть приложения перенесены процедуры решения SAT-задач из решателя PD-SAT [27]. Полученные результаты BOINC-клиент отправляет на сервер. В результате решения в общем случае всех подзадач находится решение исходной SAT-задачи.

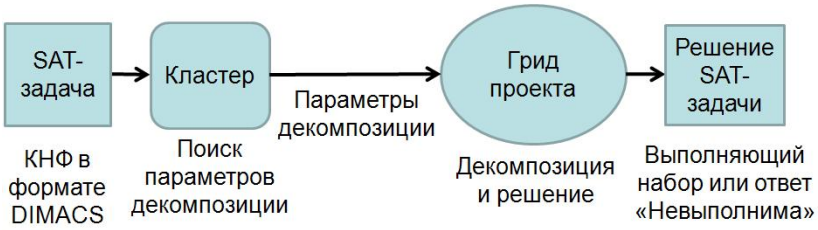


Рис. 4. Схема решения SAT-задач в проекте SAT@home

По состоянию на 14 февраля 2012 г. SAT@home имеет следующие характеристики:

- 1246 пользователей, 79 % иностранных;
- 3532 ПК, суммарно 13743 ядер процессоров, 80 % под управлением ОС Windows;
- версии клиентского приложения: windows x86, linux x86, linux x64;
- средняя реальная производительность грида проекта 1.5 терафлопс, пиковая 4,3 терафлопс.

На **Рис. 5** отображена динамика количества подключаемых к проекту ПК с 29 сентября 2011 года по 14 февраля 2012 года. Каждый столбец отображает количество подключенных ПК с момента старта проекта по конкретный день. На графике видно, что с середины октября 2011 года подключений стало значительно больше. Это объясняется тем, что 10 октября проект был добавлен на статистический сайт Free-DC [30] (содержащий информацию о проектах на платформе BOINC), а также был включен экспорт статистики, позволяющий другим статистическим сайтам добавлять проект в свои списки. Наличие информации о проекте на основных статистических сайтах – важный фактор для привлечения новых пользователей. На **Рис. 6** представлена динамика изменения реальной мощности грида проекта в гигафлопсах (с 17 января по 14 февраля 2012 года).

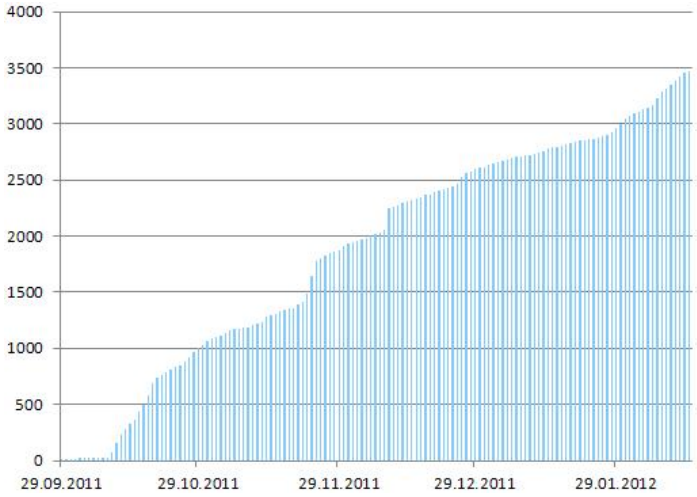


Рис. 5. Динамика количества ПК, подключенных к проекту SAT@home

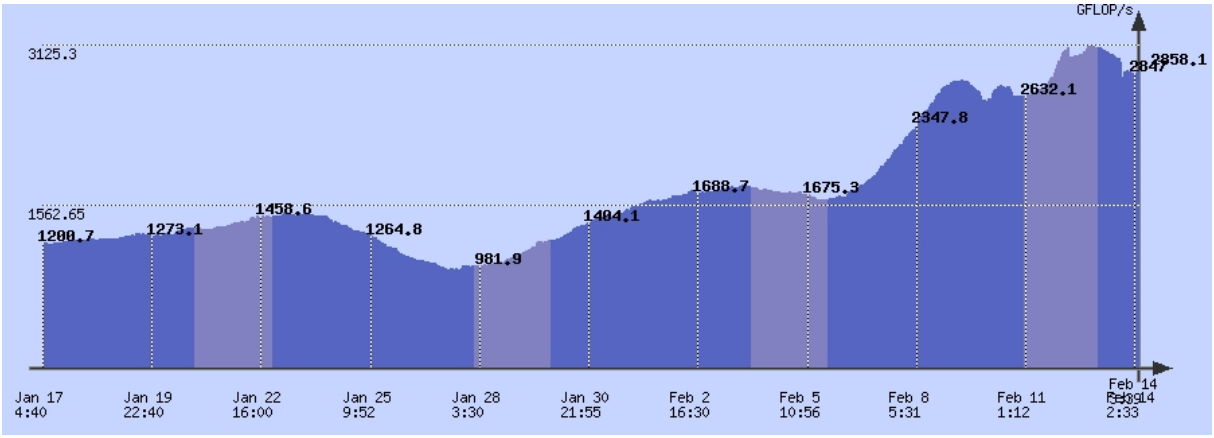


Рис. 6. Динамика изменения реальной производительности проекта (в гигафлопсах)

4.3 Результаты, полученные в проекте добровольных вычислений SAT@home

В проекте SAT@home в середине октября 2011 года запущен эксперимент, направленный на решение серии SAT-задач, кодирующих обращение функции, порождающей ключевой поток в генераторе A5/1 (длина инициализирующей последовательности 64 бита, анализируется фрагмент ключевого потока длиной 114 бит, соответствующий одному фрейму ключевого потока [31]). На сегодняшний день наиболее эффективным методом криптоанализа A5/1 является т.н. «rainbow-метод» [32]. Реализованные при этом rainbow-таблицы [32] покрывают ключевое пространство примерно на 88% (при реалистичных условиях криптоанализа). На текущей момент в проекте SAT@home решаются 10 примеров криптоанализа шифра A5/1, которые не покрываются этими rainbow-таблицами. Декомпозиция SAT-задачи, кодирующей криптоанализ A5/1, на семейство подзадач проводится по 31 переменной (структура декомпозиционного множества описана в [20]). Таким образом, для решения одного примера криптоанализа требуется решить 2^{31} (более 2 миллиардов) относительно простых подзадач, которые объединяются в пакеты по 32768 подзадачи в каждом. Каждый такой пакет является пользовательским заданием. На решение одного задания требуется обычно не более 4 часов работы одного ядра среднего по мощности процессора. На данный момент в SAT@home успешно решены пять SAT-задач из описанной выше серии. Эти результаты можно найти в разделе «Найденные решения» сайта проекта [24].

Следует отметить, что в процессе работы проектов добровольных вычислений целесообразно организовывать командные соревнования среди пользователей. За время работы проект SAT@home было организовано два таких соревнования на сайте BOINCStats [33]. На период каждого из этих соревнований производительность проекта возрастала примерно в 2 раза в сравнении с его средней производительностью. Это можно проследить на графике производительности проекта (с 10 февраля 2012 года началось семидневное соревнование).

Заключение

Концепция добровольных вычислений идеально подходит для реализации вычислительных экспериментов, привлекающих существенные ресурсы распределенных вычислителей на протяжении длительного времени (месяцы и даже годы). Описанные в работе технологии могут использоваться для организации специализированных проектов, ориентированных на решение крупномасштабных научных задач. Ключевые пункты таких технологий проиллюстрированы на примере двух реально работающих проектов – OPTIMA@home и SAT@home. Отметим, что проект SAT@home является на текущий момент единственным действующим на территории СНГ проектом добровольных распределенных вычислений, входящим в список активных проектов, составленный разработчиками платформы BOINC (внесен 7 февраля 2012 года).

Литература

1. Платформа BOINC для организации распределенных вычислений URL: http://boinc.berkeley.edu/ (дата обращения: 14.02.2012).
2. Рейтинг и описание 500 самых мощных общественно известных суперкомпьютеров URL: http://www.top500.org/ (дата обращения: 14.02.2012).
3. Formula BOINC - статистический сайт проектов добровольных вычислений URL: http://formula-boinc.org/ (дата обращения: 14.02.2012).
4. Проект распределенных добровольных вычислений OPTIMA@home URL: http://boinc.isa.ru/dcsdg/ (дата обращения: 14.02.2012).
5. Evtushenko Y., Posypkin M., Sigal I. A framework for parallel large-scale global optimization // Computer Science - Research and Development. 2009. V. 23. Issue 3-4. P. 211-215.
6. Marosi A. C., Balaton Z., Kacsuk P. GenWrapper: A Generic Wrapper for Running Legacy Applications on Desktop Grids // Third Workshop on Desktop Grids and Volunteer Computing Systems, 2009, Rome, Italy, IEEE.
7. Проект GenWrapper URL: http://genwrapper.sourceforge.net/ (дата обращения: 14.02.2012).
8. Leary R.H. Global Optima of Lennard-Jones Clusters // Journal of Global Optimization. 1997. Vol. 11, Issue 1. P. 35-53.
9. Longjiu Cheng, Jinlong Yang. Global Minimum Structures of Morse Clusters as a Function of the Range of the Potential: 81 <= N <= 160 // Journal of Physical Chemistry A. 2007. vol. 111. P. 5287-5293.
10. Посыпкин М.А. Методы и распределенная программная инфраструктура для численного решения задачи поиска молекулярных кластеров с минимальной энергией // Вестник нижегородского университета им. Н.И. Лобачевского. 2010. № 1. С. 210-219.
11. Cook S.A. The complexity of theorem-proving procedures // Third annual ACM symposium on Theory of computing, 1971, Ohio, USA. ACM. 1971. P. 151-159.
12. Thomas D., Moorby P. The Verilog hardware description language. Springer, 2002. 381 p.
13. Отпущенников И.В., Семенов А.А. Технология трансляции комбинаторных проблем в булевы уравнения // Прикладная дискретная математика. 2011. № 1. С. 96–115.
14. Davis M., Longemann G., and Loveland D. A machine program for theorem proving // Communication of the ACM. 1962. Vol. 5, Issue 7. P. 394-397.
15. Bohm M., Speckenmeyer E. A fast parallel SAT solver - efficient workload balancing // Annals of Mathematics and Artificial Intelligence. 1996. Vol. 17, No. 2, P. 381-400.
16. SATLive! – современные ссылки по задаче SAT URL: http://www.satlive.org/ (дата обращения: 14.02.2012).
17. Schulz S., Blochinger W. Parallel SAT Solving on Peer-to-Peer Desktop Grids // Journal Of Grid Computing. 2010. Vol. 8 No. 3. P. 443-471.
18. Заикин О.С., Семенов А.А. Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. № 1. С. 43–50.
19. Semenov A., Zaikin O., Bespalov D., Posypkin M. Parallel algorithms for SAT in application to inversion problems of some discrete functions. arXiv:1102.3563v1 [cs.DC].
20. Semenov A., Zaikin O., Bespalov D., Posypkin M. Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // Lecture Notes in Computer Science. 2011. Vol. 6873. P. 473-483.

21. Prestwich S. CNF encodings. In Handbook of Satisfiability (editors: A. Biere, M.Heule, H. van Maaren, T. Walsh). 2009. IOS Press. P. 75-97.

22. Посыпкин М.А., Заикин О.С., Беспалов Д.В., Семенов А.А. Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды ИСА РАН. 2009. Т. 46. С. 119-137.

23. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Seventh Fast Software Encryption Workshop, 2000, New York, USA. Springer-Verlag, 2000, P. 1-18.

24. Проект распределенных добровольных вычислений SAT@home
URL: http://sat.isa.ru/pdsat/ (дата обращения: 14.02.2012).

25. Kacsuk P., Kovács J., Farkas Z., Marosi A. Cs., Gombás G., Balaton Z. SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System // Journal Of Grid Computing. 2009. Vol. 7, No. 4. P. 439-461.

26. Balaton Z., Gombas G., Kacsuk P., Kornafeld A., Kovacs J., Marosi A. C., Vida G., Podhorszki N., Kiss T. Sztaki desktop grid: a modular and scalable way of building large computing grids // 21th International Parallel and Distributed Processing Symposium, 2007, Long Beach, California, USA. P. 1-8.

27. Заикин О.С. Реализация процедур прогнозирования трудоемкости параллельного решения SAT-задач // Вестник УГАТУ. 2010. Т. 14, № 4. С. 210-220.

28. Суперкомпьютерный центр ИДСТУ СО РАН
URL: http://www.mvs.icc.ru (дата обращения: 14.02.2012).

29. Решатель MiniSat
URL: http://minisat.se/MiniSat.html (дата обращения: 14.02.2012).

30. Free-DC - статистический сайт проектов добровольных вычислений
URL: http://www.free-dc.org/ (дата обращения: 14.02.2012).

31. Barkan E., Biham E., Keller N. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication // Proceedings in Crypto 2003. P. 600-616.

32. A5/1 Cracking project
URL: http://reflextor.com/trac/a51/wiki (дата обращения 14.02.2012).

33. BOINCStats
URL: http://ru.boincstats.com/ (дата обращения 14.02.2012).

34. Список активных BOINC-проектов
URL: http://boinc.berkeley.edu/projects.php (дата обращения 14.02.2012).

Качественное построение расчетной сетки для решения задач аэродинамики в программном комплексе

П.И. Карасев, А.С. Шишаева, А.А. Аксенов

ООО «ТЕСИС»

В работе исследовано влияние адаптации сетки в различных областях течения газа около крылового профиля на результаты решения в программном комплексе FlowVision. На основе этого исследования даны рекомендации по выбору критериев адаптации расчетной сетки для эффективного решения задач аэродинамики. Выводы, сделанные в данной статье, могут быть распространены на другие задачи внешней аэродинамики, которые исследуют обтекание объектов более сложных, чем крыловой профиль.

1. Введение

Одним из важнейших этапов при численном моделировании движения жидкости или газа является построение расчетной сетки. Качественная расчетная сетка должна быть достаточно мелкой для того чтобы выявлять в процессе проведения расчета все основные особенности течения и, таким образом, обеспечивать достаточную точность результатов расчетов. В то же время она не должна содержать слишком большое количество ячеек, так как в этом случае увеличиваются требования к ресурсам компьютера. Необходимо также отслеживать сходимость по сетке. Проект считается сошедшимся по сетке, когда результат моделирования не изменяется при дальнейшем измельчении сетки. При этом наиболее экономично задавать подробную сетку в областях высоких градиентов переменных, и более грубую – в областях, где они малы.

Области, где необходимо задавать подробную сетку, можно задавать вручную или определять автоматически, исходя из значений переменных. Первый метод имеет ряд существенных недостатков. Во-первых, не всегда заранее известно, где будут расположены зоны высоких градиентов и насколько мелкая сетка потребуется для их разрешения. Во-вторых, при сложных трехмерных течениях ручное задание областей измельчения сетки либо потребует создания сложных геометрических объектов, точно воспроизводящих соответствующую область, либо, при использовании простых объектов, приведет к избыточному измельчению сетки.

Цель данного исследования - получение алгоритма, позволяющего создавать оптимальную расчетную сетку. Исследование проводится для задач дозвукового и околозвукового обтекания известных крыловых профилей в программном комплексе FlowVision, предназначенном для численного моделирования течений жидкости и газа.

2. Особенности расчетной сетки во FlowVision

Во FlowVision применяется неструктурированная локально-адаптивная сетка с подсеточным разрешением геометрии с преобладанием шестигранных ячеек [1]. Адаптация подразумевает разбиение исходной ячейки пополам по каждому направлению. Адаптация может быть разного уровня. Уровень адаптации – это количество разбиений исходной ячейки (см. Рис. 1). Деление ячеек на равные части позволяет избежать появления вырожденных и слишком вытянутых ячеек.

.

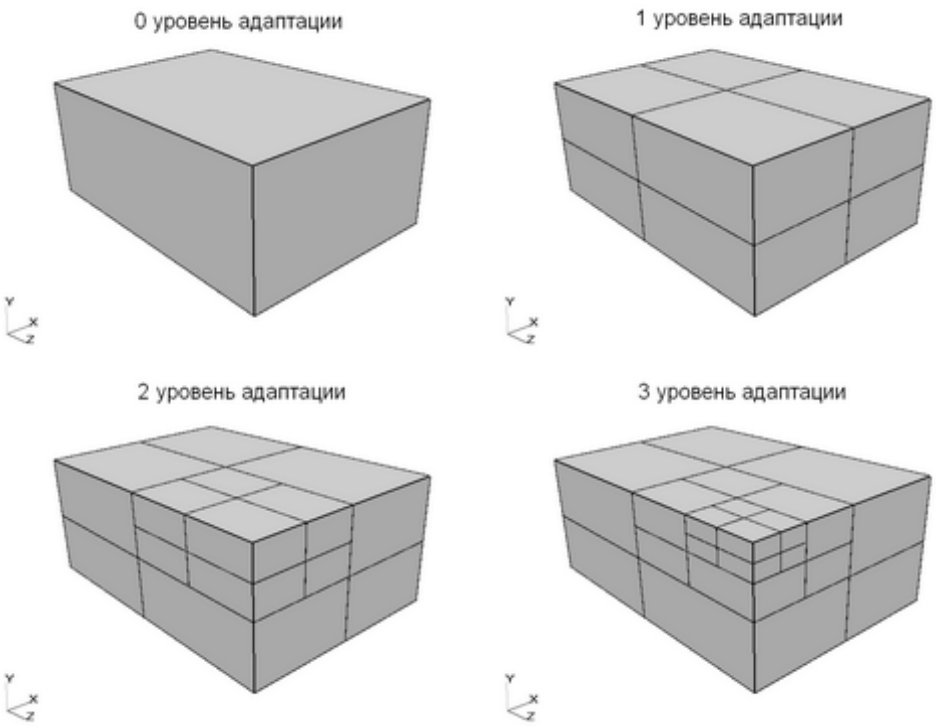


Рис. 1. Уровни адаптации в FlowVision

Вблизи границы расчетной области из ячеек начальной сетки вычитается часть геометрии, не принимающая участия в расчете. При этом расчетные ячейки принимают форму произвольных многогранников (Рис. 2). Такая сетка называется сеткой «с подсеточным разрешением» [1].

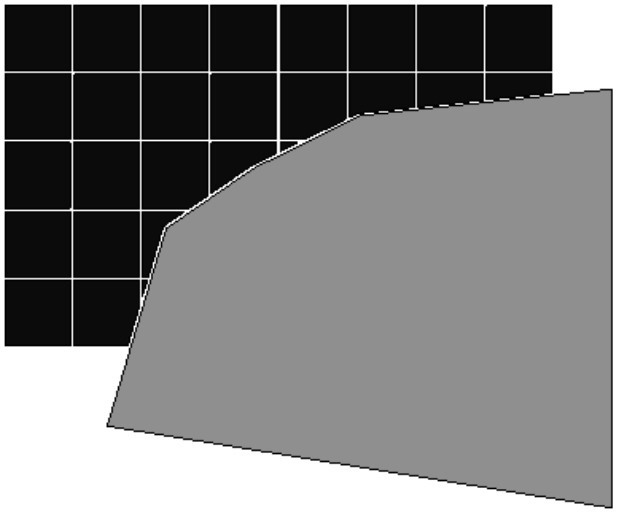


Рис. 2. Подсеточное разрешение геометрии.

3. Выделение зон течения около крылового профиля для определения относительной важности их разрешения расчетной сеткой

Для оценки степени влияния разрешения различных областей течения расчетной сеткой условно выделим несколько зон рядом с профилем крыла (Рис. 3).

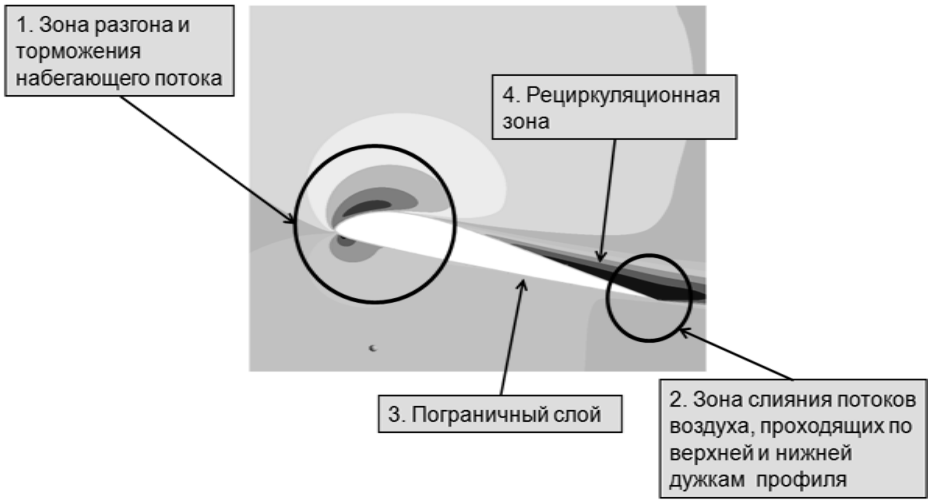


Рис. 3. Выделенные зоны около профиля.

3.1 Зона разгона и торможения набегающего потока.

В этой зоне поток, набегающий на профиль, тормозится в критической точке, расположенной рядом с передней кромкой. В критической точке поток разделяется. При удалении от нее скорость воздуха повышается из-за кривизны профиля, которая вызывает поджатие и, следовательно, ускорение потока воздуха. Так как поток, текущий по верхней части профиля испытывает большее сжатие, сильнее ускоряется и, соответственно, согласно закону Бернулли давление в нем падает сильнее, чем в потоке, текущем вдоль нижней части профиля. За счет этой зоны и возникает разность давлений, создающая подъемную силу.

3.2 Зона слияния потоков воздуха, проходящих по верхней и нижней дужкам профиля.

Эта зона также характеризуется значительными градиентами скорости и давления, особенно заметными при наличии срыва потока. Поведение потока в этой области влияет на распределение давления по всему профилю вверх по потоку и на положение скачка уплотнения в случае трансзвукового обтекания профиля. Часто задняя кромка профиля бывает срезана и за ней образуется отрывная зона как показано ниже (см. Рис. 4)

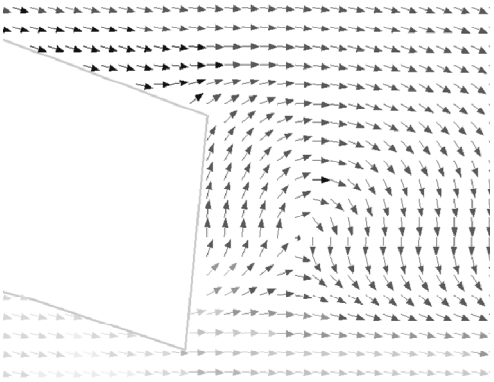


Рис. 4. Рециркуляционная зона при срезанной задней кромке профиля.

3.3 Пограничный слой

Разрешение расчетной сеткой пограничного слоя рядом с обтекаемым объектом критически важно при решении аэродинамических задач.

3.4 Зона отрыва потока

Сильное влияние на характеристики профиля оказывает наличие отрыва потока и его положение. Вихри, образующиеся при отрывах потока очень чувствительны к точности разбиения расчетной сетки.

3.5 Скачок уплотнения

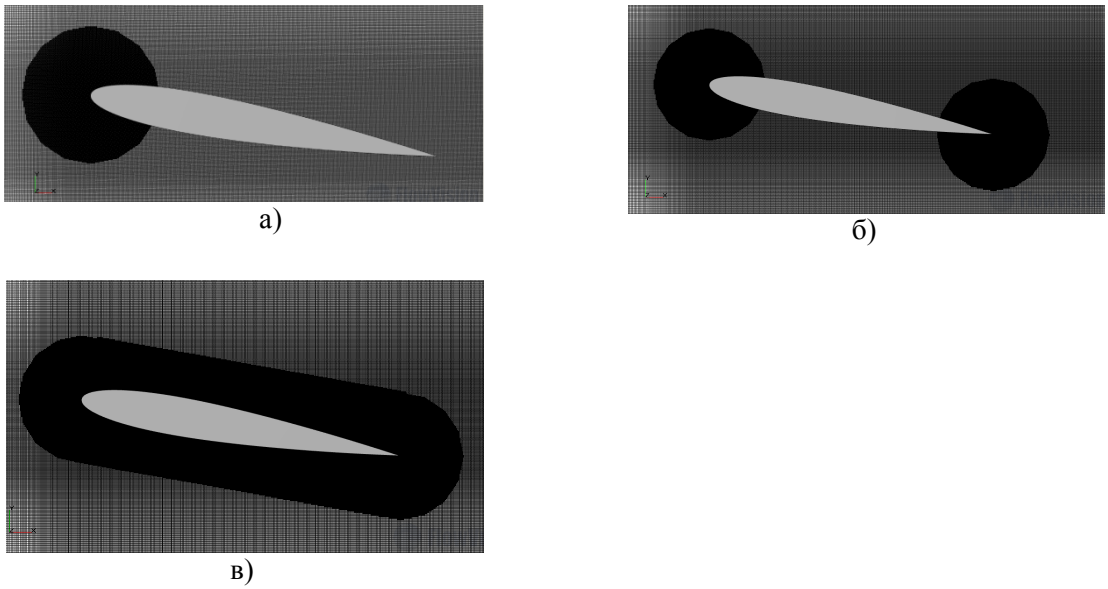
При околозвуковом обтекании профиля основное влияние на его характеристики оказывает положение скачков уплотнения рядом с верхней и нижней дужками профиля, поэтому область вокруг него можно выделить в отдельную зону (на рисунке данная зона не показана).

4. Исследование влияния на результат решения адаптации расчетной сетки в выделенных зонах около профиля NACA 0012.

Для выявления важности адаптации в выделенных зонах (Рис. 3) при безотрывном дозвуковом режиме обтекания профиля выбран профиль NACA 0012 - симметричный тонкий профиль, установленный под углом атаки 10 градусов.

Проведены расчеты для установления степени влияния на решение разрешения сеткой областей 1 и 2. Сравнение проводится с экспериментальными данными, взятыми из [2].

После предварительного расчета на начальной сетке были проведены расчеты с адаптацией в различных зонах (Рис. 5).



а) б) в)

Рис. 5. Расчетные сетки для исследования влияния адаптации различных областей около крылового профиля NACA 0012 на результаты решения: с адаптацией рядом с передней кромкой профиля (а), с адаптацией рядом с передней и задней кромками (б), с адаптацией вокруг всего крылового профиля (в).

Результаты расчетов с показанной выше адаптацией сведены в таблицу (Таблица 1).

Таблица 1. Сравнение результатов численного эксперимента для профиля NACA 0012:

	Значения			Погрешность относительно экспериментальных данных, %			Отличие погрешности от погрешности предыдущего расчета, %		
Способы адаптации	C_X	C_Y	m_Z	ΔC_X	ΔC_Y	Δm_Z	ΔC_X	ΔC_Y	Δm_Z
без адаптации	0.0298	1.007	0.01175	54.5	-1.3	9.8			
рядом с передней кромкой	0.0259	1.043	0.00974	34.3	2.3	-8.9	13.10	-3.62	17.09
рядом с передней и задней кромками	0.0258	1.037	0.01128	33.7	1.7	5.4	0.46	0.61	-15.75
вокруг всего крылового профиля	0.0255	1.044	0.01070	31.9	2.3	0.0	1.31	-0.67	5.12

Получено, что на значение сил, действующих на профиль, наибольшее влияние оказывает измельчение сетки рядом с передней кромкой (зона 1), а на значение момента помимо этого сильно влияет адаптация рядом с задней кромкой профиля (зона 2).

5. Исследование влияния на результат решения адаптации расчетной сетки в выделенных зонах около профиля ЦАГИ Р-IIIА-15.

Также был проведен численный эксперимент с адаптацией на профиле ЦАГИ Р-IIIА-15 под углом атаки 16^{o} (Рис. 6). На данном угле атаки образуется отрыв потока.

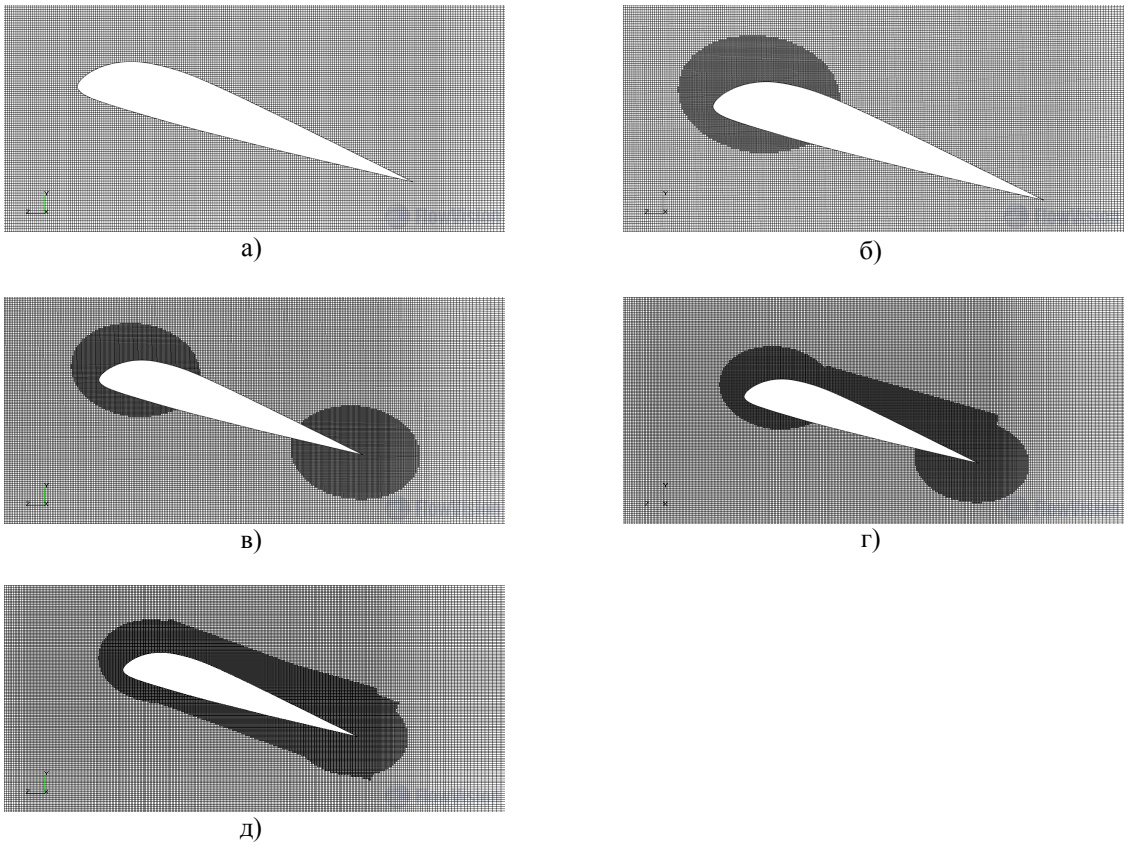


Рис. 6. Расчетные сетки при исследовании влияния адаптации различных областей около крылового профиля ЦАГИ Р-IIIА-15 на результаты решения: без адаптации (а), адаптация по передней кромке (б), адаптация по передней и задней кромке (в), адаптация по передней кромке, задней кромке и зоне отрыва потока (г), адаптация вокруг всего профиля (д).

Таблица 2. Сравнение результатов численного эксперимента для профиля ЦАГИ Р-IIIА-15:

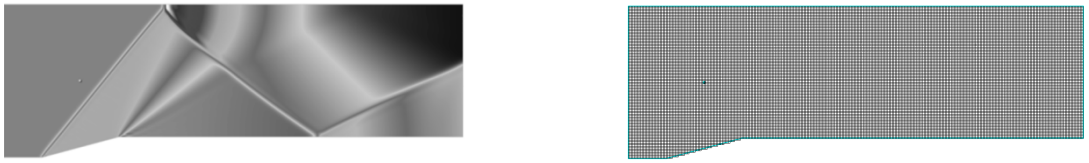
	Значения		Погрешность относительно эксперимента, %		Отличие погрешности от погрешности предыдущего расчета, %	
Способы адаптации	Cx	Cy	Cx	Cy	Cx	Cy
без адаптации	0.07	1.50	52.61	-14.65		
адаптация рядом с передней кромкой	0.09	1.38	38.48	-5.54	14.13	-9.11
адаптация рядом передней и задней кромками	0.09	1.39	38.23	-5.85	0.26	0.31
адаптация по передней кромке, задней кромке и зоне отрыва потока	0.09	1.36	34.36	-3.51	3.87	-2.34
адаптация вокруг всего профиля	0.09	1.35	34.75	-3.15	-0.39	-0.36

Экспериментальные данные, сравнение с которыми производится выше (Таблица 2) , взяты из [3]. Получено, что разрешение зоны отрыва существенно влияет на результаты решения.

6. Особенности адаптации к решению во FlowVision

Во FlowVision существует возможность автоматически адаптировать расчетную сетку в зависимости от значений и градиентов различных переменных в соответствующих ячейках. Она называется ***«адаптацией к решению».*** Уровень адаптации и максимальное количество ячеек расчетной сетки задается пользователем. Можно задавать разбиение по нескольким критериям. Вклад критерия в общее количество проадаптированных ячеек определяется весовым коэффициентом. При этом, все критерии адаптируют сетку только до одного определенного уровня.

Рассмотрим особенности адаптации во FlowVision на примере простейшей задачи, сверхзвуковом обтекании клина (см. Рис. 7).



а) б)

Рис. 7. Распределение давления а) и начальная сетка б) при сверхзвуковом обтекании клина.

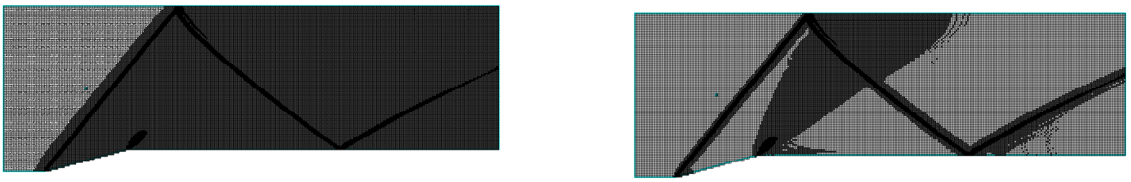
Проведем адаптацию по градиенту давления до первого уровня (Рис. 8а).

а) б)

Рис. 8. Расчетная сетка при адаптации по градиенту давления первым а) и вторым б) уровнем

При изменении уровня адаптации до второго все ячейки первого уровня сливаются, и остается только адаптация второго уровня (Рис. 8б). При увеличении уровня адаптации к решению и сохранении общего количества расчетных ячеек проадаптированные ячейки занимают меньший объем.

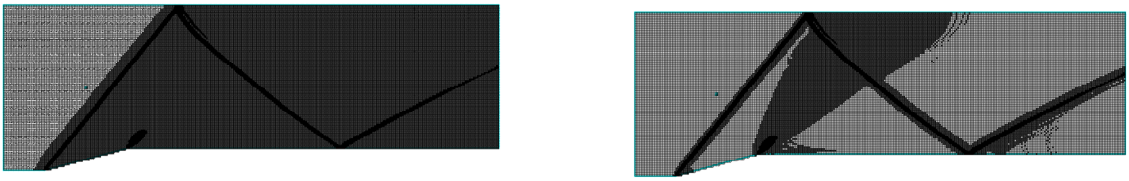
Если снова задать адаптацию первого уровня и то же количество ячеек расчетной сетки ячейки второго уровня слиты не будут. Сетка сохранит вид, показанный выше (Рис. 8б). Если количество ячеек будет увеличено, области течения с меньшим градиентом будут проадаптированы до первого уровня (



а) б)

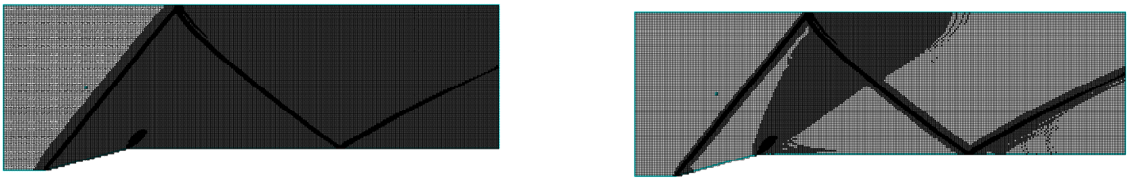
Рис. 9а). При отсутствии градиента в какой-либо зоне расчетной области адаптация в ней проходить не будет.

Если уменьшить количество ячеек, выделенных под адаптацию к решению, часть проадаптированных ячеек в зонах с малыми градиентами или вдали от выбранного значения переменной сливается (



а) б)

Рис. 9б). Важной особенностью является то, что создание адаптации к решению происходит за 1-2 шага, слитие же занимает значительно большее время.



а) б)

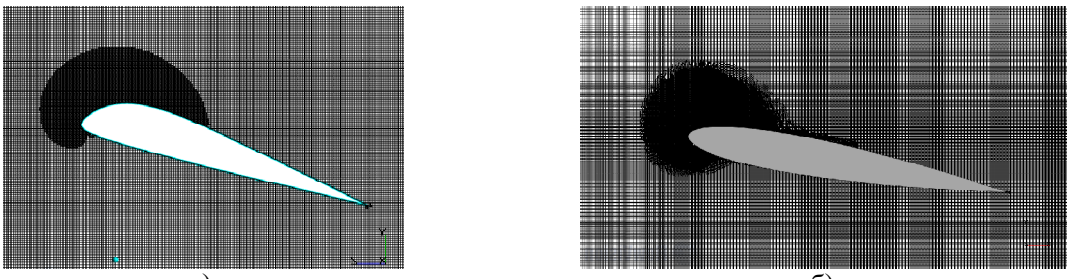
Рис. 9. Расчетная сетка при добавлении к предыдущей адаптации первого уровня а) и при уменьшении количества ячеек, использующихся при решении б).

7. Адаптация к решению для дозвукового обтекания профиля

Картина дозвукового обтекания профиля проще, чем картина трансзвукового обтекания. Поэтому разработку алгоритма применения адаптации к решению логично начать с дозвукового обтекания. Применение различных критериев адаптации при расчете дозвукового обтекания рассматривается на примере моделирования обтекания профилей ЦАГИ Р-IIIА-15 и NACA 0012.

7.1 Адаптация к решению по градиенту давления.

В результате расчетов получено, что адаптация по градиенту давления позволяет разрешать сеткой области разгона и торможения потока (см. Рис. 10). Эти области оказывают наибольшее влияние на результаты при дозвуковом обтекании (Таблица 1). Следовательно, адаптацию по градиенту давления целесообразно использовать при моделировании дозвукового обтекания.



а) б)

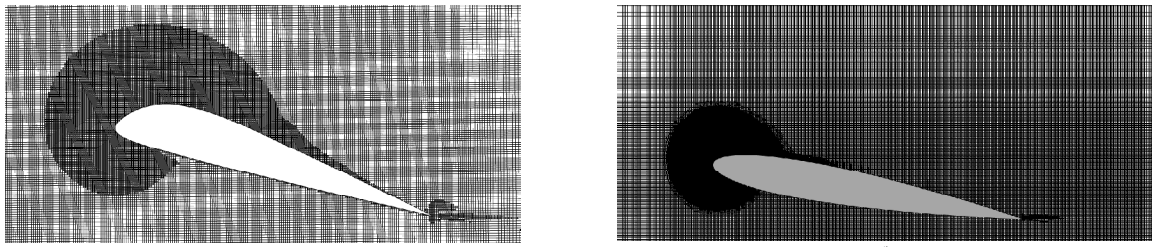
Рис. 10. Внешний вид сечения расчетной сетки при проведении адаптации к решению с использованием градиента давления а) для ЦАГИ Р-IIIА-15, б) для NACA 0012.

7.2 Адаптация к решению по градиенту скорости.

При использовании адаптации по градиенту скорости получено, что адаптация по этому критерию позволяет разрешать сеткой области разгона, торможения, пограничного слоя и слияния потока (см. Рис. 11). Однако область слияния потока адаптируется по остаточному

принципу. Пограничный слой может разрешаться частично, что может приводить к погрешностям в решении (см. Рис. 12).

По вышеперечисленным причинам использование адаптации по градиенту скорости нецелесообразно.



а) б)

Рис. 11. Внешний вид сечения расчетной сетки при адаптации к решению с использованием градиента скорости а) для ЦАГИ Р-IIIА-15, б) для NACA 0012.

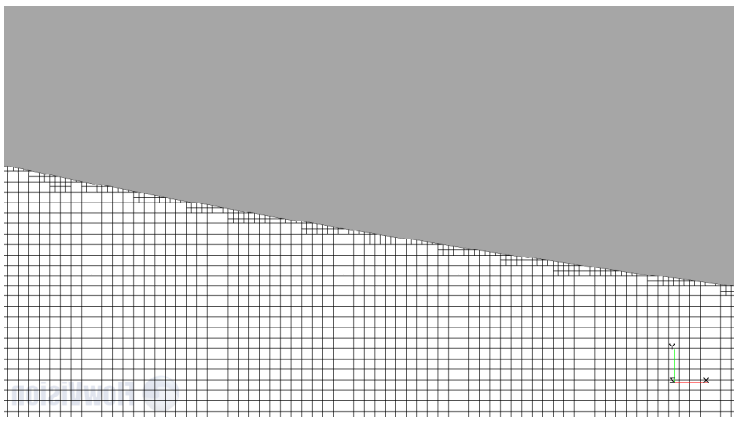
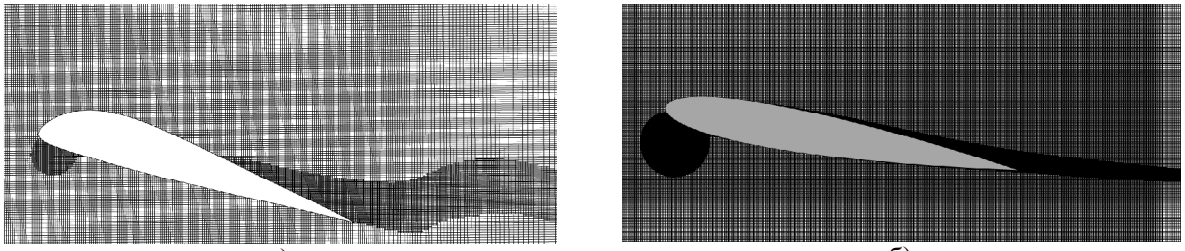


Рис. 12. Внешний вид сечения расчетной сетки рядом с профилем NACA 0012.

7.3 Адаптация к решению по значению скорости.



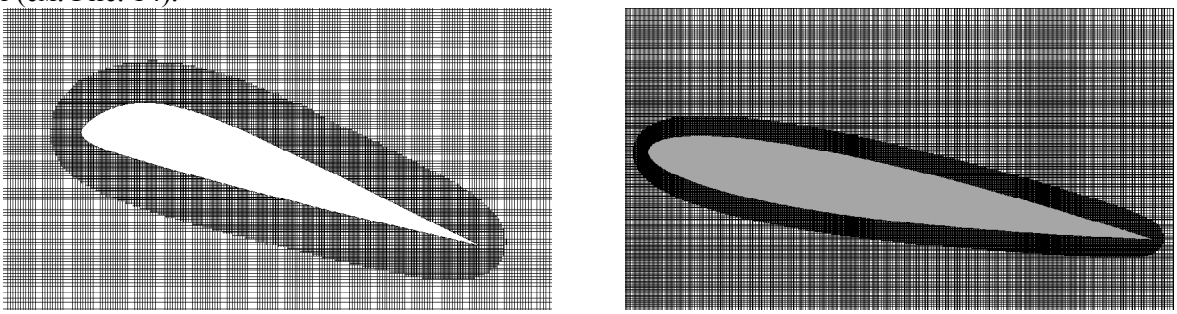
а) б)

Рис. 13. Сечение расчетной сетки при адаптации по значению скорости а) для ЦАГИ Р-IIIА-15 , б) для NACA 0012.

В результате применения адаптации по нулевому значению скорости при моделировании обтекания профиля ЦАГИ Р-IIIА-15 преимущественно были проадаптированы ячейки в зоне отрыва потока. При моделировании обтекания профиля NACA 0012, - около критической точки и в области замедления потока за профилем (см. Рис. 13). Получено, что адаптация по нулевому значению скорости позволяет разрешать сеткой области отрыва потока и критической точки. Следовательно, использование адаптации по значению скорости обосновано, если имеет место отрыв потока.

7.4 Адаптация к решению по расстоянию до стенки.

Адаптация по расстоянию до стенки позволяет разрешать пограничный слой около профиля (см. Рис. 14).



а) б)

Рис. 14. Сечение расчетной сетки при адаптации по значению расстояния от стенки а) для ЦАГИ Р-IIIА-15 , б) для NACA 0012.

8. Адаптация к решению для трансзвукового обтекания профиля.

Течение при трансзвуковом обтекании профиля отличается от течения при дозвуковом обтекании наличием скачка уплотнения. Его положение определяет значения лобового сопротивления и подъемной силы для этих профилей. В остальном трансзвуковое течение вокруг профиля сходно с рассмотренным выше дозвуковым течением. Применение различных критериев адаптации расчете трансзвукового обтекания рассматривается на примере моделирования обтекания профиля NACA 0012.

8.1 Адаптация к решению по градиенту давления.

Было проведено моделирование с использованием адаптации по градиенту давления. Получено, что она позволяет разрешать области скачка, разгона и торможения потока (см. Рис. 15). Следовательно, адаптацию по градиенту давления целесообразно использовать при моделировании трансзвукового обтекания.

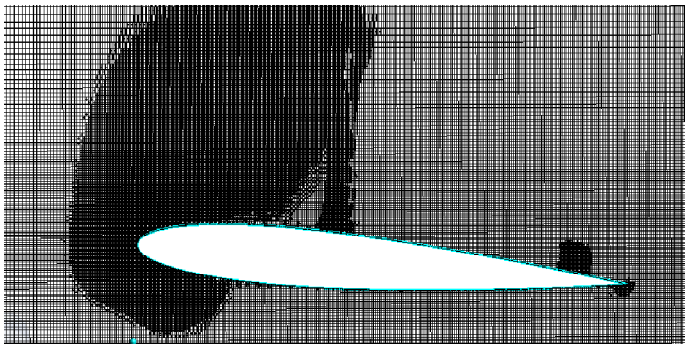


Рис. 15. Внешний вид сечения расчетной сетки при проведении адаптации к решению с использованием градиента давления для NACA 0012 при трансзвуковом режиме обтекания.

8.2 Адаптация к решению использованием градиента скорости.

При использовании адаптации по градиенту скорости, получено, что она также позволяет разрешать области скачка, разгона и торможения потока (см. **Ошибка! Источник ссылки не найден.**). Однако, поскольку эти области также хорошо разрешаются адаптацией по градиенту давления, совместное использование адаптации по градиенту давления и градиенту скорости нецелесообразно.

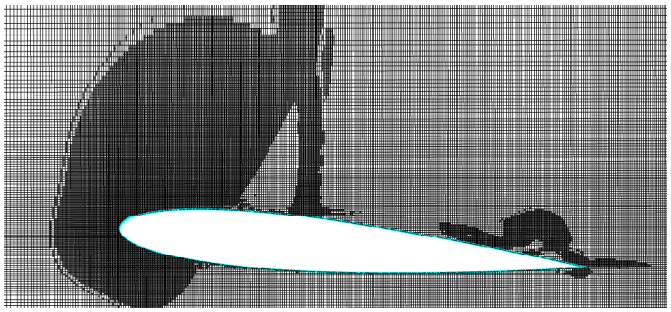


Рис. 16. Внешний вид сечения расчетной сетки при проведении адаптации к решению с использованием градиента скорости для NACA 0012 при трансзвуковом режиме обтекания.

Необходимости добавлять для трансзвуковых профилей критерии адаптации, отличные от использовавшихся для дозвуковых профилей нет.

9. Комбинация критериев адаптации.

Из вышеизложенного следует, что для создания оптимальной расчетной сетки недостаточно использовать один критерий адаптации к решению. Необходима комбинация критериев адаптации. При безотрывном обтекании профиля целесообразно использовать адаптацию по градиенту скорости и адаптацию по расстоянию до стенки (см. Рис. 17). При обтекании профиля с отрывом потока целесообразно добавлять к ним адаптацию по нулевому значению скорости. Для комбинации критериев адаптации предусмотрено задание весового коэффициента, который определяет, сколько ячеек от общего количества будет адаптировано с использованием этого критерия.

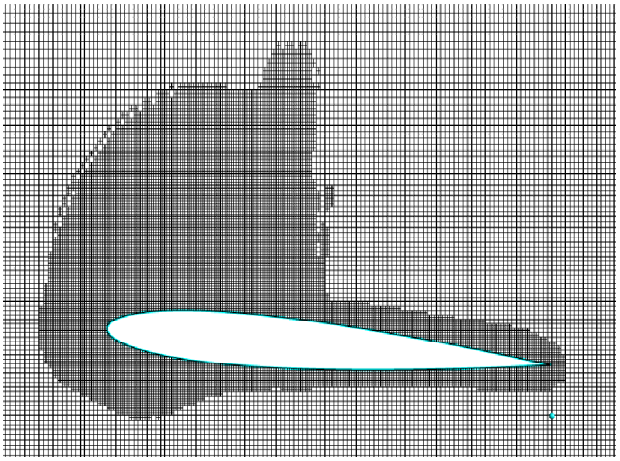


Рис. 17. Расчетная сетка при комбинации адаптации по градиенту давления и значению расстояния от стенки с весовыми коэффициентами 0.5 для каждого критерия.

Заключение

В результате проведенного исследования проанализирована применимость адаптации по различным критериям для разных типов течения, создан алгоритм, позволяющий разрешать сеткой наиболее важные для решения области течения, и сформулированы рекомендации по применению адаптации к решению.

Литература

1. *A. Aksenov, A.A. Dyadkin, V.I. Pokhilko* Overcoming of Barrier between CAD and CFD by Modified Finite Volume Method // Proc. of "1998 ASME Pressure Vessels and Piping Division Conference", San Diego, ASME PVP, 1998. V. 377-1.
2. *N. Gregory and P.G. Wilby* NPL 9615 and NACA 0012. A Comparison of Aerodynamic Data // Ministry of Defense, Aeronautical Research Council, London: Her Majesty's Stationery office // C.P. No. 1261, 1973. 56 p.
3. *С. Т. Кашафутдинов, В. Н. Лушин* Атлас аэродинамических характеристик крыловых профилей // Сибирский НИИ Авиации им. С. А. Чаплыгина, 1994. 74 с.

Статический анализ последовательных программ в системе автоматизированного распараллеливания САПФОР*

Н.А. Катаев

Московский Государственный Университет имени М.В. Ломоносова, Институт Прикладной Математики имени М.В. Келдыша РАН

Для использования последовательных языков программирования при создании параллельной программы необходимо наличие эффективных средств анализа. В статье описывается подсистема анализа, входящая в САПФОР. Разработка подсистемы осуществляется в три этапа. В статье представлены результатырешения задачи анализа скалярных приватных переменных. Приводится описание структуры подсистемы анализа, представлены разработанные алгоритмы анализа, используемые в подсистеме анализа.

1. Введение

Развитие суперкомпьютерной отрасли идет очень быстрыми темпами. Эффективное использование все возрастающих мощностей наталкивается на значительные трудности. В японском суперкомпьютереKComputer, занимающем первое место в списке Top500[1] и показавшем на тесте Linpackпроизводительность свыше 10 ПФлопс, используется 705 024 вычислительных ядер. Разрабатывать и отлаживать прикладное программное обеспечение, максимально использующее ресурсы систем такого уровня, очень сложно.

За последнее время к широко распространённым и давно развивающимся технологиям параллельного программирования (MPI, SHMEM, DVM,OpenMP) добавились низкоуровневые технологиидля гетерогенных систем (CUDA, OpenCL). Использование этих технологий требует четкого понимания“информационной структуры” [2] разрабатываемой программы, то есть связи различных элементов программы между собой. Такими элементами чаще всего являются операции. В совокупности со связями между ними операции образуют различные графовые модели программы. Такие модели оказываются достаточно сложными, и изучение их без специализированных средств тяжелая задача для разработчиков.

Часто параллельные программы создаются на основе ранее существовавших последовательных программ.В этом случае сложность изучения информационной структуры может усугубиться отсутствием разработчиков исходных кодов. Трудности возникают, когда задача по распараллеливанию решается сторонней организацией.

В таких случаях становятся незаменимы автоматизированные средства создания параллельных программ. Такие системы обращаются за помощь к программистам лишь при крайней необходимости. Неотъемлемой частью таких систем являются подсистемы анализа программ.

Анализ и преобразование программ в распараллеливающих системах может выполняться как в полностью автоматическом режиме, так и в режиме диалога с пользователем.

К полностью автоматическим средствам относятся автоматически распараллеливающие компиляторы, поставляемые компаниями Intel, Microsoft, SunMicrosystems, IBMи др. Другим примером является автоматический распараллеливатель, реализованный компанией Оптимизирующие технологии [3] в рамках компилятора GCC. В данном случае распараллеливание производится с использованием технологии OpenMPи Универсальной Библиотеки Трансляции (УБТ)[3]в рамках которой реализованы алгоритмы анализа и оптимизации.

К диалоговым распараллеливающим системам относятся такие комплексы, как ParaWise[4], ДВОР [5], САПФОР[6]. Подсистема анализа программ, разрабатываемая в рамках данного исследования, входит в состав САПФОР.

*Работа поддержана Программами президиума РАН №14, №15 и №17, грантом РФФИ № 10-07-00211.

2. САПФОР

Система Автоматизированного Распараллеливания Фортран Программ (САПФОР) разрабатывается в Институте Прикладной Математики им. М.В. Келдыша РАН. Входным языком системы является Fortran, а результат распараллеливания представляет собой программу на языкеFortranOpenMP, FortranDVM/OpenMPили FortranDVMH[7, 8, 9].Все эти языки являются расширением стандартного языка Fortran 95 директивами параллелизма. Высокий уровень выходного языка позволяет продемонстрировать пользователю результат распараллеливания в понятных для него терминах, кроме того директивы распараллеливания могут быть проигнорированы компиляторами, не поддерживающими соответствующий язык. Для данных языков существуют развитые средства отладки функциональности и эффективности.

Отличительной особенностью САПФОР является использование автоматически распараллеливающего компилятора[10], преобразующего потенциальнопараллельную программу в ее параллельную версию для заданной ЭВМ. При этом предварительный анализ программы может выполняться в полуавтоматическом режиме. Для уточнения свойств последовательной программы, выявленных анализатором, используются либо диалоговая оболочка [11], либо специальные аннотации в тексте программы. В САПФОР наиболее сложный этап распараллеливания (получение параллельной версии программы) выполняется полностью автоматически.

Основными компонентами САПФОР являются анализаторы последовательных программ, блоки преобразования последовательных программ в параллельные программы (эксперты), диалоговая оболочка для взаимодействия с пользователем, генератор кода, создающий на основе принятых экспертом решений параллельную версию программы. Связь между всеми компонентами осуществляется через базу данных, хранящую всю необходимую информацию для построения параллельной версии исходной программы.Схема САПФОР показана на **Рис. 1**.

Рис. 1. Схема САПФОР

3. Подсистемы анализа

В САПФОР может одновременно использоваться несколько подсистем анализа (анализаторов), в этом случае запуск каждого следующего анализатора уточняет результаты уже проведенного анализа.Подсистемы анализа выполняют две основные функции.

Во-первых, структура программы должна быть сохранена в соответствии с требованиями базы данных САПФОР. Из базы данных должна быть доступна информация о программных единицах (далее “ПЕ”), образующих программу и принадлежности ПЕ к конкретным файлам.Для каждой ПЕ сохраняется дерево циклов с информацией о тесной вложенности отдельных циклов. Должен быть сохранен граф управления программы и информация о COMMON-блоках (для Fortran-программ).

Во-вторых, в результате анализа должны быть выявлены зависимости по данным [12], содержащиеся в программе.

Для всех циклов программы предоставляется информация следующего вида:

- информационные зависимости между витками цикла (выходные (output), прямые (flow), обратные (anti));
- редукционные зависимости между витками цикла;
- приватные переменные (скаляры и массивы);
- регулярные зависимости по массивам. Наличие регулярной зависимости означает, чтомежду витками имеет место прямаязависимость, и виток зависит от некоторого количества (не более константы) соседних предыдущих витков. Такой цикл допускает конвейерное выполнение.

Выделяется два типа анализаторов: динамические и статические.

Динамические анализаторы выполняют анализ программы в процессе ее выполнения на представительных данных.Динамический анализатор позволяет:

- точнее оценить количество витков циклов и время их выполнения, а также размеры динамических массивов;
- обнаруживать приватные переменные и зависимости (информационные и регулярные) в случаях косвенной индексации и с учетом вводимых данных.

Основным недостатком является то, что динамический анализне способен обеспечить эксперта информацией для всевозможных наборов входных данных.

Статические анализаторы работают по тексту программы.

В САПФОР существует 4 статических анализатора, способных обнаруживать различные типы свойств исследуемых программ. Три из них выполняют анализ на основе исходного текста программы, а входом для четвертого служит база данных, подготовленная одним из трех анализаторов. После выполнения анализа любым из 4 анализаторов может быть запущен эксперт для получения параллельной версии программы, при этом эффективность распараллеливания будет зависеть от полноты выполненного анализа.

Первый анализатор основывается на Sage++[13]представлении программ. Основной его задачей является сохранение в базе данных структуры программы и выявление редукционных зависимостей без осуществления межпроцедурного анализа.

Следующий анализатор[14] является развитием первого, но применяется только для программ, написанных на Fortran 77. Данный анализатор способен обнаруживать информационные зависимости.В ряде случаев используемый в анализаторе алгоритм может быть применен для межпроцедурного анализа. Кроме того анализатор способен определять условия возникновения зависимостей в программе.

Еще один анализатор [15] в качестве основного средства анализа использует Универсальную Библиотеку Трансляции (УБТ) [3]. Анализ программ выполняется на основе внутреннего представления GIMPLEкомпилятора GCC.Единство данного представления для различных языков программирования позволяет создать единую систему анализа программ, написанных на языках Fortran и C/C++. В случае языка C++ использование GCCявляется единственным возможным решением, так как компоненты системы не рассчитаны на сложные конструкции объектно-ориентированного языка.Проблема может быть решена с помощью распараллеливания исходной программы непосредственно на уровне GIMPLE.Но поддержание данного анали-

затора в актуальном состоянии наталкивается на многочисленные трудности [15], кроме того УБТявляется коммерческим продуктом, что усложняет возможность ее использования.

В статье будет рассмотрен последний из анализаторов. В отличие от остальных он работает по информации, содержащейся в базе данных САПФОР. Вследствие этого, он потенциально не зависит от языка, на котором написана анализируемая программа. Для выполнения анализа программ, написанных на языках отличных от Fortran, достаточно создать средство сохраняющее структуру программы в терминах базы данных САПФОР. Разработку данного анализатора можно разбить на три этапа:

1. Создание подсистемы, выполняющей анализ приватных скалярных переменных с выполнением межпроцедурного анализа и учетом псевдонимов, появляющихся вследствие использования COMMON-блоков и модулей. Уточнение анализа редукционных переменных.
2. Интеграция второго из рассмотренных анализаторов[14] в разрабатываемыйанализатор, с целью определения информационных и регулярных зависимостей.
3. Создание подсистемы, выполняющейанализ приватных массивов.

4. Структура анализатора

Анализатор образуют три части:

- классы, поддерживающие внутреннее представление;
- загрузчик (frontend),осуществляющий построение внутреннего представления анализатора на основе входных данных (например, базы данных САПФОР);
- модули анализа, выполняющие различные виды анализа над внутренним представлением анализатора.

Результатом работы анализатора является обновленная база данных САПФОР. Обновление базы данных выполняется сразу в процессе анализа, чтобы снизить объем используемой анализатором памяти.

Внутреннее представлениене зависит от языка входной программы и моделирует ее информационную структуру с помощью следующих графовых моделей:

- граф оператора или выражения;
- граф управления процедуры (подпрограммы или функции);
- дерево циклов процедуры;
- граф вызовов программы;
- граф используемой в программе памяти.

Граф оператора – ориентированный граф, вершинами которого являются операции, а дуги показывают направление потока информации между операциями. Выделяется множество входных вершин, которые описывают используемые в операторе данные (переменные или константы), и множество выходных вершин, которые описывают результат применения оператора.Пример графа операториз строки 8 программы на **Рис. 2** приведен на **Рис. 3**.

Граф управления процедуры – ориентированный граф, вершинами которого являются базовые блоки [12], а дуги показывают направлениепотока управления в программе. Базовые блоки содержат операторы процедуры, каждый из которых представлен графом оператора. Выделяются начальный и конечный базовый блок, которые не содержат исполняемых операторов программы. Данные вспомогательные блоки используются модулями анализа.

Граф вызовов программы содержит информацию о точках вызовов процедур.

Граф используемой в программе памяти – ориентированные граф, вершинами которого являются участки памяти (ПЕ, COMMON-блоки, локальные переменныеи др.), а дуги показывают пересечение между участками памяти в программе. Для пересечений по памяти содержится информация об их границах. Если скалярная переменная ссылается на элемент массива, то номер данного элемент доступен из графа памяти.

Используемые в программе именованные константы в граф памяти не попадают. Если COMMON-блок или модуль явно не указан в процедуре, но используется косвенно (например, при вызовах других процедур), данная информация также отображается в графе памяти. Пример графа памяти для программы на **Рис. 2** приведен на **Рис. 4**.

```
1. program Example
2. parameter( n = 100)
3. external add
4. integeri, j, add
5.
6. i = n
7. dowhile ( i< n * n)
8.           j = add( i, n)
9. enddo
10.
11.    call print( j)
12.    end
13.
14.    subroutine print( j)
15.    integer j, k
16.    common /a/ k
17.
18.    print *, "Two last elements: ", k, j
19.    end
20.
21.    integerfunction add( x, y)
22.    integer x, y, z
23.    common /a/ z
24.
25.         add = x + y
26.         z = x
27.         x = add
28.    end
```

Рис. 2. Программа, печатающая два последних элемента арифметической прогрессии

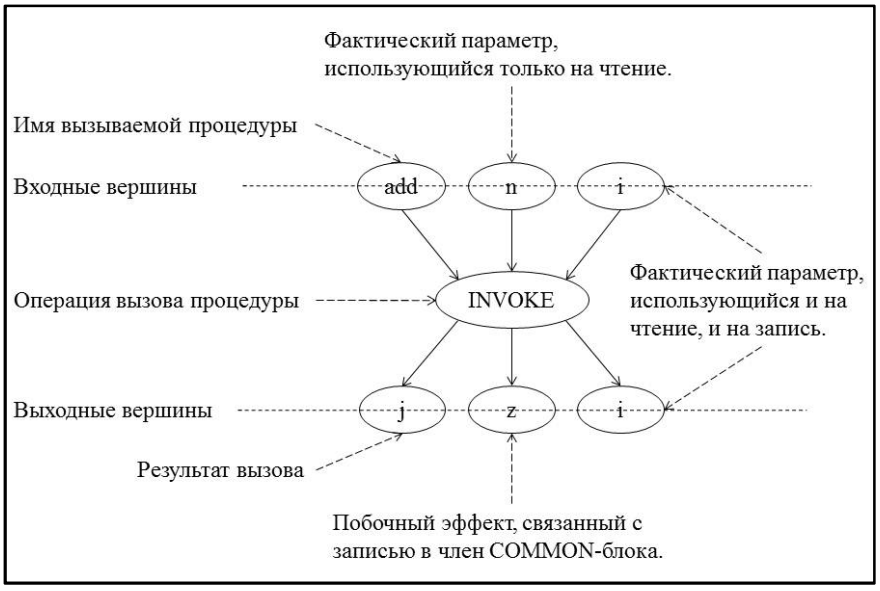


Рис.3.Граф оператора `j = add( i, n)`

В процессе анализа внутреннее представление модифицируется модулями анализа, с целью наиболее точного отражения свойств анализируемой программы. Модулями анализа выполняется следующая последовательность действий:

1. Упорядочивание процедур: построение последовательности обратного обхода глубинного остовного дерева графа вызовов [12]. Данное упорядочивание процедур используется на последующих шагах анализа.
2. Уточнение описания участков памяти, используемых в процедурах.
3. Определение возможного пересечения по памяти между формальными параметрами процедур и членами COMMON-блоков и модулей.

4. Определение приватных переменных.
5. Уточнение редукционных переменных необходимо, так как в базе данных, подаваемой на вход анализатору, сохранена информация о редукциях в программе, полученная без межпроцедурного анализа.

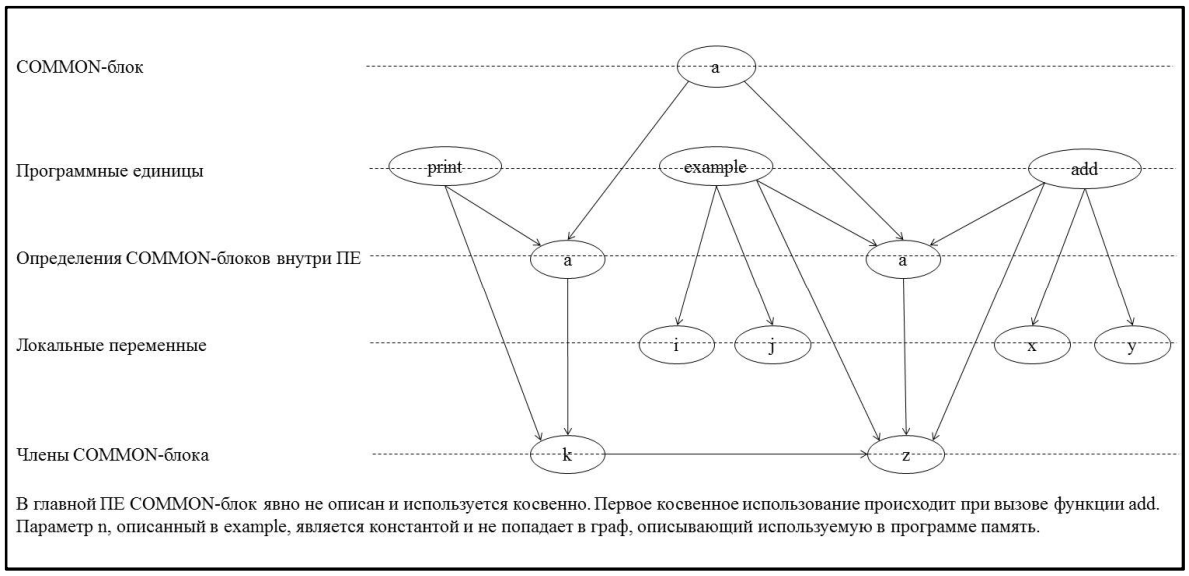


Рис. 4. Граф, используемой в программе памяти

Без использования межпроцедурного анализа невозможно определить, к каким участкам памяти будет происходить обращение при вызове некоторой процедуры. Из базы данных, подаваемой на вход анализатора, в качестве описания вызова доступен только список фактических параметров, при этом не известно: используются они на запись или на чтение. Кроме того, вызов процедуры может иметь побочный эффект, являющийся следствием использования глобальных данных (COMMON-блоков и модулей).

Анализ выполняется над вызываемой процедурой один раз, и полученные результаты используются для уточнения описания участков памяти, используемых во всех вызовах. Анализ выполняется в порядке определенном на первом шаге. При отсутствии рекурсивных вызовов процедур в программе это гарантирует, что при уточнении памяти, используемой в анализируемой процедуре, все вызываемые из нее процедуры уже проанализированы. Результаты выполненного анализа отображаются в графе оператора, содержащего вызов.

Результат выполнения данного шага виден на **Рис. 3**.В процессе анализа было установлено, что при вызове функции `add`в операторе из строки 8 программы на **Рис. 2**, параметр `n`, используется только на чтение, параметр`i`используется на чтение и на запись.Кроме того было установлено, что оператор использует на запись член COMMON-блока, явно не описанного в вызывающей процедуре.Данная информация повлекла за собой модификацию графа памяти. Обновленный граф памяти изображен на **Рис. 4**.

При некоторых вызовах процедур фактические параметры могут пересекаться по памяти. Это порождает соответствующую связь между формальными параметрами, которая должна быть учтена при последующем анализе. Фактические параметры могут быть также связаны с элементами COMMON-блоков или модулей, используемых в вызываемой процедуре.

Данная информация определяется на основе возможных значения формальных параметров. Под значениями понимаются те участки памяти из других ПЕ или COMMON-блоков, которым могут соответствовать формальные параметры ПЕ при некотором потенциально возможном вызове. Найденные связи добавляются в граф памяти программы.

Анализ выполняется в порядке противоположном порядку, определенному на первом шаге. Это гарантирует, что при анализе любой процедуры все вызывающие ее процедурыуже проанализированы, то есть известны все возможные значения формальных параметров вызывающих процедур. Это необходимо, в случае, когда формальные параметры вызывающей процедуры являются фактическими для вызываемой процедуры.

5. Анализ приватных переменных

Пусть Var_P – множество переменных некоторой программы P, MU_P – память, используемая в программе,$Operation_P$ – множество операций программы, $Loop_P$ – множество циклов программы.

Каждой переменной $v \in Var_P$ соответствует некоторый участок памяти $mu(v) \in MU_P$.При этом могут существовать переменные v_1,$v_2 \in Var$такие, что$mu(v_1) \cap mu(v_2) \neq \emptyset$.Такие переменные являются псевдонимами. К ним относятся указатели в Си, члены COMMON-блоков, эквивалентности, члены модулей в Fortran. Псевдонимы могут возникнуть при передаче параметров по ссылке.

Будем говорить, что операция $S_2 \in Operation_P$достижима изоперации $S_1 \in Operation_P$, если поток управления после прохождения операции S_1 достигает операции S_2. Отношение достижимости выполняется и в том случае, если операции S_1 и S_2 совпадают.

Рассмотрим множество итераций $Iteration_L$некоторого цикла$L \in Loop_P$.Введем на данном множестве отношение порядка в соответствии с порядком выполнения итераций в последовательной программе. Пронумеруем все итерации из данного множества (нумерация начинается с 0) и, говоря о некоторой итерации, будет иметь в виду ее номер. Тогда операцию $S \in Operation_P$, выполняющуюся в цикле $L \in Loop_P$,можно рассматривать, как множество выполнений заданной операции на различных итерациях цикла: $S = \{S^I : S^I \in I \wedge I \in Iteration_L\}$.

Переменная$v \in Var_P$ является приватной(private) для цикла $L \in Loop_P$, если:

1. Между любыми операциями$S_1^{I_1}, S_2^{I_2} \in Operation_P$, выполняющимися на разных итерациях цикла$L(I_1, I_2 \in Iteration_L \wedge I_1 \neq I_2)$, использующими память $mu(v)$ и такими, что $S_2^{I_2}$достижима из $S_1^{I_1}$, существует операция $S^{I_2} \in I_2$, использующая $mu(v)$ на запись, такая что S^{I_2} достижима из $S_1^{I_1}$, а $S_2^{I_2}$ достижима из S^{I_2}.
2. Между любыми операциями $S_1, S_2 \in Operation_P$, использующими память $mu(v)$, такими, что $S_1 \in L \wedge S_2 \notin L$, и$S_2$,достижима из$S_1$,существует операция $S \notin L$, использующая $mu(v)$ на запись.

Если выполняется условие (1) и все операцииS_2, для которых условие (2) не выполняется, читают одно и то же значение, записанное в участок памяти$mu(v)$ на последней итерации цикла, то переменная vявляется приватной по выходу (lastprivate).

Если выполняется условие (2) и все операции $S_2^{I_2}$, для которых условие (1) не выполняется, используют значение, записанное в участок памяти$mu(v)$, перед входом в цикл, то переменная vявляется приватной по входу (firstprivate).

Задачей анализа приватных переменных является обнаружение приватных переменных, переменных приватных по входу и по выходу для каждого естественногоцикла программы. Естественный цикл[12] имеет единственный входной узел, называемый заголовком цикла. Анализ выполняется над каждой из процедур, при этом процедуры рассматриваются в порядке определенном на шаге 1 последовательности действий,выполняемых модулями анализа. При этом используются результаты шагов 2 и 3.

На первом этапе разработки анализатора решалась задача анализа скалярных приватных переменных. Так как анализ условий, по которым происходят ветвления в программе, не производится (данная информация не содержится в базе данных САПФОР), анализ приватных по входу переменных не возможен.Необходимым условием приватности скалярной переменной по входу, являетсятребование того, чтобы использование данной переменной на чтение регулировалось некоторой конструкцией ветвления, расположенной в цикле.

Рассмотрим последовательность процедур $P_1^n = \{P_1, P_2, \dots, P_n\}$, такую что $\forall i = \overline{1, n-1}$существует вызов процедуры P_{i+1}из процедуры P_iи такую что $mu(P_1) \cap mu(P_2) \cap \dots \cap mu(P_n) \neq \emptyset$, где$mu(P_i)$ – участок памяти, используемой в процедуре P_i. Непустое пересечение возможно за счет передачи параметров по ссылке, обращения к членам COMMON-блоков и модулей.

Рассмотрим переменную $v \in Var_P$, описанную в P_nи такую, чтоучасток памяти$mu(v)$используется всеми процедурами последовательности P_1^n. В этом случае для решения задачи анализа приватных переменных не достаточно проанализировать только процедуру P_n.Операция

S_2 из условия (2) приватности переменной, использующая память$mu(v)$ может находиться в одной из процедур последовательности P_1^n, отличной от процедуры P_n.

Необходимо выполнить межпроцедурный анализ для всех последовательностей P_1^n, удовлетворяющих условию описанному выше. При таком анализе процедуры нужно рассматривать в направлении обратном направлению вызовов, то есть в направлении от процедуры P_n к процедуре P_1.

Отметим, что последовательности P_1^n соответствует путь в графе вызовов программы P. При этом порядок обхода процедур, определенный на первом шаге, гарантирует, что перед рассмотрением некоторой процедуры, все процедуры, вызываемые из нее, уже рассмотрены. При таком порядке обхода для любой последовательности P_1^n процедуры будут рассмотрены в направлении от процедуры P_n к процедуре P_1.

Решение задачи анализа приватных переменных для некоторой процедуры начинается с анализа внутренней области данной процедуры. Разработанный алгоритм основан на анализе потока данных (data-flowanalysis)[12]. Каждый цикл процедуры рассматривается отдельно. Анализ состоит из трех частей:

1. Анализ достигающих определений [12] определяет, в каких базовых блоках графа управления анализируемой процедуры может быть определена каждая переменная процедуры при достижении потоком управления каждого базового блока.
2. Анализ внутренней области цикла выполняется для проверки условия (1) из определения приватной переменной.
3. Анализ операций, внешних по отношению к циклу, выполняется для проверки условия (2) из определения приватных переменных.

При анализе внутренней области цикла анализируется подграф графа управления, вершинами которого являются базовые блоки, входящие в анализируемый цикл. Дуги графа управления, являющиеся для цикла обратными, входными и выходными, отбрасываются. Для каждого базового блока, входящего в подграф, определяется состояние переменных используемых процедурой. Для этого выполняется анализ потока данных в данном подграфе.Анализ выполняется в прямом направлении. Входным базовым блоком является заголовок цикла.

Множество значений потока данных описывает состояния переменных в некотором базовом блоке:

- Неопределенное состояние (UNKNOWN), означает, что анализ базового блока еще не проводился.
- Переменная не использовалась в базовом блоке, и на любом пути, ведущем к данному блоку, переменная либо не использовалась, либо была проинициализирована (NOT_USE).
- В базовом блоке или на некотором пути в графе управления, ведущем к данному блоку, переменная была использована,не будучи проинициализированной (NOT_INIT).
- Переменная была использована в базовом блоке на чтение, не будучи проинициализированной, после чего она была проинициализирована в том же базовом блоке (INIT_AFTER_USE).
- На выходе из базового блока переменная является проинициализированной (INIT).

Оператор сбора задается с помощью **Таблицы 1**.В первой строке и столбце таблицы указаны состояния переменных в базовых блоках, к которым применяется оператор сбора. Оператор сбора идемпотентен, коммутативен и ассоциативен. Верхним элементом полурешетки, образованной множеством значений потока данных и оператором сбора, является UNKNOWN, нижним –NOT_INIT.

Таблица 1. Оператор сбора для двух базовых блоков

	UNKNOWN	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
UNKNOWN	UNKNOWN	NOT_USE	NOT_INIT	INIT	INIT
NOT_USE	NOT_USE	NOT_USE	NOT_INIT	NOT_USE	NOT_USE
NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT
INIT_AFTER_USE	INIT	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
INIT	INIT	NOT_USE	NOT_INIT	INIT	INIT

Передаточнаяфункциязадается с помощью **Таблицы 2**. В первой строке таблицы показаны значения после применения оператора сбора к базовым блокам, предшествующим исследуемому базовому блоку. В первом столбце показаны значения после локального анализа исследуемого базового блока. Локальный анализ выполняется в предположении того, что используемые в базовом блоке переменные не проинициализированы и определяет локальное состояние переменных согласно следующим правилам:

- NOT_INIT – первый использующий переменную оператор использует ее на чтение и нет ни одного оператора, использующего данную переменную на запись;
- INIT_AFTER_USE – первый использующий переменную оператор, использует ее на чтение, но существуетоператор, присваивающий переменной некоторое значение;
- INIT – первый использующий переменную оператор использует ее на запись;
- NOT_USE – переменная не используется в базовом блоке.

Таблица 2. Передаточная функция для базового блока

	UNKNOWN	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
NOT_USE	UNKNOWN	NOT_USE	NOT_INIT	INIT	INIT
NOT_INIT	UNKNOWN	NOT_INIT	NOT_INIT	INIT	INIT
INTI_AFTER_USE	UNKNOWN	INIT_AFTER_USE	INIT_AFTER_USE	INIT	INIT
INIT	INIT	INIT	INIT	INIT	INIT

Начальная инициализация состояний переменных в заголовке цикла, являющемся входным базовым блоком при анализе подграфа, выполняется следующим образом:

- Состояние INIT устанавливается для индуктивных переменных цикла.
- Для остальных переменных устанавливается состояние NOT_USE.

Для остальных базовых блоков, входящих в подграф, согласно алгоритму анализа потока данных состояние переменных устанавливается равным верхнему элементу (UNKNOWN)полурешетки, образованной множеством значений потока данных и оператором сбора.

При анализе используется информация,содержащаяся в графе памяти, о пересечении между участками памяти, соответствующими разным переменным.

На основе информации о состоянии переменных, полученной в результате анализа потока данных в заданном подграфе, определяется множество переменных, являющихся кандидатами в приватные переменные, множество переменных используемых в цикле только на чтение (при распараллеливаниитакие переменные могут быть объявлены как общие), множество неиспользуемых переменных в цикле.При определении данных множеств анализатор руководствуется следующими правилами:

- Согласно условию (1)из определения приватной переменной на каждой итерации анализируемого цикла приватная переменная перед использованием должна быть проинициализирована некоторым значением. Это означает, что для каждого базового блока, входящего в цикл, приватная переменная не может находиться в состоянии NOT_INITили INIT_AFTER_USE.
- Если для всех базовых блоков, входящих в анализируемый цикл, переменная находится в состоянии NOT_USE, то данная переменная не используется в анализируемом цикле.
- Переменные, используемые в цикле только на чтение, ни в одном базовом блоке не могут находиться в состоянии INITили INIT_AFTER_USE.
- Все остальные переменные не являются ни приватными, ни приватными по выходу и вызывают в цикле зависимости по данным.

На основе кандидатов в приватные переменные определяется множество кандидатов в переменные приватные по выходу.

Данный анализ выполняется с использованием результатов анализа достигающих определений. Рассматриваются все базовые блокипроцедуры, не входящие ванализируемый цикл. Переменная, являющаяся кандидатом в приватные переменные, становится кандидатом в переменные приватные по выходу при выполнении следующих условий для некоторого базового блока, не входящего в цикл:

1. Для данной переменной перед входом в базовый блок среди достигающих определений есть определение, находящееся внутри цикла.
2. Первое использование данной переменной в базовом блоке является использованием на чтение, то есть локальный анализ базового блока показал, что переменная находится в состоянии NOT_INITили INIT_AFTER_USE.

Если для формальных параметров процедуры, элементов COMMON-блоков и модулей в конечном базовом блоке графа управления анализируемой процедуры доступны определения данных переменных,локализованные внутри анализируемого цикла, то данные переменные должны быть дополнительно проанализированы с использованием межпроцедурного анализа.В этом случаедля процедур, вызывающиханализируемую процедуру, проверяется выполнение условий аналогичных условиям, описанным выше. Базовые блоки, содержащие достигающие определения анализируемой переменной (условие 1), ищутся среди базовых блоков, содержащих вызовы анализируемой процедуры.

После того как были найдены кандидаты в переменные приватные по выходу, необходимо проверить выполнение следующих условий:

- Из анализируемого цикла существует только один выход. В противном случае нельзя гарантировать выполнение условия (2) для приватных по выходу переменных. Следовательно, все кандидаты в приватные по выходу переменные помечаются как переменные вызывающие зависимость.
- При любом выполнении анализируемого цикла (не зависимо от состояния памяти программы) на последней итерации цикла данная переменная всегда получает одно и то же значение. Нарушение данного условия возможно в случае, если в цикле несколько обратных дуг. В этом случае перед выходом могут выполняться разные базовые блоки (при этом выход из цикла только один), в которых анализируемая переменная будет определена по разному. Для такого анализа используется результат анализа достигающих определений.

Необходимый межпроцедурный анализ выполняется только в том случае, если данная проверка прошла успешно.

Для программы на **Рис. 2**проведенный анализ для цикла с заголовком в строке 7 показал:

- переменная `j`и член COMMON-блока`a`являются приватными по выходу;
- переменная `i`не является приватной и вызывает зависимость;
- именованная константа `n`только читается.

6. Программная реализация

Анализатор реализован на языке программированияC++ с использованием библиотек STL (StandardTemplateLibrary) и BGL(BoostGraphLibrary). BGLявляется одной из библиотек, входящих вBoost [16]. Кроме этого использовались и другие библиотеки, входящие в Boost.

Анализатор написан в объектно-ориентированном стиле с применением идей метапрограммирования. Одним из примеров является реализация схемы алгоритма анализа потока данных. Разработан шаблонный класс описывающий последовательность действий, выполняемую при анализе. Для использования данного класса достаточно определить передаточные функции для базовых блоков, оператор сбора и множество значений потока данных.

Программа анализатора состоит из нескольких частей, реализованных в виде библиотек:

- Библиотека, содержащая базовые структуры данных. В ней определены общие компоненты, не зависящие от анализатора и используемые остальными его частями.Например, здесь определено общее представление графов в анализаторе, построенное на основе BGL.
- Библиотека, содержащая структуры внутреннего представления анализатора.
- Библиотека, содержащая реализацию ввода/вывода. Данная библиотека предоставляет средства для доступа к базе данных САПФОР. На их основе в ней реализованfrontendанализатора, кроме того средства ввода/вывода используются для сохранения результатов анализа.
- Библиотека, содержащая модули анализа, является ядром анализа.

Все компоненты программы анализатора сопровождаются комментариями в формате, поддерживаемом средством автоматической генерации документации Doxygen [17]. Программа анализатора насчитывает 12000строк кода.

7. Экспериментальная проверка

Анализатор был проверен экспериментально на тесте Якоби, программах Каверна и Контейнер (ИПМ им. М.В. Келдыша РАН)и др.

Корректность результатов анализа для теста Якоби была проверена вручную с помощью диалоговой оболочки САПФОР, а также с помощью эксперта САПФОР. Для нее была получена корректно работающая параллельная версия программы.

Для остальных задач, в силу их объема, проверка корректности возможна только с использованием экспертов САПФОР. На основе результатов анализа, сохраненных в базе данных, были построены параллельные версии для программ Контейнер и Каверна.

Время работы анализатора на персональном компьютере с процессором Intel® Core™2DuoT8300c частотой 2,4 Ггц и 3 Гб оперативной памяти приведено в **Таблице 3**.

Таблица 3. Время работы анализатора

Анализируемая программа	Количество строк	Время, затраченное на анализ (сек.)
Якоби	40	< 1
Каверна	500	2
Контейнер	900	6
Приложение 1	3000	16
Приложение 2	19000	977 (16 минут)

Из детализации данных замеров времени,доступной в процессе анализа, видно, что основное время затрачено на анализ достигающих определений и поиск приватных переменных. Из этого следует, что деятельность по оптимизации работы анализатора должна быть сконцентрирована на этих двух стадиях анализа.

8. Заключение

Распараллеливание последовательных программ требует четкого понимания информационной структуры разрабатываемой программы. Для этого в рамках САПФОР было решено разработать статический анализатор. Разработка анализатора выполняется в три этапа:

1. Создание подсистемы, решающей задачу анализа приватных скалярных переменных.
2. Интеграция существующего анализатора в разрабатываемый анализатор с целью определения информационных и регулярных зависимостей.
3. Создание подсистемы, выполняющейанализ приватных массивов.

Первый этап разработки выполнен. В данный момент осуществляется реализация второго этапа. Выполнение третьего этапа запланировано на 2012 год.

В рамках первого этапа было разработано внутреннее представление, независящее от языка программирования, на котором написана анализируемая программа и решена задача анализа приватных скалярных переменных. Написанfrontendдля базы данных САПФОР, построенной на основе языка программирования Fortran95.

Анализатор проверен на программах, содержащихдо 20 тысяч строк. Анализ был выполнен за приемлемое время. Его корректность была проверена с помощью диалоговой оболочки и экспертов САПФОР.

Литература

1. TOP500 List– November 2011. URL:http://www.top500.org/list/2011/11/100 (датаобращения 14.01.2012)

2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2004. – 608 с.
3. OptimizingTechnologies.URL: http://www.optimitech.com/ (датаобращения: 15.01.2012).
4. ParallelSoftwareProductsInc.URL: http://www.parallelsp.com/ (датаобращения: 14.01.2012).
5. Юрушкин М.В., Петренко В.В., Штейнберг Б.Я., Алымова Е.В.,Абрамов А.А., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Полуян С.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б., Штейнберг Р.Б. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, с. 71-75.
6. Система АвтоматизированнойПараллелизации Фортран Программ.URL: http://www.keldysh.ru/dvm/SAPFOR/ (дата обращения: 02.01.2012).
7. Бахтин В.А., Коновалов Н.А., Крюков В.А., Поддерюгина Н.В. FortranOpenMP/DVM – язык параллельного программирования для кластеров// Материалы второго Международного научно-практического семинара “Высокопроизводительные параллельные вычисления на кластерных системах”, г. Нижний Новгород, 26-29 ноября 2002 г., с.28-30.
8. Бахтин В.А., Коновалов Н.А., Поддерюгина Н.В., Устюгов С.Д. Гибридный способ программирования DVM/OpenMP на SMP-кластерах // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования” (сентябрь 2006 г., г. Новороссийск), Изд-во Московского Университета, с.128-130.
9. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: экзафлопсное будущее” (19-24 сентября 2011 г., г. Новороссийск). - М.: Изд-во МГУ, с. 310-315.
10. Крюков В.А., Клинов М.С., Бахтин В.А., Поддерюгина Н.В. Автоматическое распараллеливание последовательных программ для многоядерных кластеров// Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, с. 12-15.
11. Бахтин В. А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсноебудущее” (19-24 сентября 2011 г., г. Новороссийск). – М.: Изд-во МГУ, 2011, с. 67-70
12. Ахо А.В., Лам М.С., Сети Р., УльманДж.Д. Компиляторы: принципы, технологии и инструментарий, 2-е издание: Пер. с англ.- М.ООО "И.Д.Вильямс", 2008.-1184 с.Главы 9, 11.
13. pC++/Sage++ Home Page. URL: http://www.extreme.indiana.edu/sage/ (датаобращения 12.01.2012)
14. Катаев Н.А. Система автоматизированного распараллеливания Фортран-программ: анализ многомодульных программ // Сборник тезисов лучших дипломных работ 2009 года. – М.: Издательский отдел Факультета ВМиК МГУ им. М.В. Ломоносова; МАКС Пресс, 2009. С. 141 – 142.
15. Катаев Н.А. Анализ последовательных программ с помощью средств УБТ // Труды международной научной конференции “Параллельные вычислительные технологии (ПаВТ’2011)” (Москва, 28 марта – 1 апреля 2011 г.) – Челябинск: Издательский центр ЮУрГУ, 2011, с. 697.
16. Boost C++ Libraries.URL: http://www.boost.org/ (датаобращения 12.01.2012)
17. Doxygen. URL: http://www.doxygen.org(дата обращения 12.01.2012)

Эффективные методы расчета электромагнитных полей

Д.Ю. Князьков

Институт проблем механики им. А.Ю. Ишлинского РАН

Рассматривается задача расчета электромагнитных полей от большого количества (10^8 и более) элементарных объектов для нужд голографической литографии. Анализ производительности некоторых методов расчета электромагнитных полей на кластерах МВС100-К МСЦ РАН и МИИТ Т4700 показал невозможность создания этими методами голограммы Габора для микрочипа с реальными размерами на современных суперкомпьютерах. Специально разработанный метод Большого пиксела, позволяющий рассчитывать с его помощью голограммы Габора для современных микрочипов на современных параллельных вычислительных комплексах, был реализован в составе параллельного ПК. Приведен пример синтеза голограммы Габора для топологии из $1.6 \cdot 10^9$ расчетных элементов.

1. Введение

В рамках научно-исследовательских работ по голографической литографии [1–3] необходимо уметь решать задачи создания, оптимизации и восстановления для топологий состоящих из одного или нескольких элементарных объектов (квадратиков, полосок, шевронов, тестовых мир). Такие задачи могут быть решены с использованием кластерного суперкомпьютера небольшой мощности [4].

Расчет голограммы Габора [5] является здесь необходимым технологическим этапом. В обычной проекционной литографии маска фактически является гомотетически растянутой исходной топологией, поэтому ее расчет достаточно прост. Расчет же голографической маски обычными методами для топологий реальных чипов, содержащих миллиарды элементов, будет требовать вычислительных мощностей, на порядки превышающих производительность существующих на сегодня супер-ЭВМ.

Восстановление в проекционной литографии также облегчается тем, что зависимость изображения от маски - локальна [6], тогда как в голографической литографии "информация" о каждом участке изображения распределена по всей маске и, даже при восстановлении небольшого фрагмента изображения, необходимо учитывать дифракцию от всех элементов маски. Таким образом, расчет электромагнитных полей является существенной проблемой в голографической литографии и будет рассмотрен в настоящей работе.

2. Постановка задачи

Схема голографической литографической установки [1–3] изображена на рис. 1. Излучение W падает на голографическую пластинку, которая занимает область Ω_H в плоскости $O_H xy$, а ее пропускание описывается вещественной функцией $T(x, y)$: $\Omega_H \to [0, 1]$. При этом области $\{(x, y) \mid (x, y) \in \Omega_H, T(x, y) = 1\}$ полностью прозрачны для излучения, а области $\{(x, y) \mid (x, y) \in \Omega_H, T(x, y) = 0\}$ излучение W не пропускают. В результате, в плоскости $O_I\xi\eta$ необходимо создать требуемое световое изображение.

Пусть $g_0(\xi, \eta) : \Omega_I \to \{0, 1\}$ - некоторая функция, задающая желаемую конфигурацию дорожек (топологию) на микросхеме. Возьмем в области Ω_I на плоскости $O_I\xi\eta$ соответствующее объектное поле - комплекснозначную функцию $g(\xi, \eta) : \Omega_I \to \mathbb{C}$. Тогда голограммой Габора [5] для объекта $g(\xi, \eta)$ называется функция $T_g(x, y) : \Omega_H \to [0, 1]$, рассчитанная по следующей формуле:

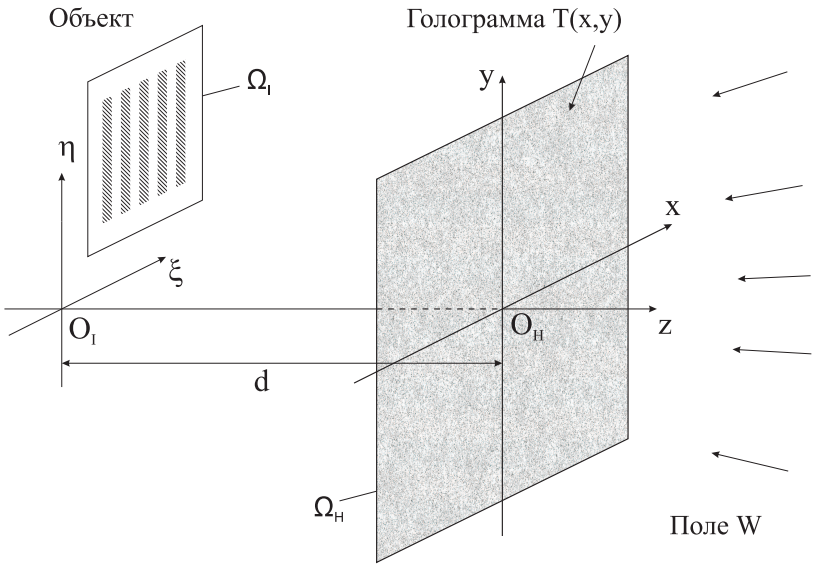


Рис. 1. Оптическая схема.

$$T_g(x,y) = \left| \iint\limits_{\Omega_I} K(x,y,\xi,\eta) g(\xi,\eta) d\xi d\eta + \overline{W}(x,y) \right|^2, \tag{1}$$

где $K(x,y,\xi,\eta) = \frac{e^{i\tilde{k}R}}{R}$, $R = \sqrt{(x-\xi)^2 + (y-\eta)^2 + d^2}$, $\tilde{k} = \frac{2\pi}{\lambda}$ - волновое число, λ - длина волны, а $\overline{W}(x,y)$ - поле, комплексно-сопряженное к восстанавливающему полю $W(x,y)$ (рис. 1).

На расстоянии d от голограммы, в объектной плоскости $O_I\xi\eta$ находится область наблюдения Ω_I, в которой регистрируется интенсивность излучения $I(\xi,\eta)$. Если поле в плоскости O_Hxy до попадания на голограмму с пропусканием $T(x,y)$ описывается функцией $W(x,y)$, то интенсивность излучения в объектной плоскости может быть в скалярном приближении вычислена с использованием интеграла Кирхгофа [1]:

$$I_T(\xi,\eta) = \left| \iint\limits_{\Omega_H} K(x,y,\xi,\eta) W(x,y) T(x,y) dxdy \right|^2. \tag{2}$$

Полученное с голограммы Габора $T_g(x,y)$ изображение $I_{T_g}(\xi,\eta)$ будет достаточно близко к заданной топологии $g_0(\xi,\eta)$.

Заметим, что при больших апертурах необходимо использовать векторную модель излучения. Формула (1) для голограммы при этом не изменится, а расчет восстановленного изображения может быть сведен к расчету нескольких интегралов типа (2), например, с использованием формул Стреттона-Чу [7]. Таким образом, основная проблема как в задаче создания голограммы (1), так и в задаче моделирования эксперимента по ее засветке (2) - это расчет интеграла

$$I(x,y) = \iint\limits_{\Omega} K(x,y,\xi,\eta) \varphi(x,y) d\xi d\eta \tag{3}$$

с сильно осцилирующим ядром

$$K(x,y,\xi,\eta) = \frac{e^{i\frac{2\pi}{\lambda}\sqrt{(x-\xi)^2+(y-\eta)^2+d^2}}}{\sqrt{(x-\xi)^2+(y-\eta)^2+d^2}}. \quad (4)$$

3. Применение прямых методов расчета и методов с БПФ

Без ограничения общнности, возьмем в качестве области Ω квадрат со стороной a. Рассмотрим самую трудоемкую операцию в задачах синтеза голограммы (1) и восстановления изображения (2) - расчет интеграла

$$f(x,y) = \int\limits_{-\frac{a}{2}+\eta_0}^{\frac{a}{2}+\eta_0} \int\limits_{-\frac{a}{2}+\xi_0}^{\frac{a}{2}+\xi_0} K(x,y,\xi,\eta)\varphi(\xi,\eta)d\xi d\eta \quad (5)$$

где (ξ_0, η_0) - центр топологии, заданной в квадрате размера $a \times a$. Конкретный вид функции $\varphi(\xi,\eta)$ зависит от решаемой задачи. Например, при создании голограммы $\varphi(\xi,\eta) = R(\xi,\eta)q_0(\xi,\eta)$, где $R(\xi,\eta)$ и $q_0(\xi,\eta)$ - функции, задающие волну подсветки и топологию соответственно. Пусть голограмма размера $b \times b$ состоит из $M \times M$ ячеек размера $\gamma \times \gamma$ (рис. 5): $M = \frac{b}{\gamma}$. Тогда для расчета серой голограммы необходимо найти значения функции f в $M \times M$ точках: $\{(-\frac{b}{2} + \gamma(i+\frac{1}{2}), -\frac{b}{2} + \gamma(j+\frac{1}{2}))|\ i,j = 0,...,M-1\}$. При этом область интеграрования будет разбиваться различным образом (см. рис. 2) в зависимости от метода, используемого при расчете интеграла (5).

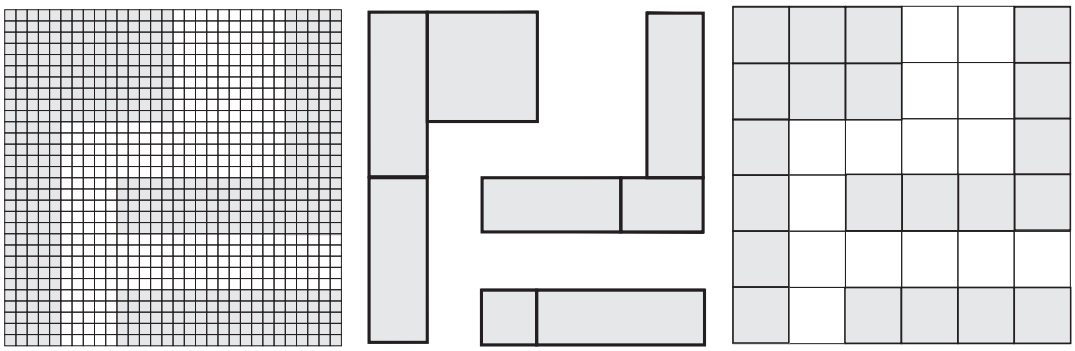


Рис. 2. Разбиение области интегрирования (расчетные сетки) для методов простого суммирования и метода с применением БПФ (слева), метода рвзбиения на прямоугольники (в центре) и метода Большого Пиксела (справа).

3.1. Метод простого суммирования

Выпишем интегральные суммы для (5). Поскольку функция ядра колеблется с периодом λ, для расчета интеграла нужно брать шаг интегрирования δ равным хотя бы $\frac{\lambda}{10}$ (см. рис. 2 слева). Тогда количество точек разбиения $N_D = \left[\frac{10a}{\lambda}\right]$ и интегральные суммы:

$$f(i,j) = \sum_{k,l=0}^{N_D-1} \varphi(k,l)K(i,j,k,l), \quad i,j = 0,...,M-1 \quad (6)$$

где под $f(i,j)$, $\varphi(k,l)$, $K(i,j,k,l)$, $i,j = 0,...,M-1$, $k,l = 0,...,N_D-1$ подразумеваются следующие обозначения:

$$
\begin{aligned}
f(i,j) &= f(-\tfrac{b}{2}+\gamma(i+\tfrac{1}{2}),-\tfrac{b}{2}+\gamma(j+\tfrac{1}{2})) \\
\varphi(k,l) &= \varphi(\xi_0-\tfrac{a}{2}+\delta(k+\tfrac{1}{2}),\eta_0-\tfrac{a}{2}+\delta(l+\tfrac{1}{2})) \\
K(i,j,k,l) &= \\
= K(-\tfrac{b}{2} &+ \gamma(i+\tfrac{1}{2}),-\tfrac{b}{2}+\gamma(j+\tfrac{1}{2}),\xi_0-\tfrac{a}{2}+\delta(k+\tfrac{1}{2}),\eta_0-\tfrac{a}{2}+\delta(l+\tfrac{1}{2}))
\end{aligned}
$$

где (ξ_0,η_0) - центр изображения (топологии).

Для расчета по формуле (6) понадобится $\Upsilon_D^* = (C_\varphi^* + C_K^* + 1)M^2N_D^2$ умножений и $\Upsilon_D^+ = (C_\varphi^+ + C_K^+ + 1)M^2N_D^2 - M^2$ сложений, где C_φ^* и C_φ^+ - количество умножений и сложений для вычисления значения функции φ, а C_K^* и C_K^+ - количество умножений и сложений для вычисления значения функции K. Упрощенно можно записать

$$\Upsilon_D = C_D M^2 N_D^2 \,. \tag{7}$$

3.2. Разбиение топологии на прямоугольники

Представим теперь топологию g_0 в виде суммы N_R прямоугольников P_m, $m = 1,...,N_R$:

$$\{(\xi,\eta)|\ g_0(\xi,\eta)=1\} = \bigsqcup_{m=1}^{N_R} P_m \,.$$

Пример такого разбиения показан на рис. 2 в центре.

При вычислении интеграла (5) можно воспользоваться тем, что подсветка объекта $R(\xi,\eta)$ мало меняется на этих прямоугольниках и можно считать $R(\xi,\eta)|_{P_m} = R_m$, $\forall m$. Тогда, поскольку $\varphi(\xi,\eta) = R(\xi,\eta)g_0(\xi,\eta)$, получаем $\varphi(\xi,\eta)|_{P_m} = \varphi_m$, $\forall m$ и

$$f(x,y) = \int\limits_{-\frac{a}{2}+\eta_0}^{\frac{a}{2}+\eta_0}\int\limits_{-\frac{a}{2}+\xi_0}^{\frac{a}{2}+\xi_0} K(x,y,\xi,\eta)\varphi(\xi,\eta)d\xi d\eta = \sum_{m=1}^{N_R}\varphi_m \iint\limits_{P_m} K(x,y,\xi,\eta)d\xi d\eta \,.$$

Соотношение геометрических параметров оптической схемы голографической установки таково, что интегралы, участвующие в последней сумме могут быть вычислены аналитически в приближении дальней зоны [8,9]: $\iint\limits_{P_m} K(x,y,\xi,\eta)d\xi d\eta = I_{P_m}(x,y)$. Тогда

$$f(x,y) = \sum_{m=1}^{N_R}\varphi_m I_{P_m}(x,y)\,,$$

откуда получаем расчетную формулу для (5):

$$f(i,j) = \sum_{m=1}^{N_R}\varphi_m I_{P_m}(i,j), \quad i,j = 0,...,M-1, \tag{8}$$

где под $f(i,j)$, $I_{P_m}(i,j)$, $i,j = 0,...,M-1$ подразумеваются следующие обозначения:

$$
\begin{aligned}
f(i,j) &= f(-\tfrac{b}{2}+\gamma(i+\tfrac{1}{2}),-\tfrac{b}{2}+\gamma(j+\tfrac{1}{2})), \\
I_{P_m}(i,j) &= I_{P_m}(-\tfrac{b}{2}+\gamma(i+\tfrac{1}{2}),-\tfrac{b}{2}+\gamma(j+\tfrac{1}{2})).
\end{aligned}
$$

Для осуществления вычислений по формуле (8) понадобится совершить $\Upsilon^*_\Pi = (C^*_\varphi + C^*_I + 1)M^2 N_R$ умножений и $\Upsilon^+_\Pi = (C^+_\varphi + C^+_I + 1)M^2 N_R - M^2$ сложений, где C^*_I и C^+_I - количество операций для вычисления значения функции $I_{P_m}(x, y)$. Если в качестве топологии взять структуру, представляющую собой квадрат размером $a \times a$, заполненный маленькими квадратиками размера $\sigma \times \sigma$, получим $N_R = \frac{a^2}{4\sigma^2}$. Здесь и далее σ - это минимальный характеристический размер элемента на топологии, $N = \frac{a}{\sigma}$. Тогда получим упрощенную формулу для количества операций:

$$\Upsilon_R = \frac{1}{4} C_R N^4 \,. \tag{9}$$

Такой метод расчета был реализован на языке программирования C++ в составе параллельного програмного комлекса BinNet [4]. Этот ПК успешно использовался на кластерах МВС100-К МСЦ РАН и МИИТ Т4700 при проведении исследовательских работ. С его помощью расчитывались файлы голограмных масок для их дальнейшего изготовления, моделировались процессы создания голограмм, восстановления изображений и проводились различные численные эксперименты для нужд исследовательского проекта по голографической литографии (некоторые результаты этих работ приведены в [1]).

Из формулы (8) видно, что вычисление результатов дифракции от разных прямоугольников в разых точках голограммы совершенно независимы, поэтому можно ожидать, что соответствующий алгоритм будет обладать хорошей масштабируемостью на кластерных ВС. Действительно, результаты численных экспериментов (рис. 3) показывают, что эффективность алгоритма близка к 1 (замерялось суммарное время необходимое для расчета по (8) и межпроцессорного взаимодействия). Размер задачи n - это количество сумм в (8), остальные параметры близки к параметрам экспериментальной установки.

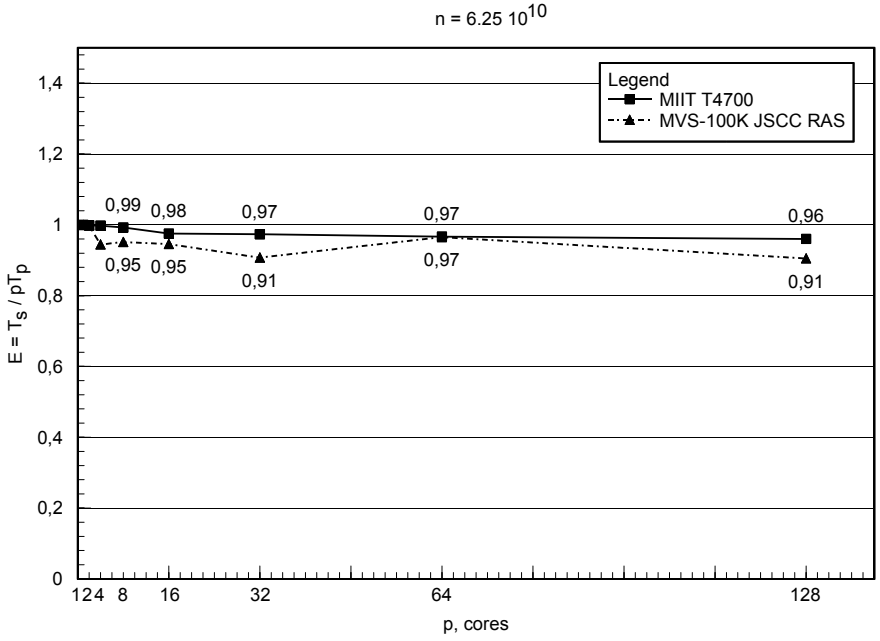


Рис. 3. Эфффективность работы алгоритма суммирования излучения от прямоугольников. Количество вычислительных ядер изменяется от 1 до 128, размер задачи постоянен.

3.3. Использование быстрого преобразования Фурье

В описанных ранее алгоритмах расчета по формулам (6) и (8) никак не используется то, что исходный интеграл является, как легко увидеть из (4), сверткой с ядром $K_1(x_1, y_1) =$

$K(x_1, y_1, 0, 0)$. Перепишем теперь (6) с учетом этого свойства:

$$f(i,j) = \sum_{k,l=0}^{N_{FFT}-1} \varphi(k,l) K_1(i-k, j-l), \quad i,j = 0, ..., M_{FFT} - 1, \tag{10}$$

где $N_{FFT} = \frac{a}{\delta}$, $M_{FFT} = \frac{b}{\delta}$, δ - шаг интегрирования, а под $f(i,j)$, $\varphi(k,l)$, $K_1(s,t)$, $i,j = 0, ..., M_{FFT} - 1$, $k,l = 0, ..., N_{FFT} - 1$ подразумеваются следующие обозначения:

$$\begin{aligned} f(i,j) &= f(-\tfrac{b}{2} + \delta(i + \tfrac{1}{2}), -\tfrac{b}{2} + \delta(j + \tfrac{1}{2})) \\ \varphi(k,l) &= \varphi(-\tfrac{a}{2} + \delta(k + \tfrac{1}{2}), -\tfrac{a}{2} + \delta(l + \tfrac{1}{2})) \\ K_1(s,t) &= K_1(-\tfrac{b}{2} + \delta(s + \tfrac{1}{2}), -\tfrac{b}{2} + \delta(t + \tfrac{1}{2})) \end{aligned}$$

Заметим, что в (10) шаг расчета интеграла на голограмме совпадает с шагом интегрирования по плоскости топологии (рис. 2 слева): голограмма рассчитывается не как ранее в узлах сетки со стороной, равной ячейке голограммы γ, а в узлах сетки со стороной, равной δ. Таким образом теперь расчитывается очень много "лишних" точек на голограмме, однако появляется возможность расчитывать свертку (10) с помощью преобразования Фурье:

$$\mathbf{f} = F^{-1}[F(\boldsymbol{\varphi}) \otimes F(\mathbf{K_1})], \tag{11}$$

где F и F^{-1} - прямое и обратное двумерное дискретное преобразование Фурье, $\otimes$ - поэлементное умножение, а $\mathbf{f}$, $\boldsymbol{\varphi}$, $\mathbf{K_1}$ - соответствующие матрицы. Использование быстрого преобразования Фурье [10] позволяет получить следующее количество операций: $\Upsilon_{\text{FFT}} = 6C_1((k+1)N_{FFT} - 1)^2 log_2((k+1)N_{FFT} - 1)$, где $k = \frac{b}{a}$. Если отбросить несущественные члены, принять $\delta = \frac{\lambda}{10}$, $N_{FFT} = 10N$, а в качестве константы C_1 выбрать $C_1 = 5$ [11], то получим

$$\Upsilon_{\text{FFT}} = 30(k+1)^2 N^2 10^2 log_2((k+1)^2 N), \tag{12}$$

то есть $N^2 log_2 N$ в качестве асимптотики количества операций, тогда как простое суммирование (как по формуле (6), так и по (10)) приводит к ассимтотике N^4 (здесь принято $N = \frac{a}{\sigma}$, где σ - размер минимального элемента на микросхеме и учтено, что $\sigma \approx \lambda$, $N_D \sim N$, $M \sim N$, $N_{FFT} \sim N$, $M_{FFT} \sim N$).

К сожалению, реальная производительность алгоритма БПФ на кластерной ВС при распределенном хранении соответствующей матрицы будет невысока. На рис. 4 приведен график зависимости эффективности работы такого алгоритма от количества используемых вычислительных ядер (от 1 до 512) при постоянном размере задачи (матрица 15999×15999 комплексных чисел двойной точности). Видно, что эффективность падает в 4 раза при увеличении количества процессов от 1 до 8 на одном узле (вычислительный узел МВС100-К имеет два 4-х ядерных процессора), далее держится примерно на уровне 25% при увеличении количества процессов до 64 (8 узлов) и затем продолжает падать, достигая 10% на 512 вычислительных ядрах.

3.4. Трудоемкость алгоритмов и ассимптотика количества операций

Проанализируем возможности использования описанных выше методов для расчета топологий, состоящих из большого количества элементов. Количество операций при использовании метода простого суммирования ($\Upsilon_D^* = (C_\varphi^* + C_K^* + 1)M^2 N_D^2$, $\Upsilon_D^+ = (C_\varphi^+ + C_K^+ + 1)M^2 N_D^2 - M^2$) пропорционально $M^2 N_D^2$. Это слишком много даже если для вычислений используется современный суперкомпьютер. Действительно, пусть рассчитывется топология с минимальным размером элемента $\sigma = 100$ нм на чипе со стороной $a = 1$ см. Тогда

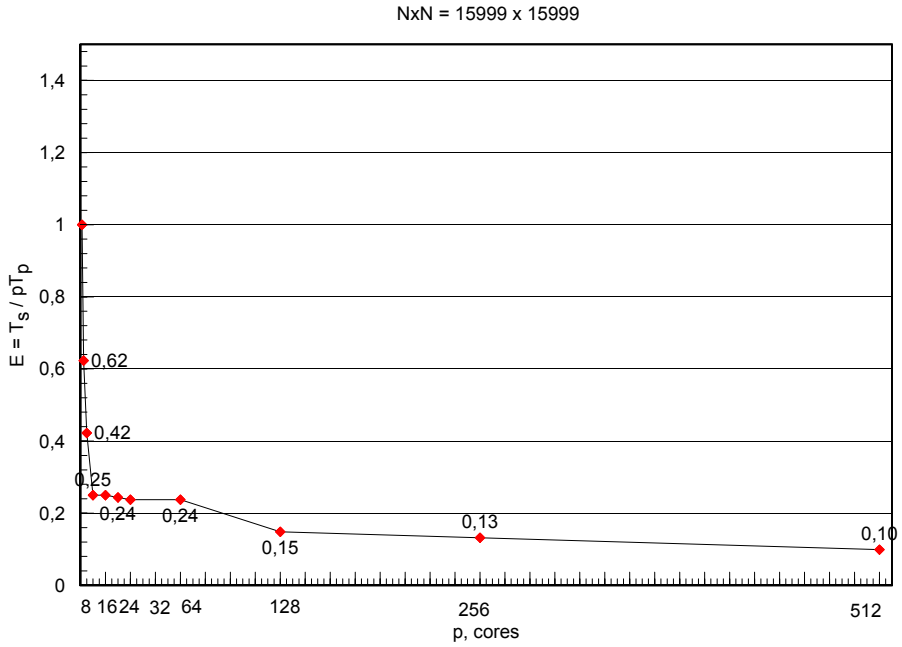


Рис. 4. Эффективность двумерного параллельного БПФ на МВС100-К МСЦ РАН.

$N = 10^5$ и для реальной оптической схемы $M = 10^5$. При длине волны $\lambda = 193$ нм, получаем $N_D \approx 5 \cdot 10^5$, а количество операций при $C_D = 10$: $\Upsilon_D = 2.5 \cdot 10^{22}$. Пусть такой алгоритм реализуется на суперкомпьютере с производительностью 100 TFlops. Тогда, даже если бы при вычислениях достигалась пиковая производительность (а в случае расчета по формуле (6) этого, на самом деле, не сложно добиться), на расчет одной голограммы ушло бы около 8 лет.

Количество операций для формулы излучения от прямоугольников (8) $\Upsilon_{\mathrm{R}} \sim M^2 N_R$. Поскольку количество прямоугольников, из которых состоит топология N_R в общем случае пропорционально N^2, и для оптической схемы, которая используется в эксперименте $M \sim N$, получаем такую же неприемлимую ассимптотику, как и в предыдущем случае: $\Upsilon_{\mathrm{R}} \sim N^4$. Для топологии с $\sigma = 100$ нм и $a = 1$ см из оценки количества операций (9) для синтеза голограммы Габора получается, что расчет займет более 100 суток на суперкомпьютере с производительностью 100 TFlops.

Аналогичные оценки для расчета по методу БПФ (11) дают время около 30 мин, но, учитываю низкую эффективность этого алгоритма на кластерной ВС (см. рис. 4), реальное время работы составит около 10 часов. Этот алгоритм имеет также повышенные требования к общему объему оперативной памяти.

Для расчета голограмм Габора можно предложить ряд методов, использующих предварительно расчитанные поля от некоторых повторяющихся структур. Такими структурами могут могут быть как просто прямоугольники (со сторонами, параллельными осям координат или имеющими с ними угол 45 градусов - именно из таких элементов состоят топологии слоев реальных микросхем), так и повторяющиеся структурные элементы (например одинаковые элементарные ячейки в схемах памяти). Количество таких структур N_Ω, на которые разбивается топология пропорционально в общем случае N^2. Получается, что для всех способов с предварительно расчитанными полями (голограммами) верна та же ассимптотика количества операций - $M^2 N^2$, а с учетом того, что на практике $M \sim N$, имеем ассимптотику N^4. Более того при использовании этих методов для расчета голограмм топологий, близких по сложности к топологиям реальных слоев микрочипов, возникает проблема хранения и загрузки в память огромных массивов данных.

Метод расчета (11) с помощью БПФ, если потребовать размещения всех необходимых

матриц в ОЗУ, успешно может быть применен на персональном компьютере только для расчета голограмм и восстановления с них для маленьких топологий (полосок, квадратиков, уголков, шевронов). Для работы с подобными и более сложными - средними топологиями (например тестовая мира рис. 6) использовался упомянутый выше ПК BinNet. Такие расчеты обычно занимали от 5-10 минут на 32 вычислительных ядрах до суток на всем кластере МИИТ Т4700, состоящем из 64 узлов с двумя 4-х ядерными процессорами и 16 Гб ОЗУ каждый. Однако, из оценок количества необходимых арифметических операций получается, что ни один из описанных выше методов не может быть применен для расчета голограмм реальных чипов на существующих на данный момент суперкомпьютерах.

4. Метод большого пиксела

Рассмотрим два из описанных выше методов создания серой голограммы: метод расчета свертки (10) с помощью БПФ (11) и метод сумирования излучения от прямоугольников, на которые разбивается топология (8). Можно заметить, что в первом случае получается большой выигрыш в ассимптотике ($C_{FFT}N^2log_2N$ по сравнению с C_DN^4 для метода простого суммирования (6)), но при этом необходимо брать очень маленький шаг по сеткам на топологии и голограмме. Во втором случае, на голограмме не считаются "лишние" точки и излучение от целых прямоугольников расчитывается по аналитической формуле, но ассомптотика остается неприемлимой - C_RN^4. Метод большого пиксела (БП), предложенный А.С. Шамаевым, использует преимущества этих двух подходов: с одной стороны излучение от минимального характерного квадратика топологии - большого пиксела размера $\sigma \times \sigma$ - считается аналитически, а с другой - расчет сводится к вычислению свертки, что позволяет воспользоваться БПФ и добиться ассимптотики $C_{BP}N^2log_2N$.

4.1. Разбиение топологии на большие пикселы

Разобьем всю область объекта (квадрат $a \times a$ с центром в точке (ξ_0, η_0)) на большие пикселы - квадратики $\square_{kl}$ размера $\sigma \times \sigma$ с центрами в $(\xi_0 - \frac{a}{2} + \sigma(k + \frac{1}{2}), \eta_0 - \frac{a}{2} + \sigma(l + \frac{1}{2}))$, $k, l = 0, ..., N-1$, где $N = [\frac{a}{\sigma}]$. Таким образом, на каждом $\square_{kl}$ функция топологии q имеет постоянное значение q_{kl}: 1, если квадратик $\square_{kl}$ принадлежит топологии и 0 в противном случае (см. рис. 2 справа). При задании топологии некоторым ее элементам может быть приписана отличная от нуля фаза ψ (например, когда при реализации техники phaseshift соседним элементам приписываются фазы 0 и π). Эта фаза также будет постоянной на $\square_{kl}$. Подсветка объекта R практически не меняется на площади большого пиксела поэтому можно считать, что функция $\varphi = q\psi R$ на каждом из $\square_{kl}$ постоянна и равна φ_{kl}. Тогда исходный интеграл (5) можно преобразовать следующим образом:

$$f(x,y) = \iint\limits_{\mathrm{O}} \varphi(\xi,\eta)K(x,y,\xi,\eta)d\xi d\eta = \sum_{k,l=0}^{N-1} \varphi(k,l) \iint\limits_{\square_{kl}} K(x-\xi, y-\eta)d\xi d\eta. \qquad (13)$$

Обозначим $S_{kl}(x,y) = \iint\limits_{\square_{kl}} K(x,y,\xi,\eta)d\xi d\eta$ - результат дифракции на квадратном отверстии - большом пикселе $\square_{kl}$. В приближении дальней зоны этот интеграл может быть посчитан аналитически. Возьмем теперь шаг расчетной сетки на голограмме равным шагу по топологии σ. Тогда на голограмме будет $H \times H$ расчетных узлов, где $H = [\frac{b}{\sigma}]$. Заметим, что интеграл $S_{kl}(x,y)$ инвариантен относительно одновременного сдвига по объекту и по голограмме: $\forall \delta_k, \delta_l \in \mathbb{Z}$ имеем $S_{k+\delta_k\, l+\delta_l}(x + \delta_k\sigma, y + \delta_l\sigma) = S_{kl}(x,y)$. Используя это свойство и обозначив $S(i,j) = S_{00}(i\sigma, j\sigma)$ сумму (13) можно преобразовать следующим образом:

$$f(i,j) = \sum_{k,l=0}^{N-1} \varphi(k,l)S(i-k, j-l), \quad i,j = 0, ..., H-1, \qquad (14)$$

где $f(i,j) = f(-\frac{b}{2} + \sigma(i+\frac{1}{2}), -\frac{b}{2} + \sigma(j+\frac{1}{2}))$.

Для прямого расчета свертки (14) понадобится $N^2H^2 = [\frac{a}{\sigma}]^2[\frac{b}{\sigma}]^2$ операций. Таким образом, так как $a \sim b$ при увеличении размера топологии a или уменьшении характерного размера σ количество операций будет расти как 4-ая степень, что при работе с большими топологиями приводит к невозможности произвести расчет за приемлемое время даже на суперкомпьютере. Однако, для вычисления свертки (14) можно применить описанный выше способ, использующий 3 двумерных БПФ (11). Тогда количество операций составит $\Upsilon_{\text{БП}} = 3 \cdot 2C_1((k+1)N-1)^2 log_2((k+1)N-1)$. Если отбросить несущественные члены и принять $C_1 = 5$, будем иметь

$$\Upsilon_{\text{БП}} = 30(k+1)^2N^2 log_2((k+1)N)\,. \tag{15}$$

Таким образом, алгоритм большого пиксела дает выигрыш в два порядка по времени счета и по требуемому объему оперативной памяти по сравнению с обычным методом расчета свертки с БПФ (11), (12).

4.2. Реализация метода Большого Пиксела на кластерной ВС

При реализации метода большого пиксела на кластерной ВС, поле на голограмме расчитывалось по зонам, линейный размер каждой из которых совпадал с размером топологии (рис. 5). Всего тогда надо расчитать k^2 таких зон. Тогда количество операций составит $\Upsilon_{\text{БПЗ}} = (2k^2+1)2C_1(2N-1)^2 log_2(2N-1)$. При $C_1 = 5$ получим

$$\Upsilon_{\text{БП}} = (80k^2+40)N^2 log_2(2N)\,. \tag{16}$$

Это несколько больше, чем для метода большого пиксела (15), но в 37.5 раз меньше, чем для обычного метода с БПФ (12). Более того, в требование к общему объему оперативной памяти для алгоритма БП в таком виде снижается еще на два порядка (точнее, в $\frac{(k+1)^2}{4}$ раз).

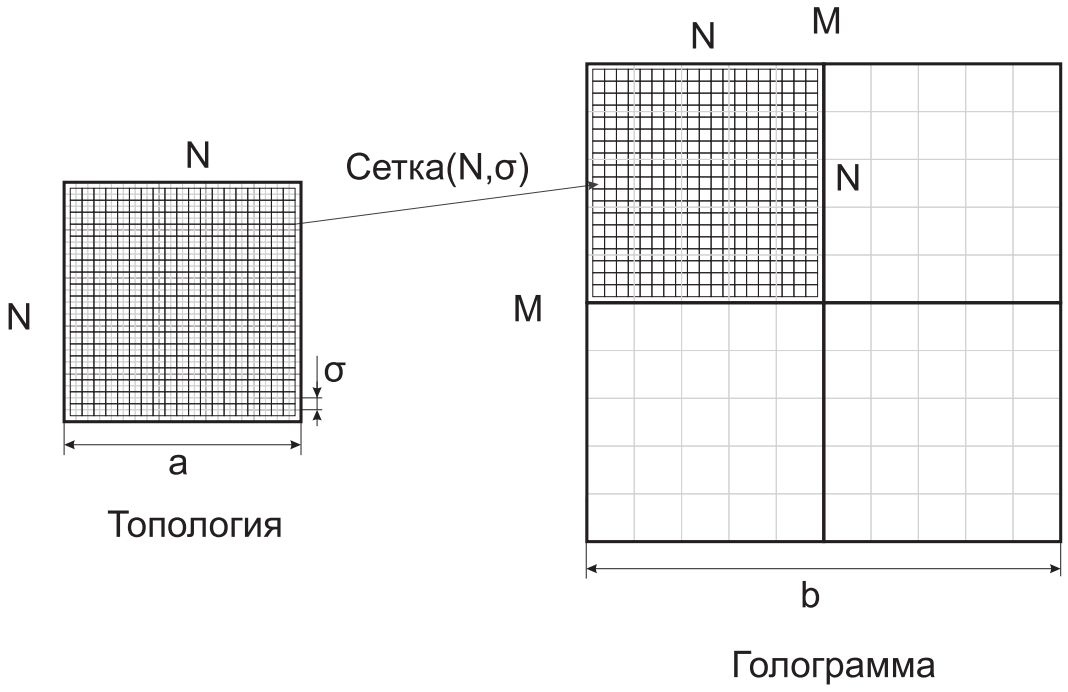


Рис. 5. Расчетные сетки при синтезе голограммы методом большого пиксела с разбиением на зоны.

Метод большого пиксела был реализован на языке C++ в составе ПК BigPixel. Для осуществления 2-мерного распределенного параллельного БПФ в программе использовались пакеты FFTW и Intel MKL.

Поведение эффективности алгоритма практически идентично поведению эффективности алгоритма БПФ (см. рис. 4), так как на каждом из k^2 шагов основное время занимает расчет двух двумерных БПФ, при этом время поточечного умножение матриц пренебрежимо мало, а БПФ матрицы φ осуществляется лишь один раз и результат его запоминается.

4.2.1. Пример численного расчета

В качестве тестовой топологии был выбран объект, состоящий из 40000×40000 расчетных пикселей и заполненный тестовыми мирами размером 57×37 пикселей каждая. Расчитываемая голограмма Габора была в 10 раз больше объекта, то есть $k = 10$. Этот расчет проводился на 640 вычислительных ядрах суперкомпьютера МВС100-К МСЦ РАН и занял 100 минут. При этом время осуществления каждого БПФ матрицы размера 79999×79999 из комплексных чисел двойной точности составляло 29 секунд. Для контроля правильности расчета голограммы проводилось восстановление (уже по формулам типа (8)) небольшого участка изображения, содержащего 9 мир. Результат этого восстановления показан на рис. 6.

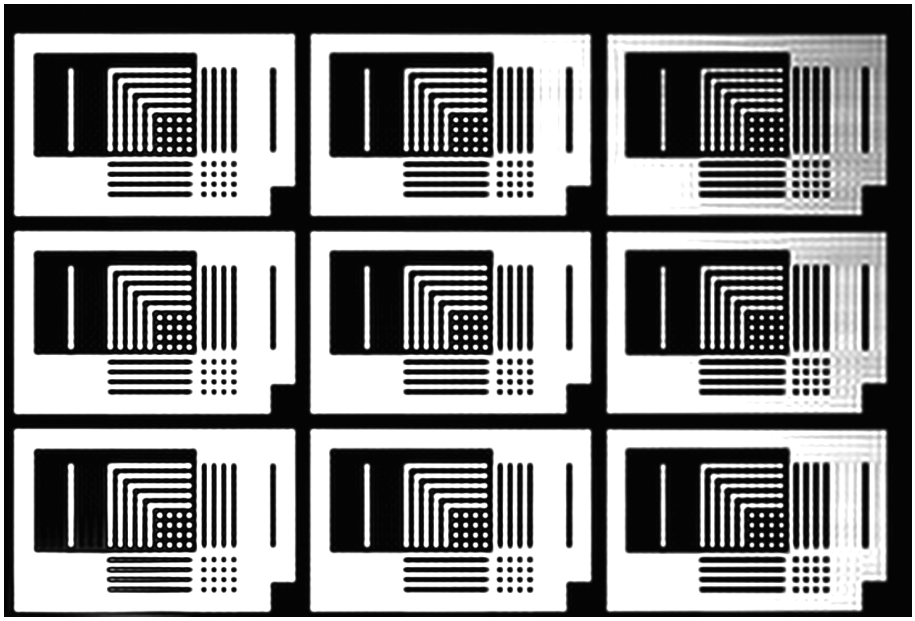


Рис. 6. Результат восстановления участка изображения для топологии, составленной из тестовых мир.

5. Заключение

В работе была исследована эффективность работы алгоритма расчета свертки суммированием излучения от прямоугольников и алгоритма двумерного БПФ на параллельных вычислительных системах кластерного типа. Показана хорошая масштабируемость первого алгоритма и низкая эффективность второго (БПФ) при увеличении кол-ва вычислительных ядер как внутри вычислительного узла, так и при увеличении кол-ва ВУ. Показана невозможность расчета обычными методами голограммы Габора для микрочипа с реальными размерами.

В статье описан метод Большого пиксела, позволяющий рассчитывать с его помощью

голограммы Габора для современных микрочипов с произвольной топологией на существующих параллельных вычислительных комплексах. Модифицированный, позволяющий еще на два порядка снизить объем требуемой оперативной памяти, вариант метода большого пиксела был реализован в составе параллельного программного комплекса BigPixel.

В случае, когда топология микросхемы имеет периодическую (регулярную) структуру, можно добиться повышения производительности алгоритма большого пиксела, если в качестве элементарных пикселов брать не маленькие квадратики, а эти повторяющиеся структуры.

В данной работе не затрагивались вопросы, связанные с улучшением получающегося изображения [12], тогда как из рис. 6 видно, что качество изображения оставляет желать лучшего. Можно предложить градиентный метод оптимизации, где для вычисления вариации соответствующего функционала необходимо будет рассчитывать несколько сверток типа (3). Следовательно, метод большого пиксела может быть применен в решении таких оптимизационных задач.

Литература

1. Борисов М.В., Боровиков В.А., Гавриков А.А, Князьков Д.Ю., Раховский В.И., Челюбеев Д.А., Шамаев А.С. Методы создания и коррекции качества голографических изображений геометрических объектов с элементами субволновых размеров // Доклады Академии Наук, 2010, т. 434, № 3, стр. 332-336.

2. Борисов М.В., Гавриков А.А, Князьков Д.Ю., Раховский В.И., Челюбеев Д.А., Шамаев А.С. Способ изготовления голографических изображений рисунка // Патент РФ 2396584, опубликован 10.08.2010г.

3. Rakhovskiy V.I., Borisov M.V., Shamaev A.S., Chelubeev D.A., Gavrikov A.A., Knyazkov D.U. Method of producing holographic images of IC topologies // US Patent App. №12836670, 2011.

4. Knyazkov D.Yu. Simulation of Holography Using Multiprocessor Systems // Springer LNCS, Mathematical Modeling and Computational Science, 2011, Vol. 7125 (in print).

5. Gabor D. A new microscopic principle // Nature, 1948, Vol. 161, pp. 777-778.

6. Singh V., Hu B., Bollepalli S., Wagner S., Borodovsky Y. Making a trillion pixels dance // Proc. of SPIE Vol. 6924, 0S1–0S12, 2008.

7. Stratton, J.A., Chu L.J. Diffraction theory of electromagnetic waves // Phys. Rev., 1939, Vol. 56, pp. 99-107.

8. Born M., Wolf E. Principles of Optics. Oxford, 1964

9. Басс Ф.Г., Фукс И.М. Рассеяние волн на статистически неровной поверхности. 1972. М.: Наука. 424 с.

10. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. М.: Бином, 2003, 632 с.

11. Нуссбаумер Г. Быстрое преобразование Фурье и алгоритмы вычисления сверток. М.: Радио и связь, 1985, 248 с.

12. Князьков Д.Ю. Оптимизация электромагнитных полей в голографической литографии с помощью метода локальных вариаций // Известия РАН. Теория и системы управления, 2011, № 6, стр. 99-109.

Подходы к оптимизации и распараллеливанию вычислений в задаче детектирования объектов разных классов на изображении*

Е.А. Козинов, В.Д. Кустикова, И.Б. Мееров, А.Н. Половинкин, А.А. Сиднев

Нижегородский государственный университет им. Н.И. Лобачевского

В работе рассматривается задача детектирования объектов разных классов на статических изображениях. Описывается схема решения данной задачи с использованием алгоритма Latent SVM. Используется известный подход к ускорению вычислений – построение каскада классификаторов. Обсуждаются проблемы распараллеливания и оптимизации времени расчета. Проводится анализ вариантов решения указанных проблем. Даются результаты вычислительных экспериментов, формулируются выводы и планы по дальнейшему развитию.

1. Введение

В настоящее время при решении многих практических задач используются системы компьютерного зрения (системы видеонаблюдения, управления процессами, организации информации и др.). Ядром вычислений в таких системах является обработка видеоданных. Поскольку основное применение результатов компьютерного зрения сосредоточено в области промышленной робототехники (автономные роботы, системы визуального контроля и измерений), то ключевым становится не только качество, но и скорость обработки видеоданных. Центральная задача, которая решается многими исследовательскими группами, – это анализ качественного состава сцены. При этом исследования ведутся как в направлении повышения точности распознавания объектов на кадрах видеопотока, так и в направлении снижения времени обработки видеоинформации.

В работе рассматривается задача детектирования объектов разных классов (автомобили, автобусы, люди и др.) на изображениях с использованием алгоритма поиска Latent SVM [4]. Приводится вычислительная схема решения данной задачи. Описывается один из наиболее общих алгоритмических подходов к ускорению вычислений в задачах компьютерного зрения – использование каскада классификаторов – на примере модификации Latent SVM [5]. Обсуждаются проблемы оптимизации и распараллеливания вычислений в системах с общей памятью. Выполняется анализ различных вариантов решения указанных проблем, предлагаются результаты вычислительных экспериментов на базе изображений PASCAL Visual Object Challenge 2007 (VOC2007, http://pascallin.ecs.soton.ac.uk/challenges/VOC). Разработка выполнена в рамках широко известной библиотеки компьютерного зрения OpenCV (http://sourceforge.net/projects/opencvlibrary).

2. Задача детектирования объектов на изображении с использованием методов, основанных на извлечении признаков

2.1 Постановка задачи

Задача детектирования объектов состоит в том, чтобы определить наличие объекта на изображении и найти его положение в системе координат пикселей исходного изображения. Положение объекта в зависимости от выбора алгоритма детектирования может определяться координатами прямоугольника, окаймляющего объект, либо контуром этого объекта, либо

* Работа выполнена в рамках программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы», государственный контракт № 11.519.11.4015.

координатами набора точек, наиболее характерных для объекта. В данной работе считается, что положение определяется координатами окаймляющего прямоугольника.

Все вычисления должны выполняться в реальном времени, поэтому основная цель настоящей работы состоит в том, чтобы максимально уменьшить среднее время поиска за счет оптимизации и распараллеливания вычислений.

2.2 Общая схема решения

Один из возможных подходов к решению задачи детектирования состоит в том, чтобы использовать алгоритмы машинного обучения для построения моделей классов объектов и алгоритмы вывода для поиска объектов на изображении.

Построение модели состоит из двух этапов:

1. Извлечение признаков, характерных для объектов класса, – построение характеристических векторов-признаков для особых точек объекта (углов, линий, ребер [15] или контуров объектов [16]) или для всего объекта.
2. Построение модели объекта на полученных признаках для последующего распознавания объектов. На данном этапе выполняется тренировка какого-либо классификатора.

Методы, основанные на извлечении признаков, описывают объект с использованием векторов-признаков. Вектора строятся на основании цветовой информации (гистограмма ориентированных градиентов (Histogram of Oriented Gradients или HOG) – один из наиболее популярных способов). Также может быть использована контекстная информация (context based) [17,18], а в некоторых случаях – данные о геометрии и взаимном расположении частей объекта (part-based) [4]. Тем не менее, все эти методы строят некоторую математическую модель объекта на каждом изображении тренировочной выборки, содержащем объект. Формально признак является числовой характеристикой. Таким образом, объект описывается набором векторов признаков в характерных точках. В результате тренировки строится модель, содержащая «усредненные» вектора признаков.

Алгоритм вывода (поиска) по существу включает два этапа:

1. Извлечение признаков объекта из тестового изображения согласно алгоритму, который использовался в процессе тренировки. При извлечении признаков возникает две основные проблемы:
   - на изображении может быть много объектов одного класса, а необходимо найти всех представителей. Поэтому необходимо просматривать все части изображения, проходя «бегущим» окном (sliding window) от левого верхнего до правого нижнего угла. При этом размер окна определяется размером изображений тренировочной выборки;
   - объекты на изображении могут иметь разный масштаб. Самое распространенное решение – масштабирование изображения и построение пирамиды изображений.
2. Поиск объектов на изображении. Входными данными этапа поиска являются формальное описание объекта – набор признаков, которые выделены из тестового изображения, – и модель класса объектов. На основании этой информации классификатор принимает решение о принадлежности объекта классу. Некоторые методы поиска также оценивают степень достоверности того, что объект принадлежит рассматриваемому классу.

Качество методов в основном зависит от того, насколько хорошо выбраны признаки, т.е. насколько хорошо эти признаки дифференцируют классы объектов. Существуют специализированные методы, основанные на извлечении признаков, например, для детектирования лиц [7, 8, 10], транспортных средств [11] и пешеходов [12, 13, 14].

3. Поиск объектов с использованием алгоритма Latent SVM

3.1 Модель объектов некоторого класса

При построении алгоритма Latent SVM модель объекта некоторого класса определяется набором компонент, каждая из которых соответствует одному из ракурсов этого объекта. Компонента включает совокупность фильтров [4]:

- грубый фильтр, определяющий набор векторов признаков, наиболее характерных для всего объекта;
- совокупность точных фильтров, описывающих отдельные части объекта.

3.2 Вычислительная схема поиска объектов

Алгоритм поиска Latent SVM состоит из двух этапов:

1. Построение пирамиды признаков.
2. Определение наиболее вероятных положений объектов заданного класса.

Построение пирамиды признаков включает масштабирование изображений – формирование пирамиды изображений [1, 4] – и построение матриц векторов-признаков для каждого изображения в пирамиде. Вектор признаков представляет гистограмму ориентации градиентов (Histogram of Oriented Gradients (HOG), [4]) для некоторого блока пикселей исходного изображения.

Смысл этапа поиска положения объекта состоит в том, чтобы некоторым образом оценить вероятность нахождения объекта во всех возможных положениях пирамиды признаков и выбрать среди всего множества наиболее вероятное. Для определения положения объекта в алгоритмах [4, 5] вычисляется значение оценочной функции для каждого возможного положения. Расположение объекта определяется положением левого верхнего угла грубого фильтра в наборе векторов матрицы признаков какого-либо уровня. Оценочная функция строится как сумма скалярных произведений наборов векторов признаков грубого и точных фильтров модели с соответствующими наборами векторов признаков, построенных на изображении. В дальнейшем сумму скалярных произведений векторов фильтра и изображения будем называть сверткой. Дополнительно оценочная функция содержит слагаемое, которое определяет штраф за чрезмерное удаление частей объекта от самого объекта. Коэффициенты функции штрафа являются параметрами компоненты модели.

4. Каскад классификаторов

4.1 Схема построения каскада классификаторов

Построение и использование каскада классификаторов – один из наиболее распространенных подходов [2, 3] к ускорению вычислений при решении различных задач классификации. По существу идея построения аналогична AdaBoost-классификатору (adaptive boost, [6]). На каждой стадии тренировки строится некоторый классификатор, при этом учитывается информация об ошибках предшествующего классификатора. Итоговый классификатор представляется комбинацией построенных классификаторов.

При решении задач детектирования и классификации объектов на изображениях последовательность классификаторов, полученных на этапе тренировки, сортируется в зависимости от величины ошибки. Применение первых более слабых классификаторов позволяет последовательно отсекать положения, в которых заведомо не может быть обнаружен объект. Таким образом, сильные классификаторы (как правило, вычислительно более трудоемкие) применяются не ко всем возможным положениям объекта, а только к некоторому небольшому набору.

Каскад классификаторов нашел применение при решении многих задач компьютерного зрения. Наиболее известным алгоритмом, в основе которого лежит каскад классификаторов, является алгоритм Виолы-Джонса для детектирования лиц [7, 8]. Идея построения каскада также используется при реализации каскадного преобразования Хафа [9], которое позволяет искать структурные объекты на изображении (линии, окружности и т.п.), и многих других алгоритмов. Далее рассмотрим каскадную схему алгоритма Latent SVM, предназначенного для детектирования объектов разных классов.

4.2 Детектирование объектов с использованием каскадного Latent SVM

4.2.1 Модель объектов некоторого класса

В каскадном Latent SVM [5] в модель объектов каждого класса включается дополнительная информация:

- PCA-проекции грубого и точных фильтров в пространство меньшей размерности;
- по два порога для каждого фильтра и его проекции:
  - порог идентификации присутствия объекта или его части, который определяет возможность присутствия объекта или его части в определенном месте изображения;
  - порог гипотетического отклонения определяет, насколько часть объекта может быть смещена от своего идеального расположения относительно грубого фильтра.

4.2.2 Вычислительная схема поиска объектов

Каскадный алгоритм Latent SVM [5] включает два этапа каскадной схемы, на которых выполняются следующие действия:

1. Построение множества возможных положений объекта. Для этого применяется преобразованная дополнительная пирамида признаков и соответствующие ей фильтры. На данном этапе осуществляется:

   1.1. Вычисление свертки грубого фильтра (PCA) в первом возможном положении (например, в левом верхнем углу изображения на одном из уровней). Если свертка меньше порога идентификации для грубого фильтра, то необходимо выполнить переход к рассмотрению следующего возможного положения грубого фильтра.

   1.2. Перебор сверток точных фильтров.

   1.2.1. Выбор первого фильтра и вычисление его свертки с векторами матрицы признаков в некоторой окрестности грубого фильтра.

   1.2.2. Проверка условия: если сумма свертки точного фильтра и текущей оценки положения объекта меньше порога гипотетического отклонения, то положение грубого фильтра не рассматривается.

   1.2.3. Выбор положения с максимальной суммой текущей оценки и свертки среди всех подсчитанных положений. Сумма принимается за текущую оценку положения объекта.

   1.2.4. Проверка условия: если полученная оценка положения объекта меньше порога деформации для текущего точного фильтра, то рассматривается следующее положение грубого фильтра, иначе анализируется положение следующего точного фильтра.

   1.3. Если для всех точных фильтров было найдено расположение, то положение грубого фильтра считается возможным положением объекта на изображении текущего масштаба.
2. Определение наиболее вероятных положений объекта среди множества, построенного на предыдущем шаге. На данном этапе используется полная пирамида признаков. Как правило, количество возможных положений не очень большое. Процесс проверки существования объекта на полной пирамиде производится по той же схеме, что и для свернутой пирамиды признаков.

5. Этапы оптимизации и распараллеливания последовательной реализации алгоритма Latent SVM

5.1 Тестовая инфраструктура

На каждом этапе оптимизации и распараллеливания последовательной реализации Latent SVM в качестве тестового набора использовалась база данных конкурса PASCAL VOC 2007, содержащая различные изображения объектов двадцати классов (aeroplane, bicycle, bird, bottle и др.). Представленные фотографии различаются размером изображенных на них объектов, их

положением на сцене, ракурсом и степенью освещенности. Указанные факторы оказывают значительное влияние на точность построенной модели.

Так как исходная задача состояла в реализации алгоритма поиска и не включала в себя реализацию алгоритма обучения, для проведения экспериментов были использованы модели авторов статей [4, 5], преобразованные в формат xml в соответствии с разработанной структурой.

Оценка качества детектирования объектов с помощью реализованного алгоритма вывода выполнялась средствами VOC Development Kit, в состав которого входит модуль, позволяющий вычислить среднюю точность предсказания (average precision). Указанная метрика определяется как математическое ожидание точностей следующим образом:

$$AP = \frac{1}{11} \sum_{r \in [0;0.1;...;1]} p(r), \qquad p(r) = \max_{\bar{r}:\bar{r} \geq r} p(\bar{r})$$

где $p(\bar{r}) = \frac{a}{a+c}$ – точность, $\bar{r}$ – процент перекрытия детектированного окаймляющего прямоугольника и прямоугольника, который был размечен на исходном изображении как окаймляющий $\bar{r} \in [0;0.1;0.2;...;1]$, a – количество объектов, для которых процент перекрытия не меньше, чем $\bar{r}$ (т. е. считается, что объект детектирован правильно), c – количество объектов с процентом перекрытия, меньшим, чем $\bar{r}$ (объект найден ошибочно).

Вычислительные эксперименты проводились с использованием следующей инфраструктуры:

- Язык разработки: C.
- Среда разработки: Microsoft Visual Studio 9.0.
- Компилятор: Microsoft C/C++ Compiler Version 15.00.30729 (x64).
- Процессор: 2 двухъядерных процессора Intel Xeon 5150 (2.66 GHz).
- Память: 4 Gb.
- Библиотеки: Intel Threading Building Blocks (TBB) 3.0 for Windows, v.3.0.2010.707.
- Технологии: OpenMP.
- Операционная система: Microsoft Windows Server 2008 Standard SP1 x64.

Далее в работе приводится последовательность шагов оптимизации и распараллеливания. На каждом этапе, который дает выигрыш по отношению к реализациям авторов [4, 5], приводятся результаты экспериментов по средней точности детектирования и среднему времени детектирования объектов на одном изображении. Объем тестовой выборки составляет примерно 5000 изображений.

5.2 «Горячие» точки последовательной реализации

Первый шаг при выполнении оптимизации заключается в локализации «горячих» точек программы. Для этого мы воспользовались инструментом Intel Parallel Amplifier в режиме “Hotspots”. Результат анализа показал, что в разработанной вычислительной схеме метода Latent SVM основная операция – это вычисление сверток. Входными данными операции являются:

1. Двумерная матрица векторов признаков *featureMap* (матрица признаков на некотором уровне пирамиды признаков). Данная матрица может быть представлена в виде трехмерного прямоугольного параллелепипеда размерности $N_y^f N_x^f p$, где N_x^f – число столбцов матрицы векторов, N_y^f - количество строк матрицы векторов, $p = 31$ – размерность вектора признаков. Двумерную матрицу, полученную при каждом фиксированном p, будем называть каналом.
2. Двумерная матрица весовых векторов фильтра *filter* (это может быть точный или грубый фильтр). Данная матрица может быть представлена в виде трехмерного прямоугольного параллелепипеда размерности $N_y^g N_x^g p$, где N_x^g – число столбцов матрицы весовых

векторов, N_y^g - количество строк матрицы весовых векторов, $p = 31$ – размерность вектора весов.

Результатом выполнения операции свертки является двумерная матрица свертки *conv* размерности $\left(N_y^f - N_y^g + 1\right)\left(N_x^f - N_x^g + 1\right)$.

Задача вычисления свертки матрицы векторов признаков и фильтра сводится к тому, чтобы определить значения элементов матрицы свертки в соответствии с формулой:

$$conv[i,j] = \sum_{di=0}^{\left(N_y^g-1\right)} \sum_{dj=0}^{\left(N_x^g-1\right)} \sum_{k=0}^{p-1} filter[di,dj,k] \cdot featureMap[i+di, j+dj, k]$$

По существу с точки зрения программной реализации получается цикл вложенности 5. Чтобы ускорить вычисление сверток, были выполнены некоторые компиляторные оптимизации (упрощение циклов, вынос инвариантов), но значительный выигрыш не был получен, т.к. очевидно, что компилятор выполнял эти оптимизации автоматически. Распараллеливание циклов в функции вычисления сверток не дает выигрыша, т.к. свертки вычисляются многократно, как следствие, возникают значительные накладные расходы на организацию параллелизма (возобновление и остановка потоков). Именно такие результаты были получены в процессе анализа степени параллелизма разработанной реализации. Таким образом, распараллеливание необходимо выполнять на более высоком уровне.

5.3 Распараллеливание по уровням пирамиды признаков

Как выяснилось ранее, основное время занимает процедура поиска положения объектов (второй этап вычислительной схемы) и наиболее трудоемкой операцией является вычисление сверток. Заметим, что поиск объектов «бегущим» окном выполняется на каждом уровне пирамиды признаков независимо. Таким образом, можно выполнить распараллеливание разработанной последовательной реализации по уровням пирамиды. Более того, на каждом уровне можно подсчитать количество выполняемых операций, как следствие, достаточно просто построить статическое расписание распределения нагрузки между потоками. Предлагается использовать следующую схему распределения – каждому потоку отдавать на обработку набор уровней так, чтобы суммарное количество операций было приближенно одинаковым. Такой подход обеспечит равномерность распределения нагрузки между потоками. Описанная схема распараллеливания была реализована с помощью средств библиотеки TBB.

Таблица 1. Средняя точность детектирования объектов на данных VOC 2007

Название класса объектов	Средняя точность			Время работы (с)		Ускорение (отношение времени последовательной к параллельной)
	Последовательная реализация	Реализация авторов [4]	Параллельная реализация в 4 потока	Последовательная реализация	Текущая реализация в 4 потока	
aeroplane	0,278	0,28	0,279	4,205	1,263	3,329
bicycle	0,525	0,544	0,525	4,412	1,303	3,386
bird	0,006	0,007	0,007	3,746	1,174	3,191
boat	0,123	0,145	0,124	3,623	1,157	3,131
bottle	0,26	0,262	0,261	3,438	1,125	3,056
bus	0,409	0,398	0,409	4,435	1,304	3,401
car	0,457	0,463	0,457	3,215	1,079	2,98
cat	0,163	0,16	0,164	5,032	1,402	3,589
chair	0,154	0,163	0,155	4,89	1,374	3,559
cow	0,167	0,167	0,167	4,696	1,355	3,466

diningtable	0,238	0,243	0,239	5,185	1,432	3,621
dog	0,07	0,05	0,07	4,85	1,372	3,535
horse	0,437	0,438	0,437	5,117	1,422	3,598
motorbike	0,384	0,382	0,385	5,12	1,42	3,606
person	0,338	0,342	0,338	3,747	1,188	3,154
pottedplant	0,101	0,079	0,101	3,417	1,117	3,059
sheep	0,166	0,172	0,166	3,13	1,061	2,95
sofa	0,214	0,221	0,215	4,998	1,399	3,573
train	0,331	0,34	0,331	4,208	1,362	3,089
tvmonitor	0,38	0,393	0,381	4,208	1,334	3,154
Средние значения:				4,283	1,282	3,321

Таблица 1 содержит результаты вычислительных экспериментов, проведенных над разработанной последовательной и параллельной реализациями. Очевидно, что средняя точность детектирования объектов с помощью подготовленных реализаций (второй и четвертый столбцы) приближенно совпадает с точностью, показанной авторами статьи [4] (третий столбец). Отметим, что, несмотря на практически идентичное качество детектирования, последовательная реализация в среднем обрабатывает одно изображение в 2-2.5 раза медленнее, чем реализация авторов алгоритма, которая решает задачу детектирования объектов одного класса на изображении примерно за 2 секунды [4]. Но дальнейшее распараллеливание последовательной версии по уровням пирамиды признаков в среднем дает ускорение в 3,3 раза на четырех потоках (последний столбец таблицы), в результате чего время обработки было уменьшено в среднем до 1,28 секунды. Указанное ускорение – неплохой результат при условии, что распараллелены не все этапы алгоритма, а только поиск объектов, который занимает ориентировочно 80% времени от общего времени поиска объектов заданного класса на изображении.

5.4 Реализация каскадного алгоритма Latent SVM

Следующим шагом оптимизации является реализация каскадной схемы, описанной в предыдущем разделе.

В таблице 2 приведены численные значения средней точности, полученные на каскадной реализации алгоритма вывода Latent SVM. Сравнение с результатами, приведенными в [5], показывает, что практически на всех классах объектов наблюдаются незначительные отклонения. Также, в таблице представлены результаты точности детектирования для выполненной последовательной реализации алгоритма [4] на моделях из [5], содержащих большее количество компонент. Из результатов сравнения можно видеть, что точность детектирования во всех трех реализациях практически совпадает. В таблице 1 полужирным начертанием выделены наилучшие точности детектирования для каждого класса объектов.

Наибольший интерес в данной работе представляет эффект от применения каскадной схемы. Поэтому обратим внимание на ускорение, которое получено по отношению к предшествующей реализации. Максимальное ускорение составляет 8,45 раза, среднее – примерно 6,5 раза. Очевидно, что применение каскадной схеме позволяет значительно уменьшить среднее время детектирования без потери в средней точности.

Таблица 2. Средняя точность детектирования объектов на данных VOC 2007

Название класса объектов	Средняя точность		Время работы (с)		Ускорение (отношение времени каскадной к предыдущей)
	Каскадная реализация	Реализация авторов [5]	Каскадная реализация	Предыдущая реализация	
aeroplane	0,28	0,23	1,37	10,37	7,57
bicycle	**0,57**	0,49	1,38	11,42	8,28
bird	0,09	**0,11**	1,72	10,76	6,26

boat	**0,15**	0,13	2,14	11,2	5,23
bottle	0,24	**0,27**	2	10,19	5,1
bus	0,47	**0,47**	1,46	11,05	7,57
car	**0,54**	0,50	1,78	9,45	5,31
cat	0,16	**0,19**	1,35	10,8	8
chair	**0,20**	0,16	2,28	10,94	4,8
cow	**0,24**	0,23	1,57	10,88	6,93
diningtable	**0,22**	0,11	1,61	11,14	6,92
dog	0,11	**0,12**	1,35	11,41	8,45
horse	**0,56**	0,36	1,54	10,88	7,06
motorbike	**0,45**	0,37	1,29	10,87	8,43
person	**0,41**	0,38	2,82	10,12	3,59
pottedplant	0,12	**0,14**	2,38	9,71	4,08
sheep	0,18	**0,23**	1,73	9,18	5,31
sofa	0,29	0,23	1,52	10,57	6,95
train	**0,44**	0,34	1,42	11,53	8,12
tvmonitor	0,40	**0,40**	1,66	11,35	6,84
Средние значения:			1,72	10,69	6,54

5.5 Оптимизация и распараллеливание каскадного алгоритма Latent SVM

На данный момент необходимо оценить среднее время обработки одного изображения с использованием разработанной каскадной реализации. Первый шаг состоит в том, чтобы определить «горячие» точки в процедуре поиска. Аналогично предыдущей реализации наиболее затратной операцией является операция вычисления сверток. Отметим, что при этом свертки вычисляются не во всех возможных положениях объекта (не на всей матрице признаков), а только на некотором наборе наиболее вероятных, которые остались после отсечения. Соотношение времени построения пирамиды признаков и времени вычисления сверток близко к единице. На данном этапе операция вычисления сверток была переписана с использованием SSE. В результате в среднем выигрыш от такой оптимизации составил 0,1 секунды (четвертый и пятый столбцы таблицы). Далее было выполнено распараллеливание оптимизированной версии.

Таблица 3. Средняя точность детектирования объектов на данных VOC 2007

Название класса объектов	Средняя точность		Время работы (с)				Ускорение	
	Параллельная каскадная реализация	Реализация авторов [5]	Исходная реализация	Последовательная каскадная реализация	Каскадная реализация после оптимизации	Параллельная реализация в 4 потока	Отношение параллельной каскадной к последовательной	Отношение предыдущей к параллельной
aeroplane	0,28	0,23	10,37	1,37	1,37	0,51	2,69	20,33
bicycle	**0,57**	0,49	11,42	1,38	1,30	0,50	2,60	22,84
bird	0,09	**0,11**	10,76	1,72	1,60	0,56	2,86	19,21
boat	**0,15**	0,13	11,2	2,14	1,97	0,77	2,56	14,55
bottle	0,24	**0,27**	10,19	2,00	1,81	0,66	2,74	15,44
bus	0,47	**0,47**	11,05	1,46	1,35	0,55	2,45	20,09
car	**0,54**	0,50	9,45	1,78	1,62	0,60	2,70	15,75
cat	0,16	**0,19**	10,8	1,35	1,29	0,53	2,43	20,38
chair	**0,20**	0,16	10,94	2,28	2,08	0,75	2,77	14,59
cow	**0,24**	0,23	10,88	1,57	1,47	0,55	2,67	19,78

diningtable	**0,22**	0,11	11,14	1,61	1,49	0,58	2,57	19,21
dog	0,11	**0,12**	11,41	1,35	1,31	0,50	2,62	22,82
horse	**0,56**	0,36	10,88	1,54	1,44	0,54	2,67	20,15
motorbike	**0,45**	0,37	10,87	1,29	1,23	0,49	2,51	22,18
person	**0,41**	0,38	10,12	2,82	2,55	0,86	2,97	11,77
pottedplant	0,12	**0,14**	9,71	2,38	2,17	0,72	3,01	13,49
sheep	0,18	**0,23**	9,18	1,73	1,57	0,59	2,66	15,56
sofa	0,29	0,23	10,57	1,52	1,43	0,55	2,60	19,22
train	**0,44**	0,34	11,53	1,42	1,33	0,53	2,51	21,75
tvmonitor	0,40	**0,40**	11,35	1,66	1,62	0,60	2,70	18,92
Средние значения:			10,69	1,72	10,69	0,597	2,66	18,40

Обратимся к анализу каскадной схемы и возможности применения распараллеливания по уровням пирамиды признаков. На первом этапе каскадной схемы отсекаются наименее вероятные положения объектов. Как следствие, при переходе на следующий этап на каждом уровне пирамиды останутся достоверные положения объекта, количество которых заранее не известно. При этом невозможно спрогнозировать, в какой момент произойдет отсечение (отбрасывание) следующего положения, а потому нельзя вычислить количество выполняемых операций. Реализация такой схемы распараллеливания не даст выигрыша, т.к. невозможно достигнуть равномерного распределения нагрузки между потоками. Поэтому было принято решение параллелить каскадную схему на уровне компонент модели объектов класса. В таблице 3 показаны результаты экспериментов с параллельной реализацией каскадного Latent SVM, выполненной с использованием технологии OpenMP. Разработанная реализация не уступает по точности детектирования реализации авторов [5], а на некоторых классах объектов, напротив, показывает более высокие результаты (выделены полужирным начертанием во втором столбце). В среднем ускорение составило 2,66 раза на четырех потоках. Отметим, что указанная величина во многом определяется количеством возможных положений, которые попадают на вход второго этапа каскадной схемы в каждой компоненте. Таким образом, ускорение может принимать недетерминированные значения. В результате распараллеливания оптимизированной версии каскада в среднем время поиска уменьшилось в 18,4 раза по сравнению с исходной последовательной реализацией (последний столбец таблицы 3).

6. Заключение

В ходе работы выполнены последовательная и параллельная реализации алгоритма вывода Latent SVM [4], позволяющие детектировать объекты с точностью, которая не уступает реализации разработчиков алгоритма, о чем свидетельствуют результаты проведенных экспериментов. Реализации интегрированы в библиотеку с открытыми исходными кодами OpenCV, использование которой не требует дополнительного дорогостоящего программного обеспечения. Использование разработанных реализаций осложняется существенным временем поиска. Реализация каскадной схемы алгоритма Latent SVM [5] позволила уменьшить время поиска объектов на тестовой базе VOC 2007 в среднем в ~18 раз. В настоящее время выполняется интеграция каскадного Latent SVM в библиотеку OpenCV. Изучается возможность повышения качества указанного алгоритма для детектирования пешеходов и транспортных средств, а также исследуются направления дальнейшей оптимизации вычислений при решении задачи многоклассового детектирования объектов.

Литература

1. Форсайт Д., Понс Ж. Компьютерное зрение. Современный подход. – М.: Изд. д. Вильямс, 2004. – 465с.
2. Szeliski R. Computer Vision: Algorithms and Applications. – Springler, 2010. – 979p.
3. Sonka M., Hlavac V., Boyle R. Image Processing, Analysis and Machine Vision. – Thomson, 2008. – 866p.

4. Felzenszwalb P. F., Girshick R. B., McAllester D., Ramanan D. Object Detection with Discriminatively Trained Part Based Models // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2010. Vol.32, No.9. – pp. 1627–1645.
5. Felzenszwalb P. F., Girshick R. B., McAllester D., Ramanan D. Cascade object detection with deformable path model// Proceedings of the IEEE CVPR 2010.
6. Hastie T., Tibshirani R., Freidman J. The elements of statistical learning. Data mining, inference and prediction. – 2001. – 745p.
7. Viola P., Jones M.J. Robust Real-Time Face Detection // international Journal of Computer Vision 57(2). – 2004. – pp. 137-154.
8. Viola P., Jones M.J. Rapid object detection using a boosted cascade of simple features // In Proceedings IEEE Conf. on Computer Vision and Pattern Recognition. – 2001.
9. T. Tuytelaars, M. Proesmans, L. Van Gool The cascaded Hough transform, Int'l Conf. on Image Processing, ICIP-97, vol. II, pp. 736-739, 1997.
10. Pentland A., Choudhury T. Face Recognition for Smart Environments // IEEE Computer Vision. – 2000. – pp. 50-55.
11. Alonso D., Saldaro L., Nieto M. Robust Vehicle Detection through Multidimensional Classification for On Broad Video Based Systems // IEEE. – 2007.
12. Dalal N., Triggs B. Histograms of oriented gradients for human detection // In Proceedings CVPR'05. – 2005.
13. Viola P., Jones M.J., Snow D. Detecting pedestrians using patterns of motion and appearance // In: Proceedings of the 9th International Conference on Computer Vision (ICCV), Vol. 1 – 2003. – pp. 734-741.
14. Gavrila D. M., Giebel J., Munder S. Vision-based pedestrian detection: the protector system // Proceedings of the IEEE Intelligent Vehicles Symposium, Parma, Italy. – 2004. – pp. 13-18.
15. Amit Y. 2D Object Detection and Recognition: models, algorithms and networks. – The MIT Press, 2002. – 325p.
16. Shotton J., Blake A., Cipolla R. Contour-based Learning for Object Detection // 10th IEEE International Conference on Computer Vision (ICCV'05). 2005. Vol.1. – P. 503-510.
17. Torralba A., Murphy K.P., Freeman W.T., Rubin M.A. Contex-based Vision System for Place and Object Recognition // 9th IEEE International Conference on Computer Vision (ICCV'03). 2003. Vol.1. – pp. 273-283.
18. Myung Jin Choi, Lim, J.J., Torralba, A., Willsky, A.S. Exploiting Hierarchical Contex on a large database of object categories // IEEE Computer Vision and Pattern Recognition (CVPR'10). 2010. – pp. 129-136.

MPIPerf: пакет оценки эффективности коммуникационных функций стандарта MPI*

М.Г. Курносов[1]

Институт физики полупроводников им. А.В. Ржанова СО РАН[1]

Рассмотрены методы измерения времени выполнения коллективных операций обмена информацией (Collective communications) между ветвями параллельных MPI-программ. Отражены систематические ошибки измерений характерные для программных пакетов, реализующих эти методы. Предложен подход к измерению времени выполнения коллективных операций обменов, основанный на синхронизации моментов запуска коммуникационных функций в ветвях MPI-программы. Приведено описание разработанного пакета MPIPerf и результаты натурных экспериментов на вычислительных кластерах с сетями связи Gigabit Ethernet и InfiniBand QDR.

1. Введение

Коммуникационные библиотеки стандарта MPI (MPICH2, Open MPI, Cray MPI и др.) являются основным средством создания параллельных программ для распределенных вычислительных систем (ВС). Основу этих библиотек составляют коммуникационные функции дифференцированных (Point-to-point communications) и коллективных операций (Collective communications) обмена информацией между ветвями параллельных программ. Коллективные операции подразделяются на несколько видов [1, 2]: трансляционный (ТО, One-to-all broadcast), трансляционно-циклический (ТЦО, All-to-all broadcast), коллекторный обмены (КО, All-to-one broadcast) и коллективные операции синхронизации ветвей (Barrier, Eureka). Коллективные операции реализуется на основе дифференцированных обменов (Point-to-point communications) или аппаратурно с использованием специализированных коммуникационных сетей (например, на базе сети с древовидной топологией в системах семейства IBM Blue Gene). Для широкого класса параллельных алгоритмов время выполнения коллективных операций является критически важным и определяет их масштабируемость [3, 4].

Коллективные операции обменов информацией реализуются программно в виде коммуникационных функций. Например, операции ТО и КО в библиотеках стандарта MPI реализуются функциями: `MPI_Bcast`, `MPI_Gather`; в библиотеках Cray Shmem: `shmem_broadcast`, `shmem_collect`; в языке параллельного программирования Unified Parallel C: `upc_all_broadcast`, `upc_all_gather`.

Пользователями и разработчиками системного программного обеспечения распределенных ВС востребованы средства для оценки времени выполнения функций, реализующих коллективные операции обменов. Такая информация необходима для определения оптимального алгоритма реализации коллективной операции – алгоритма, обеспечивающего минимум времени её выполнения, для определения эффективности реализаций коммуникационных библиотек и runtime-систем языков параллельного программирования, а также для верификации аналитических (LogP, PLogP, LogGP) и имитационных моделей (LogGOPSim, PSINS, DIMEMAS, BigSim) выполнения параллельных программ [5].

В данной работе приводится обзор известных методов измерения времени выполнения коллективных операций обмена информацией. Отражены систематические ошибки измерений характерные для программных пакетов реализующих эти методы. Предложен подход к измерению времени выполнения коллективных операций, в котором учтены известные систематические ошибки измерений, характерные для существующих пакетов.

* Работа выполнена при поддержке РФФИ (грант № 11-07-00105), Совета по грантам Президента РФ для поддержки ведущих научных школ (грант НШ-2175.2012.9) и в рамках госконтракта № 07.514.11.4015 с Минобрнауки РФ.

2. Методы измерения времени выполнения коллективных операций обмена информацией

Сформулируем суть задачи измерения времени выполнения функции, реализующей коллективную операцию обмена информацией между ветвями параллельной программы. Задана функция и значения ее входных параметров (буферы приема-передачи, размеры сообщений и номера ветвей, участвующих в операции). Требуется измерить время $t_0, t_1, \ldots, t_{n-1}$ выполнения функции в n ветвях программы на заданной подсистеме ЭМ.

В основе большинства известных методов (пакеты Intel MPI Benchmarks [6], mpptest [7], LLCBench [8], Phloem [9], MPIBlib [10], OSU Micro-Benchmarks [11] и др.) время выполнения коллективной операции оценивается путем измерения времени k выполнений функции (рис. 1, здесь и далее примеры приводятся для функций стандарта MPI). За время t выполнения функции в ветви принимается среднее время одного её запуска (среднее время одной итерации цикла измерений).

```
MPI_Bcast(buf, count, MPI_BYTE, root, comm)              /* Инициализация */
MPI_Barrier(comm)                                        /* Синхронизация */
t = MPI_Wtime()
for i = 1 to k do                                       /* Цикл измерений */
    MPI_Bcast(buf, count, MPI_BYTE, root, comm)
end for
t = (MPI_Wtime() – t) / k                 /* Среднее время одного запуска */
```

Рис. 1. Процедура измерения времени выполнения функции `MPI_Bcast`

Время выполнения коллективной операции зависит от значений её входных параметров. Так, на время выполнения функции трансляционного обмена `MPI_Bcast` влияют способ выделения памяти для буфера приема-передачи `buf` (выбор границы выравнивания адреса буфера, размещение передаваемой информации в кэш-памяти процессора), тип передаваемых данных (встроенный тип данных MPI с непрерывным размещением элементов в памяти или производный тип данных с размещением элементов в памяти по несмежным адресам), выбор корневого процесса `root` и коммуникатора `comm`, который задает распределение ветвей программы по ЭМ системы. Кроме того, на время выполнения функции оказывают влияние и ветви других программ, которые загружают совместно используемые ресурсы системы (коммуникационную сеть, контроллер(ы) оперативной памяти вычислительных узлов и др.). По этой причине результаты измерений времени выполнения коллективных операций известными пакетами различны. Кроме этого, существующим методам присущи систематические ошибки измерений.

3. Систематические ошибки измерения времени выполнения коллективных операций обмена информацией

Рассмотрим распространенные систематические ошибки измерений времени выполнения коллективных операций обмена информацией, характерные для существующих методов и программных пакетов, реализующих их.

3.1 Игнорирование отложенной инициализации MPI-функций

Время выполнения коллективной операции измеряется без её предварительной инициализации (пакеты Intel MPI Benchmarks, Phloem, MPIBlib, SKaMPI [12]). Во многих библиотеках стандарта MPI и системах параллельного программирования ветви программы, участвующие в коллективной операции, устанавливают между собой сетевое соединение только при её первом вызове (Late initialization). Поэтому время выполнения первого вызова может значительно превосходить время выполнения последующих обращений к функции. Результаты первого вызова не должны учитываться в итоговой оценке времени выполнения операции. В табл. 1 приведены отношения времени выполнения ветвью 0 первого вызова MPI-функции ко второму вызову.

Результаты получены на вычислительном кластере с коммуникационной сетью InfiniBand QDR (64 процесса – 8 узлов по 8 ядер, при повторном вызове функции использовался буфер такого же размера – 8192 байт, но размещенный по другому адресу в оперативной памяти).

Таблица 1. Отложенная инициализация MPI-функций

MPI-функция	Отношение времени выполнения ветвью 0 первого вызова MPI-функции к её второму вызову	
	Библиотека OpenMPI 1.4.4	**Библиотека Intel MPI 4.0.0.028**
`MPI_Bcast`	6178	2,59
`MPI_Allgather`	5,61	131,8
`MPI_Barrier`	5,50	9,01
`MPI_Reduce`	6944	12901
`MPI_Allreduce`	11,71	353,37

3.2 Учёт времени выполнения коллективной операции только в одной ветви

В качестве конечного значения времени выполнения коллективной операции обмена информацией принимается только время определенной ветви (например, ветви 0). Однако, время выполнения операции в ветвях различно. Это обусловлено тем, что алгоритмы коллективных операций реализуются на основе дифференцированных обменов. В зависимости от используемого алгоритма коллективной операции ветви выполняют различное количество таких обменов [2, 13, 14]. Например, на рис. 2 приведена диаграмма выполнения коллекторного приема информации (`MPI_Reduce`) алгоритмом биномиального дерева (Binomial tree) [14].

За время выполнения коллективной операции обмена информацией следует принимать максимальное из времен выполнения операции в ветвях программы.

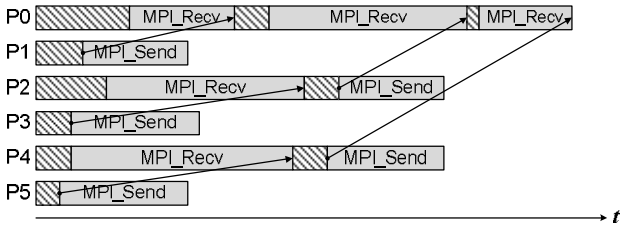


Рис. 2. Диаграмма выполнения коллекторного приема информации (`MPI_Reduce`) ветвью 0 (6 ветвей, результаты получены пакетом Intel Trace Analyzer and Collector)

3.3 Измерение времени выполнения MPI-функции при разных значениях входных параметров

На каждой итерации цикла измерений (рис. 1) используются различные значения входных параметров. Например, в пакетах Intel MPI Benchmarks, mpptest при измерении времени выполнения функции `MPI_Bcast` на каждой итерации цикла номер `root` корневой ветви изменяется. Это приводит к тому, что при выполнении коллективной операции используются различные каналы связи (меняется последовательность обменов сообщениями между ветвями).

3.4 Игнорирование иерархической организации подсистемы памяти вычислительных узлов

В некоторых пакетах не учитывается то, что при повторном использовании одного и того же буфера приема/передачи его данные с большой вероятностью будут размещены в кэш-памяти процессора. Это приводит к тому, что время выполнения первого и последующих вызовов коллективной операции могут значительно отличаться.

3.5 Некорректная синхронизация ветвей программы перед выполнением коллективной операции

Синхронизация ветвей параллельной программы перед и/или в цикле измерений выполняется при помощи барьерной синхронизации (`MPI_Barrier`). Это приводит к неравномерному смещению моментов запуска коллективной операции в ветвях. Причина в том, что барьерная синхронизация реализуется путем дифференцированных обменов сообщениями нулевой длины [13, 14]. Количество таких обменов, выполняемых каждой ветвью, различно и зависит от используемого алгоритма барьерной синхронизации. В конечном счете, ветви осуществляют выход из функции синхронизации в разные моменты времени. На рис. 3 приведен пример выполнения барьерной синхронизации "рассеивающим" алгоритмом (Dissemination barrier) [15]. Каждая из 8 ветвей выполняет два вызова операции приёма-передачи (`MPI_Sendrecv`). Видно, что ветви заканчивают выполнение операции в различные моменты времени.

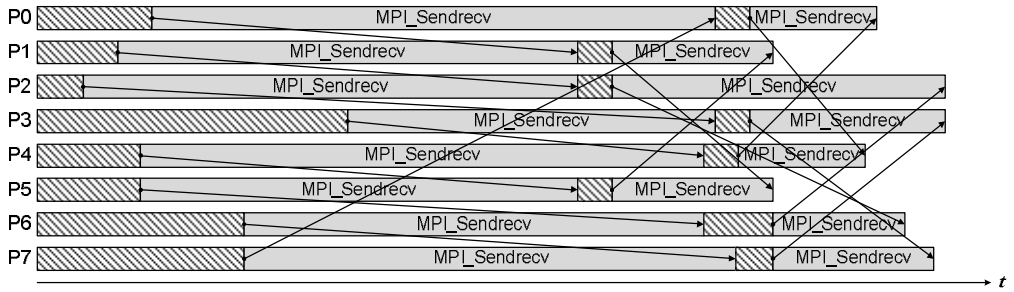


Рис. 3. Диаграмма выполнения "рассеивающего" алгоритма барьерной синхронизации между 8 параллельными ветвями (результаты получены пакетом Intel Trace Analyzer and Collector)

3.6 Измерение времени выполнения коллективной операции без синхронизации моментов её запуска в ветвях

Измерение времени выполнения MPI-функции выполняется в цикле без синхронизации моментов её запуска в ветвях программы (рис. 1). Это приводит к тому, что результирующая оценка времени выполнения операции включает и время ожидания начала следующей итерации цикла измерений. Рассмотрим измерение времени выполнения функции `MPI_Bcast`, в соответствие с процедурой на рис. 1. Для определенности будем полагать, что `MPI_Bcast` реализуется линейным алгоритмом, при котором сообщение размером m байт из ветви 0 передается в ветвь 1, затем в ветвь 2 и т.д. На рис. 4 показана диаграмма выполнения этого алгоритма. Время t_i выполнения обменов ветвью $i \in \{0,1,\ldots,n-1\}$ выражено в модели Дж. Хокни, в которой α – латентность канала связи, β – время передачи одного байта по каналу связи.

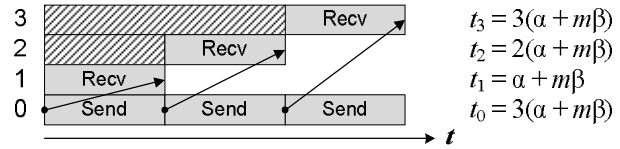


Рис. 4. Диаграмма выполнения линейного алгоритма трансляционного обмена (`MPI_Bcast`) между 4 ветвями параллельной программы:
– обмен информацией; – ожидание

На рис. 5 приведена диаграмма выполнения четырех итераций цикла измерений (рис. 1). Из диаграммы видно, что измеренное время T_1 выполнения функции в ветви 1 в 2,5 раза превосходит "истинное" время t_1 (рис. 4). Это объясняется тем, что после выполнения первой итерации, ветвь 1 сразу (без синхронизации) переходит к итерации 2 и запускает операцию, в которой она ожидает сообщение от ветви 0. В состоянии ожидания ветвь 1 прибывает до тех пор, пока ветвь 0 не передаст сообщения ветвям 2 и 3. Таким образом, на каждой итерации к истинному време-

ни выполнения операции добавляется время ожидания. Измеренное таким способом время выполнения функции включает систематическую ошибку, связанную с несинхронизированным запуском операций в ветвях программы. Сказанное справедливо не только для измерения времени выполнения алгоритмов ТО, но и для других коллективных операций.

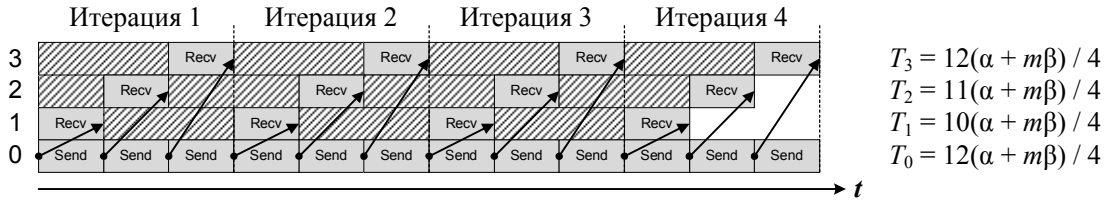


Рис. 5. Диаграмма выполнения четырех итераций цикла измерений времени выполнения линейного алгоритма трансляционного обмена

Обозначим через $t_i(n)$ время выполнения линейного алгоритма ТО в ветви $i \in \{0,1,\ldots,n-1\}$. Аналогично, $T_i(n,k)$ – время выполнения линейного алгоритма ТО в ветви i, измеренное k итерациями цикла (рис. 1). Нетрудно показать, что

$$t_i(n) = \begin{cases} (n-1)(\alpha + m\beta), & \text{при } i = 0, \\ i(\alpha + m\beta), & \text{при } i > 0, \end{cases} \qquad T_i(n,k) = \begin{cases} (n-1)k(\alpha + m\beta)/k, & \text{при } i = 0, \\ ((n-1)(k-1)+i)(\alpha + m\beta)/k, & \text{при } i > 0. \end{cases}$$

Тогда отношение $r_i(n,k)$ времени $T_i(n,k)$ к $t_i(n)$ позволяет судить об ошибке метода, приведенного на рис. 1:

$$r_i(n,k) = \frac{T_i(n,k)}{t_i(n)} = \begin{cases} 1, & \text{при } i = 0, \\ \dfrac{(n-1)(k-1)+i}{ik} & \text{при } i > 0. \end{cases}$$

Аналогичные оценки систематической ошибки метода можно построить, используя более точные модели дифференцированных обменов: LogGP [16], PLogP [17].

Как было отмечено выше, причина возникновения рассмотренной ошибки заключается в несинхронизированном запуске коллективной операций в цикле измерений (рис. 1). Одним из известных подходов к устранению этой ошибки является организация одновременного запуска операций в ветвях. Это достигается за счет синхронизации показаний локальных часов ветвей и формирования расписаний запуска операции в них. Такой подход лежит в основе пакетов SKaMPI, MPIBench [18] и Netgauge [19]. Однако эти пакеты не лишены перечисленных выше систематических ошибок измерений.

Автором предложен метод измерения времени выполнения коллективных операций обменов информацией, основанный на синхронизации моментов запуска функций в ветвях. В методе учтены известные систематические ошибки измерений времени выполнения таких операций.

4. Описание метода

Разработанный метод включает нижеследующие шаги.

1. *Синхронизация показаний локальных часов ветвей.* Каждая ветвь i вычисляет смещение o_i показаний своих локальных часов относительно часов ветви 0, показания которых принимаются за глобальное время. Зная свое локальное время T_i, ветвь i может вычислить показания глобальных часов $T_0 = T_i + o_i$ и наоборот: $T_i = T_0 - o_i$.

2. *Оценка времени выполнения ТО*. Формируется оценка сверху времени t_{bcast} выполнения трансляционной передачи (`MPI_Bcast`) из ветви 0 сообщения с показанием её часов. Как правило, показания часов представляется вещественным числом двойной точности в формате IEEE 754.

3. *Измерение времени выполнения коллективной операции.* Процесс измерения времени выполнения коллективной операции разбивается на этапы (рис. 6).

3.1. На каждом этапе $j = 0,1,\ldots$ ветвь 0 запрашивает показание T_0 своих локальных часов и используя трансляционный обмен (`MPI_Bcast`) передает ветвям время τ_j первого запуска операции: $\tau_j = T_0 + t_{bcast}$.

3.2. По значению τ_j ветви формируют расписание k запусков операции на этапе j. Запуск $l = 0,1,\ldots,k-1$ осуществляется по глобальным часам в момент времени $\tau_{jl} = \tau_j + l\delta_j$. На каждый запуск отводится δ_j секунд, таким образом, запуск l должен завершиться до момента $\tau_{j,l+1} = \tau_j + (l+1)\delta_j$. Если ветвь осуществляет запуск l позднее момента τ_{jl} или завершает выполнение операции после $\tau_j + (l+1)\delta_j$, то результат выполнения операции считается некорректным. Такие ситуации могут возникать по причине динамического характера загрузки ресурсов распределенных ВС.

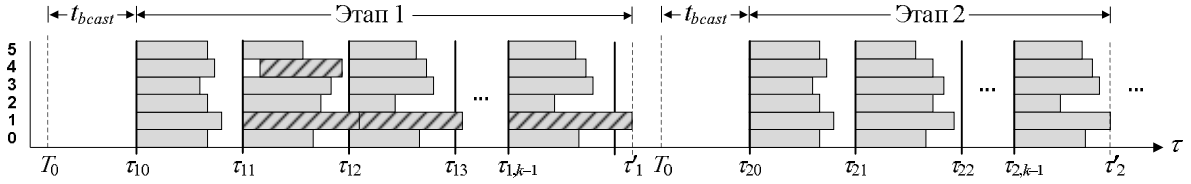


Рис. 6. Этапы измерения времени выполнения коллективной операции обмена информацией:
▨ – некорректное выполнение; ▭ – корректное выполнение

3.3. После окончания этапа осуществляется анализ результатов измерений. За время выполнения операции на запуске l принимается максимальное из времен выполнения ветвей. Если на этапе j количество некорректных запусков превышает g процентов, то выполняется корректировка длины интервала $\delta_{j+1} = \gamma(\tau'_j - \tau_j)/k$, где τ'_j – время завершения этапа j (максимальное из времени завершения выполнения ветвей на k-ом запуске этапа j), γ – масштабный коэффициент ($1,1 < \gamma < 2$, подбирается эмпирически).

3.4. Проверяются условия окончания измерений. Если стандартная ошибка среднего времени выполнения коллективной операции достигла заданного уровня или количество запусков превысило максимально допустимое, то измерения прекращаются.

Инициализация операции осуществляется на этапе 0. Выполняется k' её запусков при длине интервала $\delta_0 = 0$. Результаты этого этапа не учитываются при формировании итоговых результатов измерений. Время выполнения этапа 0 используется для формирования начального значения длины интервала $\delta_1 = \gamma(\tau'_0 - \tau_0)/k'$.

4. *Статистическая обработка результатов измерений.* После завершения измерений осуществляется обработка их результатов. Пусть Q количество корректных запусков операции за время выполнения всех этапов, а t^q – время выполнения операции на запуске $q \in \{1,2,\ldots,Q\}$. Из последовательности значений $t^1, t^2, \ldots, t^Q$ удаляется g' процентов минимальных и максимальных значений. За итоговое время t выполнения коллективной операции принимается статистическая оценка математического ожидания времени выполнения корректных запусков операции. Для измеренного времени t формируется доверительный интервал с заданной доверительной вероятностью.

5. Пакет MPIPerf

Описанный метод реализован автором в пакете MPIPerf. В текущей версии реализованы тесты для всех коллективных операций стандарта MPI 2.2.

Синхронизация локальных часов ветвей в пакете MPIPerf осуществляется по выбору пользователя линейным или кольцевым алгоритмом. Оба алгоритма основаны на процедуре син-

хронизации часов двух ветвей – корневой ветви 0 и ветви i. Смещение o_i показаний своих локальных часов ветвь i рассчитывается следующим образом (рис. 7)

$$o_i = T_0 - \frac{T_{\text{RTT}}}{2} - T_i', \quad T_{\text{RTT}} = T_i'' - T_i',$$

где T_0 – показание часов ветви 0, а T_i' и T_i'' показания часов ветви i. Величина T_{RTT} – это суммарное время передачи и последующего приема сообщения с показаниями часов (RTT – Round Trip Time). За значение T_{RTT} принимается минимальное из измеренных. Причем измерения величины T_{RTT} заканчиваются, если её текущее минимальное значение не изменялось на протяжении последних d итераций (по умолчанию $d = 100$).

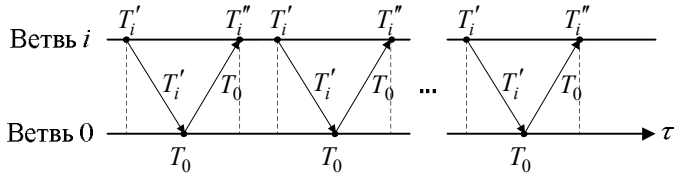


Рис. 7. Синхронизация локальных часов ветвей i и 0

Линейный алгоритм последовательно выполняет синхронизацию часов ветвей 1, 2, …, $n-1$ с часами ветви 0. Коммуникационная сложность (асимптотическая оценка количества выполняемых дифференцированных обменов) линейного алгоритма составляет $T_{linear} = O(n \cdot T_{sync})$, где T_{sync} – коммуникационная сложность процедуры синхронизации двух ветвей (рис. 7).

В кольцевом алгоритме каждая ветвь i аналогично рассчитывает отклонение $o_{i,(i-1+n) \bmod n}$ показаний своих часов от ветви $(i-1+n) \bmod n$. Далее ветвь 0, используя функцию трансляционного обмена (`MPI_Gather`), принимает смещения времени o_{ij} всех ветвей. Итоговое смещение локальных часов ветви i рассчитывается ветвью 0 как $o_i = o_{10} + o_{21} + \ldots + o_{i,(i-1+n) \bmod n}$. Затем значения o_i рассылаются ветвям функцией трансляционного обмена (`MPI_Scatter`). Коммуникационная сложность кольцевого алгоритма складывается из сложности T_{sync} и сложности алгоритмов коллекторного и трансляционного обменов. Как правило, в современных реализациях библиотек стандарта MPI функции `MPI_Scatter` и `MPI_Gather` реализуются алгоритмами с логарифмической коммуникационной сложностью, в этом случае сложность кольцевого алгоритма $T_{ring} = O(T_{sync} + \log_2 n)$.

Ниже приведены значения параметров предложенного метода, используемые по умолчанию в пакете MPIPerf (пользователю доступна регулировка этих значений):

- количество k' запусков операции в процессе её инициализации: 4;
- количество k запусков операции на каждом этапе измерений: 8;
- количество некорректных запусков, по достижению которого корректируется длина интервала δ_j: 25% от общего количества запусков на этапе j;
- значение масштабного коэффициента γ для корректировки длины интервала δ_j: 1,1;
- условия окончания измерений: вариант 1 – количество измерений больше 100 или корректных измерений больше 30; вариант 2 – относительная ошибка среднего значения времени выполнения операции не превосходит 0,05 и количество успешных измерений не менее 10;
- количество g' отбрасываемых минимальных и максимальных значений измеренного времени выполнения операции: 25% от общего числа корректных измерений.

По окончанию работы пакет MPIPerf формирует отчет включающий: значение варьируемого параметра (размер сообщения или количество ветвей в программе); общее количество n_t запусков коллективной операции; количество n_c корректных запусков; количество n_s корректных запусков после статистической обработки; статистическая оценка T математического

ожидания времени выполнения операции (формируется по n_s значениям); стандартная ошибка SE измерения времени T ($SE = \sigma[T] / \sqrt{n_c}$, где $\sigma[T]$ – несмещённая оценка среднеквадратичного отклонения T относительно его математического ожидания); минимальное значение T, максимальное значение T, абсолютная ошибка E измерений времени T ($E = \alpha \cdot SE$, где α – коэффициент Стьюдента), доверительный интервал $P(T - \alpha \cdot SE \le T \le T + \alpha \cdot SE) = p$, где p – заданная пользователем надежность ($p \in \{0,9; 9,95; 0,99\}$).

Реализована возможность формирования отчетов о времени выполнения операции в каждой ветви программы.

6. Экспериментальное исследование

Эксперименты с пакетом MPIPerf проводились на двух вычислительных кластерах:

– кластер А (Центр параллельных вычислительных технологий ФГОБУ ВПО "Сибирский государственный университет телекоммуникаций и информатики"): 10 вычислительных узлов, на каждом узле установлено 2 четырехъядерных процессора Intel Xeon E5420, коммуникационная сеть – Gigabit Ethernet, операционная система – CentOS 5.2 x86_64 (ядро linux 2.6.18-92.el5);

– кластер Б (Информационно-вычислительный центр ФГОБУ ВПО "Новосибирский национальный исследовательский государственный университет"): использована подсистема из 96 узлов HP BL2x220 G7, на каждом узле 2 шестиядерных процессора Intel Xeon X5670, коммуникационная сеть – InfiniBand 4x QDR, операционная система – SUSE Linux Enterprise Server 11 x86_64 (ядро linux 2.6.27.19-5).

6.1. Исследование точности синхронизации моментов запуска коллективной операции в ветвях программы

Точность синхронизации моментов запуска операции в ветвях можно оценить при помощи специальных коллективных функций ожидания, время выполнения которых известно априори. В первой функции `WaitPatternUp` [12] каждая ветвь i ожидает $(i+1) \cdot 10^{-6}$ секунд (выполняет пустой цикл). Если все ветви запустили эту функцию в один момент времени, то время её выполнения составит $n \cdot 10^{-6}$ секунд, где n – количество ветвей в программе. Во второй функции `WaitPatternNull` каждая ветвь засекает время запуска операции и сразу завершает её выполнение. Время работы этой функции 0 секунд.

На рис. 8 показано время выполнения теста `WaitPatternUp` на вычислительном кластере А с использованием библиотеки MPICH2 1.2.1. Видно, что время операции, измеренное пакетом MPIPerf, совпадает с её действительным временем выполнения. На рис. 9 показано время выполнения того же теста на кластере Б (использована библиотека OpenMPI 1.4.4) пакетами MPIPerf (кривая 1) и SKaMPI (кривая 2). Результаты не совпадают с реальным временем выполнения операции. Причина в том, что на узлах кластера Б операционная система в качестве источника времени (clock source) использует высокоточный таймер событий HPET – High Precision Event Timer, с которым она некорректно работает [20] и формирует неточные значения функции `gettimeofday`. В свою очередь, эта функция используется во многих библиотеках MPI для реализации подпрограммы `MPI_Wtime`, средствами которой осуществляются измерения времени в большинстве известных пакетов (Intel MPI Benchmarks, mpptest, Phloem, MPIBlib, OSU Micro-Benchmarks). Для решения этой проблемы в MPIPerf реализована возможность выбора таймера, который будет использован для измерения времени выполнения операций и синхронизации ветвей. Поддерживаются таймеры на основе функций `gettimeofday`, `MPI_Wtime` и значений регистра TSC – Time Stamp Counter [21] современных процессоров. На рис. 9 (случай 3) видно, что пакет MPIPerf с таймером TSC корректно измеряет время выполнения операции.

Перед использованием пакета MPIPerf рекомендуется проверить корректность показаний системных таймеров, путем измерения ими времени выполнения операции `WaitPatternNull`. На рис. 10 приведены результаты выполнения теста на кластере Б.

По виду кривой 1 можно судить о некорректной работе функции `MPI_Wtime`, в тоже время кривая 2 свидетельствует о корректной работе таймера на основе значения регистра TSC.

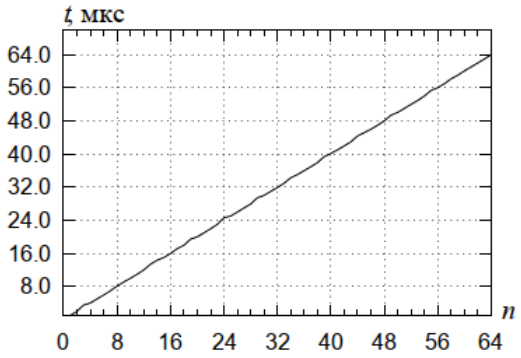


Рис. 8. Зависимость времени t выполнения на кластере А теста `WaitPatternUp` от количества n ветвей в программе (таймер `MPI_Wtime`)

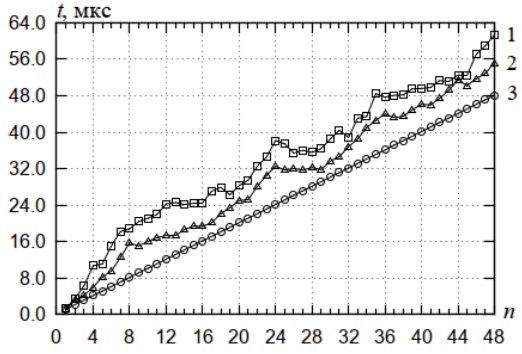


Рис. 9. Зависимость времени t выполнения на кластере Б теста `WaitPatternUp` от количества n ветвей в программе: *1* – MPIPerf, таймер `MPI_Wtime`; *2* – пакет SKaMPI, таймер `MPI_Wtime`; *3* – пакет MPIPerf, таймер TSC

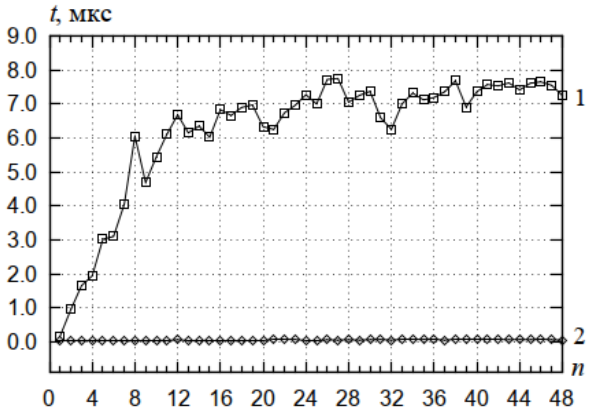


Рис. 10. Зависимость времени t выполнения на кластере Б теста `WaitPatternNull` от количества n ветвей в программе: *1* – MPIPerf, таймер MPI_Wtime; *2* – MPIPerf, таймер TSC

6.2. Сравнение с пакетом SKaMPI

На рис. 11 приведены результаты измерения времени выполнения на кластере А функции `MPI_Bcast` (библиотека MPICH2 1.2.1) пакетами MPIPerf и SKaMPI. Результаты измерений пакетами различны. Это объясняется разными подходами к выделению памяти под буферы приёма и передачи сообщений, а также динамическим характером загрузки ресурсов кластера.

Важным вопросом при разработке средств измерения времени выполнения коллективных операций является воспроизводимость результатов. На рис. 12 показана относительная ошибка измерения среднего времени выполнения функции `MPI_Barrier` пакетами SKaMPI и MPIPerf (использован кластер Б и библиотека Intel MPI 4.0.0.028 с таймером `MPI_Wtime` на основе значения регистра TSC). Оба пакета запускались по 10 раз. На каждом запуске измерялось время выполнения функции `MPI_Barrier` при различном количестве n ветвей в программе. По результатам 10 запусков для каждого значения n рассчитывалось среднее значение $\mathrm{M}[t]$ времени t выполнения функции, среднее квадратическое отклонение $\sigma[t]$ времени выполнения и относительная ошибка RSE измерения среднего времени выполнения функции

$$\mathrm{RSE} = \frac{\sigma[t]}{\mathrm{M}[t] \cdot \sqrt{10}}.$$

Кривые 1 и 2 на рис. 12 свидетельствуют об удовлетворительной для практики воспроизводимости результатов измерений пакетом MPIPerf.

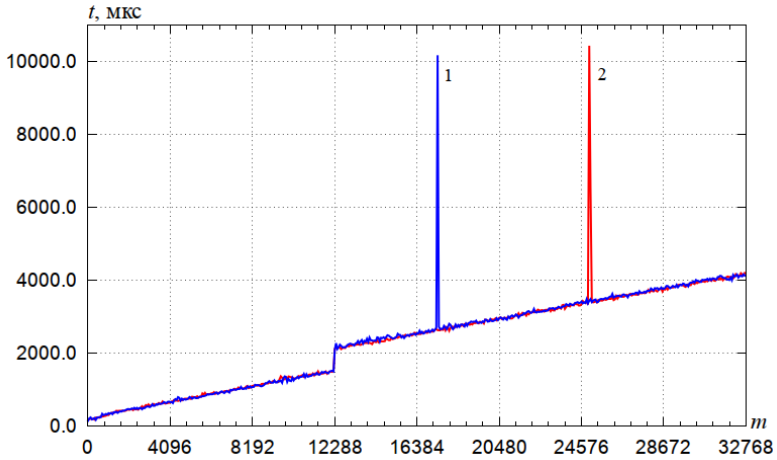


Рис. 11. Зависимость времени t выполнения функции `MPI_Bcast` от размера m передаваемого сообщения: *1* – пакет SKaMPI; *2* – пакет MPIPerf

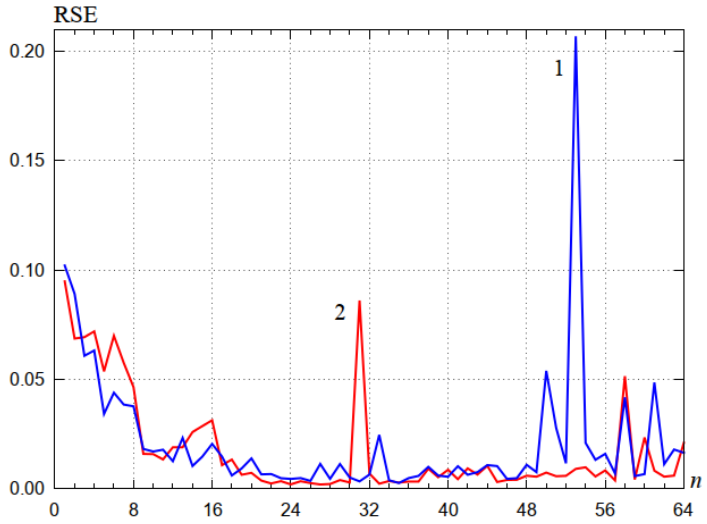


Рис. 12. Зависимость относительной ошибки RSE измерения среднего времени выполнения функции `MPI_Barrier` от количества n ветвей в программе: 1 – пакет SKaMPI; 2 – MPIPerf

Отличительными особенностями созданного пакета MPIPerf являются нижеследующие.

1. Реализованная методика измерения времени выполнения коллективных операций на основе синхронизации моментов их запуска по показаниям глобальных часов; учет отложенной инициализации MPI-функций и иерархической организации подсистемы памяти вычислительных узлов ВС.

2. Расширяемая архитектура пакета, обеспечивающая возможность создания тестов для производных типов данных (MPI Derived data types) и виртуальных топологий (MPI Virtual topologies).

3. Набор тестов для всех коллективных операций стандарта MPI 2.2.

4. Возможность выбора и оценки корректности показаний таймера для измерения времени выполнения коллективных операций.

Пакет распространяется в исходных кодах на условиях лицензии BSD (`http://mpiperf.cpct.sibsutis.ru`).

7. Заключение

Предложенный метод измерения времени выполнения коммуникационных функций основан на синхронизации моментов запуска операции в ветвях MPI-программы и учитывает известные систематические ошибки измерений, характерные для существующих пакетов (OSU MPI Benchmarks, SKaMPI, Netgauge, Intel MPI Benchmarks, Phloem MPI Benchmarks, MPIBLib). Разработанный подход применим и для измерения времени выполнения коммуникационных функций в runtime-системах языков параллельного программирования (IBM X10, Cray Chapel, Unified Parallel C), а также для оценки производительности коммуникационных сетей при реализации коллективных операций обмена информацией.

Программная реализация метода в пакете MPIPerf обеспечивает возможность измерения времени выполнения всех коллективных операций стандарта MPI 2.2. Результаты натурных экспериментов на вычислительных кластерах с сетями связи InfiniBand QDR и Gigabit Ethernet подтвердили эффективность предложенного метода.

В будущих версиях MPIPerf планируется реализовать измерение времени выполнения неблокирующих коллективных операций (Nonblocking collective communications), появление которых ожидается в стандарте MPI 3.0, а также реализовать модуль оценки потребления оперативной памяти библиотеками MPI для исследования их масштабируемости.

Литература

1. Хорошевский В.Г. Распределенные вычислительные системы с программируемой структурой // Вестник СибГУТИ. - 2010. - № 2 (10). - С. 3-41.
2. Курносов М.Г. Алгоритмы трансляционно-циклических информационных обменов в иерархических распределенных вычислительных системах // Вестник компьютерных и информационных технологий. - 2011. - № 5. - С. 27-34.
3. Rabenseifner R.. Automatic MPI Counter Profiling // Proceedings of the 42nd Cray User Group. - Noorwijk, The Netherlands, 2000. - 19 pp.
4. Han D., Jones T.. MPI Profiling // Technical Report UCRL-MI-209658 - Lawrence Livermore National Laboratory, USA, 2004. - 15 pp.
5. Иванников В.П., Аветисян А.И., Гайсарян С.С., Падарян В.А. Прогнозирование производительности MPI-программ на основе моделей // Автоматика и телемеханика. - 2007. - №5. - С. 8-17.
6. Intel MPI Benchmarks 3.2.2 // Intel Software Network. URL: http://software.intel.com/en-us/articles/intel-mpi-benchmarks.
7. Gropp W., Lusk E.L. Reproducible Measurements of MPI Performance Characteristics // In Proc. of the 6th European PVM/MPI Users Group Meeting, 1999. - P. 11-18.
8. LLCbench - Low Level Architectural Characterization Benchmark Suite // LLCBench Home Page. URL: http://icl.cs.utk.edu/projects/llcbench.
9. Phloem MPI Benchmarks // The ASCI Purple Benchmark Codes. URL: https://asc.llnl.gov/sequoia/benchmarks/#phloem.
10. Lastovetsky A., Rychkov V., O'Flynn M. MPIBlib: Benchmarking MPI Communications for Parallel Computing on Homogeneous and Heterogeneous Clusters // In Proc. Of 15th European PVM/MPI User's Group Meeting, Berlin, 2008. - P. 227-238.
11. OSU Micro-Benchmarks // OSU Network-Based Computing Laboratory. URL:http://mvapich.cse.ohio-state.edu/benchmarks/.
12. Worsch T., Reussner R., Werner A. On Benchmarking Collective MPI Operations // Proceedings of the 9th EuroPVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. - 2002. - P. 271-279.

13. Pjesivac-Grbovic J., Angskun T., Bosilca G., Fagg G. Gabriel E. and Dongarra J. Performance Analysis of MPI Collective Operations // Cluster Computing. - 2007. -Vol. 10, No. 2. - P. 127-143.

14. Thakur R., Rabenseifner R., and Gropp W. Optimization of collective communication operations in MPICH // Int. Journal of High Performance Computing Applications. - 2005. - Vol. 19, No. 1. - P. 49 66.

15. Hensgen D., Finkel R. and Manbet U. Two Algorithms for Barrier Synchronization // International Journal of Parallel Programming. - 1988. - Vol. 17(1). - P. 1-17.

16. Alexandrov A., Mihai I., Schauser K. and Scheiman C. LogGP: Incorporating Long Messages into the LogP Model // Technical Report, University of California at Santa Barbara. - 1995. - 21 p.

17. Kielmann T., Bal H. E., Kees V. Fast Measurement of LogP Parameters for Message Passing Platforms // Proceedings of the 15 Workshops on Parallel and Distributed Processing. - London, UK, 2000. - P. 1176-1183.

18. MPIBench // MPIBench Home Page. URL: http://www.dhpc.adelaide.edu.au/projects/mpibench.

19. Hoefler T., Mehlan T., Lumsdaine A. and Rehm W. Netgauge: A Network Performance Measurement Framework // Proceedings of High Performance Computing and Communications, 2007. - P. 659-671.

20. Bug 444496 "hpet increasing min_delta_ns" // Novell Bugzilla. - URL: https://bugzilla.novell.com/show_bug.cgi?id=444496.

21. Intel's Applications Notes. Using the RDTSC Instruction for Performance Monitoring // URL: http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf.

Двухсеточные параллельные алгоритмы для решения дробно-дифференциальных уравнений аномальной диффузии

С.Ю. Лукащук

Уфимский государственный авиационный технический университет

Приводятся описание и анализ параллельных алгоритмов решения начально-краевых задач для уравнений аномальной диффузии, содержащих производные дробного порядка типа Римана-Лиувилля по пространственным и/или временной переменным. Параллельные алгоритмы построены на основе двухсеточных методов и, в случае распараллеливания по оси времени, представляют собой модификации известного алгоритма PARAREAL.

1. Введение

Во многих неупорядоченных сложных средах, таких как пористые и трещиновато-пористые среды, аморфные полупроводники, жидкие полимеры и стекла, перколяционные кластеры и самоподобные фрактальные среды, турбулентные потоки жидкости, газа и плазмы, кинетика протекания процессов диффузионного переноса часто отличается от классической, соответствующей нормальному закону распределения [1, 2]. Такие процессы принято называть процессами аномальной диффузии. Аномальный диффузионный перенос обычно обусловлен эффектами памяти среды, пространственной нелокальности и перемежаемости, имеет немарковскую стохастическую природу и поэтому не может быть описан классическими эволюционными уравнениями переноса [3].

Одним из эффективных и широко используемых подходов для описания процессов аномальной диффузии является использование аппарата интегро-дифференцирования дробного порядка [4–6]. В этом случае уравнение процесса является интегро-дифференциальным уравнением, содержащим производные дробного порядка по временной и/или пространственным переменным.

Основной проблемой численного моделирования процессов аномальной диффузии, описываемых уравнениями дробного порядка, является нелокальный характер последних. При использовании конечно-разностной дискретизации обычных диффузионных уравнений значение в текущем узле расчетной сетки зависит только от значений в соседних узлах. При конечно-разностной дискретизации уравнений дробного порядка значение в текущем узле будет зависеть от всех остальных пространственных узлов (при наличии в уравнении дробных производных по пространственным переменным) и всех предыдущих временных слоев (при наличии дробной производной по времени) [4]. В результате при численных расчетах аномального переноса объем вычислений оказывается существенно (на порядки!) большим по сравнению с объемом вычислений в расчетах классических диффузионных процессов. При наличии дробной производной по времени объем вычислений также растет с увеличением номера временного слоя, поскольку увеличивается диапазон памяти системы, которую нужно учитывать. Существенно возрастает и объем необходимой оперативной памяти для хранения результатов расчета. Использование параллельных алгоритмов, построенных только на принципах пространственной декомпозиции расчетной области (domain decomposition), ситуацию не спасает, поскольку приводит к чрезвычайно большим объемам передачи данных между процессорами.

Указанные проблемы делают актуальной задачу разработки новых последовательных и параллельных численных алгоритмов решения уравнений аномальной диффузии с производными дробного порядка. В данной работе предлагается подход к построению таких

алгоритмов, основанный на использовании идеи двухсеточных методов. Грубая сетка используется в этом случае как основа для расчета эффектов пространственного и временного дальнодействия, а мелкая сетка служит для непосредственных вычислений. По узлам грубой сетки может производиться пространственная и, при необходимости, временная декомпозиция расчетной области, что приводит к легко реализуемым параллельным алгоритмам. При распараллеливании по временной оси данный подход приводит к модификации алгоритма PARAREAL для уравнений с производными дробного порядка [8].

2. Постановка задачи

При описании процессов аномальной диффузии наиболее часто используются левосторонняя и правосторонняя производные дробного порядка α типа Римана-Лиувилля [5]:

$$\left({}_aD_x^{\alpha}y\right)(x) = \frac{1}{\Gamma(n-\alpha)}\left(\frac{d}{dx}\right)^n \int_a^x \frac{y(\xi)}{(x-\xi)^{\alpha-n+1}}d\xi, \tag{1}$$

$$\left({}_xD_b^{\alpha}y\right)(x) = \frac{(-1)^n}{\Gamma(n-\alpha)}\left(\frac{d}{dx}\right)^n \int_x^b \frac{y(\xi)}{(\xi-x)^{\alpha-n+1}}d\xi, \tag{2}$$

где $n = [\alpha] + 1$ и $[\alpha]$ – целая часть числа α.

В дальнейшем для иллюстрации предлагаемого подхода будем рассматривать уравнение аномальной диффузии дробного порядка следующего вида:

$$\left({}_0D_t^{\alpha}u\right)(x,t) = \gamma\left({}_0D_x^{1+\beta}u\right)(x,t) + (1-\gamma)\left({}_xD_L^{1+\beta}u\right)(x,t) + f(x,t), \tag{3}$$
$$t \in (0,T], \quad x \in (0,L), \quad \alpha,\beta,\gamma \in (0,1).$$

Для уравнения (3) поставим первую краевую задачу:

$$u(x,0) = u^0(x), \quad u(0,t) = u_0(t), \quad u(L,t) = u_L(t), \tag{4}$$

где $u^0(x),\ u_0(t),\ u_L(t)$ – заданные функции. Отметим, что постановка таких краевых условий предъявляет определенные дополнительные требования к виду функции $f(x,t)$, которые однако не являются важными для рассматриваемых алгоритмов и поэтому здесь не приводятся.

При построении численных схем для дробно-дифференциальных уравнений используются различные конечно-разностные аппроксимации дробных производных. Для левосторонней дробной производной (1) в случае равномерной сетки все они могут быть представлены в следующем общем виде:

$$\left({}_aD_x^{\alpha}y\right)(x_k) \approx h^{-\alpha}\sum_{j=0}^{k+m} A_j y_{k+m-j}, \quad y_j = y(x_j), \quad x_j = a + jh \tag{5}$$

(здесь h – шаг сетки). Аналогичная формула имеет место и для правосторонней дробной производной (2). При $m = 0$ и $A_j = (-1)^j\binom{\alpha}{j}$ имеем наиболее часто используемую классическую аппроксимацию, получаемую из формулы Грюнвальда-Летникова [5], и обеспечивающую первый порядок точности аппроксимации дробной производной для аналитических функций [4]. Повысить порядок точности можно использованием полиномиальной интерполяции искомой функции между узлами расчетной сетки с последующим аналитическим вычислением соответствующих интегралов. Случай линейной межузельной интерполяции, обеспечивающий второй порядок точности, рассмотрен в [7].

3. Декомпозиция задачи на основе двухсеточного подхода

Введем в рассматриваемой пространственно-временной области $\bar{\Omega} = [0,L] \times [0,T]$ равномерную конечно-разностную сетку с шагом ΔX по пространственной и ΔT по временной

переменным:

$$\omega_G = \{(X_i = i\Delta X,\ T_j = j\Delta T),\ i = 0, 1, \ldots, N,\ j = 0, 1, \ldots, M,\ \Delta X = L/N,\ \Delta T = T/M\}. \tag{6}$$

Сетку (6) в дальнейшем будем называть грубой сеткой. Каждую ячейку, ограниченную смежными узлами грубой сетки, будем рассматривать как отдельную расчетную подобласть:

$$\bar{\Omega}_{i,j} = [X_{i-1}, X_i] \times [T_{j-1}, T_j], \quad \bar{\Omega} = \bigcup_{i=1}^{N} \bigcup_{j=1}^{M} \bar{\Omega}_{i,j}.$$

Для $(t, x) \in \Omega_{i,j}$ производные дробного порядка, входящие в уравнение (3), могут быть записаны следующим образом:

$${}_0D_t^\alpha u = {}_{T_{j-1}}D_t^\alpha u + {}_0I_{T_{j-1}} u,$$

$${}_0D_x^{1+\beta} u = {}_{X_{i-1}}D_x^{1+\beta} u + {}_0J_{X_{i-1}} u, \quad {}_xD_L^{1+\beta} u = {}_xD_{X_i}^{1+\beta} u + {}_{X_i}J_L u,$$

где введены следующие обозначения для интегралов:

$${}_0I_{T_{j-1}} u = \frac{1}{\Gamma(-\alpha)} \int_0^{T_{j-1}} \frac{u(x,\tau)}{(t-\tau)^{\alpha+1}} d\tau,$$

$${}_0J_{X_{i-1}} u = \frac{1}{\Gamma(-\beta-1)} \int_0^{X_{i-1}} \frac{u(\xi,t)}{(x-\xi)^{\beta+2}} d\xi, \quad {}_{X_i}J_L u = \frac{1}{\Gamma(-\beta-1)} \int_{X_i}^{L} \frac{u(\xi,t)}{(\xi-x)^{\beta+2}} d\xi.$$

Уравнение (3) в области $\Omega_{i,j}$ может быть теперь переписано в следующем виде:

$${}_{T_{j-1}}D_t^\alpha u = \gamma\ {}_{X_{i-1}}D_x^{1+\beta} u + (1-\gamma)\ {}_xD_{X_i}^{1+\beta} u + f(x,t) + Lu, \tag{7}$$

где

$$L = \gamma\ {}_0J_{X_{i-1}} + (1-\gamma)\ {}_0J_{X_{i-1}} - {}_0I_{T_{j-1}}$$

- линейный интегральный оператор. Принципиально важным является тот факт, что вычисление Lu в области $\Omega_{i,j}$ производится по значениям функции $u(x,t)$ при $(x,t) \notin \Omega_{i,j}$.

Обозначим через $U = \{U_{i,j}, i = 0, 1, \ldots, N,\ j = 0, 1, \ldots, M\}$ сеточную функцию решения задачи на грубой сетке ω_G. В предлагаемом подходе ее значения используются для постановки начальных и граничных условий для уравнения (7), а также вычисления интегралов, входящих в Lu.

После того, как задача для уравнения (7) замыкается с помощью начальных и граничных условий и указан способ вычисления Lu по значениям U, численное решение задач в каждой подобласти $\Omega_{i,j}$ или соответствующим образом выбранной группе подобластей может выполняться параллельно. Для этого в каждой подобласти вводится точная сетка (для простоты считаем, что разбиение всех подобластей идентично):

$$\omega_F = \left\{(x_k = k\delta x,\ t_l = l\delta t),\ k = 0, 1, \ldots, n,\ l = 0, 1, \ldots, m,\ \delta x = \frac{\Delta X}{n},\ \delta t = \frac{\Delta T}{m}\right\}. \tag{8}$$

С использованием аппроксимаций вида (5) на сетке ω_F строится конечно-разностная аппроксимация уравнений (7) требуемого уровня точности. Как и в случае классического уравнения диффузии, могут быть построены явные, неявные или полу-явные схемы. Сеточную функцию задачи на точной сетке будем обозначать через $u = \{u_{k,l},\ k = 0, 1, \ldots, n,\ l = 0, 1, \ldots, m\}$.

Таким образом, точная сетка ω_F используется для параллельного решения задач в отдельных подобластях, которые могут выполняться параллельно, а грубая сетка ω_G используется для постановки краевых условий и вычисления дополнительной функции источника Lu.

На основе такого двухсеточного подхода возможно построение алгоритмов двух видов. В алгоритмах первого вида счет ведется последовательно (например, по времени) и в узлы грубой сетки просто копируются значения, полученные при расчете на точной сетке. Затем эти значения рассылаются по всем подобластям, в которых они необходимы для вычисления соответствующих элементов. Второй вид алгоритмов в более полной мере реализует идею двухсеточного подхода и основан на схемах типа „предиктор-корректор“. В этом случае каким-либо образом рассчитываются начальные приближения в узлах грубой сетки, а затем проводятся расчеты (параллельные) на точных сетках и уточняются приближения на грубой сетке. К алгоритму второго типа относится, в частности, модификация алгоритма PARAREAL для распараллеливания по времени. Далее будут рассмотрены характерные представители алгоритмов обоих видов.

4. Описание параллельных алгоритмов

4.1. Параллельный алгоритм с декомпозицией по пространству

Данный алгоритм относится к алгоритмам первого вида. Расчет по времени ведется последовательно, поэтому шаги по времени грубой и точной сеток совпадают: ($\delta t = \Delta T,\ M = m$). Характерными особенностями алгоритма, обеспечивающими требуемый уровень точности вычислений, являются: 1) использование кубической сплайн-интерполяции по значениям в узлах грубой сетки на одном временном слое, 2) использование полиномов второй и третьей степени для аппроксимации в окрестности границ подобластей $\Omega_{i,j}$ производных дробного порядка по пространственной переменной.

Сохранение заданной точности вычислений производных дробного порядка по пространству после декомпозиции на подобласти $\Omega_{i,j}$ является главной проблемой при разработке соответствующих параллельных алгоритмов. Использование классических аппроксимаций вида (5) (типа Грюнвальда-Летникова) для смещенных дробных производных, входящих в уравнение (7), может приводить к катастрофической потере точности вычислений в окрестности границ областей $\Omega_{i,j}$ несмотря на сохранение общего порядка точности аппроксимации. Для подавления ошибки предлагается использовать следующие вычислительные приемы.

Во-первых, необходимо выполнить сдвиг вычисляемой функции на величину соответствующего граничного значения:

$$ {}_{X_{i-1}}D_x^{1+\beta}u = {}_{X_{i-1}}D_x^{1+\beta}\left[u(x,t) - u(X_{i-1},t)\right] + \frac{1}{\Gamma(-\beta)}\frac{u(X_{i-1},t)}{(x-X_{i-1})^{\beta+1}}, \tag{9} $$

$$ {}_{x}D_{X_i}^{1+\beta}u = {}_{x}D_{X_i}^{1+\beta}\left[u(x,t) - u(X_i,t)\right] + \frac{1}{\Gamma(-\beta)}\frac{u(X_i,t)}{(X_i-x)^{\beta+1}}. \tag{10} $$

В результате основная составляющая производной оказывается вычисленной аналитически, а по приближенным формулам рассчитываются уже относительно небольшие поправки.

Во-вторых, при расчете левосторонней производной из правой части (9) в точке x_1 и правосторонней производной из правой части (10) в точке x_{n-1} коэффициенты A_r соответствующих аппроксимаций вида (5) находятся на основе явного вычисления этих производных по трехузловой квадратичной интерполяции. Для следующих точек x_2 и x_{n-2} используется четырехузловая кубическая интерполяция. Для остальных точек может быть использована классическая аппроксимация первого или второго порядка точности.

В-третьих, вычисление интегралов ${}_0J_{X_{i-1}}u$ и ${}_{X_i}J_Lu$ производится аналитически для кубической сплайн-интерполяции, построенной по значениям на грубой сетке U. Известно [9], что оценка погрешности аппроксимации функции $f(x)$ кубическим сплайном $s(x)$ на равномерной сетке с шагом H имеет вид

$$ |s(x) - f(x)| \le \frac{5}{384}AH^4, $$

где $\left|f^{IV}(x)\right| \le A$. Тогда легко получается оценка точности вычисления интеграла ${}_0J_{X_{i-1}}u$:

$$\left|{}_0J_{X_{i-1}}u(x,t_l) - s(x,t_l)\right| \le \frac{5}{384}\frac{A(t)(\Delta X)^4}{(\delta x)^{1+\beta}},$$

где $|u_{xxxx}(x,t)| \le A(t)$. Аналогичная оценка справедлива для интеграла ${}_{X_i}J_Lu$. Поскольку множитель $(\delta x)^{-1-\beta}$ является общим множителем при вычислении дробных производных на точной сетке по аппроксимациям вида (5), то точность вычисления интегралов по сплайн-интерполяции в разностной схеме решения уравнения (7) на точной сетке имеет порядок $O(\Delta X^4)$. Если для аппроксимации дробных производных используется аппроксимация первого порядка точности по пространству, то для сохранения этого порядка у всей схемы для уравнения (7) необходимо выполнение следующего очевидного неравенства: $(\Delta X)^4 \le \delta x$. При использовании аппроксимации второго порядка точности неравенство становится более жестким: $(\Delta X)^2 \le \delta x$.

Далее представлены основные шаги алгоритма при использовании явной схемы для дискретизации уравнений (7).

1. По заданному начальному условию задачи $u^0(x)$ (4) каждый процессор параллельно вычисляет значения сеточной функции U на нулевом временном шаге.

   Следующие шаги алгоритма являются едиными для временных слоев $l = 1, 2, \ldots, m$.

2. По значениям U для $(l-1)$-го временного слоя каждый процессор вычисляет коэффициенты соответствующего кубического интерполяционного сплайна.

3. На точной сетке все процессоры параллельно выполняют расчет $u_{k,l}$ $(k = 1, 2, \ldots, n-1)$ на временном слое l. При этом производная по времени аппроксимируется формулой Грюнвальда-Летникова

   $$\left({}_0D_t^{\alpha}u\right)(x_k,t_l) \approx (\delta t)^{-\alpha}\sum_{r=0}^{l}(-1)^r\binom{\alpha}{r}u_{k,l-r},$$

   а производные по пространству аппроксимируются на $(l-1)$-м временном слое с учетом (9) и (10) по формулам

   $$\begin{aligned}
   \left({}_{X_{i-1}}D_x^{1+\beta}u\right)(x_k,t_{l-1}) &\approx (\delta x)^{-(1+\beta)}\sum_{r=0}^{k+1}A_r\left(u_{k+1-r,l-1}-u_{0,l-1}\right)+\\
   &+ \frac{1}{\Gamma(-\beta)}\frac{u_{0,l-1}}{(x_k-X_{i-1})^{1+\beta}},\\
   \left({}_xD_{X_i}^{1+\beta}u\right)(x_k,t_{l-1}) &\approx (\delta x)^{-(1+\beta)}\sum_{r=0}^{n-k+1}A_r\left(u_{k-1+r,l-1}-u_{0,l-1}\right)+\\
   &+ \frac{1}{\Gamma(-\beta)}\frac{u_{0,l-1}}{(X_i-x_k)^{1+\beta}},
   \end{aligned}$$

   Коэффициенты A_r выбираются так, как описано выше. Интегралы ${}_0J_{X_{i-1}}u$ и ${}_{X_i}J_Lu$ вычисляются по аналитическим формулам для сплайн-интерполяции, построенной на шаге 2 .

4. Процессор, отвечающий за расчет подобласти $\Omega_{i+1,l}$ $(i = 1, 2, \ldots, N-1)$, пересылает процессору, отвечающему за расчет соседней подобласти $\Omega_{i,l}$ значение приращения за временной шаг искомой функции в первом узле точной сетки: $u_{1,l}^{i+1} - u_{1,l-1}^{i+1}$ (здесь верхний индекс соответствует пространственному номеру области). Процессоры, получившие значения, вычисляют сеточную функцию в узлах грубой сетки на l-м временном слое по формуле

   $$U_{i,l} = U_{i,l-1} + 0.5\left(u_{1,l}^{i+1} - u_{1,l-1}^{i+1} + u_{n-1,l}^{i} - u_{n-1,l-1}^{i}\right).$$

5. Выполняется взаимный обмен рассчитанными значениями сеточной функции U таким образом, чтобы полный вектор этих значений находился на каждом процессоре.

Далее переход к следующему временному слою и повтор алгоритма с шага 2.

Отметим, что условие устойчивости использованной явной конечно-разностной схемы имеет вид

$$\frac{(\delta t)^\alpha}{(\delta x)^{1+\beta}} \le \frac{\alpha}{1+\beta} \tag{11}$$

(для случая $\alpha = 1$ это условие приведено в [7]). В предельном случае классического уравнения диффузии $\alpha = 1,\ \beta = 1$ это условие переходит в хорошо известное условие устойчивости явной разностной схемы для уравнения параболического типа.

4.2. Параллельный алгоритм с декомпозицией по времени

Известно, что параллельные алгоритмы для решения эволюционных задач могут быть построены не только на основе декомпозиции пространственной расчетной области, но и на основе декомпозиции по временной оси. Одним из наиболее широко используемых алгоритмов данного вида является алгоритм PARAREAL, впервые предложенный в работе [10]. В настоящее время данный алгоритм хорошо изучен, получены теоретические оценки его точности и сходимости [11–14], предложен целый ряд его модификаций для решения различных классов задач эволюционного типа [11, 15, 16]. В работе [8] была предложена модификация алгоритма PARAREAL для решения обыкновенных дифференциальных уравнений с производной дробного порядка $\alpha \in (0, 1)$. В данном разделе дается описание новой модификации алгоритма PARAREAL, совмещенной с приведенным в предыдущем разделе алгоритмом декомпозиции по пространству, и предназначенной для решения уравнений вида (3).

Алгоритм базируется на пространственно-временной декомпозиции расчетной области Ω на основе грубой сетки ω_G, представленной в разделе 3, и двумерной сплайн-интерполяции решения по значениям сеточной функции U в узлах грубой сетки.

Пусть с использованием некоторого сеточного метода G, который в дальнейшем будем называть „грубым“, построено приближенное решение U краевой задачи (3), (4) в узлах грубой сетки. Рассмотрим теперь задачу для уравнения (7) в области $\Omega_{i,j}$. Обозначим через $u^{i,j}$ функцию решения этого уравнения. С использованием сплайн-интерполяции по значениям U на $(j-1)$-м временном слое можно записать начальное условие для уравнения (7) в виде

$$u^{i,j}(x, T_{j-1}) = \phi(x, U_{0,j-1}, U_{1,j-1}, \ldots, U_{N,j-1}). \tag{12}$$

Пусть интерполяция по временной переменной поля значений U выполняется таким образом, что значения интерполяционных коэффициентов для произвольного $(j-1)$-го слоя зависят только от значений U на этом и предыдущих временных слоях. Эта интерполяция используется для вычисления интегралов ${}_0I_{T_{j-1}}$. Тогда решение уравнения (7) с начальным условием (12) будет зависеть только от значений U на первых $(j-1)$-м временных слоях:

$$u^{i,j} = u^{i,j}(x, t, U_{k,l}),\ k = 0, 1, \ldots, N,\ l = 0, 1, \ldots, j-1,\ (x, t) \in \Omega_{i,j}. \tag{13}$$

Отметим, что для получения решений вида (13) необходимо, вообще говоря, на каждом временном слое решать систему уравнений вида (7) для всех подобластей, принадлежащих данному временному слою, с условиями сопряжения решений на границах подобластей. Однако, при численном решении этой задачи на мелкой сетке может быть эффективно использован алгоритм, приведенный в п. 4.1.

Зная решения (13), можно записать уравнения для нахождения значений в узлах грубой сетки:

$$U_{i,j} = u^{i,j}(X_i, T_j, U_{k,l}),\ k = 0, 1, \ldots, N,\ l = 0, 1, \ldots, j-1,\ i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, M. \tag{14}$$

Система (14) является системой нелинейных уравнений относительно сеточной функции U и может быть решена методом Ньютона:

$$U_{i,j}^{q+1} = u^{i,j}(X_i, T_j, U^q) + \sum_{k=0}^{N} \sum_{l=0}^{j-1} \frac{\partial u^{i,j}(X_i, T_j, U^q)}{\partial U_{k,l}} (U_{k,l}^{q+1} - U_{k,l}^q),\ i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, M \tag{15}$$

(здесь q – номер итерации метода Ньютона).

Ключевым моментом алгоритма PARAREAL является способ вычисления производных, входящих в (15):

$$\sum_{k=0}^{N} \sum_{l=0}^{j-1} \frac{\partial u^{i,j}(X_i, T_j, U^q)}{\partial U_{k,l}} (U_{k,l}^{q+1} - U_{k,l}^q) \approx G(X_i, T_j, U^{q+1}) - G(X_i, T_j, U^q), \tag{16}$$

где через G обозначено решение задачи на грубой сетке, получаемое с помощью „грубого“ метода. Таким образом, для вычисления производных на каждом следующем временном слое используется разность „грубых“ решений задачи, полученных по старым и новым, уже вычисленным на предыдущих временных слоях, значениям U.

Обозначим через $F(X_i, T_j, U)$ численное решение задачи на точной сетке в области $\Omega_{i,j}$. Полагая $u^{i,j}(X_i, T_j, U) \approx F(X_i, T_j, U)$, на основе (15), (16) получаем основную расчетную формулу алгоритма PARAREAL:

$$\begin{aligned} &U_{i,j}^{q+1} = F(X_i, T_j, U_{k,l}^q) + G(X_i, T_j, U_{k,l}^{q+1}) - G(X_i, T_j, U_{k,l}^q), \\ &k = 0, 1, \ldots, N,\ l = 0, 1, \ldots, j-1,\ i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, M. \end{aligned} \tag{17}$$

Данная вычислительная процедура позволяет параллельно по времени итерационно находить приближенные решения задачи в узлах грубой сетки, сопоставимые по точности со значениями, получаемыми при расчете на точной сетке во всей расчетной области.

В результате приходим к следующему параллельному алгоритму (полагаем, что имеется $N \times M$-процессорная решетка $\{p_{i,j}\}$ и за вычисление каждой области $\Omega_{i,j}$ отвечает свой процессор).

1. С использованием классической явной конечно-разностной схемы G, построенной на основе аппроксимаций дробных производных вида (5), каждый процессор находит начальное приближение решения в узлах грубой сетки U^0.

2. Организуется глобальная итерационная по q процедура метода Ньютона. На первом шаге $q = 0$.

3. По значениям $U_{k,l}^q$ $(k = 0, 1, \ldots, N,\ l = 0, 1, \ldots, j-1)$ каждый процессор $p_{i,j}$ строит двумерную кубическую по каждой переменной сплайн-интерполяцию поля грубого решения.

4. Все процессоры параллельно производят расчет каждый своей подобласти на точной сетке. При этом группы процессоров, отвечающие за расчет одинаковых временных слоев, реализуют алгоритм с декомпозицией по пространству, приведенный в п. 4.1. Интегралы, образующие Lu в уравнениях (3), рассчитываются по аналитическим формулам, полученным по принятому виду сплайн-интерполяции.

5. Последовательно (по временным слоям) рассчитывается следующее приближение в узлах грубой сетки U^{q+1} по формуле (17). При этом каждый процессор $p_{i,j}$ отвечает за вычисление одного значения $U_{i,j}^{q+1}$ и выполняет один дополнительный расчет „грубым“ методом $G(X_i, T_j, U_{k,l}^{q+1})$. После расчета каждого нового временного слоя полученные обновленные значения U^{q+1} рассылаются каждому процессору.

6. Проверяется условие остановки итерационного процесса: $\left\|U^{q+1} - U^q\right\| \le \varepsilon$. Если условие не выполнено, то возврат к шагу 2. При этом временной слой $q+1$ из расчета исключается (на нем уже получено решение, соответствующее решению на точной сетке, которое не может быть уточнено по (17)).

Описанный алгоритм допускает простую модификацию для случая, когда количество доступных процессоров достаточно для расчета только нескольких временных слоев одновременно. Кроме того, данный алгоритм существенно упрощается в следующих двух случаях. 1) При $\alpha = 1$, т.е. когда исходное уравнение (3) содержит обыкновенную (не дробную) производную по времени первого порядка – в этом случае отсутствует необходимость интерполяции по времени и алгоритм становится прямой комбинацией алгоритма п. 4.1 и классического алгоритма PARAREAL. 2) Если отсутствует пространственная декомпозиция – в этом случае отсутствует необходимость интерполяции по пространственной переменной и алгоритм становится прямой комбинацией модифицированного на уравнения дробного порядка алгоритма PARAREAL и классического последовательного алгоритма конечно-разностного решения таких уравнений.

5. Оценка параллельной эффективности алгоритмов

5.1. Оценка эффективности параллельного алгоритма с декомпозицией по пространству

Сначала приведем оценку трудоемкости последовательного алгоритма решения задачи (3), (4), построенного по явной схеме с аппроксимацией дробных производных по формуле Грюнвальда-Летникова. Шаг сетки по пространственной и временной переменным для последовательного алгоритма примем равным соответствующим шагам точной сетки параллельного алгоритма (в этом случае последовательное и параллельное решения будут иметь одинаковый порядок точности). Таким образом, имеем $Nn-1$ расчетных узлов по пространству и M расчетных узлов по времени. Расчет l-го временного слоя по последовательному алгоритму требует $2(Nn-1)(Nn+l+4)$ операций. Принимая, как обычно при таких оценках, время выполнения одной операции за единицу, получаем время выполнения последовательного алгоритма равным $T_1 = M(Nn-1)(2Nn+M+9)$.

Теперь оценим время выполнения параллельного алгоритма, приведенного в п. 4.1. Будем полагать, что количество используемых для его выполнения процессоров равно количеству пространственных подобластей: $P = N$. Количество операций, выполняемых каждым процессором на l-м временном слое, складывается из четырех составляющих: 1) операций, необходимых для вычисления дробных производных: $2(n-1)(n+l+11)$, 2) операций, необходимых для вычисления интегралов ${}_0J_{X_{i-1}}u$ и ${}_{X_i}J_Lu$ по явным аналитическим формулам через сплайн-интерполяцию: $228(n-1)(N-1)$, 3) операций, необходимых для расчета значений в узлах грубой сетки: $5(N-1)$ и 4) операций, необходимых для вычисления коэффициентов интерполяционного сплайна: $22(N-1)$ (для решения системы используется метод прогонки). Таким образом, трудоемкость расчета l-го временного слоя в параллельном алгоритме составляет $2(n-1)(n+l+11) + [228(n-1)+27](N-1)$ операций, которые выполняются параллельно. Тогда время расчета всех M временных слоев в параллельном алгоритме составит $T_p = M[(n-1)(2n+M+228N-206)+27(N-1)]$. Долей последовательного расчета, выполняемого на этапе инициализации алгоритма до начала основной вычислительной процедуры, в данной оценке пренебрегаем.

Тогда для ускорения параллельного алгоритма получаем следующую оценку:

$$S_p \equiv \frac{T_1}{T_p} = \frac{(Nn-1)(2Nn+M+9)}{(n-1)(2n+M+228N-206)+27(N-1)} \approx \frac{N(2Nn+M)}{2n+M}$$

(последняя оценка становится справедливой при $N \ll n$). Заметим, что число временных слоев M при использовании явной схемы может оказаться достаточно большим.

При стремлении к нулю шага точной сетки, т.е. при $n \to \infty$, и неизменных остальных параметрах получаем следующую асимптотическую оценку ускорения $S_p \approx N^2 \equiv P^2$. Таким образом, по сравнению с классическим алгоритмом, предлагаемый параллельный алгоритм демонстрирует асимптотически квадратичное ускорение. Причина такого нелинейного ускорения очевидна: в то время как классический алгоритм использует для расчета все пространственные узлы, параллельный алгоритм рассчитывает дальнодействующие нелокальные эффекты с использованием сплайн-аппроксимации, что приводит к существенному уменьшению количества операций в параллельном алгоритме по сравнению с последовательным.

Приведенная оценка не учитывает времени, затрачиваемого на обмен данными между процессорами, и является справедливой для систем с общей памятью. Для систем с распределенной памятью это время необходимо учитывать. Тем не менее, на величину асимптотической оценки это время влияния не оказывает, поскольку пропорционально $M(N-1)$.

5.2. Оценка эффективности параллельного алгоритма с декомпозицией по времени

Время выполнения последовательного алгоритма T_1 оценивается аналогично п. 5.1 и составляет $T_1 = Mm(Nn-1)(2Nn+Mm+9)$ (оценка изменилась, поскольку теперь имеем Mm слоев точной сетки по времени).

Оценим время исполнения параллельного алгоритма. Как и ранее, долей последовательного расчета, выполняемого на этапе инициализации алгоритма до начала основной вычислительной процедуры, пренебрегаем. Расчет выполняется на грубой ω_G и точной ω_F сетках, определенных в (6) и (8), соответственно.

Наибольшее число операций будут выполнять процессоры, отвечающие за расчет подобластей $\Omega_{i,M}$, соответствующих последнему M-му временному слою, поэтому именно они и будут определять время реализации алгоритма. Для каждой итерации q количество операций, выполняемых каждым таким процессором, складывается из четырех составляющих: 1) операций, необходимых для вычисления дробных производных: $m(n-1)(2n+m+23)$, 2) операций, необходимых для вычисления интегралов ${}_0J_{X_{i-1}}u$, ${}_{X_i}J_L u$ и ${}_0I_{T_{M-1}}$ по явным аналитическим формулам через сплайн-интерполяцию: $114m(n-1)[2(N-1)+(M-1)]$, 3) операций, необходимых для расчета на грубой сетке по „грубому“ методу и уточнению значений U по формуле (17): $M(N-1)(2N+M+9)+2$ и 4) операций, необходимых для вычисления коэффициентов интерполяционного сплайна: $22(2MN-M-1)$. Сумма этих значений дает трудоемкость одной итерации метода Ньютона в параллельном алгоритме. Если для реализации алгоритма требуется Q итераций, то общее время выполнения параллельного алгоритма равно

$$T_p = Q[m(n-1)(2n+m+128N+114M-319)+M(N-1)(2N+M+53)+22(M-1)].$$

По классической формуле $S_p = T_1/T_p$ может быть легко оценено ускорение параллельного алгоритма. Если $N \ll n,\ M \ll m$, то $T_1 \approx MNmn(2Nn+Mm),\ T_p \approx Qmn(2n+m)$ и получаем следующую оценку ускорения:

$$S_p = \frac{MN(2Nn+Mm)}{Q(2n+m)}.$$

Для случая квадратной грубой сетки ($M=N$) получаем $S_p = N^3/Q$. Учитывая, что общее количество процессоров, задействованных для выполнения параллельного алгоритма, равно $P = MN = N^2$, получаем следующую асимптотическую оценку ускорения: $S_p \approx P^{3/2}/Q$. Максимально возможное количество итераций $Q = M = \sqrt{P}$. Однако не практике количество итераций всегда оказывается существенно меньше, поэтому действительная асимптотическая оценка ускорения получается сверхлинейной.

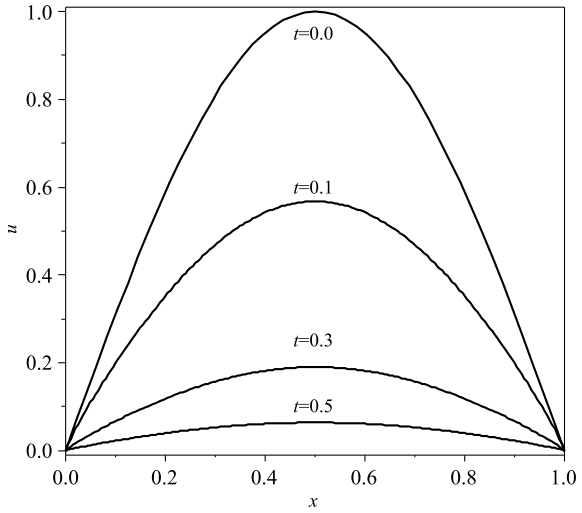


Рис. 1. Решение тестовой задачи для различных моментов времени

6. Результаты тестовых расчетов

Точность и эффективность предлагаемых алгоритмов была проверена рядом тестовых расчетов. Ниже представлены некоторые результаты проведенных вычислительных экспериментов.

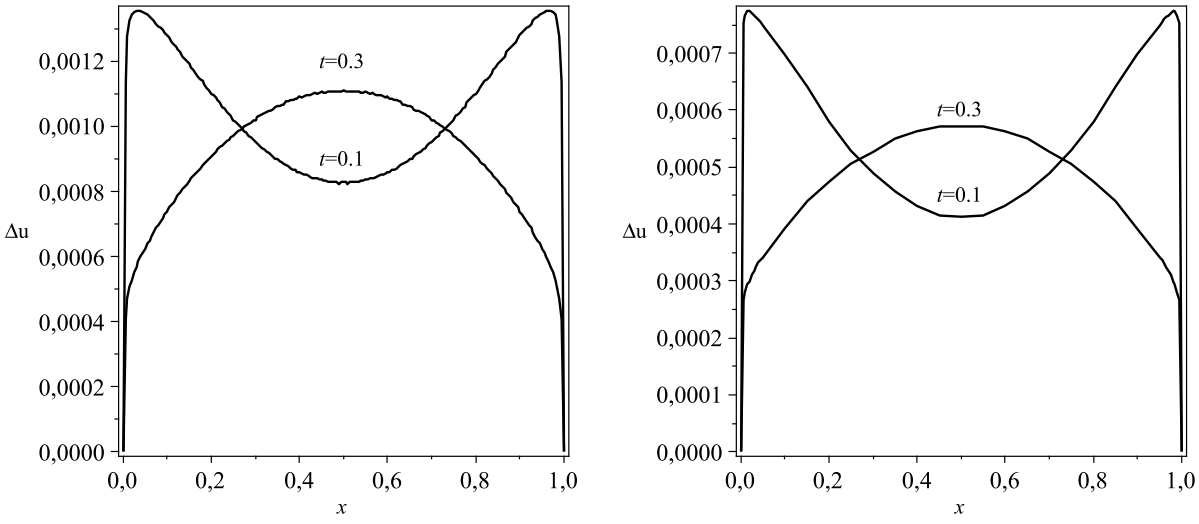
Решается уравнение (3) в единичном квадрате ($L = 1,\ T = 1$) при нулевых граничных условиях $u_0(t) = 0,\ u_L(t) = 0$, без функции источника $f(x,t) = 0$, с начальным условием $u^0(x) = \sin(\pi x)$. На Рис. 1 показано решение этой задачи в различные моменты времени при $\alpha = 0.95,\ \beta = 0.7,\ \gamma = 0.5$.

На Рис. 2 показаны ошибки приближенных решений Δu для двух моментов времени $t = 0.1$ и $t = 0.3$, полученных с помощью классического последовательного алгоритма при $m = 10^5$ шагов по времени и количестве узлов сетки по пространству а) $n = 200$ и б) $n = 400$. Как и следовало ожидать, измельчение сетки приводит к уменьшению ошибки при неизменном характере ее распределения.

На Рис. 3 показаны ошибки приближенных решений Δu для тех же моментов времени, полученных при использовании параллельного алгоритма с декомпозицией по пространству при $m = M = 10^5$ и а) количестве процессоров $P = N = 4$ и количестве узлов сетки по пространству в каждой подобласти $n = 100$; б) количестве процессоров $P = N = 8$ и количестве узлов $n = 50$. Расчеты проводились на одном узле суперкомпьютера Уфимского государственного авиационного технического университета, объединяющего на общей памяти объемом 4 Гб два четырехядерных процессора Intel Xeon 5300. Из приведенных рисунков видно, что параллельный алгоритм показывает тот же порядок точности, что и последовательный.

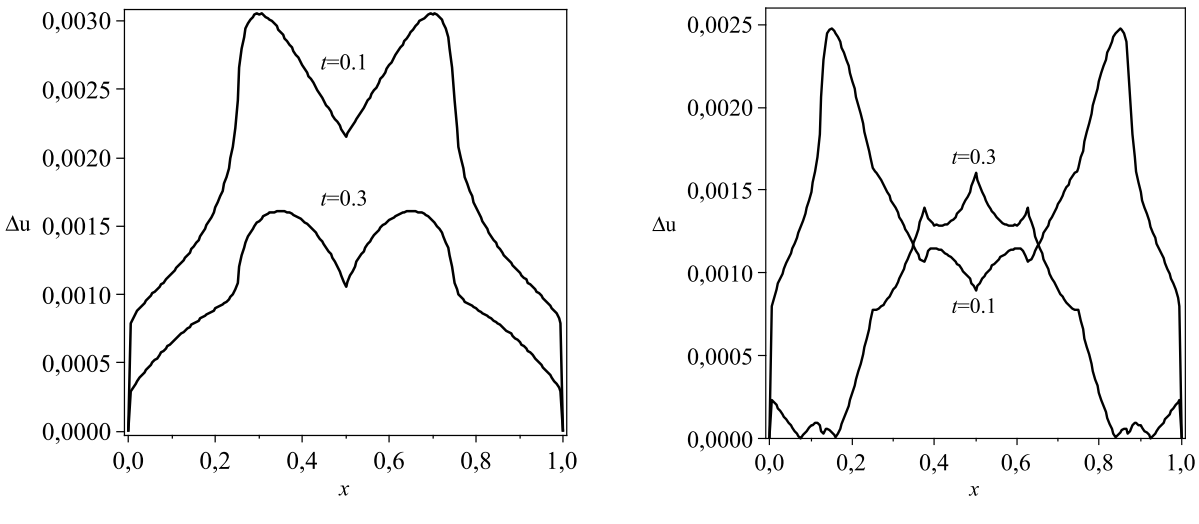
7. Заключение

Разработанные двухсеточные параллельные алгоритмы решения уравнений аномальной диффузии дробного порядка показывают сверхлинейное ускорение по сравнению с классическим последовательным конечно-разностным алгоритмом и обеспечивают тот же порядок точности вычислений при условии согласованного выбора шагов точной и грубой сеток. Это свидетельствует о неоптимальности классического алгоритма. Поэтому представляется целесообразным и при последовательных расчетах использовать двухсеточные алгоритмы, получаемые очевидной редукцией предложенных параллельных алгоритмов на однопроцессорный случай.



а) б)

Рис. 2. Ошибка решения Δu, полученного по классическому последовательному алгоритму: а) $n = 200$, б) $n = 400$



а) б)

Рис. 3. Ошибка решения Δu, полученного с использованием параллельного алгоритма с декомпозицией по пространству: а) $P = N = 4,\ n = 100$, б) $P = N = 8,\ n = 50$

Литература

1. Bouchaud J.P., Georges A. Anomalous diffusion in disordered media: statistical mechanisms, models and physical applications // Phys. Rep. 1990. Vol. 195. P. 127–293.

2. Metzler R., Klafter J. The random walk's guide to anomalous diffusion: A fractional dynamic approach // Phys. Rep. 2000. Vol. 339. P. 1–77.

3. Учайкин В.В. Автомодельная аномальная диффузия и устойчивые законы. // УФН. 2003. Т. 173, No. 8. С. 847–876.

4. Podlubny I. Fractional differential equations. Academic press, San Diego, 1999. 365 p.

5. Самко С.Г., Килбас А.А., Маричев О.И. Интегралы и производные дробного порядка и некоторые их приложения. Минск: Наука и техника, 1987. 688 с.

6. Учайкин В.В. Метод дробных производных. Ульяновск: Изд-во „Артишок“, 2008. 512 с.

7. Головизнин В.М., Киселев В.П., Короткин И.А., Юрков Ю.И. Прямые задачи неклассического переноса радионуклидов в геологических формациях // Известия Академии наук, серия "Энергетика". 2004. №. 4. С. 121–132.

8. Лукащук С.Ю. Модификация алгоритма PARAREAL для решения дифференциальных уравнений дробного порядка. // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции. Челябинск: Изд. ЮУрГУ, 2010. С. 519–524.

9. Завьялов Ю.С., Квасов Б.И., Мирошниченко В.Л. Методы сплайн-функций. М.: Наука, 1980. 352 с.

10. Lions J.-L., Maday Y., Turinici G. Resolution d'edp par un schema en temps parareal // C.R. Acad Sci. Paris. Ser. I Math. 2001. Vol. 332. P. 661–668.

11. Maday Y., Turinici G. The Parareal in Time Iterative Solver: a Further Direction to Parallel Implementation // Lecture Notes in Computational Science and Engineering: Domain Decomposition Methods in Science and Engineering XVIII. 2005. Vol. 40. P. 441–448.

12. Staff G.A., Ronquist E.M. Stability of the Parareal Algorithm // Lecture Notes in Computational Science and Engineering: Domain Decomposition Methods in Science and Engineering XVIII. 2005. Vol. 40. P. 449–456.

13. Gander M.J., Vandewalle S. On the Superlinear and Linear Convergence of the Parareal Algorithm // Lecture Notes in Computational Science and Engineering: Domain Decomposition Methods in Science and Engineering XVI. 2007. Vol. 55. P. 291–298.

14. Gander M.J., Vandewalle S. Analysis of the parareal time-parallel time-integration method // SIAM J. Sci. Comput. 2007. Vol. 29, N. 2. P. 556–578.

15. Fischer P.F., Hecht F., Maday Y. A Parareal in Time Semi-implicit Approximation of the Navier-Stokes Equations // Lecture Notes in Computational Science and Engineering: Domain Decomposition Methods in Science and Engineering XVIII. 2005. Vol. 40. P. 433–440.

16. Bal G., Wu Q. Symplectic Parareal // Lecture Notes in Computational Science and Engineering: Domain Decomposition Methods in Science and Engineering XVII. 2008. Vol. 60. P. 401–408.

Параллельный программный комплекс POLARA для моделирования обтекания профилей и исследования расчетных схем метода вихревых элементов*

И.К. Марчевский, В.С. Морева

МГТУ им. Н. Э. Баумана

Разработан программный комплекс POLARA для моделирования обтекания профилей бессеточным лагранжевым методом вихревых элементов (МВЭ). Его основная задача — построение зависимостей аэродинамических коэффициентов профиля от угла атаки, сводящееся к решению в параллельном режиме серии независимых задач по расчету обтекания профиля. Реализованные алгоритмы позволяют также распараллеливать каждый расчет, и комплекс POLARA может применяться для решения задач, когда в расчетной схеме присутствуют десятки-сотни тысяч вихревых элементов и вычислительная сложность чрезвычайно высока. Не менее важна возможность проведения серий методических экспериментов по анализу влияния параметров расчетных схем МВЭ.

1. Введение

Метод вихревых элементов (МВЭ) является весьма эффективным численным методом моделирования двумерных течений несжимаемой среды; в настоящее время он интенсивно развивается как в нашей стране, так и за рубежом [1, 2]. Возможности метода достаточно широки: он с одной стороны позволяет исследовать фундаментальные вопросы гидромеханики вплоть до эффектов, которые невозможно или чрезвычайно трудно смоделировать при использовании других численных методов [3], а с другой стороны — дает возможность решать многие инженерные задачи с приемлемой для практических приложений точностью при достаточно низких затратах машинного времени (по сравнению с сеточными методами).

В частности, одним из актуальных направлений исследований является разработка эффективных методов моделирования поведения конструкций в потоке жидкости или газа, когда требуется решать сопряженную задачу движения профиля в потоке под действием гидродинамических сил. Во многих математических моделях учет влияния набегающего потока на обтекаемый профиль приближенно, но вполне обоснованно осуществляется при помощи безразмерных стационарных аэродинамических коэффициентов C_{xa}, C_{ya} и C_m (коэффициенты лобового сопротивления, подъемной силы и аэродинамического момента соответственно). Движение профиля сопровождается изменением его угла атаки по отношению к набегающему потоку, что влечет за собой изменение аэродинамических нагрузок, которое может быть учтено через зависимости аэродинамических коэффициентов от угла атаки. Для некоторых представляющих практический интерес профилей эти зависимости определены экспериментально; однако в соответствующих справочниках приводятся данные лишь для определенного класса профилей, чаще всего — крыловых, а также для простейших форм плохообтекаемых профилей [4, 5].

В ряде случаев решаемые задачи сводятся к исследованию влияния потока среды на систему профилей, при этом одни профили (подветренные) попадают в спутный след других (наветренных). Зависимости их стационарных аэродинамических коэффициентов от взаимного расположения профилей с учетом взаимовлияния подробно исследованы экспериментально лишь для круговых профилей [6]. Таким образом, задача численного моделирования

*Работа выполнена при частичной финансовой поддержке РФФИ (проект № 11-08-00699-а) с использованием вычислительных ресурсов Межведомственного суперкомпьютерного центра РАН.

обтекания профилей и определения их аэродинамических характеристик в зависимости от угла атаки профиля либо его расположения по отношению к другим обтекаемым профилям является актуальной.

Для получения стационарных значений C_{xa}, C_{ya} и C_m для фиксированного угла атаки профиля требуется решить нестационарную задачу о моделировании обтекания неподвижного профиля, установленного под данным углом, а затем осреднить вычисленные нестационарные значения аэродинамических коэффициентов по большому промежутку времени. Для получения достаточно точных зависимостей аэродинамических характеристик от угла атаки может потребоваться значительное число расчетов. Скорость решения всей задачи, т. е. построения зависимостей $C_{xa}(\alpha)$, $C_{ya}(\alpha)$ и $C_m(\alpha)$ для определенного диапазона углов атаки α, становится особенно важной, когда требуется провести анализ большого количества возможных вариантов, например, при решении задачи оптимизации и поиска оптимальной формы профиля. Аналогичная ситуация возникает при проведении вычислительных экспериментов, чрезвычайно важных при разработке новых или модификации существующих расчетных схем МВЭ. К этому же классу относятся "методические" задачи подбора оптимальных параметров расчетной схемы численного метода.

Целью настоящей работы является анализ способов ускорения вычислений, проводимых с помощью метода вихревых элементов, а также создание алгоритмов оптимизации использования имеющихся вычислительных ресурсов.

2. Метод вихревых элементов

Метод вихревых элементов [1–3] — это бессеточный лагранжев метод численного моделирования нестационарного обтекания профилей, основанный на моделировании эволюции завихренности. Распределение завихренности моделируется набором вихревых элементов (ВЭ), для каждого из которых задается положение в пространстве и интенсивность. Скорость среды в любой точке течения вычисляется по закону Био — Савара по известным характеристикам ВЭ, а давление определяется при помощи аналога интеграла Коши — Лагранжа.

Метод вихревых элементов особенно эффективен при решении внешних задач обтекания, поскольку на каждом шаге расчета по времени выполнение уравнения неразрывности и граничного условия на бесконечности происходит автоматически. Граничное условие непротекания или прилипания обеспечивается генерацией ВЭ на поверхности обтекаемых профилей, а течение идеальной либо вязкой среды, описываемое уравнениями Эйлера либо Навье — Стокса, моделируется движением имеющихся ВЭ. Интенсивности рождаемых ВЭ находятся из решения соответствующей системы линейных алгебраических уравнений, аппроксимирующей граничное условие на профиле, а движение ВЭ описывается системой обыкновенных дифференциальных уравнений, правые части которой представляют собой скорости движения ВЭ.

Именно процедура определения скоростей вихревых элементов является наиболее затратной с вычислительной точки зрения, особенно при увеличении числа вихревых элементов в расчете, поскольку для этого приходится учитывать взаимное влияние всех пар ВЭ. Несколько менее трудоемкими, но тем не менее достаточно затратными являются операции реструктуризации вихревой пелены и вычисления аэродинамических нагрузок, действующих на профиль.

Таким образом, решение серии задач по расчету обтекания профилей под различными углами атаки требует выполнения большого объема вычислительной работы, но в силу однотипности и независимости отдельных задач процедуру их решения можно оптимизировать путем автоматизации проведения серии расчетов. Одним из перспективных направлений представляется применение параллельных алгоритмов, позволяющих оптимизировать использование имеющихся вычислительных ресурсов.

Помимо распараллеливания вычислений для сокращения времени счета возможно использование так называемого быстрого метода в “задаче N тел”. Аналог задачи N тел в методе вихревых элементов возникает при вычислении скоростей движения ВЭ, индуцируемых другими вихревыми элементами. Алгоритм приближенного решения данной задачи быстрым методом предлагается в работе [7]. Его использование позволяет существенно уменьшить вычислительную сложность задачи и дает возможность, к примеру, увеличить число вихревых элементов и тем самым повысить точность результатов.

3. Использование параллельных вычислений

3.1. Общая схема распараллеливания вычислений в комплексе POLARA

Рассмотрим один из способов ускорения вычислений в методе вихревых элементов — использование параллельных алгоритмов [8]. В каждой из задач с помощью алгоритма метода вихревых элементов моделируется обтекание профиля под фиксированным углом атаки и определяются аэродинамичексие нагрузки, действующие на него. В силу независимости этих задач наиболее очевидным является внешнее распараллеливание — одновременное решение нескольких задач. Для его реализации в разработанном программном комплексе POLARA предусмотрена возможность работы на современных многопроцессорных (многоядерных) вычислительных системах, что позволяет осуществлять конвейерную обработку очереди задач: использование большого числа вычислительных ядер дает возможность проводить несколько расчетов одновременно; по мере завершения отдельных расчетов и освобождения соответствующих ядер на них запускается процесс решения следующих задач.

В программном комплекс POLARA реализовано также внутреннее распараллеливание — параллельное выполнение наиболее трудоемких этапов алгоритма (например, вычисление скоростей ВЭ) внутри решения одной задачи обтекания профиля. Использование данного механизма позволяет оптимизировать загрузку вычислительных ядер при решении большой серии задач, а также сократить временные затраты при решении малой серии задач или единственной задачи, которая может иметь чрезвычайно высокую вычислительную сложность.

Программный комплекс POLARA позволяет использовать оба типа параллелизма одновременно, причем число вычислительных ядер, выделяемых под решение различных задач обтекания профиля под конкретными углами атаки может выбираться пользователем произвольным образом. Это связано с тем, что для одного и того же профиля вычислительная сложность моделирования его обтекания может существенно зависеть от угла атаки.

Рассмотрим основные этапы алгоритма определения аэродинамических характеристик некоторого профиля в заданном диапазоне углов атаки. Для этого требуется решить серию из N_{task} задач, каждая из которых соответствует обтеканию под определенным углом атаки. Пусть общее число доступных процессоров (вычислительных ядер) равно N_{proc}, а для решения i-й задачи выделяется $n_{proc}^{(i)}$ процессоров (рис. 1).

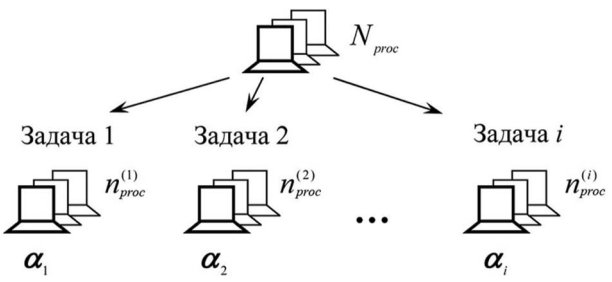


Рис. 1. Схема распараллеливания при решении задачи

Этап 1. На этапе запуска расчета из N_{proc} процессоров выделяется головной процессор, на который из текстового файла загружается очередь задач — список углов атаки α_i с указанием чисел $n_{proc}^{(i)}$. На этот процессор также загружаются из соответствующих файлов общие параметры расчетной схемы, геометрия обтекаемого профиля и значение "кванта времени" t_{kvant} — времени, через которое происходит синхронизация работы параллельных ветвей программы, контроль загрузки процессоров и обновление очереди задач. Если в каких-то из запущенных расчетов выполняется условие прекращения счета, то их решение останавливается, а задействованные в них процессоры высвобождаются. Таким образом, глобальный головной процессор контролирует решение серии задач и осуществляет пересылки необходимых данных между другими процессорами.

Этап 2. Данный этап повторяется циклически, пока не будут решены все задачи из очереди. Если в результате анализа состояния вычислительной системы и очереди задач обнаруживается наличие необходимого числа свободных процессоров для запуска решения последующих задач, формируются подгруппы из $n_{proc}^{(i)}$ процессоров и в каждой такой подгруппе выделяется головной процессор. Локальный головной процессор управляет ходом моделирования обтекания профиля на данном угле атаки и обеспечивает необходимые пересылки информации внутри своей подгруппы. Головные процессоры всех подгрупп обмениваются данными с глобальным головным процессором, передавая ему информацию о необходимости продолжения счета в данной задаче в течение следующего кванта времени или выполнении критерия останова счета.

Этап 3. На данном уровне осуществляется непосредственное численное моделирование обтекания профиля с помощью метода вихревых элементов. Шаги расчета по времени повторяются до тех пор, пока не закончится квант времени t_{kvant} или выполнится критерий останова, в качестве которого может использоваться признак выполнения заданного числа временны́х шагов.

3.2. Анализ эффективности распараллеливания

Наиболее затратными с вычислительной точки зрения являются следующие операции алгоритма МВЭ.

Операция 1. Вычисление скоростей ВЭ как результата их взаимного влияния друг на друга.

Операция 2. Реструктуризация вихревой пелены, приводящая к уменьшению числа ВЭ за счет объединения близкорасположенных ВЭ или исключения из расчетной схемы ВЭ, удалившихся от обтекаемого профиля на значительное расстояние.

Операция 3. Вычисление нагрузок, которые действуют на профиль со стороны потока, по рассчитанному полю скоростей и известному распределению завихренности.

Исследуем эффективность использования параллельных алгоритмов при решении плоских задач МВЭ (внутреннего распараллеливания). На рис. 2 представлены оценки вычислительной трудоемкости различных этапов последовательного алгоритма МВЭ.

Видно, что наиболее трудоемким является вычисление скорости ВЭ. Другие две операции — реструктуризация вихревой пелены и вычисление аэродинамических нагрузок — менее трудоемки, в то время как затратами машинного времени на прочие операции (например, решение системы линейных уравнений) можно пренебречь. Аналогичные соотношения трудоемкостей были получены и при решении пространственных задач обтекания тел методом вихревых элементов [9].

Согласно хорошо известному закону Амдаля [8], для вычислительной системы с S процессорами (вычислительными ядрами) максимально возможное ускорение компьютерной программы с долей P параллельного кода и $(1-P)$ последовательного кода равно

$$\alpha = \frac{1}{\frac{P}{S} + (1-P)}.$$

Рис. 2. Трудоемкость операций в методе вихревых элементов

С помощью этой формулы вычислено максимальное ускорение, которое теоретически достижимо при распараллеливании только операций 1, операций 1, 2 или операций 1, 2, 3 алгоритма, например, для вычислительных систем с 4, 16 и 64 ядрами. Результаты представлены в таблице 1.

Таблица 1. Максимальное ускорение параллельного кода

Распараллеленные операции	Доля параллельного кода	Максимальное ускорение		
		4 ядра	16 ядер	64 ядра
1	95,3 %	3,5	9,4	16,2
1, 2	98,2 %	3,8	12,6	30,0
1, 2, 3	99,5 %	3,9	14,9	48,7

Видно, что для вычислений с использованием небольшого числа ядер максимальные значения ускорений близки. Но при использовании системы с десятками вычислительных ядер распараллеливание операций 2 и 3 метода вихревых элементов дает значительный вклад в ускорение, несмотря на то, что трудоемкость этих операций составляют вместе лишь малую долю от трудоемкости операции 1 (приблизительно 4,4 %).

Реальное ускорение, наблюдаемое в расчетах, несколько меньше, чем предсказывает закон Амдаля, поскольку оценки ускорения по этому закону получены для “идеального” распараллеливания, т. е. ситуации, когда параллельный код выполняется в S раз быстрее на S-ядерной вычислительной системе.

На рис. 3 для программного комплекса POLARA представлена зависимость ускорения от числа вычислительных ядер при распараллеливании операций 1, 2 и 3 алгоритма, полученная при решении одной, весьма сложной с вычислительной точки зрения задачи обтекания профиля под фиксированным углом атаки. Для сравнения показано максимальное значение ускорения для случаев распараллеливания только операции 1, операции 1, 2 и операций 1, 2, 3.

Реальное ускорение при распараллеливании операций 1, 2, 3 близко к оценке по закону Амдаля для распараллеливания только операций 1, 2 (сказывается наличие пересылок данных, затрат на синхронизацию вычислений и прочих “накладных расходов”), поэтому, можно сделать вывод о том, что распараллеливание операции 3 важно для достижения

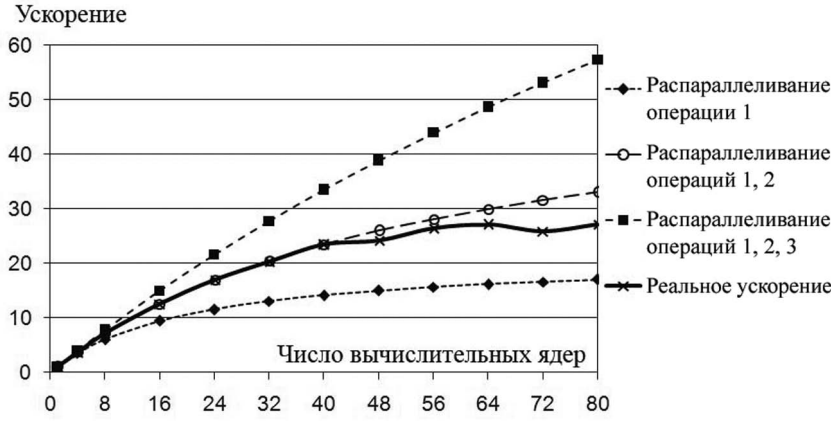


Рис. 3. Реальное ускорение и оценки по закону Амдаля (внутреннее распараллеливание)

высокой эффективности, хотя ее трудоемкость крайне мала — всего около 1,3 %.

Если требуется решить не одну задачу, пусть и весьма трудоемкую, а серию задач по расчету обтекания профиля под различными углами атаки, эффективность применения параллельных алгоритмов (внешнего распараллеливания) может быть существенно выше. В качестве тестового примера рассматривалась серия из 91 задачи по моделированию обтекания крылового профиля под различными углами атаки; в каждом расчете выполнялось 5000 временны́х шагов. На рис. 4 показаны затраты машинного времени при решении этой серии задач с помощью программного комплекса POLARA на 1, 32 и 64 вычислительных ядрах, а также зависимость ускорения вычислений от числа используемых ядер при проведении расчетов на вычислительных комплексах МВС-100К (МСЦ РАН), СКИФ-МГУ “Чебышев” (МГУ им. М.В. Ломоносова) и учебно-экспериментальном вычислительном кластере (МГТУ им. Н.Э. Баумана) [10].

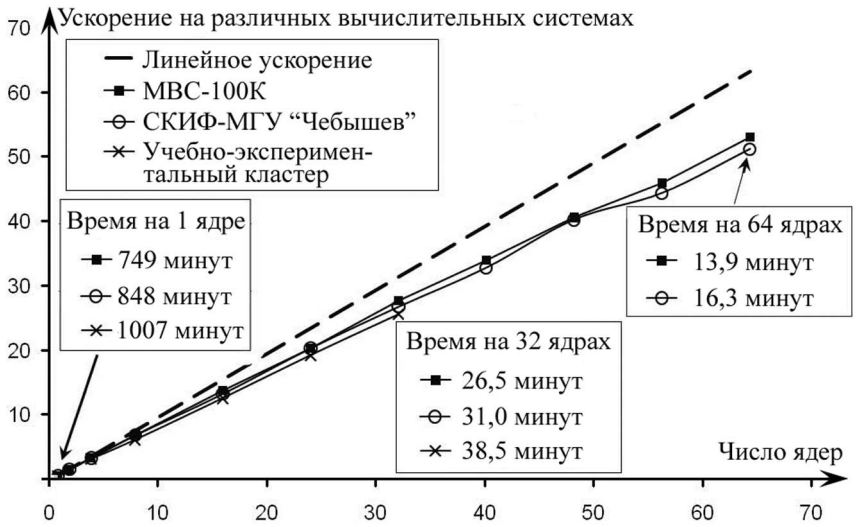


Рис. 4. Ускорение при решении серии задач (внешнее распараллеливание)

Видно, что на всех вычислительных комплексах ускорение растет практически линейно с ростом числа задействованных ядер, при использовании 64 вычислительных ядер удается получить более, чем 50-кратное ускорение расчетов.

3.3. Возможности применения различных моделей параллельного программирования

В рассмотренном выше примере для распараллеливания вычислений использовалась библиотека MPI как для внешнего, так и для внутреннего распараллеливания. Целесообразность ее применения для внешнего распараллеливания, т.е. для одновременного решения нескольких полностью независимых задач, не вызывает сомнений ввиду практически отсутствующего межпроцессорного обмена данными. В то же время при распараллеливании решения каждой задачи (внутреннем) интенсивность обменов данными между задействованными в расчете ядрами становится существенной, поэтому для повышения эффективности работы программного комплекса была исследована возможность применения для этих целей технологии OpenMP. С ее помощью были распараллелены те же 3 наиболее трудозатратные операции, что и при использовании MPI.

При проведении расчетов на вычислительных машинах с общей памятью на 2–8 вычислительных ядрах было установлено, что выигрыш по времени, достигаемый за счет использования OpenMP, не превышает нескольких процентов по сравнению с применением MPI. В связи с этим данная возможность практически не использовалась, поскольку применение MPI позволяет сделать программу универсальной — она может работать как на системах с общей, так и с распределенной памятью (кластерах). При этом для целей внутреннего распараллеливания может использоваться произвольное число вычислительных ядер, которые могут находиться на различных узлах кластера. В последнем случае эффективность распараллеливания, очевидно, снизится, но такая возможность необходима для решения сложных задач, когда в расчетной схеме присутствуют десятки-сотни тысяч вихревых элементов.

4. Использование быстрого метода вычисления вихревого влияния

4.1. Схема применения быстрого метода

Помимо применения параллельных алгоритмов, в программном комплексе POLARA предусмотрена возможность использования так называемого быстрого метода в “задаче N тел”, позволяющего сократить временны́е затраты на выполнение наиболее трудоемкой операции алгоритма — вычисление скоростей вихревых элементов, обусловленных их взаимным влиянием друг на друга. Быстрые методы решения “задачи N тел” основаны на том, что при расчете влияния удаленных ВЭ их коллективное воздействие рассчитывается приближенно согласно формулам, представленным в [7].

Первым этапом алгоритма быстрого метода является построение дерева — иерархической структуры прямоугольных областей (рис. 5), при этом прямоугольник нулевого уровня содержит все ВЭ. Он делится по длинной стороне на два одинаковых прямоугольника первого уровня. Путем перебора вихревых элементов определяется их принадлежность к одному из них. Далее аналогичным образом эти прямоугольники делятся пополам, образуя области второго уровня. Деление прекращается при выполнении заданного критерия по размеру прямоугольника, количеству ВЭ в нем и (или) номеру уровня. На втором этапе непосредственно вычисляются суммарные циркуляции и “центры тяжести” положительных и отрицательных ВЭ в каждом прямоугольнике нижнего уровня, а аналогичные параметры областей более высокого уровня находятся по характеристикам дочерних областей. Далее находятся скорости ВЭ в областях нижнего уровня: влияние ВЭ, находящихся в той же области, а также близкорасположенных ВЭ из других областей рассчитывается непосредственно по закону Био — Савара, а далее осуществляется обход дерева, и влияние ВЭ, расположенных в достаточно удаленных областях, учитывается при помощи 4 коэффици-

ентов, которые одинаковы для всех ВЭ из данной области.

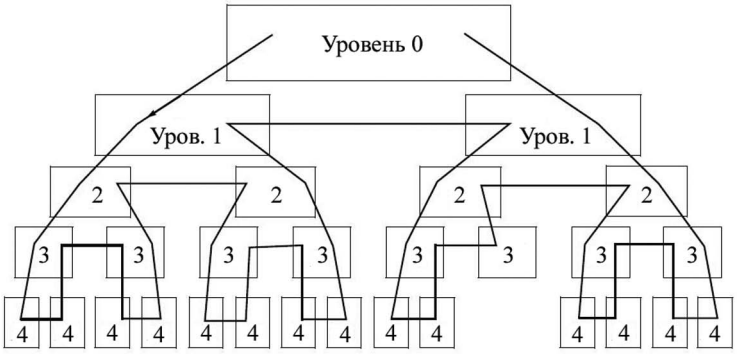


Рис. 5. Структура дерева, имеющего глубину 4 уровня, и направление его обхода

Соотношение трудоемкостей различных этапов алгоритма МВЭ при использовании быстрого метода, полученное при решении той же задачи, что была рассмотрена выше (рис. 2), имеет следующий вид (рис. 6).

Рис. 6. Трудоемкость операций в методе вихревых элементов при использовании быстрого метода

В сравнении с предыдущими оценками видно, что доля затрат на вычисление скоростей ВЭ сократилась, хотя и осталась доминирующей. Соотношение трудоемкостей операций при этом несколько изменилось, что связано с использованием при их выполнении оптимизированных алгоритмов, использующих имеющуюся иерархическую структуру дерева в вихревом следе. Использование быстрого метода решения "задачи N тел" позволило сократить общее время счета в рассмотренной тестовой задаче в 6 раз, поэтому его внедрение является целесообразным при решении "тяжелых" задач с достаточно большим числом ВЭ.

4.2. Оценка оптимальной глубины дерева

Актуальной задачей является выбор максимальной глубины построения дерева, при котором вычисление скоростей всех ВЭ может быть выполнено максимально быстро. Для этого оценим число арифметических операций (умножения и деления), выполняемых при вычислении скоростей ВЭ. Рассмотрим N ВЭ, равномерно распределенных в круге радиуса $a/\sqrt{\pi}$, тогда ячейка нулевого уровня — это квадрат со стороной a. Приближенно будем считать, что все ячейки всех уровней являются квадратными, тогда на k-м уровне имеется $N_k = \frac{\pi}{4}\cdot 2^k = \pi\cdot 2^{k-2}$ ячеек — квадратов со стороной $a_k = \frac{a}{(\sqrt{2})^k}$ (учитываются только ячейки,

попадающие внутрь круга). Пусть построено дерево глубиной k уровней; рассмотрим одну его ячейку и рассчитаем число ячеек этого же уровня, попадающих в ближнюю зону (т. е. тех ячеек, влияние ВЭ из которых будет учитываться непосредственно по закону Био — Савара). Под ближней зоной будем понимать круг с центром в центре данного квадрата и радиусом $\Delta = \frac{h}{\theta}$, где h — сумма габаритов ячеек (в данном случае $h = 4a/(\sqrt{2})^k = a/(\sqrt{2})^{k-4}$, $0 < \theta \leq 4$ — критерий близости. Число ячеек k-го уровня дерева, попадающих в этот круг, составляет

$$N_{\text{ближ}} = \frac{2^4\pi}{\theta^2}.$$

Пусть $j = m_{ks}$ — расстояние, выражаемое в длинах стороны квадрата k-го уровня, при котором квадрат s-го уровня попадает в дальнюю зону (т. е. относится к ячейкам, влияние ВЭ из которых вычисляется приближенно и быстро)

$$j = m_{ks} = \frac{2}{\theta}(1 + (\sqrt{2})^{k-s}),$$

тогда на расстоянии $j = m_{ks}$ ячеек k-го уровня к дальней зоне будут относиться ячейки уровня

$$s = k + 2 - 2\log_2(\theta j - 2).$$

Радиусы ближней области и всей исходной области, выраженные в длинах стороны квадрата k-го уровня, равны соответственно $p_k = \frac{4}{\theta}$ и $P_k = \frac{(\sqrt{2})^k}{\sqrt{\pi}}$, тогда приближенно количество ячеек (максимально высокого уровня), находящихся в дальней зоне, можно оценить следующим образом:

$$N_{\text{дал}} = \sum_{j=p_k+1}^{P_k} \frac{(2\pi a_k \cdot j)a_k}{a_s^2} = 8\pi \sum_{j=\frac{4}{\theta}+1}^{\frac{(\sqrt{2})^k}{\sqrt{\pi}}} \frac{j}{(\theta j - 2)^2} \approx$$

$$\approx \int_{\frac{4}{\theta}}^{\frac{(\sqrt{2})^k}{\sqrt{\pi}}} \frac{y}{(\theta y - 2)^2} dy = \frac{8\pi}{\theta^2}\left[\ln\left(\frac{\theta(\sqrt{2})^k}{2\sqrt{\pi}} - 1\right) - \left(\frac{\theta(\sqrt{2})^k}{2\sqrt{\pi}} - 1\right)^{-1} + 1\right].$$

Вычисление четырех коэффициентов влияния от каждой ячейки дальней зоны требует 28 арифметических операций, таким образом общее число операций, равно

$$Q_1 = 28 \cdot N_{\text{дал}} \cdot \pi 2^{k-2} = \frac{56\pi^2 2^k}{\theta^2}\left[\ln\left(\frac{\theta(\sqrt{2})^k}{2\sqrt{\pi}} - 1\right) - \left(\frac{\theta(\sqrt{2})^k}{2\sqrt{\pi}} - 1\right)^{-1} + 1\right].$$

Вычисление скоростей ВЭ, находящихся внутри ячеек нижнего уровня, через рассчитанные коэффициенты требует

$$Q_2 = 4 \cdot \frac{N}{\pi \cdot 2^{k-2}} \cdot \pi 2^{k-2} = 4N$$

арифметических операций.

Влияние ВЭ ближней зоны, рассчитываемое напрямую по закону Био — Савара, требует

$$Q_3 = 6 \cdot N_{\text{ближ}} \cdot \left(\frac{N}{\pi 2^{k-2}}\right)^2 \cdot \pi 2^{k-2} = \frac{3N^2}{\theta^2 2^{k-7}}$$

арифметических операций.

Итого, общее число операций, необходимое для вычисления скорости на одном шаге расчета составляет

$$Q = Q_1 + Q_2 + Q_3 = \frac{224\pi^3 p^2}{\theta^4}\left[\ln(p-1) - \frac{1}{p-1} + 1\right] + 4N + \frac{96N^2}{\pi p^2},$$

где $p = \frac{\theta \cdot 2^{k/2}}{2\sqrt{\pi}}$.

Для поиска значения p, при котором число арифметических операций минимально, следует приравнять к нулю производную $Q'(p)$. В результате получается алгебраическое уравнение

$$p^4 \left(2\ln(p-1) + 3 + \frac{1}{(p-1)^2} \right) = \frac{6N^2\theta^4}{7\pi^4}, \tag{1}$$

левая часть которого при $p > 1{,}5$ является возрастающей функцией; следовательно оно имеет единственное решение $p = p^*$, которое может быть найдено численно. Поскольку оптимальное значение глубины дерева k^* — натуральное число, оно вычисляется по формуле

$$k^* = \left[2 \left(\log_2 p^* + \log_2 \frac{2\sqrt{\pi}}{\theta} \right) \right], \tag{2}$$

в которой квадратные скобки обозначают целую часть числа.

Для проверки полученной оценки проведены расчеты при $N = 30\,000$; $60\,000$; $120\,000$. В табл. 2 приведены зависимости времени выполнения алгоритма быстрого метода от максимальной глубины дерева, при этом время счета для оптимальной глубины дерева, найденной по формулам (1)–(2), принято за единицу.

Таблица 2. Оптимальное число уровней дерева

Число уровней дерева	Время счета t_k/t_{k^*}		
	$N = 30\,000$ $k^* = 13$	$N = 60\,000$ $k^* = 14$	$N = 120\,000$ $k^* = 15$
11	2,20	4,01	7,12
12	1,18	2,04	3,55
13	1,00	1,30	2,07
14	1,27	1,00	1,11
15	1,88	1,38	1,00
16	2,04	2,04	1,34

Видно, что полученная оценка оптимального числа уровней дерева k^* верна, и именно при данной глубине дерева время счета является минимальным. Данная оценка является более точной, чем в работе [11].

Совместное использование быстрого метода вычисления вихревого влияния и параллельных вычислительных алгоритмов позволяет получить еще больший выигрыш по времени проведения расчета. Собственно быстрый метод хорошо распараллеливается, при этом используются те же подходы, что и в „прямом“ методе вычисления вихревого влияния. Процедуру построения иерархической структуры дерева распараллелить существенно сложнее, однако на практике в этом нет острой необходимости, поскольку время построения дерева (при числе уровней дерева, близком к оптимальному) обычно составляет доли процента от времени вычисления вихревого влияния, и эту процедуру можно осуществлять на всех вычислительных ядрах, занятых в параллельном расчете.

5. Исследование расчетных схем метода вихревых элементов

При создании программных комплексов для решения инженерных задач по моделированию обтекания профилей потоком с использованием МВЭ, а также при создании новых и модификации существующих расчетных схем метода возникает задача верификации разработанных моделей и оптимального подбора параметров, входящих в расчетную схему МВЭ. Расчетная схема МВЭ содержит ряд задаваемых расчетчиком параметров, наиболее важные из которых представлены в таблице 3.

Таблица 3. Основные параметры расчетной схемы МВЭ

Δt	шаг расчета по времени
ε	радиус вихревого элемента
ε_{col}	радиус коллапса — расстояние, на котором соседние ВЭ объединяются
`tik type`	метод контроля условия „непротыкания“
`odu ord`	порядок точности метода решения ОДУ, описывающих движение ВЭ [12]

Для верификации расчетной схемы и метода в целом необходимо проведение серии методических экспериментов. Опыт показывает, что при расчете обтекания существенно различных профилей (например, цилиндра или профиля крыла) параметры расчетной схемы следует выбирать по-разному. Для их подбора обычно решается задача о расчете обтекания „эталонного“ профиля с заранее известными характеристиками. Параметры расчетной схемы выбираются таким образом, чтобы согласовать результаты расчета с ними. Далее с этими же параметрами можно проводить исследование характеристик целого класса профилей, близких к „эталонному“ по форме, относительной толщине и т.п. Результаты расчетов показывают, что в этом случае характеристики исследуемых профилей удается вычислять достаточно точно. Проведение методических экспериментов требует выполнения большого числа расчетов, поскольку при подборе параметров расчетной схемы МВЭ нельзя ограничиться проведением серии расчетов для одного фиксированного угла атаки; для каждого исследуемого набора параметров требуется построение всей поляры или некоторой ее части. Это связано с тем, что режим обтекания профиля может существенно зависеть от угла атаки, и при некотором наборе параметров расчетной схемы результаты моделирования будут приемлемыми для одних углов и существенно неверными для других.

При выполнении расчетов указанного типа общая логика работы программного комплекса POLARA сохраняется, при этом отдельные независимые расчеты отличаются не только углом атаки, но также и набором параметров расчетной схемы, перечисленных в таблице 3. К их числу при необходимости могут быть отнесены и другие параметры; это потребует лишь незначительной переработки алгоритма комплекса POLARA. Следует отметить, что если при построении поляры профиля число расчетов редко превосходит несколько десятков, то при проведении методических исследований расчетных схем метода при варьировании хотя бы 2–3 параметров в некоторых, даже довольно узких пределах, число проводимых расчетов может составлять несколько сотен и даже тысяч. Поэтому использование параллельных вычислительных технологий, которые позволяют с высокой эффективностью выполнять указанные серии расчетов на многопроцессорных вычислительных комплексах, здесь особенно актуально.

6. Выводы

Предложены подходы, позволяющие сократить временны́е затраты при проведении расчетов методом вихревых элементов. С использованием параллельных вычислительных алгоритмов разработан программный комплекс POLARA, автоматизирующий и оптимизирующий процесс решения серии однотипных и независимых задач моделирования обтекания профиля.

В программном комплексе POLARA реализована возможность применения быстрого метода решения "задачи N тел", который позволяет сократить время вычислений (по сравнению с прямым методом), что важно при проведении расчетов с большим числом вихревых элементов. Построена оценка оптимального числа уровней дерева, при которой время счета максимально сокращается. Данная оценка подтверждена на тестовых примерах.

Литература

1. Cottet G.-H., Koumoutsakos P.D. Vortex Methods: Theory and Practice. Cambridge University Press, 2008. 328 p.
2. Андронов П.Р., Гувернюк С.В., Дынникова Г.Я. Вихревые методы расчета нестационарных гидродинамических нагрузок. М.: Изд-во МГУ, 2006. 184 с.
3. Дынникова Г.Я. Вихревые методы исследования нестационарных течений вязкой несжимаемой жидкости: Дис. ... докт. физ.-мат. наук. М., 2011. 269 с.
4. Случановская З.П. Распределение давления на поверхности прямоугольного, трехгранного и полукруглого цилиндров и их аэродинамические коэффициенты // Тр. Инс-та механики МГУ, № 24. М.: Изд-во МГУ, 1973. С. 52–60.
5. Кашафутдинов С.Т., Лушин В.Н. Атлас аэродинамических характеристик крыловых профилей. Новосибирск: СО РАН, 1994. 78 с.
6. Здравкович М.М. Обзор исследований интерференции между двумя круглми цилиндрами при различном их взаимном расположении // Труды Америк. общества инж.-механиков, сер. "Д", рус. перевод. 1977. Т. 99, № 4. С. 119–137.
7. Дынникова Г.Я. Использование быстрого метода решения "задачи N тел" при вихревом моделировании течений // Журнал вычислительной математики и математической физики. 2009. Т 49, № 8. С. 1458–1465.
8. Гергель В.П. Теория и практика параллельных вычислений. М.: Бином, 2007. 424 с.
9. Марчевский И.К., Щеглов Г.А. Применение параллельных алгоритмов при решении задач гидродинамики методом вихревых элементов // Вычислительные методы и программирование. 2010. Т 11. С. 105–110.
10. Лукин В.В., Марчевский И.К. Учебно-экспериментальный вычислительный кластер. Ч.1. Инструментарий и возможности // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2011. № 4. С. 28–43.
11. Гирча А.И. Быстрый алгоритм решения „Задачи N тел" в контексте численного метода вязких вихревых доменов // Информационные технологии моделирования и управления. 2008. № 1. С. 47–52.
12. Марчевский И.К., Морева В.С. Численное моделирование обтекания системы профилей методом вихревых элементов // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2010. № 1. С. 12–20.

Высокопроизводительные вычисления в моделировании динамики и сейсмичности систем тектонических плит*

В.Л. Розенберг, Л.А. Мельникова

Институт математики и механики УрО РАН, Екатеринбург

Представлено краткое описание разработанной авторами сферической блоковой модели динамики и сейсмичности литосферы. Приводятся новые результаты моделирования на МВС, полученные для аппроксимации глобальной системы тектонических плит посредством модификации, учитывающей неоднородность земной коры. В частности, реализованы возможности определения механизма модельного землетрясения и учета случайных факторов, влияющих на параметры модели. Предполагается, что результаты вычислительных экспериментов могут использоваться в экспертных системах мониторинга регионального и глобального сейсмического риска.

1. Введение

Изучение сейсмичности и особенно ее вариаций во времени на основе каталогов зарегистрированных землетрясений затруднено крайне короткой историей инструментальных наблюдений (период порядка 100 лет), по сравнению со временем действия тектонических факторов, обуславливающих сейсмичность в том или ином регионе. Явления, обнаруженные в таком каталоге землетрясений, могут быть единичными и не повторяться в будущем. Искусственные же каталоги, полученные путем численного моделирования, могут покрывать длительные интервалы времени, что позволяет анализировать статистическую значимость исследуемых свойств реального сейсмического потока, в частности, выявлять/подтверждать закономерности, предшествующие сильным толчкам (так называемые «предвестники»). Таким образом, результаты вычислительных экспериментов могут быть востребованы в экспертных системах мониторинга регионального и глобального сейсмического риска [1, 2]. Отметим, что основным результатом моделирования сейсмичности литосферы является искусственный каталог землетрясений, в котором каждое событие характеризуется моментом времени, координатами эпицентра, глубиной, магнитудой и, в некоторых моделях, учитывающих геологическое строение региона, интенсивностью. Моделирование динамики земной коры предполагает получение поля скоростей движения точек на разных глубинах, действующих сил, обусловленных ими смещений, а также характера взаимодействия структурных элементов.

До сих пор не существует адекватной теории сейсмотектонического процесса, но на основе имеющихся данных можно предположить, что различные свойства литосферы (пространственная неоднородность, иерархическая блоковая структура, типы нелинейной реологии, гравитационные и термодинамические процессы, коррозия под напряжением и др.) соотносятся со свойствами последовательностей землетрясений. Устойчивость этих свойств на количественном уровне в различных регионах позволяет сделать вывод, что литосферу можно моделировать как большую диссипативную систему, поведение которой не зависит существенно от частных деталей конкретных процессов, протекающих в геологической системе. Среди различных подходов к моделированию литосферных процессов (см., например, работу [1] и библиографию к ней) можно выделить два основных направления. Первое, традиционное, направление опирается на детальное исследование одного специфического тектонического разлома или, нередко, одного конкретного сильного землетрясения с целью воспроизведения определенных пре- и/или пост-сейсмических явлений, характерных для данного разлома или события. Модели второго направления, предложенные относительно недавно, трактуют сейсмотектонический

* Работа выполнена в рамках Программы научно-исследовательских работ Президиума РАН № 15 «Информационные, управляющие и интеллектуальные технологии и системы» при поддержке УрО РАН (проект 12-П-1-1023) и Программы государственной поддержки ведущих научных школ (проект НШ-6512.2012.1).

процесс гораздо более абстрактно; основной задачей моделирования является получение универсальных свойств сейсмичности, обнаруженных эмпирическим путем (прежде всего, степенного закона распределения «размера» событий, именно, закона повторяемости Гутенберга–Рихтера, кластеризации, миграции событий, сейсмического цикла и т. д.). Представляется, однако, что адекватная модель, разрабатываемая в рамках второго направления, должна не только отражать некоторые общие свойства нелинейных динамических систем, но и учитывать геометрию взаимодействующих тектонических разломов. Блоковые модели динамики и сейсмичности литосферы [2, 3] разрабатывались с учетом обоих требований. Подход к моделированию опирается на концепцию иерархической блоковой структуры литосферы [4]. Тектонические плиты представляются в виде системы абсолютно жестких блоков, находящейся в состоянии квазистатического равновесия; при этом модельное событие представляет собой резкий сброс напряжений, возникающих на разломах, разделяющих блоки, под действием внешних сил. Два главных механизма, включенных в сейсмотектонический процесс, именно, тектоническое нагружение с характеристической скоростью в несколько см/год и перераспределение упругого напряжения с характеристической скоростью в несколько км/сек, рассматриваются в модели в стандартной временной шкале как, соответственно, равномерное движение и мгновенный сброс напряжения. Плоская блоковая модель [2, 3], в которой структура ограничена двумя горизонтальными плоскостями, является наиболее изученной; на ее основе построены аппроксимации реальных сейсмических регионов. Однако, при попытке моделирования динамики глобальных тектонических плит обнаруживаются существенные неточности, для преодоления которых введена сферическая геометрия. Компьютерная реализация сферического варианта требует гораздо больше памяти и времени работы процессора, чем плоская блоковая модель. Поэтому для работы со сферической модификацией используются многопроцессорные машины и параллельные вычислительные технологии. Настоящая работа фактически является продолжением исследований [5–8]; ее новизна состоит в описании новых возможностей математической модели и в презентации результатов новых вычислительных экспериментов.

2. Сферическая блоковая модель: краткое описание

Подробное описание всех модификаций сферической блоковой модели динамики и сейсмичности литосферы приведено в [5]. В данной работе ограничимся изложением основных понятий и принципов, при этом особое внимание уделим нововведениям в модели, касающимся определения модельного землетрясения. Блоковая структура является ограниченной и односвязной частью шарового слоя глубиной H, заключенного между двумя концентрическими сферами, одна из которых (внешняя) интерпретируется как поверхность Земли, другая (внутренняя) ─ как нижняя граница упругой литосферы (см. Рис. 1). Разделение структуры на блоки определяется пересекающими этот слой бесконечно тонкими разломами, каждый из которых представляет собой коническую поверхность, наклоненную под определенным углом к внешней сфере. Общие точки двух разломов на внешней и внутренней сферах называются вершинами. Участки разломов, ограниченные соответствующими парами соседних вершин, называются сегментами. Пересечения блока с ограничивающими сферами представляют собой сферические многоугольники, при этом пересечение с нижней (для блока) сферой называется подошвой. Предполагается, что вне блоковой структуры могут находиться граничные блоки, примыкающие к внешним сегментам. Другая возможность состоит в рассмотрении блоковой структуры, замкнутой на сфере. Блоки считаются абсолютно жесткими, все их смещения ─ бесконечно малыми по сравнению с линейными размерами, поэтому геометрия блоковой структуры не меняется в процессе моделирования, и структура не движется как единое целое. Гравитационными силами можно пренебречь, так как они слабо зависят от смещений блоков и блоковая структура в начальный момент времени находится в состоянии квазистатического равновесия. Блоки (в том числе и граничные) имеют шесть степеней свободы. Смещение каждого блока состоит из поступательной и вращательной компонент. Предполагается, что законы движения граничных блоков и подстилающей среды известны, при этом движение описывается как вращение на сфере, т.е. задаются положение оси вращения и угловая скорость.

Разработано несколько модификаций модели, зависящих от способа трактовки глубины сферического слоя. В первой модификации модели (без глубины), считалось, что все характе-

ристики точек структуры определяются только их координатами и не зависят от глубины сферического слоя, поскольку данная глубина значительно меньше линейных размеров блоков. Главное преимущество модификации состоит в значительной экономии времени счета при моделировании, что может быть существенно при большом количестве запусков в эксперименте по вариации того или иного параметра; основной недостаток — формальный учет углов наклона разломов, фактически определяющих характер сейсмичности. Появившаяся позже модификация с постоянной глубиной использовала предположение об однородности литосферы по глубине (все блоки имели одну и ту же глубину H, а свойства всех частей блока (разлома) были одинаковыми). Модификация, разрабатываемая в настоящее время, предусматривает возможность задания различных глубин (в пределах H) для разных блоков и учета зависимости вязкоупругих свойств разлома от его глубины. Отметим, что по существу это является первой попыткой учета неоднородности литосферы (например, различий в строении континентальной и океанической коры и уменьшения вязкости коры с глубиной) в блоковых моделях.

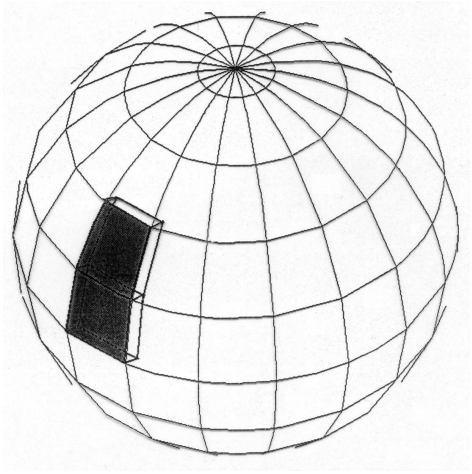


Рис. 1. Пример блоковой структуры на сфере

Поскольку блоки являются абсолютно жесткими, все деформации имеют место на разломах и подошвах блоков; силы возникают на подошвах из-за смещения блоков относительно подстилающей среды и на поверхностях ограничивающих их разломов из-за смещений соседних блоков или их подстилающей среды. Приведем формулы для определения упругой силы (f_t, f_l, f_n), действующей на единицу площади разлома:

$$f_t = K_t(\Delta_t - \delta_t), \quad f_l = K_l(\Delta_l - \delta_l), \quad f_n = K_n(\Delta_n - \delta_n). \tag{1}$$

Здесь (t,l,n) — система координат, связанная с точкой приложения силы (оси t, l лежат в плоскости, касательной к поверхности разлома, ось n ей перпендикулярна, см. Рис. 2); Δ_t, Δ_l, Δ_n — компоненты относительного смещения в системе (t, l, n) (а) соседних блоков в случае, если точка принадлежит части разлома, разделяющей блоки, и (б) блока и подстилающей среды соседнего блока в случае, если точка принадлежит части разлома, отделяющей блок от подстилающей среды соседнего блока; δ_t, δ_l, δ_n — соответствующие неупругие смещения, зависимость от времени которых описывается уравнениями

$$\frac{d\delta_t}{dt} = W_t f_t, \quad \frac{d\delta_l}{dt} = W_l f_l, \quad \frac{d\delta_n}{dt} = W_n f_n. \tag{2}$$

Коэффициенты K_t, K_l, K_n (1), характеризующие упругие свойства разлома, и коэффициенты W_t, W_l, W_n (2), характеризующие вязкие свойства разлома, могут быть различными для разных разломов и, кроме того, могут изменяться в зависимости от глубины.

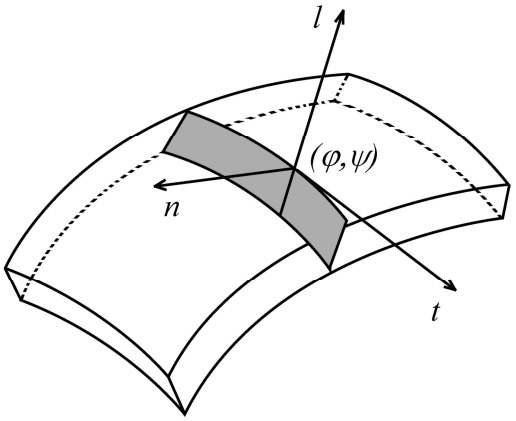


Рис. 2. Локальная система координат, связанная с точкой разлома (φ,ψ)

Аналогично выглядят формулы для вычисления сил и неупругих смещений на подошвах блоков. Смещения любого внутреннего блока и углы его поворотов находятся из условия равенства нулю суммы всех сил, действующих на блок, и суммарного момента этих сил. Это условие обеспечивает состояние квазистатического равновесия системы и одновременно является условием минимума энергии. Поскольку в рассматриваемой модели зависимость сил от смещений и поворотов блоков является линейной (явные формулы опущены ввиду их громоздкости, см. [5]), то система уравнений для определения этих величин также линейна и имеет вид

$$Aw = b\,. \tag{3}$$

Компонентами неизвестного вектора $w = (w_1, w_2, \ldots, w_{6n})$ являются смещения и углы поворота внутренних блоков (n — число таких блоков). Элементы матрицы A (размерности $6n \times 6n$) не зависят от времени и могут быть вычислены один раз в начале процесса. Для подсчета различных криволинейных интегралов выполняется дискретизация (разбиение на ячейки) сферической поверхности подошв блоков и сегментов разломов, при этом предполагается, что значения сил и неупругих смещений совпадают для всех точек ячейки. Система (3) решается в дискретные моменты времени t_i.

В каждый момент t_i при вычислении компонент силы, действующей на разломе, определяется безразмерная величина κ, трактуемая как модельное напряжение:

$$\kappa = \frac{\sqrt{f_t^2 + f_l^2}}{P - f_n}\,. \tag{4}$$

Здесь P — одинаковый для всех разломов параметр, который может интерпретироваться как разность между литостатическим и гидростатическим давлением. Таким образом, фактически величина κ является отношением модуля силы, стремящейся сдвинуть блоки вдоль разлома, к модулю силы, прижимающей блоки друг к другу. Для каждого разлома задаются значения трех порогов прочности, вообще говоря, зависящие от времени:

$$B > H_f \geq H_s\,,$$
$$B = B(t_i) = B_0(t_i) + \sigma X(t_i)\,, \quad H_f = H_f(t_i) = aB(t_i)\,, \quad H_s = H_s(t_i) = bH_f(t_i)\,. \tag{5}$$

Для всех i выполняется: $0 < B_0(t_i) < 1$, $0 < \sigma << 1$, $X(t_i)$ – случайная величина, распределенная по стандартному нормальному закону $N(0;1)$, $0 < a < 1$, $0 < b \le 1$. Предполагается, что начальные условия таковы, что неравенство $\kappa < B$ имеет место во всех ячейках структуры.

Взаимодействие между блоками (между блоком и соседней подстилающей средой) полагается вязкоупругим (нормальное состояние) до тех пор, пока величина κ (4) на части разлома, разделяющего элементы структуры, не достигает заданного порога B. Такая ситуация интерпретируется как землетрясение. В ячейках, попавших в «критическое» состояние, в соответствии с законом сухого трения, происходит резкий сброс напряжения посредством изменения значений неупругих смещений δ_t, δ_l, δ_n по формулам:

$$\delta_t^e = \delta_t + \gamma^e \xi_t f_t, \quad \delta_l^e = \delta_l + \gamma^e f_l, \quad \delta_n^e = \delta_n + \gamma^e \xi_n f_n, \tag{6}$$

где $\delta_t, f_t, \delta_l, f_l, \delta_n, f_n$ – значения неупругих смещений и компонент вектора силы непосредственно перед землетрясением. Коэффициенты $\xi_t = K_l / K_t$ ($\xi_t = 0$ при $K_t = 0$) и $\xi_n = K_l / K_n$ ($\xi_n = 0$ при $K_n = 0$) отражают предположение о неоднородности смещений в разных направлениях в том смысле, что одно и то же значение упругой силы приводит к различным скоростям изменения различных неупругих смещений. Коэффициент γ^e задается формулой:

$$\gamma^e = \frac{\sqrt{f_t^2 + f_l^2} - H_f (P - f_n)}{K_l (\sqrt{f_t^2 + f_l^2} + H_f f_n)}, \tag{7}$$

при этом для нового значения модельного напряжения κ справедливо равенство $\kappa = H_f$, что следует из (1), (4)–(7). После описанных выше пересчетов находится правая часть системы (3), затем определяются векторы сдвига и углы поворота блоков. Если вновь в какой-либо ячейке $\kappa \ge B$, то вся процедура повторяется. Когда во всех ячейках на разломах $\kappa < B$, вычисления продолжаются по обычной схеме. Считается, что ячейки, в которых произошли землетрясения, находятся в состоянии крипа. Это означает, что для них в уравнениях (2) для вычисления значений неупругих смещений используются параметры W_t^s ($W_t^s >> W_t$), W_l^s ($W_l^s >> W_l$) и W_n^s ($W_n^s >> W_n$), обеспечивающие значительно более быстрый, по сравнению с нормальным состоянием, рост неупругих смещений и, следовательно, уменьшение значений сил и напряжений. Состояние крипа продолжается до тех пор, пока $\kappa > H_s$, после чего ячейка возвращается в нормальное состояние с использованием при расчетах W_t, W_l и W_n. Качественный характер зависимости величины κ от времени в случае постоянных значений порогов B, H_f, H_s показан на Рис. 3.

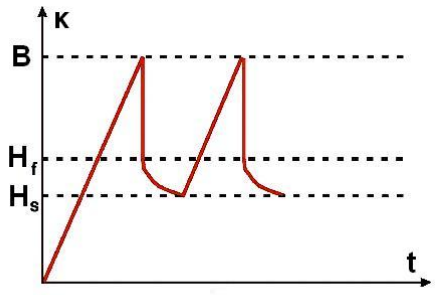


Рис. 3. Зависимость величины κ от времени

Основным результатом процесса моделирования является искусственный каталог землетрясений. Принадлежащие одному разлому ячейки, в которых произошло землетрясение в момент времени t_i, объединяются в одно событие. Географические координаты его эпицентра и глубина вычисляются как взвешенные суммы координат и глубин ячеек (вес ячейки определяется как отношение ее площади к сумме площадей всех ячеек, вовлеченных в землетрясение). Взвешенная сумма векторов $(\gamma^e \xi_t f_t, \gamma^e f_l)$ добавок к неупругим смещениям в ячейках δ_t и δ_l при пересчете по формулам (6) аппроксимирует случившуюся подвижку блоков вдоль разлома, и позволяет определить механизм модельного события. Механизм землетрясения ─ важная его характеристика, информирующая о процессе распространения различных сейсмических волн от очага. В зависимости от направления подвижки и угла наклона разлома принято выделять следующие основные механизмы: сдвиг, сброс и взброс [9]. В новой версии модели магнитуда землетрясения вычисляется в зависимости от его механизма с использованием известных в сейсмологии эмпирических формул [10]

$$M = D \lg S + E, \tag{8}$$

где S ─ сумма площадей ячеек (в км^2), $D = 1.02$, $E = 3.98$ для сдвига, $D = 1.02$, $E = 3.93$ для сброса, $D = 0.90$, $E = 4.33$ для взброса.

Отметим, что в каждый момент времени модель дополнительно позволяет получить картину мгновенной кинематики блоков и информацию о характере их взаимодействия вдоль границ.

3. Результаты вычислительных экспериментов

Сферическая блоковая модель активно применялась к исследованию динамики и сейсмичности глобальной системы крупнейших тектонических плит, покрывающих всю поверхность Земли [5–8]. Рассматриваемая структура включала 15 плит (это Наска, Южноамериканская, Кокос, Карибская, Североамериканская, Тихоокеанская, Африканская, Антарктическая, Евразийская, Аравийская, Индийская, Сомалийская, Филиппинская, Австралийская, Хуан де Фука) и 199 разломов. Движение подстилающей среды определялось как вращение на сфере согласно модели HS3-NUVEL1 [11]. Модельные глубины плит выбирались с учетом распределения по глубине реальных землетрясений, углы наклоны разломов были приближены к реальным [8].

В данной работе представлены результаты новых вычислительных экспериментов с указанной блоковой структурой. Анализ проводился по двум основным направлениям: во-первых, исследование механизмов модельных событий и использование новой схемы подсчета магнитуды (8), а, во-вторых, введение случайных значений для порогов прочности (см. (5)), что отражает изменения свойств среды, неподдающиеся аналитическому описанию, и изучение чувствительности модели к неточностям в задании ключевых параметров.

В качестве основных параметров, определяющих качество моделирования, рассматриваются пространственное распределение сильных событий и параметры закона Гутенберга–Рихтера, характеризующего распределение землетрясений по магнитуде. Был проведен сравнительный анализ искусственных каталогов землетрясений, которые создавались с использованием формул (8), и реального, извлеченного из глобального каталога NEIC [12] и включающего события за период времени с 01.01.1900 по 31.12.2010 без ограничений по глубине и местоположению, см. Рис. 4. Полученные искусственные каталоги землетрясений (на Рис. 5, 6 приведены данные для «лучшего» из них в смысле соответствия реальному) обнаружили ряд черт, присущих наблюденной сейсмичности, например: (а) наличие двух основных сейсмических поясов, Тихоокеанского и Средиземноморско-Трансазиатского, где происходит большая часть сильных событий; (б) увеличение сейсмической активности вблизи точек, где сходятся три и более плит. Установлено соответствие сейсмически активных (границы Кокос/Карибы, Индия/Евразия, Наска/Южная Америка, район Калифорнии, Аравия/Евразия, юго-восток, восток, северо-восток и особенно север Австралийской плиты, вокруг Филиппин) и «спокойных» (юг Тихоокеанской плиты, Наска/Тихий Океан, восток и юго-запад Африки, Индия/Австралия, Северная Америка/Евразия) регионов реальным [13], что следует считать позитивным фактом.

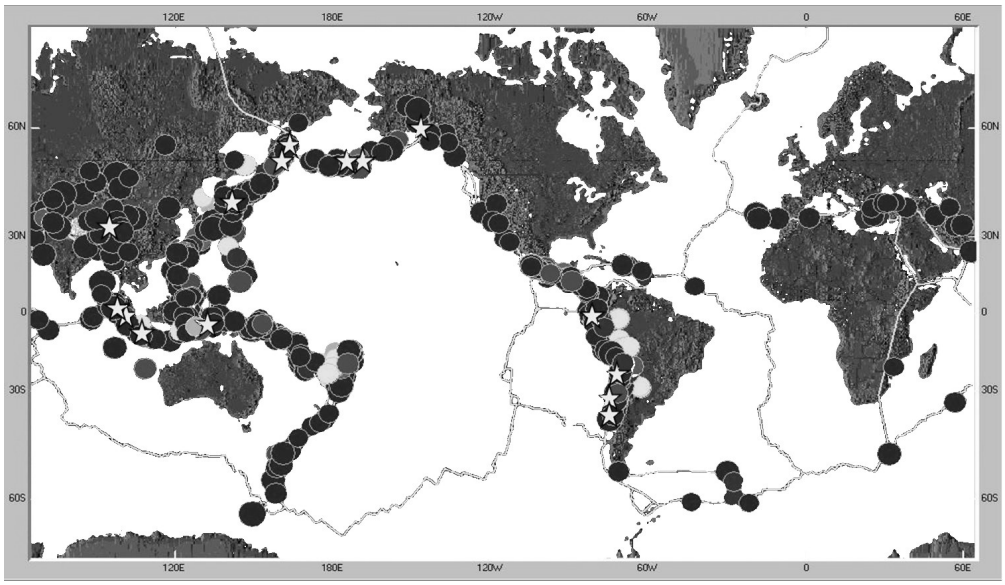


Рис. 4. Зарегистрированная сейсмичность: эпицентры сильных землетрясений с магнитудой не менее 7.5, каталог NEIC, период 01.01.1900–31.12.2010. Звездочками обозначены 15 сильнейших событий

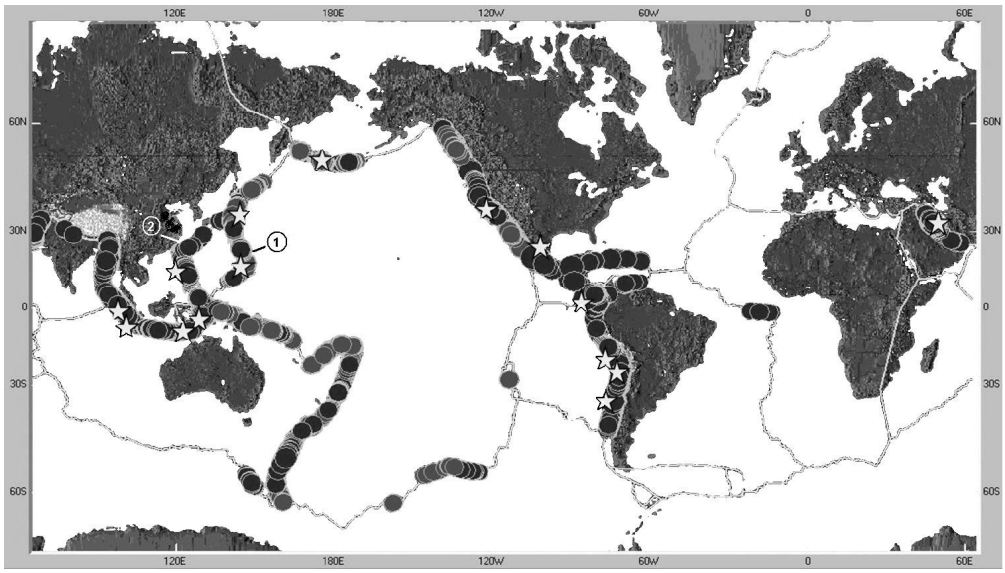


Рис. 5. Модельная сейсмичность: эпицентры сильных землетрясений с магнитудой не менее 7.0 за 100 единиц безразмерного времени. Звездочками обозначены 15 сильнейших событий

Более того, учет механизмов землетрясений в модели позволил получить удовлетворительное соответствие сильнейших модельных событий реальным (Рис. 4, 5) и близость углов наклона графиков повторяемости для реального и модельного каталогов на достаточно большом магнитудном интервале (Рис. 6), чего не удавалось на предыдущем этапе [6, 8]. В то же время, сравнение механизмов конкретных реальных и модельных событий выявило неудовлетворительные результаты, что стимулирует изучение влияния параметров модели на тип механизма события. Некоторые результаты в этом направлении приведены в таблице 1. Отметим, что для большинства землетрясений, зарегистрированных на разломах 1 и 2 (Рис. 5), механизм является «сбросом» [14]. Из таблицы следует, что одинаковые изменения коэффициентов на разных разломах (хотя и очень похожих) приводят к различной динамике механизмов, что, с одной стороны, соответствует представлениям о неустойчивости механизмов событий, а, с другой стороны, предполагает планирование более сложных вычислительных экспериментов.

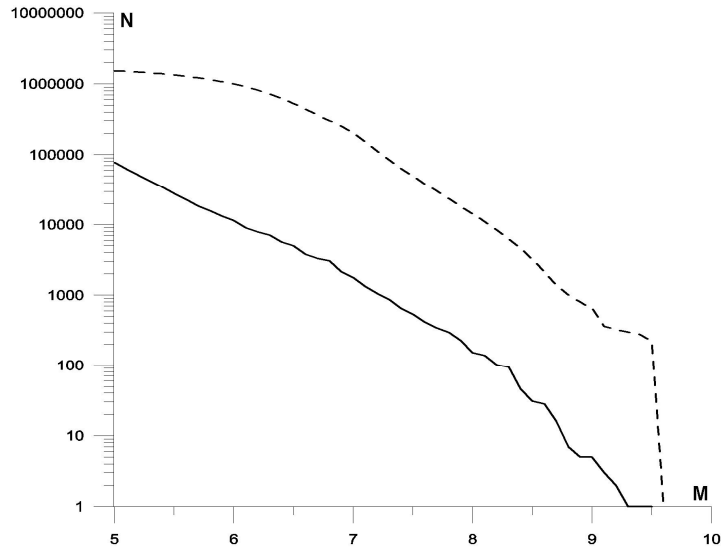


Рис. 6. Графики повторяемости, построенные по реальному (NEIC, события с магнитудой не менее 5.0, период 01.01.1900–31.12.2010, сплошная линия) и модельному (события с магнитудой не менее 5.0, 100 единиц безразмерного времени, штриховая линия) каталогам; N — аккумулированное число землетрясений, M — магнитуда

Таблица 1. Зависимость механизмов модельных событий от коэффициентов из формулы (1) (все остальные параметры фиксированы) для разломов 1 и 2 (см. Рис. 5)

Разлом	Коэффициенты на разломе	Общее количество событий на разломе (15 ед. безразм. вр.) и доля событий со сдвигом M_1, со сбросом M_2, с взбросом M_3
1	$K_t = 15,\ K_l = 5,\ K_n = 10$	1771: M_3 – 100%
2	$K_t = 30,\ K_l = 10,\ K_n = 20$	2364: M_1 – 80.63%, M_2 – 13.03%, M_3 – 6.34%
1	$K_t = 30,\ K_l = 10,\ K_n = 20$	2790: M_3 – 100%
2	$K_t = 15,\ K_l = 5,\ K_n = 10$	2903: M_1 – 70.58%, M_2 – 6.24%, M_3 – 23.18%
1	$K_t = 5,\ K_l = 15,\ K_n = 10$	3595: M_3 – 100%
2	$K_t = 5,\ K_l = 15,\ K_n = 10$	2961: M_3 – 100%
1	$K_t = 15,\ K_l = 5,\ K_n = 10$	2599: M_3 – 100%
2	$K_t = 15,\ K_l = 5,\ K_n = 10$	3705: M_1 – 73.36%, M_3 – 26.64%

Опишем один из экспериментов по использованию случайных значений для порогов прочности (5). Были выбраны разломы, окружающие Филиппинскую плиту (всего их 10), и для них рассмотрены различные способы изменения параметров (5). Моделирование проводилось до 100 единиц модельного времени с шагом дискретизации 1/365. Посчитаны три варианта (во всех $a = 0.85$, $b = 0.7$): (1) $B(t_i) = 0.1$ для всех t_i; (2) $B(t_i)$ для каждого t_i есть значение нормально распределенной случайной величины $N(0.1; 0.005/3)$, причем, с учетом «правила 3σ», взятое из отрезка $[0.095; 0.105]$; (3) $B(t_i)$ выбирается как в варианте (2), но через 0.5 единиц модельного времени. Оказалось, что каталог (1) содержит 152142 события, каталог (2) — 14361, каталог (3) — 110105. Однако, детальный анализ показал, что общее число ячеек в критическом состоянии, формирующих модельные землетрясения, приблизительно совпадает, что свидетельствует об их перераспределении между соседними моментами дискретного времени. Кроме того, близость графиков повторяемости в диапазоне информативных магнитуд (Рис. 7) говорит об идентичности распределения землетрясений по магнитуде.

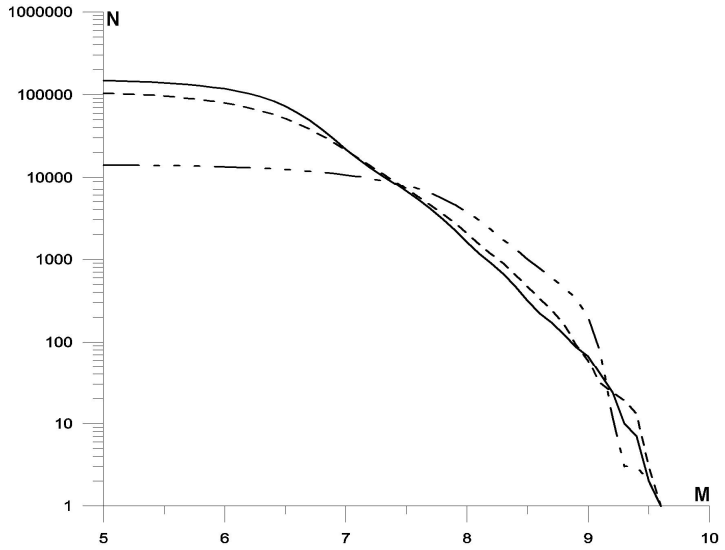


Рис. 7. Графики повторяемости, построенные по каталогам из вариантов (1) ─ сплошная линия, (2) ─ пунктирная линия и (3) ─ штриховая линия; N ─ аккумулированное число землетрясений, M ─ магнитуда

4. Некоторые аспекты распараллеливания

Вычислительные эксперименты показали, что сферическая блоковая модель является достаточно ресурсоемкой при расчетах на последовательных машинах, однако допускает эффективное применение параллельных технологий. Для распараллеливания алгоритма, реализующего блоковую модель, используется стандартная схема «мастер-рабочий» [5, 6, 15] с единым загрузочным MPI-модулем, почти равномерным распределением вычислительной нагрузки по всем процессорам и отсутствием обменов между рабочими процессорами. Поскольку подробное описание параллельного алгоритма и некоторых его характеристик (ускорения, эффективности и масштабируемости) приведено в [6], ограничимся кратким изложением основных аспектов. Блок-схема главной вычислительной процедуры представлена на Рис. 8. В начале своей работы программа назначает один из процессоров мастером. Затем загружается информация о блоковой структуре, выполняются предварительные вычисления (например, считается матрица A системы (3)). На каждом шаге дискретного времени наиболее трудоемкой процедурой является определение значений сил и неупругих смещений во всех ячейках структуры (так, в расчетном типовом варианте имеем около 200 000 ячеек на подошвах 15 блоков и около 3 500 000 – на сегментах 199 разломов). Поскольку эти вычисления могут быть проведены независимо друг от друга, их необходимо равномерно разделить между процессорами. Именно распределение ячеек на равные порции по процессорам и оптимальная организация межпроцессорных коммуникаций являются ключевыми моментами описываемого алгоритма распараллеливания. Обмен информацией реализован по схеме, отраженной на Рис. 8, где операции, выполняемые только мастером, помечены символом «M», только рабочими ─ символом «W» (остальные операции выполняются на всех процессорах). На каждом шаге мастер вычисляет новые значения смещений блоков, граничных блоков и подстилающей среды (что требует незначительного времени из-за малой размерности системы (3)), после чего рассылает их на рабочие процессоры. Пересчитанные значения сил, неупругих смешений и вектора b возвращаются мастеру, происходит переход к следующему шагу. Отметим, что при реализации межпроцессорных пересылок используются как коллективные обмены (например, при рассылке мастером решения системы (3)), так и обмены типа «точка-точка» (например, при передаче мастеру информации о модельных событиях).

При обработке ситуации, трактуемой как землетрясение, схема несколько усложняется, поскольку в этом случае мастер опрашивает все рабочие процессоры до тех пор, пока существуют ячейки сегментов, находящиеся в критическом состоянии; после чего мастер получает инфор-

мацию о происшедших событиях и записывает ее в файл специальной структуры. При такой организации время вычислительной работы на каждом рабочем процессоре оказывается значительно больше времени обмена данными с мастером, и за счет этого достигается довольно высокая полезная загрузка отдельного процессора.

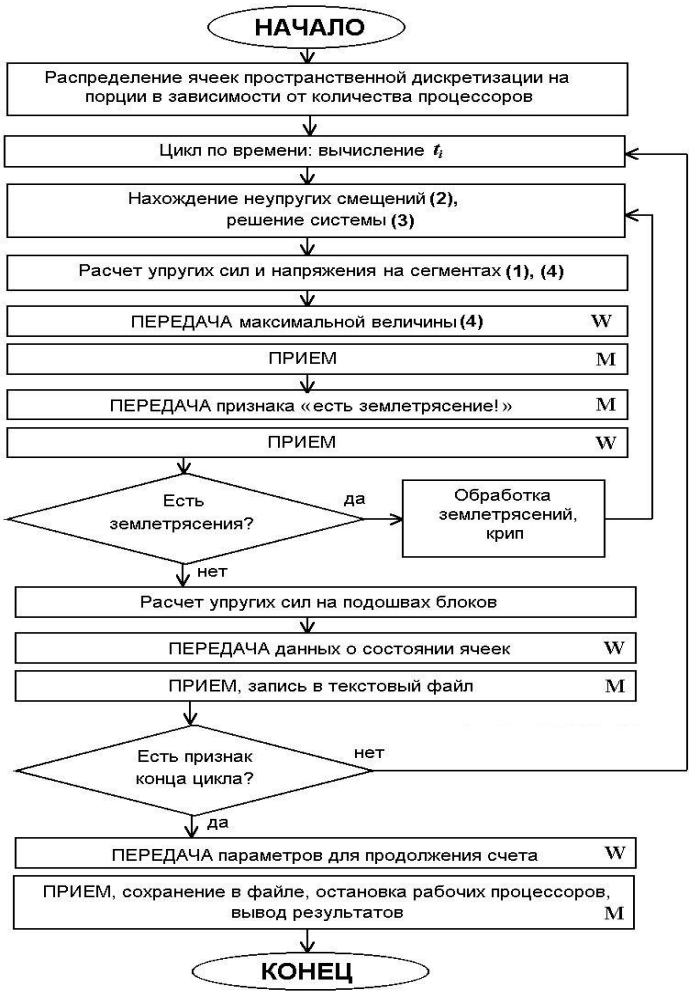


Рис. 8. Схема основной вычислительной процедуры

Однако, остаются две причины, негативно влияющие на качество распараллеливания: во-первых, равные порции ячеек обеспечивают равные объемы вычислений только при отсутствии землетрясений (а их местоположение непредсказуемо) и, во-вторых, мастер, поддерживая коммуникации и выполняя некоторые специфические операции, всегда работает дольше рабочих.

Для новой версии модели был проведен эксперимент по изучению качества распараллеливания в следующем смысле: анализировалось соотношение реального ускорения и максимального теоретически достижимого значения. Для вычисления последнего использовался известный закон Амдала [16]:

$$S_p^* = \frac{1}{\alpha + \dfrac{1-\alpha}{p}}, \tag{9}$$

где S_p^* – максимальное теоретически достижимое ускорение на p процессорах , α – доля от общего объема вычислений, которая может быть реализована только последовательно, т.е

$\alpha = \dfrac{\tau}{\tau + \pi}$, τ — время работы последовательной части программы, π — время работы параллельной части программы, причем распараллеливание полагается идеальным. Оценки времен τ и π можно получить исходя из замеров времен работы отдельных участков программы, запущенной на одном процессоре.

Для эксперимента был выбран наиболее «времяемкий» вариант из просчитанных (с большим числом модельных землетрясений) и рассмотрена 1 единица безразмерного времени с шагом дискретизации 1/365 (типовой расчетный вариант содержит 36500 или 73000 шагов, количество ячеек указано выше). Моделирование проводилось на введенном в 2010-м году в эксплуатацию в Институте математики и механики УрО РАН (г. Екатеринбург) гибридном вычислителе кластерного типа «Уран» (состоит из 208 вычислительных узлов, каждый из которых оснащен двумя четырехъядерными процессорами Intel Xeon, работающими на частоте 3.0 ГГц, и 16–32 Гб оперативной памяти; имеет пиковую производительность порядка 20 Тфлопс). Результаты тестирования представлены в таблице 2. Здесь T_p — время выполнения программы на многопроцессорной машине для p процессоров, T_1 — время работы последовательного алгоритма, $S_p = T_1 / T_p$ и $E_p = S_p / p$ — полученные ускорение и эффективность, S_p^* (см. (9)) и $E_p^* = S_p^* / p$ — максимальные теоретически достижимые величины. Для выбранного модельного варианта доля α оценивается величиной 0.006.

Таблица 2. Время счета, ускорение, эффективность и их оценки сверху для разного числа процессоров

p	T_p (сек)	S_p	S_p^*	E_p	E_p^*
1	8588.94	—	—	—	—
2	4337.48	1.98	1.99	0.990	0.995
4	2278.15	3.77	3.93	0.943	0.983
8	1427.74	6.02	7.68	0.753	0.96
16	697.45	12.31	14.68	0.769	0.918
32	379.58	22.63	26.98	0.707	0.843
64	195.98	43.82	46.44	0.685	0.726
96	141.71	60.61	61.15	0.631	0.637

Из таблицы следует, что эффективность распараллеливания достаточно высока, причем она, уменьшаясь с ростом числа задействованных процессоров, не падает ниже разумного уровня. Более того, при увеличении числа процессоров отклонение полученной эффективности от теоретически возможной уменьшается. Представляется, что резервом для улучшения качества распараллеливания является оптимизации передачи/записи информации о модельных землетрясениях.

5. Заключительные замечания

На данном этапе тщательный количественный анализ распределений реальных и модельных землетрясений представляется преждевременным, поскольку, во-первых, зарегистрированная сейсмичность мала на многих границах плит (ввиду недостаточной длины интервала наблюдения), а во-вторых, диапазон изменения модельной магнитуды не вполне соответствует реальному. Задача приведения магнитудных интервалов в соответствие требует дополнительного анализа в том числе и потому, что такое соответствие может зависеть от конкретного сейсмического региона. Введение в модель возможностей определения механизма события и учета случайных факторов призвано облегчить решение этой задачи. Поскольку проведенный анализ (в частности, результаты, представленные в настоящей статье) свидетельствует о достаточной степени адекватности модели в смысле воспроизведения важных закономерностей, обнаруженных в зарегистрированной сейсмичности эмпирическим путем, то результаты модели-

рования могут быть использованы для исследования характера сейсмического потока как на глобальном уровне, так и в конкретных регионах (например, с целью мониторинга глобального и регионального сейсмического риска). Отметим, что речь идет не о краткосрочном или среднесрочном прогнозировании отдельных землетрясений, а об изучении тенденций развития сейсмического процесса.

Литература

1. Gabrielov A.M., Newman W.I. Seismicity Modeling and Earthquake Prediction: a Review // Geophysical Monograph 83, IUGG. 1994. Vol. 18. P. 7-13.
2. Keilis-Borok V.I., Soloviev A.A. (Eds.) Nonlinear Dynamics of the Lithosphere and Earthquake Prediction. Springer, 2003. 337 p.
3. Габриэлов А.М., Кейлис-Борок В.И., Левшина Т.А., Шапошников В.А. Блоковая модель динамики литосферы // Математические методы в сейсмологии и геодинамике: Выч. Сейсмология. Вып. 19. М.: Наука, 1986. С. 168-178.
4. Алексеевская М.А. [и др.] Морфоструктурное районирование горных стран по формализованным признакам // Распознавание и спектральный анализ в сейсмологии: Выч. Сейсмология. Вып. 10. М.: Наука, 1977. С. 33-49.
5. Мельникова Л.А., Розенберг В.Л. Сферическая блоковая модель динамики и сейсмичности литосферы: различные модификации и вычислительные эксперименты // Труды ИММ УрО РАН. 2007. Т. 13. №3. С. 95-120.
6. Розенберг В.Л., Мельникова Л.А. Применение параллельных технологий к моделированию глобальной сейсмичности // Параллельные вычислительные технологии (ПаВТ-2010): Тр. междунар. конф. Челябинск, 2010. ISBN 978-5-696-03987-9. С. 311-321.
7. Rozenberg V.L., Sobolev P.O., Soloviev A.A., and Melnikova L.A. The Spherical Block Model: Dynamics of the Global System of Tectonic Plates and Seismicity // Pure Appl. Geophys. 2005. No. 162. P. 145-164.
8. Rozenberg V.L., Melnikova L.A., and Soloviev A.A. Spherical Block-and-Fault Model: Basic Principles, Different Modifications, and Simulation of Global Seismicity // Advanced School on Understanding and Prediction of Earthquakes and other Extreme Events in Complex Systems, Trieste, Italy, 26 September - 8 October, 2011. SMR.2265-05. 42 p.
9. Аки К., Ричардс П. Количественная сейсмология: теория и методы. М.: Мир, 1983. Т. 1-2. 880 с.
10. Wells D.L., Coppersmith K.L. New Empirical Relationships among Magnitude, Rupture Length, Rupture Width, Rupture Area, and Surface Displacement // Bull. Seism. Soc. of America. 1994. Vol. 84, No. 4. P. 974-1002.
11. Gripp A.E., Gordon R.G. Young Tracks of Hotspots and Current Plate Velocities // Geophys. J. Int. 2002. Vol. 150. P. 321-361.
12. Global Hypocenters Data Base, NEIC/USGS. Denver, CO. URL: http://earthquake.usgs.gov/regional/neic/ (дата обращения: 12.09.2011).
13. Mutter J.C. Seismic Images of Plate Boundaries // Scient. American. 1986. Vol. 254. P. 66-75.
14. World Stress Map Project. Potsdam, Germany. URL: http://dc-app3-14.gfz-potsdam.de/ (дата обращения: 12.09.2011).
15. Digas B.V., Melnikova L.A., and Rozenberg V.L. Application of Parallel Technologies to Modeling Lithosphere Dynamics and Seismicity // Lecture Notes in Computer Science 6068. Volume on Parallel Processing and Applied Mathematics. Berlin-Heidelberg, Springer, 2010. P. 340-349.
16. Гергель В.П. Теория и практика параллельных вычислений. М.: БИНОМ. Лаборатория знаний, 2007. 424 с.

Ускорение расчетов на графических процессорах при исследовании течения Стокса методом граничных элементов*

О.А. Солнышкина[1,2], Ю.А. Иткулова[1,2], Н.А. Гумеров[1,3]

Центр "Микро- и наномасштабная динамика дисперсных систем", Башкирский государственный университет, Уфа[1],
Институт механики, Уфимского научного центра РАН, Уфа[2]
Institute for Advanced Computer Studies, University of Maryland, USA[3]

В работе исследуется динамика двух вязких несмешивающихся жидкостей в неограниченной области и течение вязкой жидкости в канале сложной геометрии в трехмерной постановке. Численная методика основывается на методе граничных элементов, используются неструктурированные треугольные сетки. При увеличении масштаба задач возникает нехватка памяти вычислительной системы. Эта проблема решалась заменой прямого метода решения итерационным GMRES. Для ускорения расчетов разработан модуль матрично-векторного произведения «на лету» и распараллелен на графических процессорах (GPU). Проведено сравнение полученных численных результатов с аналитическими решениями. Представлены результаты по эффективности использования GPU.

1. Введение

Детальное исследование динамики дисперсных систем является актуальной проблемой современной науки и техники. Так, например, эмульсии широко используются в нефтяной промышленности. Существуют два типа эмульсий: эмульсии, закачиваемые в пористый пласт для увеличения нефтеотдачи, и эмульсии, образующиеся при нагнетании в пласт воды для вытеснения нефти, которая перемешивается с нефтью. Поэтому важно разработать эффективный численный аппарат, позволяющий детально исследовать поведение эмульсий, включая взаимодействие капель, их коалесценцию, и наблюдать многие эффекты на микроуровне в подобных системах «жидкость-жидкость».

В то же время на сегодняшний день нет однозначных представлений об их поведении при движении в пористых средах или микроканалах сложной геометрии, которые моделируют пористый пласт. Недавно был обнаружен эффект динамического запирания водонефтяных эмульсий в микроканалах, который заключается в том, что течение эмульсий со временем прекращается, несмотря на постоянно действующий перепад давления [9]. Этот эффект носит как положительные, так и отрицательные черты, которые влияют на величину дебита скважин, но его механизм до сих пор не изучен.

В работе исследуются трехмерные течения Стокса двух вязких несмешивающихся жидкостей в неограниченной области [6] и несжимаемой вязкой жидкости в канале переменного сечения сложной геометрии. Рассматриваемые задачи будут лежать в основе численного исследования эффекта динамического запирания эмульсий в микроканалах. Метод граничных элементов для Стоксовых течений изложен в [4] и успешно применялся для расчета течений в каналах [5] и взаимодействия капель и частиц в дисперсных течениях [8], в которых также можно найти обзор литературы.

Для решения поставленных задач используются новые эффективные подходы к численному моделированию трехмерных задач и современные информационные технологии. Численная методика основывается на методе граничных элементов, что уменьшает раз-

*Работа выполнена при поддержке Министерства по образованию и науки Российской Федерации, грант №11.G34.310040

мерность задачи на единицу. Используются неструктурированные треугольные сетки, точно описывающие границу области, что позволяет моделировать процессы в геометрически сложных объектах.

Программная реализация задач предусматривает выбор оптимальных алгоритмов в зависимости от количества узлов сетки. Для ускорения расчетов разработан модуль матрично-векторного произведения без хранения матрицы в памяти вычислительной системы, который используется в итерационном решателе GMRES, и распараллелен как на обычном многоядерном процессоре (CPU), так и на графических процессорах (GPU) с использованием технологии CUDA.

2. Трехмерное численное моделирование динамики капли в неограниченном потоке жидкости

2.1. Математическая модель

Постановка задачи об обтекании капли ньютоновской жидкости плотности ρ_2 и вязкости μ_2 потоком другой ньютоновской жидкости плотности ρ_1 и вязкости μ_1 приведена, например, в [6] и заключается в следующем. Рассматриваемые жидкости занимают объём V_2 и V_1 соответственно, S — граница раздела фаз. Гравитационная постоянная g и коэффициент поверхностного натяжения капли γ постоянны. На бесконечности скорость потока обтекающей каплю жидкости 1 равна $\mathbf{u}_\infty(\mathbf{x})$. При числах Рейнольдса Re_1, Re_2 значительно меньше 1, поведение потока жидкости 1, обтекающего каплю, и поведение жидкости 2 внутри капли описывается уравнениями Стокса

$$\nabla \cdot \boldsymbol{\sigma}_i = -\nabla p + \mu_i \nabla^2 \mathbf{u}_i = 0, \qquad \nabla \cdot \mathbf{u} = 0, \qquad i = 1, 2\,, \tag{1}$$

где $\boldsymbol{\sigma}$ — тензор напряжений; p — давление, которой включает гидродинамические компоненты.

Условия на границе двух жидкостей

$$\begin{aligned} &\mathbf{u}_1 = \mathbf{u}_2 = \mathbf{u}, \qquad \mathbf{f} = \boldsymbol{\sigma} \cdot \mathbf{n} = \mathbf{f_1} - \mathbf{f_2} = f\mathbf{n}, \\ &f = \gamma(\nabla \cdot \mathbf{n}) + (\rho_1 - \rho_2)(\mathbf{g} \cdot \mathbf{x}), \qquad \mathbf{x} \in S, \end{aligned} \tag{2}$$

где $\mathbf{f}$ — поверхностное напряжение; $\mathbf{n}$ — нормаль к поверхности, направленная в жидкость 1.

Условие на бесконечности

$$\mathbf{u}_1(\mathbf{x}) \;\to\; \mathbf{u}_\infty(\mathbf{x}), \qquad \mathbf{x} \;\to\; \infty. \tag{3}$$

Кинематическое условие, описывающее динамику капли

$$\frac{d\mathbf{x}}{dt} = (\mathbf{u} \cdot \mathbf{n})\mathbf{n}, \qquad \mathbf{x} \in S. \tag{4}$$

2.2. Численное моделирование

Задача решалась методом граничных элементов [10]. Суть метода состоит в преобразовании дифференциального уравнения в частных производных, описывающего поведение неизвестной функции внутри и на границе области, в интегральное уравнение, определяющее только граничные значения, и поиске численного решения этого уравнения. Если требуется найти значения потенциала во внутренних точках области, то их можно вычислить, используя известные решения на границе. Таким образом, размерность задачи уменьшается на единицу.

Для данной поверхности S скорость $\mathbf{u}$ в произвольной точке $\mathbf{y}$ может быть вычислена через граничные интегралы [8], [6]

$$\left.\begin{array}{c} \mathbf{y}\in V_1, \quad \mathbf{u}(\mathbf{y}) - 2\mathbf{u}_\infty(\mathbf{y}) \\ \mathbf{y}\in V_2, \quad \lambda\mathbf{u}(\mathbf{y}) \\ \mathbf{y}\in S, \quad \frac{1+\lambda}{2}\mathbf{u}(\mathbf{y}) - \mathbf{u}_\infty(\mathbf{y}) \end{array}\right\} = \tag{5}$$

$$= \int_S \left\{ -\frac{1}{\mu}\mathbf{G}(\mathbf{x},\mathbf{y})\cdot\mathbf{f}(\mathbf{x}) - (1-\lambda)\left[\mathbf{T}(\mathbf{x},\mathbf{y})\cdot\mathbf{n}(\mathbf{x})\right]\cdot\mathbf{u}(\mathbf{x}) \right\} dS(\mathbf{x}),$$

где $\mu = \mu_1$ — вязкость обтекающей жидкости; $\lambda = \mu_2/\mu_1$ — отношение вязкостей внутренней и внешней жидкостей; $\mathbf{G}$ — фундаментальное решение уравнения Стокса; $\mathbf{T}$ — тензор напряжений, определяются как

$$\begin{array}{c} \mathbf{G}(\mathbf{x},\mathbf{y}) = \frac{1}{8\pi}\left(\frac{\mathbf{I}}{r} + \frac{\mathbf{rr}}{r^3}\right), \quad \mathbf{T}(\mathbf{x},\mathbf{y}) = -\frac{3}{4\pi}\frac{\mathbf{rrr}}{r^5}, \\ \mathbf{r} = \mathbf{x} - \mathbf{y}, \quad r = |\mathbf{r}|. \end{array} \tag{6}$$

Поверхность S разбивается на N узлов $\mathbf{x}_i$, по которым мы строим квадратурные формулы граничных интегралов.

Используя метод вершинных коллокаций, последнее уравнение в граничных интегралах (5) в точках $\mathbf{y} = \mathbf{x}_j$ можно записать

$$\sum_{i=1}^{N}\left[\frac{1+\lambda}{2}\mathbf{I}_{ji} + (1-\lambda)\mathbf{I}_{ji}^{(T)}\right]\cdot\mathbf{u}_i = \mathbf{u}_{\infty,j} - \frac{1}{\mu}\sum_{i=1}^{N}\mathbf{I}_{ji}^{(G)} f_i, \quad j = 1,...,N, \tag{7}$$

$$\mathbf{I}_i^{(G)}(\mathbf{y}) = \int_{S_i}\mathbf{G}(\mathbf{x},\mathbf{y})\cdot\mathbf{n}(\mathbf{x})\,dS(\mathbf{x}), \tag{8}$$

$$\mathbf{I}_i^{(T)}(\mathbf{y}) = \int_{S_i}\mathbf{T}(\mathbf{x},\mathbf{y})\cdot\mathbf{n}(\mathbf{x})\,dS(\mathbf{x}), \tag{9}$$

где $\mathbf{I}_{ji}$ — единичная матрица; остальные пространственные функции записаны в рассматриваемой точке коллокации.

Вычисление сингулярных элементов производилось методом пробных функций на основе известных интегральных тождеств [4].

Решая систему линейных алгебраический уравнений (7), получаем компоненты скорости на границе.

Поверхность аппроксимируется сеткой с треугольными элементами. Начальная сетка создается стандартными процедурами триангуляции поверхности, в которых особое внимание уделяется качеству сетки, необходимому для аккуратного вычисления поверхностных интегралов. В процессе эволюции узлы сетки движутся согласно уравнению (4).

Наиболее трудоемкой процедурой является вычисление средней кривизны поверхности в каждом узле сетки. Средняя кривизна вычислялась двумя методами: контурных интегралов и установления параболоида [8]. Ошибка вычисления кривизны находилась из сравнения с аналитическим выражением для эллипсоида. Метод установления параболоида дает более точное значение средней кривизны с погрешностью не более 3%, но для его использования валентность всех узлов сетки должна быть не менее пяти.

2.3. Результаты тестирования для одиночной капли

На рис. 1 показана капля в начальный момент времени $t = 0$ и деформированная капля при $t = 10^{-5}$ с одинаковыми плотностями жидкостей $\rho_1 = \rho_2 = 10^3$, с вязкостью

обтекающей жидкости $\mu_1 = 10^{-3}$ и капли $\mu_2 = 2 \cdot 10^{-3}$, коэффициентом поверхностного натяжения $\gamma = 0,005$. Все переменные заданы в системе Си.

(a) Форма начальной капли (b) Форма деформированной капли

Рис. 1. Динамика капли

Из рисунков видно, что изначально эллипсоидальная капля под действием поверхностного натяжения приобретает сферическую форму. Число узлов сетки N варьировалось в пределах 600–3000.

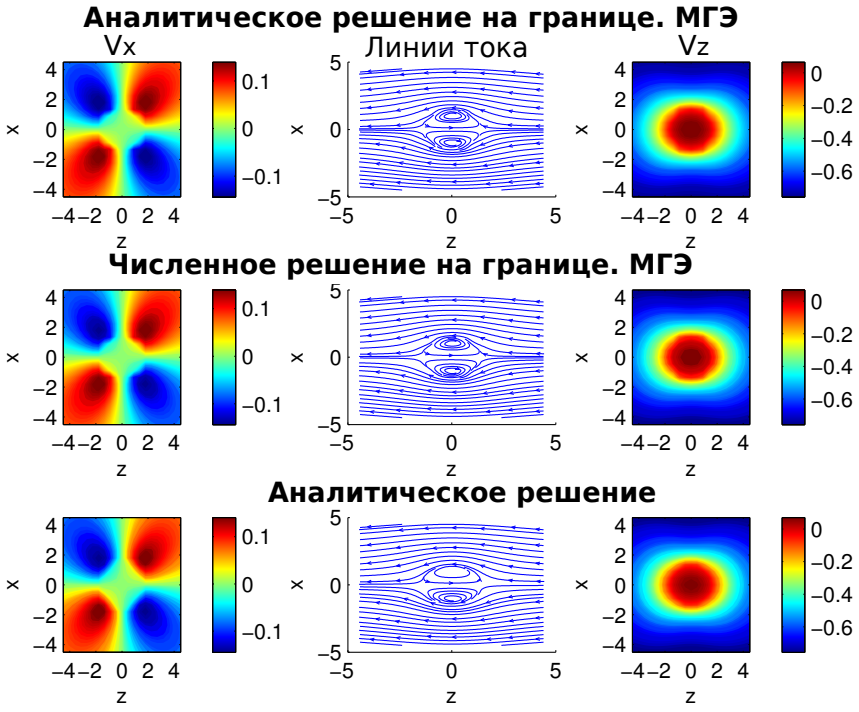


Рис. 2. Компоненты скоростей внутри и вне капли

Было проведено сравнение численного решения с аналитическим [11] в 3 этапа:

1. Компоненты скорости на границе капли, полученные аналитически, сравнивались с численными значениями, полученными методом граничных элементов. Погрешность составила 0,5 процента для $N = 2562$.

2. Компоненты скорости внутри и вне капли, полученные соответственно из первого и второго уравнений (5), используя для вычисления правой части точные значения $\mathbf{f}(\mathbf{x})$, $\mathbf{u}(\mathbf{x})$ сравниваются с аналитическими решениями.

3. Компоненты скорости внутри и вне капли, полученные соответственно из первого и второго уравнений (5), используя для вычисления правой части численные значения $\mathbf{f}(\mathbf{x})$, $\mathbf{u}(\mathbf{x})$ сравниваются с аналитическими решениями.

На рис. 2 показаны результаты, полученные на 2-ом и 3-ем этапах сравнения с аналитическим решением. Результаты сравнения показывают хорошее приближение численных расчетов к аналитическим с некоторой погрешностью, которая зависит от соотношения вязкостей рассматриваемых жидкостей λ и количества узлов дискретизации.

3. Трехмерное численное моделирование течения вязкой жидкости в канале переменного сечения

3.1. Математические модели периодического и непериодического потока

Рассматривалось течение вязкой несжимаемой жидкости с вязскостью μ в ограниченном канале переменного сечения с гладкой поверхностью S при малых числах Рейнольдса (рис. 3).

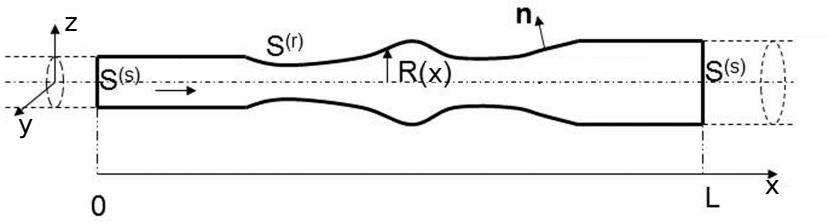


Рис. 3. Канал переменного сечения

Движение жидкости описывалось уравнениями Стокса (1).

Поверхность канала разбивалась на два подмножества: $S = S^{(s)} \cup S^{(r)}$, где $S^{(r)}$ — жесткая граница (боковая поверхность канала) и $S^{(s)}$ — мягкая граница (торцы канала). Задача решалась со следующими граничными условиями:

для непериодического потока

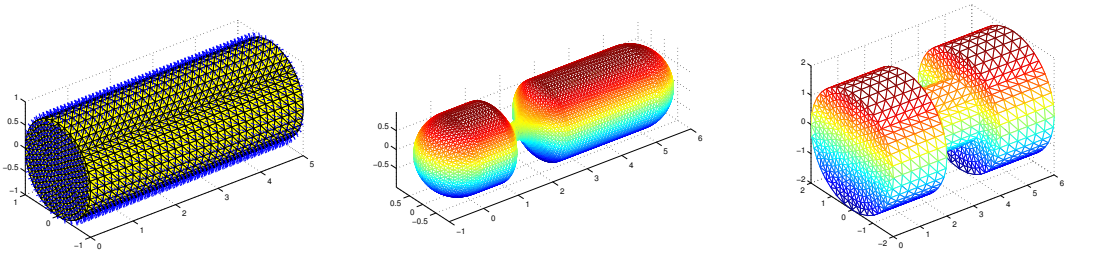
$$\mathbf{u}(\mathbf{x}) = \mathbf{u^0}(\mathbf{x}),\ \mathbf{x} \in S, \qquad \mathbf{f}(\mathbf{x_0}) = \mathbf{f^0}(\mathbf{x_0}),\ \mathbf{x_0} \in S^{(s)}. \tag{10}$$

для периодического потока

$$\begin{gathered} \mathbf{u}(\mathbf{x}) = 0,\ \mathbf{x} \in S^{(r)}, \quad \mathbf{f}(\mathbf{x_0}) = \mathbf{f^0}(\mathbf{x_0}),\ \mathbf{x_0} \in S^{(s)}, \\ \mathbf{u}\,(0,y,z) = \mathbf{u}\,(L,y,z)\,, \qquad \mathbf{f}\,(0,y,z) = -\mathbf{f}\,(L,y,z) + \mathbf{i}_x \Delta p. \end{gathered} \tag{11}$$

Обе постановки решались численно методом граничных элементов в нескольких модификациях.

Методы воплощены для произвольной трехмерной геометрии, в тестовой задаче аналитическая форма канала была осесимметричной.



(a) Цилиндр с нормалями (b) Сглаженный канал (c) Экспериментальный канал

Рис. 4. Качественная триангуляция канала

На рис. 4 слева представлена триангуляция цилиндра с нормалями для моделирования течения Пуазейля, в центре — модель канала для непериодического потока, справа — модель экспериментального канала для периодического потока.

3.2. Метод граничных элементов в нескольких модификациях

Уравнения Стокса (1) переписывались в граничных интегралах [10], [4]. Сначала определялся вектор нормального напряжения $\mathbf{f}$ на границе

$$\frac{1}{\mu}\int_S \mathbf{G}\left(\mathbf{y},\mathbf{x}\right)\cdot\mathbf{f}\left(\mathbf{x}\right)dS(\mathbf{x}) = \frac{1}{2}\mathbf{u^0}\left(\mathbf{y}\right) - \int_S \mathbf{K}\left(\mathbf{y},\mathbf{x}\right)\cdot\mathbf{u}^0\left(\mathbf{x}\right)dS(\mathbf{x}),\ \mathbf{y}\in S, \tag{12}$$

где $\mathbf{G}$, $\mathbf{T}$ находились по формуле (6).

Затем, значения скорости $\mathbf{u}$ в любой точке области $\mathbf{y}$ вычислялись через граничные интегралы

$$\mathbf{u}\left(\mathbf{y}\right) = \frac{1}{\mu}\int_S \mathbf{G}\left(\mathbf{y},\mathbf{x}\right)\cdot\mathbf{f}\left(\mathbf{x}\right)dS(\mathbf{x}) + \int_S \mathbf{K}\left(\mathbf{y},\mathbf{x}\right)\cdot\mathbf{u}^0\left(\mathbf{x}\right)dS(\mathbf{x}),\ \mathbf{y}\in V. \tag{13}$$

Для проверки достоверности полученных результатов, задача также была решена модифицированным методом граничных элементов, который заключается в следующем.

Нам известно точное решение для течения Пуазейля, которому задавалось возмущение

$$\mathbf{f} = \mathbf{f^0} + \mathbf{f}', \quad \mathbf{u} = \mathbf{u^0} + \mathbf{u}'. \tag{14}$$

Граничные условия (10) для возмущений $\mathbf{u}'$ и $\mathbf{f}'$ выглядят следующим образом

$$\begin{gathered}\mathbf{u}' = 0,\ \mathbf{x}\in S^{(s)}, \quad \mathbf{u}' = -\mathbf{u^0},\ \mathbf{x}\in S^{(r)},\\ \mathbf{f}'(\mathbf{x_0})\cdot\mathbf{n}(\mathbf{x_0}) = 0,\ \mathbf{x_0}\in S^{(s)}.\end{gathered} \tag{15}$$

Из интегральных уравнений (12)–(13), которым удовлетворяют возмущения $\mathbf{u}'$ и $\mathbf{f}'$, находились их значения. С учетом (14) определялся вектор нормального напряжения $\mathbf{f}$ на границе и компоненты скорости $\mathbf{u}$ в любой точке области.

Для периодического потока уравнения Стокса (1) переписывались в граничных интегралах

$$\frac{1}{2}\mathbf{u}\left(\mathbf{y}\right) - \int_S \mathbf{K}\left(\mathbf{y},\mathbf{x}\right)\cdot\mathbf{u}\left(\mathbf{x}\right)dS\left(\mathbf{x}\right) = \frac{1}{\mu}\int_S \mathbf{G}(\mathbf{y},\mathbf{x})\cdot\mathbf{f}\left(\mathbf{x}\right)dS\left(\mathbf{x}\right), \quad \mathbf{y}\in S. \tag{16}$$

Интегральные уравнения (16) переписываются в матричном виде

$$\mathbf{Au} = \mathbf{Bf}, \qquad \mathbf{A} = \frac{1}{2}\mathbf{I} - \mathbf{K}, \qquad \mathbf{B} = \frac{1}{\mu}\mathbf{G}. \tag{17}$$

Граничные условия будут выглядеть

$$\mathbf{u}_1 = 0, \quad \mathbf{u}_2 = \mathbf{u}_3 = \mathbf{u}_s, \quad \mathbf{f}_1 = \mathbf{f}_r, \quad \mathbf{f}_2 = \mathbf{f}_s, \quad \mathbf{f}_3 = -\mathbf{f}_s + \mathbf{f}_p, \quad \mathbf{f}_p = \mathbf{i}_x \Delta p\,, \tag{18}$$

где 1 — боковая поверхность; 2 — входное сечение; 3 — выходное сечение.

Из (16) получим систему линейных алгебраических уравнений относительно неизвестных $(\mathbf{u}_s, \mathbf{f}_s, \mathbf{f}_r)$

$$\begin{pmatrix} \mathbf{A}_{12} + \mathbf{A}_{13} & \mathbf{B}_{13} - \mathbf{B}_{12} & -\mathbf{B}_{11} \\ \mathbf{A}_{22} + \mathbf{A}_{23} & \mathbf{B}_{23} - \mathbf{B}_{22} & -\mathbf{B}_{21} \\ \mathbf{A}_{32} + \mathbf{A}_{33} & \mathbf{B}_{33} - \mathbf{B}_{32} & -\mathbf{B}_{31} \end{pmatrix} \begin{pmatrix} \mathbf{u}_s \\ \mathbf{f}_s \\ \mathbf{f}_r \end{pmatrix} = \begin{pmatrix} \mathbf{B}_{13}\mathbf{f}_p \\ \mathbf{B}_{23}\mathbf{f}_p \\ \mathbf{B}_{33}\mathbf{f}_p \end{pmatrix}. \tag{19}$$

Сингулярные интегралы рассчитывались, используя известные тестовые решения: постоянное и линейное течения.

3.3. Результаты тестирования для канала

Для тестирования программы использовался канал с входным и выходным сечением радиуса $R = 1$, радиусом сужения $r = 0,5$ и длинной $L = 5$.

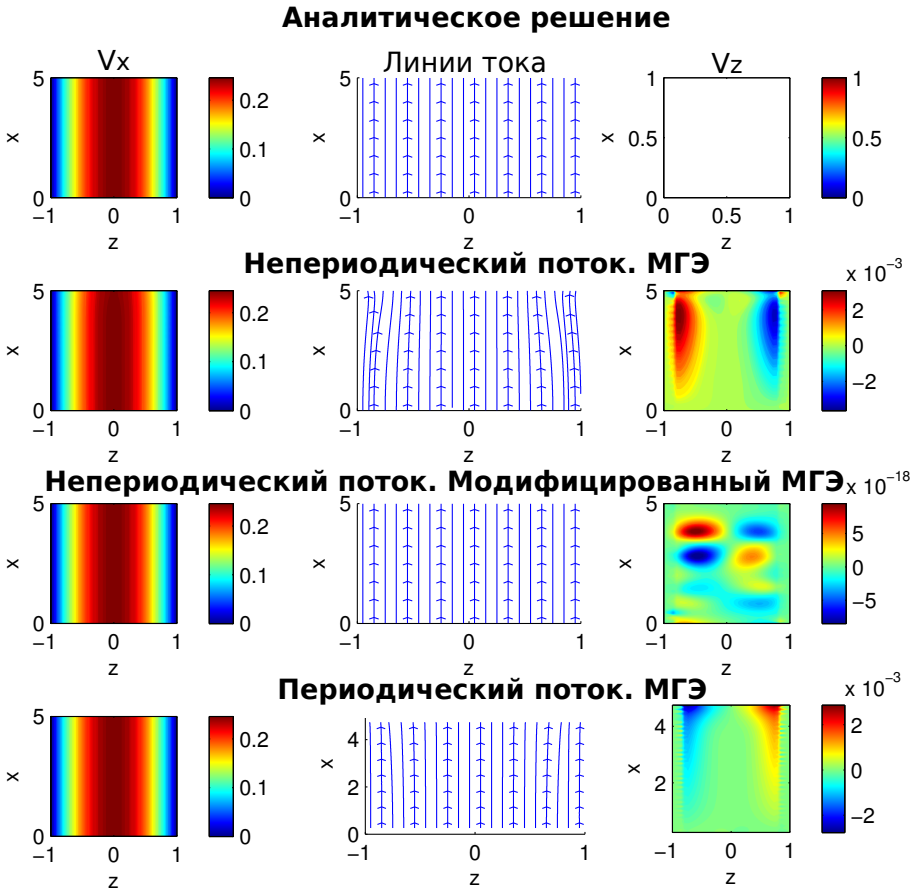


Рис. 5. Сравнение модификаций МГЭ с аналитическим решением

Проведено сравнение компонент вектора нормального напряжения $\mathbf{f}$ на границе методом граничных элементов в нескольких модификациях с аналитическим решения для непериодического течения Пуазейля. Погрешность вычисления неизвестных значений на границе составила порядка 0,7%. Для периодического потока погрешность компонент вектора нормального напряжения $\mathbf{f}$ на границе составила порядка 0,24%, компонент вектора скорости $\mathbf{u}$ около 0,9%.

На рис. 5 представлены поля скорости и линии тока, вычисленные в осевом сечении цилинда плоскостью Oxz, для течения Пуазейля с $N = 2598$ в методе вершинных колокаций и $N = 2704$ в методе колокаций с центром в грани элемента. Из рисунка видно, что численные расчеты хорошо приближаются к аналитическому решению. Модифицированный метод граничных элементов, особенно для компонент скорости вдоль оси Oz, дает более точные результаты, по сравнению с обычным методом граничных элементов.

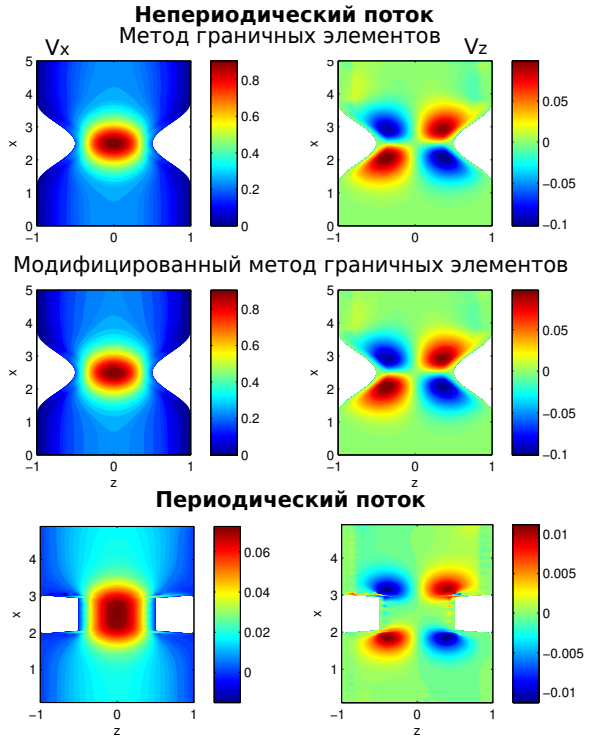


Рис. 6. Канал переменного сечения

Получены численные результаты для канала различного радиуса сужения методом граничых элементов в двух модификациях и проведено сравнение этих вариантов метода. В результате, относительная погрешность составила порядка 0,1-1,7% в зависимости от формы канала. По мере уменьшения радиуса сужения канала погрешность увеличивается и максимальное значение 1,7 % принимает на цилидрическом канале. На рис. 6 представлены поля скорости для каналов разной геометрии и радиусом сужения $r = 0,5$. Сглаженный канал состоит из 2598 вершин и 5192 граней, экспериментальный — из 1354 вершин и 2704 граней. Таким образом, количество расчетных узлов N находится в пределах 3000 узлов. На границе сингулярные интегралы вычисляются неточно, что ведет к большой погрешности вычисления поля скорости в точках, лежаших близко к стенке канала. Наблюдаемый эффект на входе для периодического потока качественно совпадает с экспериментальным [1] и приближенным аналитическим решением [2].

4. Ускорение расчетов на графических процессорах

При трехмерном численном моделировании физических процессов для областей со сложной геометрией, например течение эмульсии в микроканалах переменного сечения, необходимо построение сеток с большим количеством узлов. Решение подобных многомасштабных задач требует разработки и применения эффективных численных методов. Для сеток маленького размера при решении СЛАУ применялись прямые методы, но при увеличении масштаба задачи их использование затрудняется. Это связано с тем, что размер необходимой памяти пропорционален квадрату числа узлов сетки, также при их увеличении возрастает время вычислений. При использовании прямых методов, начиная с некоторого количества узлов, возникает нехватка памяти вычислительной системы.

Эту проблему можно решить используя итерационные методы решения, которые существенно снижают затраты памяти и времени. Наиболее эффективными и устойчивыми среди итерационных методов решения таких систем уравнений являются так называемые проекционные методы, и особенно тот их класс, который связан с проектированием на подпространства Крылова (например, GMRES). Эти методы обладают целым рядом достоинств: они устойчивы, допускают эффективное распараллеливание, работу с различными строчными (столбцовыми) форматами и предобуславливателями разных типов [7].

Для эффективной программной реализации итерационного метода необходимо решить две проблемы:

1. разработать подпрограмму, быстро умножающую матрицу на вектор;
2. ускорить сходимость метода с помощью предобуславливателя.

В рамках данной работы для решения проблем, связанных с использованием памяти, в среде Matlab был разработан программный модуль умножения матрицы на вектор без хранения матрицы в памяти системы («MV product on the fly»), который используется в GMRES при решении СЛАУ. Каждый элемент матриц G и K вычислялся по следующим формулам:

$$G^{ij}_{mn} = S_n \mathbf{G}(\mathbf{x_m} - \mathbf{x_n}) = \frac{1}{8pi} S_n \left(\frac{\delta_{ij}}{|\mathbf{X}_m - \mathbf{X}_n|} - \frac{(x^i_m - x^i_n)(x^j_m - x^j_n)}{|\mathbf{X}_m - \mathbf{X}_n|^3} \right),$$

$$K^{ij}_{mn} = S_n \mathbf{K}(\mathbf{x_m} - \mathbf{x_n}) = -\frac{3}{4pi} S_n \frac{(x^i_m - x^i_n)(x^j_m - x^j_n)}{|\mathbf{X}_m - \mathbf{X}_n|^5} \sum_{i=1}^{N} (x^k_m - x^k_n) n^k_n, \tag{20}$$

$$m, n = 1, \ldots, N, \;\; i, j, k = 1, 2, 3,$$

где N — количество узлов дискретизации области. Таким образом матрицы G и K имеют размер $3N \times 3N$ и $3N \times N$.

Применение модуля матрично-векторного произведения позволяет решить проблему ограничения по памяти вычислительной системы, но матричные вычисления являются вычислительно-трудоемкими, поэтому представляют собой классическую область применения параллельных вычислений. Для ускорения расчетов модуль был распараллелен на центральном многоядерном процессоре (CPU) средствами Matlab Parallel Computing Toolbox и с помощью программно-аппаратной технологии CUDA на графических процессорах (GPU) [3].

Использование графических процессоров для данной задачи обусловлено тем, что выполнение расчётов на GPU показывает отличные результаты в алгоритмах, использующих параллельную обработку данных (применение одной и той же последовательности математических операций к множеству данных). При этом лучшие результаты достигаются, если отношение числа арифметических инструкций к числу обращений к памяти достаточно велико.

Для многих методов матрично-векторных вычислений характерно наличие параллелизма по данным и в большинстве случаев распараллеливание таких операций сводится к разделению обрабатываемых матриц между процессорами используемой вычислительной системы. Наиболее общие способы разделения матриц состоят в разбиении данных на полосы (горизонтальные или вертикальные) или на прямоугольные фрагменты (блоки).

В рамках данной работы распараллеливание модуля «MV product on the fly» на GPU основывалось на разбиении матриц горизонтальными полосами на m частей так, что $N = m \times L$, где N — размерность матрицы, L — число строк матрицы в блоке, m — количество потоков на GPU.

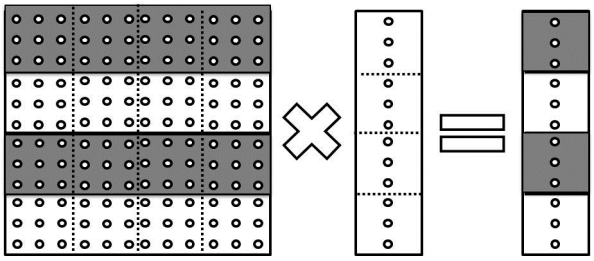


Рис. 7. Разделение данных и организация вычислений при выполнении параллельного модуля матрично-векторного произведения на GPU

На каждой итерации каждый из m потоков вычисляет свою часть вектора решения. То есть при умножении матрицы G размера $3N \times 3N$ на вектор b размера $3N \times 1$ на одной итерации поток вычисляет очередной свой блок матрицы размером 3×3 по формулам (20) и умножает полученный блок на соответствующую часть вектора b размером 3×1 и прибавляет результат к полученному на предыдущей итерации вектору (рис. 7). Каждый поток хранит часть результирующего вектора, которая по окончании вычислений копируется в global memory для получения полного вектора решения.

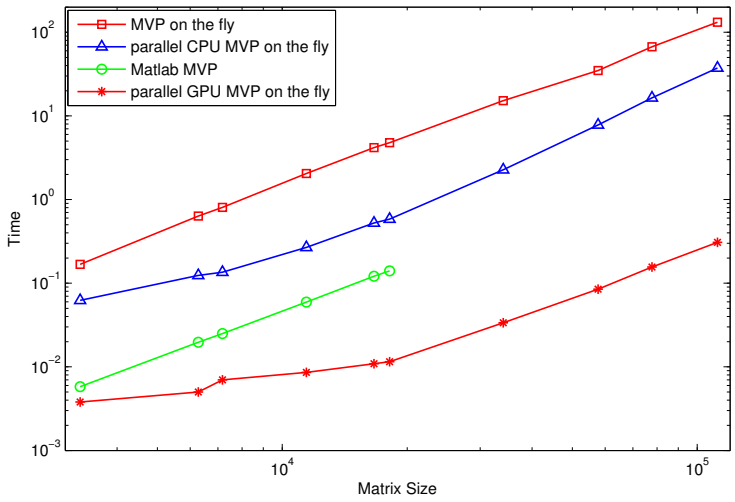


Рис. 8. Время вычисления матрично-векторного произведения

На рис. 8 показано сравнение времени выполнения модуля матрично-векторного произведения на одном ядре CPU (MVP on the fly), на 8 ядрах CPU (parallel MVP on the fly), на GPU (parallel GPU MVP on the fly) и встроенной функции умножения матрицы на вектор Matlab (Matlab MVP).

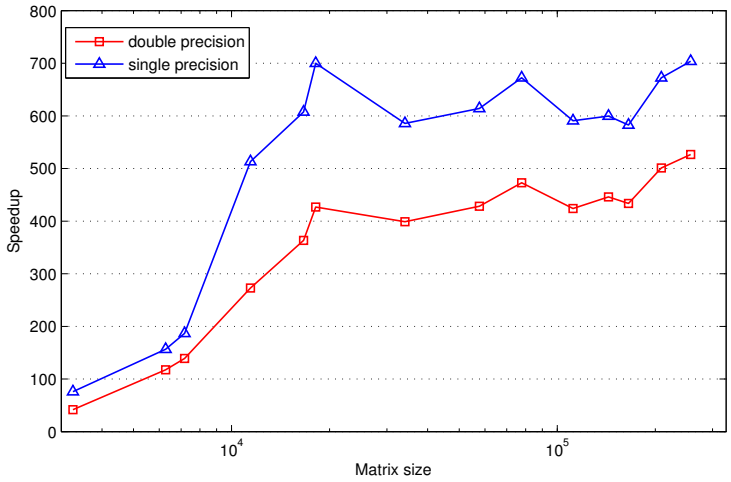


Рис. 9. Ускорение расчетов в зависимости от размера матрицы

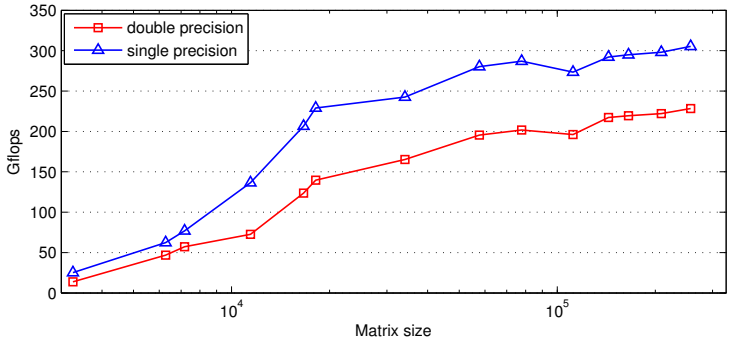


Рис. 10. Производительность в зависимости от размера матрицы

Все расчеты проводились на вычислительной системе с CPU Intel Xeon 5660, 2.8GHz, GPU NVIDIA Tesla C2050. При проведении вычислительных экспериментов было установлено, что для различных размеров задачи наилучшие результаты по времени достигаются при размере блока равным 256 потоков.

На рис. 9 приведено ускорение расчета на GPU по сравнению с CPU для операций с двойной и одинарной точностью. При расчетах на графической карте NVIDIA Tesla C2050 для количества узлов сетки N от 1 000 до 100 000 для уравнений Стокса получено ускорение до 520 раз для операций с двойной точностью и до 700 раз для операций с одинарной точностью.

При проведении тестовых расчетов достигнута следующая производительность: до 220 Gflops для операций с двойной точностью и 300 Gflops для операций с одинарной точностью (рис. 10). Учитывая, что пиковая производительность графической карты Tesla C2050 для чисел с плавающей точкой двойной точности составляет 515 Gflops, одинарной точности — 1,03 Tflops, получены хорошие результаты.

Результаты тестов на графической карте NVIDIA Tesla C2050 показали возможность решения граничных задач для уравнений Стокса размером до 100 000 элементов на одной рабочей станции.

5. Заключение

В среде Matlab разработаны программные продукты для исследования динамики одиночной капли вязкой жидкости в неограниченном потоке другой жидкости, течения Стокса в канале переменного сечения в нескольких модификациях и периодического течения Стокса в канале переменного сечения в трехмерном случае методом граничных элементов. При численном решении были использованы вспомогательные методы установления параболоида и метод контурных интегралов для вычисления средней кривизны области. Смоделирована качественная триангуляция канала произвольного переменного сечения для метода вершинных коллокаций и экспериментального канала для метода коллокаций в центрах граней. Сравнение результатов решения ряда тестовых задач с аналитическими решениями показало высокую эффективность выбранных методов и разработанных алгоритмов.

Для ускорения расчетов разработан модуль матрично-векторного произведения «на лету», то есть без хранения матрицы в памяти вычислительной системы, который используется в GMRES, и распараллелен на графических процессорах (GPU). Представлены результаты по эффективности использования GPU.

Решенные задачи являются первым и главным шагом для дальнейшего моделирования поведения водонефтяных эмульсий в микроканалах.

Литература

1. Boger D.V. Viscoelastic flows through contractions //J. Fluid Mech. 1987. Vol. 19. P. 157–182

2. Lubansky A.S., Boger D.V., Servais C. Burbidge A.S. Cooper-White J.J. An approximate solution to flow through a contraction for high Trouton ratio fluids// Non-Newtonian Fluid Mech. 2007. Vol. 144. P. 87–97

3. NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Version 3.2.2010.

4. Pozrikidis C. Boundary Integral and Singularity Methods for Linearized Viscous Flow. 1992(Cambridge University Press, Cambridge, MA).

5. Pozrikidis C. Creeping flow in two—dimensional channels// J. Fluid Mech. Vol. 180. 1987. P. 495–514.

6. Rallison J.M., Acrivos A. A numerical study of the deformation and burst of a viscous drop in an extensional flow // J. Fluid Mech. Vol. 89. part 1, 1978. P. 191–200.

7. Saad Y. Iterative Methods for Sparse Linear System. 2000, SIAM.

8. Zinchenko A.Z. and Davis R.H. An efficient algorithm for hydrodynamical interaction of many deformable drops // J. Comp. Phys. vol. 157, 2000. 539-587 p.

9. Ахметов А. Т., Саметов С. П. Особенности течения дисперсии из микрокапель воды в микроканалах// Письма в ЖТФ. Том 36, вып. 22, 2010. С. 21–28.

10. Бреббия К. Методы граничных элементов. Пер. с анг. М.: Мир, 1987. С. 524.

11. Хаппель Дж., Бреннер Г. Гидродинамика при малых числах рейнольдса. Пер. с анг. М.: Мир, 1976. С. 623.

Параллельная реализация трехмерной модели гидродинамики мелководных водоемов на супервычислительной системе

А.И. Сухинов, А.Е. Чистяков

Технологический институт Южного федерального университета в г. Таганроге

Работа посвящена разработке математической модели для расчета полей скоростей применительно к прибрежным системам и мелководным водоемам, таким как Азовское море. Отличительными особенностями разрабатываемых алгоритмов являются: высокая производительность, достоверность и точность получаемых результатов. В работе представлены параллельный метод и численные результаты моделирования гидродинамических процессов в Азовском море на основе программного комплекса «Azov3D», созданного в ТТИ ЮФУ. Для построенного параллельного алгоритма выполнены теоретические и экспериментальные оценки ускорения и эффективности.

1. Введение

Исследования Азовского моря проводятся уже более ста лет. За это время накоплен большой объем натурных данных обо всех компонентах морской экосистемы, выявлены многие закономерности их функционирования и взаимодействия. Наряду с натурными и экспериментальными исследованиями, одной из основных задач является архивация и систематизация первичной информации, накопленной исследователями ТТИ ЮФУ. Такой подход к изучению гидродинамических и гидробиологических явлений предусматривает формирование базы данных и применение математического моделирования морских процессов.

В ходе экспедиционных исследований в 2001 была обнаружена зона анаэробного заражения и наблюдалась массовая гибель ихтиофауны в восточной части Азовского моря[1]. Для реконструкции сценария экологической катастрофы, а также для моделирования возможных вариантов биологической реабилитации водоема был создан ряд высокоточных математических моделей гидрофизических процессов в мелководных водоемах. Данные модели описывают движение водной среды с учетом следующих факторов: ветровые течения и трение о дно, стоки рек, испарение, сила Кориолиса, турбулентный обмен, сгонно-нагонные явления, сложная геометрия дна и береговой линии. Другое актуальное назначение высокоточных математических моделей гидрофизических процессов связано со своевременным предсказанием различных природных катаклизмов, связанных с изменением уровня воды: затоплением прибрежных районов, обмелением судоходных каналов и др.

2. Математическая модель гидродинамики мелководных водоемов

Азовское море расположено на юго–западе России и имеет максимальную протяженность с севера на юг - 250 км., с запада на восток - 350 км., а максимальная глубина равна 14 м. Модель предназначена для оценки и прогнозирования состояния водной среды водоема. Математическое описание основано на выделении осредненных составляющих параметров течения среды (скорости, давления). Уравнения модели движения жидкости рассматриваются в прямоугольной области геоинформационной системы Азовского моря. Оси Ox и Oy имеют горизонтальные направления с запада на восток и с севера на юг соответственно. Ось Oz направлена вертикально вниз.

Исходными уравнениями гидродинамики мелководных водоемов являются[2-3]:

– уравнение движения (Навье – Стокса):

$$u'_t + uu'_x + vu'_y + wu'_z = -\frac{1}{\rho}p'_x + \left(\mu u'_x\right)'_x + \left(\mu u'_y\right)'_y + \left(\nu u'_z\right)'_z + 2\Omega(v\sin\theta - w\cos\theta),$$

$$v'_t + uv'_x + vv'_y + wv'_z = -\frac{1}{\rho}p'_y + \left(\mu v'_x\right)'_x + \left(\mu v'_y\right)'_y + \left(\nu v'_z\right)'_z - 2\Omega u\sin\theta, \qquad (1)$$

$$w'_t + uw'_x + vw'_y + ww'_z = -\frac{1}{\rho}p'_z + \left(\mu w'_x\right)'_x + \left(\mu w'_y\right)'_y + \left(\nu w'_z\right)'_z + 2\Omega u\cos\theta;$$

– уравнение неразрывности в случае переменной плотности:

$$\rho'_t + \left(\rho u\right)'_x + \left(\rho v\right)'_y + \left(\rho w\right)'_z = 0, \qquad (2)$$

где $V = \{u, v, w\}$ – компоненты вектора скорости, p – превышение давления над гидростатическим давлением невозмущенной жидкости, ρ – плотность, Ω – угловая скорость вращения земли, θ – угол между вектором угловой скорости и вертикалью, μ,ν – горизонтальная и вертикальная составляющие коэффициента турбулентного обмена.

Система уравнений (1) – (2) рассматривается при следующих граничных условиях:

– на входе (устье рек Дон и Кубань) –

$$u(x,y,z,t) = u(t), \quad v(x,y,z,t) = v(t), \quad p'_n(x,y,z,t) = 0, \quad V'_n(x,y,z,t) = 0,$$

– боковая граница (берег и дно) –

$$\rho\mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho\mu(v')_n(x,y,z,t) = -\tau_y(t),\ V_n(x,y,z,t) = 0,\ p'_n(x,y,z,t) = 0,$$

– верхняя граница –

$$\rho\mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho\mu(v')_n(x,y,z,t) = -\tau_y(t), \qquad (3)$$

$$w(x,y,t) = -\omega - p'_t / \rho g,\ p'_n(x,y,t) = 0,$$

– на выходе (Керченский пролив) –

$$p'_n(x,y,z,t) = 0, \quad V'_n(x,y,z,t) = 0,$$

где ω - интенсивность испарения жидкости, τ_x, τ_y – составляющие тангенциального напряжения (закон Ван-Дорна).

Составляющие тангенциального напряжения для свободной поверхности:

$$\tau_x = \rho_a C_p\left(|\vec{w}|\right) w_x |\vec{w}|,\ \tau_y = \rho_a C_p\left(|\vec{w}|\right) w_y |\vec{w}|,$$

где $\vec{w}$ – вектор скорости ветра относительно воды, ρ_a – плотность атмосферы,

$C_p(x) = \begin{cases} 0.0088, & x < 6,6\ м/с \\ 0.0026, & x \geq 6,6\ м/с \end{cases}$ — безразмерный коэффициент.

Составляющие тангенциального напряжения для дна с учетом введенных обозначений могут быть записаны следующим образом:

$$\tau_x = \rho_v C_p\left(|V|\right) u|V|,\ \tau_y = \rho_v C_p\left(|V|\right) v|V|,$$

где ρ_v – плотность донных отложений.

Рассмотренная ниже аппроксимация позволяет на основании измеренных пульсаций скоростей строить коэффициент вертикального турбулентного обмена, неоднородный по глубине[4]:

$$\nu = C_s^2 \Delta^2 \frac{1}{2}\sqrt{\left(\frac{\partial\overline{U}}{\partial z}\right)^2 + \left(\frac{\partial\overline{V}}{\partial z}\right)^2}, \qquad (4)$$

где $\overline{U}, \overline{V}$ – осредненные по времени пульсации горизонтальных компонент скорости, Δ - характерный масштаб сетки, C_s - безразмерная эмпирическая константа, значение которой обычно определяется на основе расчета процесса затухания однородной изотропной турбулентности.

3. Дискретная модель гидродинамики мелководных водоемов

Расчетная область вписана в параллелепипед. Для численной реализации дискретной математической модели поставленной задачи гидродинамики вводится равномерная сетка:

$$\overline{w}_h = \left\{ t^n = n\tau, x_i = ih_x, y_j = jh_y, z_k = kh_z;\ n = \overline{0..N_t}, i = \overline{0..N_x}, j = \overline{0..N_y}, k = \overline{0..N_z};\right.$$

$$\left. N_t\tau = T, N_x h_x = l_x, N_y h_y = l_y, N_z h_z = l_z \right\},$$

где τ – шаг по времени, h_x, h_y h_z – шаги по пространству, N_t – количество временных слоев, T – верхняя граница по временной координате, N_x, N_y N_z – количество узлов по пространственным координатам, l_x, l_y l_z – границы по параллелепипеда в направлении осей Ox, Oy и Oz соответственно.

Для решения задачи гидродинамики использовался метод поправки к давлению[5]. Вариант данного метода в случае переменной плотности примет вид:

$$\begin{aligned}
&\frac{\tilde{u}-u}{\tau} + u\overline{u}'_x + v\overline{u}'_y + w\overline{u}'_z = \left(\mu\overline{u}'_x\right)'_x + \left(\mu\overline{u}'_y\right)'_y + \left(\nu\overline{u}'_z\right)'_z + 2\Omega(v\sin\theta - w\cos\theta),\\
&\frac{\tilde{v}-v}{\tau} + u\overline{v}'_x + v\overline{v}'_y + w\overline{v}'_z = \left(\mu\overline{v}'_x\right)'_x + \left(\mu\overline{v}'_y\right)'_y + \left(\nu\overline{v}'_z\right)'_z - 2\Omega u\sin\theta,\\
&\frac{\tilde{w}-w}{\tau} + u\overline{w}'_x + v\overline{w}'_y + w\overline{w}'_z = \left(\mu\overline{w}'_x\right)'_x + \left(\mu\overline{w}'_y\right)'_y + \left(\nu\overline{w}'_z\right)'_z + 2\Omega u\cos\theta,
\end{aligned} \qquad (5)$$

$$\begin{aligned}
&p''_{xx} + p''_{yy} + p''_{zz} = \frac{\hat{\rho}-\rho}{\tau^2} + \frac{\left(\hat{\rho}\tilde{u}\right)'_x}{\tau} + \frac{\left(\hat{\rho}\tilde{v}\right)'_y}{\tau} + \frac{\left(\hat{\rho}\tilde{w}\right)'_z}{\tau},\\
&\frac{\hat{u}-\tilde{u}}{\tau} = -\frac{1}{\rho}\hat{p}'_x,\ \frac{\hat{v}-\tilde{v}}{\tau} = -\frac{1}{\rho}\hat{p}'_y,\ \frac{\hat{w}-\tilde{w}}{\tau} = -\frac{1}{\rho}\hat{p}'_z,
\end{aligned}$$

где $V = \{u, v, w\}$ – компоненты вектора скорости, $\{\hat{u}, \hat{v}, \hat{w}\}, \{\tilde{u}, \tilde{v}, \tilde{w}\}$ – компоненты полей вектора скорости на «новом» и промежуточном временных слоях соответственно, $\overline{u} = (\tilde{u} + u)/2$, $\hat{\rho}$ и ρ - распределение плотности водной среды на новом и предыдущем временных слоях соответственно.

При построении дискретных математических моделей гидродинамики учитывалась «заполненность» контрольных ячеек, что позволяет повысить реальную точность решения в случае сложной геометрии исследуемой области за счет улучшения аппроксимации границы.

Через $o_{i,j,k}$ обозначена «заполненность» ячейки (i, j, k). Степень «заполненности» ячейки определяется давлением столба жидкости внутри данной ячейки. Если среднее давление в узлах, которые относятся к вершинам рассматриваемой ячейки, больше давления столба жидкости внутри ячейки, то ячейка считается заполненной полностью $(o_{i,j,k} = 1)$. В общем случае «заполненность» ячеек можно вычислить по следующей формуле:

$$o_{i,j} = \frac{P_{i,j,k} + P_{i-1,j,k} + P_{i,j-1,k} + P_{i-1,j-1,k}}{4\rho g h_z}, \qquad (6)$$

где $P = p + \rho g z$ – давление.

Вводятся коэффициенты q_0, q_1, q_2, q_3, q_4, q_5, q_6, описывающие «заполненность» областей, находящихся в окрестности ячейки (контрольных областей). Значение q_0 характеризует «заполненность» области D_0: $\{x \in (x_{i-1}, x_{i+1}),\ y \in (y_{j-1}, y_{j+1}),\ z \in (z_{k-1}, z_{k+1})\}$, q_1 – D_1: $\{x \in (x_i, x_{i+1}),\ y \in (y_{j-1}, y_{j+1}),\ z \in (z_{k-1}, z_{k+1})\}$, q_2 – D_2: $\{x \in (x_{i-1}, x_i),\ y \in (y_{j-1}, y_{j+1}),\ z \in (z_{k-1}, z_{k+1})\}$, q_3 – D_3: $\{x \in (x_{i-1}, x_{i+1}),\ y \in (y_j, y_{j+1}),\ z \in (z_{k-1}, z_{k+1})\}$, q_4 – D_4: $\{x \in (x_{i-1}, x_{i+1}),\ y \in (y_{j-1}, y_j),\ z \in (z_{k-1}, z_{k+1})\}$, q_5 – D_5: $\{x \in (x_{i-1}, x_{i+1}),\ y \in (y_{j-1}, y_{j+1}),\ z \in (z_k, z_{k+1})\}$, q_6 – D_6:

$\{ x \in (x_{i-1}, x_{i+1}),\ y \in (y_{j-1}, y_{j+1}),\ z \in (z_{k-1}, z_k) \}$. Заполненные части областей D_m будем называть Ω_m, где $m = \overline{0..6}$. В соответствии с этим коэффициенты q_m можно вычислить по формулам:

$$(q_m)_{i,j,k} = \frac{S_{\Omega_m}}{S_{D_m}},\ (q_1)_{i,j,k} = \frac{o_{i+1,j,k} + o_{i+1,j+1,k} + o_{i+1,j,k+1} + o_{i+1,j+1,k+1}}{4},$$

$$(q_2)_{i,j,k} = \frac{o_{i,j,k} + o_{i,j+1,k} + o_{i,j,k+1} + o_{i,j+1,k+1}}{4},\ (q_3)_{i,j,k} = \frac{o_{i+1,j+1,k} + o_{i,j+1,k} + o_{i+1,j+1,k+1} + o_{i,j+1,k+1}}{4},$$

$$(q_4)_{i,j,k} = \frac{o_{i,j,k} + o_{i+1,j,k} + o_{i,j,k+1} + o_{i+1,j,k+1}}{4},\ (q_5)_{i,j,k} = \frac{o_{i,j,k+1} + o_{i+1,j,k+1} + o_{i+1,j+1,k+1} + o_{i,j+1,k+1}}{4},$$

$$(q_6)_{i,j,k} = \frac{o_{i,j,k} + o_{i+1,j,k} + o_{i+1,j+1,k} + o_{i,j+1,k}}{4},\ (q_0)_{i,j,k} = \frac{1}{2}\left((q_1)_{i,j,k} + (q_2)_{i,j,k}\right).$$

В случае граничных условий третьего рода

$$c'_n(x,y,t) = \alpha_n c + \beta_n$$

дискретные аналоги операторов конвективного uc'_x и диффузионного $(\mu c'_x)'_x$ переноса, полученные при помощи интегро-интерполяционного метода[6], учитывающие частичную «заполненность» ячеек, могут быть записаны в следующем виде[7]:

$$uc'_x \Box (q_1)_{i,j} u_{i+1/2,j} \frac{c_{i+1,j} - c_{i,j}}{2h_x} + (q_2)_{i,j} u_{i-1/2,j} \frac{c_{i,j} - c_{i-1,j}}{2h_x},$$

$$(\mu c'_x)'_x \Box (q_1)_{i,j} \mu_{i+1/2,j} \frac{c_{i+1,j} - c_{i,j}}{h_x^2} - (q_2)_{i,j} \mu_{i-1/2,j} \frac{c_{i,j} - c_{i-1,j}}{h_x^2} - \left|(q_1)_{i,j} - (q_2)_{i,j}\right| \mu_{i,j} \frac{\alpha_x c_{i,j} + \beta_x}{h_x}.$$

Аналогичным образом запишутся аппроксимации по оставшимся координатным направлениям.

Погрешность аппроксимации математической модели равна $O\left(\tau + \|h\|^2\right)$, где $\|h\| = \sqrt{h_x^2 + h_y^2 + h_z^2}$. Доказано сохранение потока на дискретном уровне разработанной гидродинамической модели, а также отсутствие неконсервативных диссипативных слагаемых, полученных в результате дискретизации системы уравнений. Достаточное условие устойчивости и монотонности разработанной модели определяется на основе принципа максимума [6] при ограничениях на шаг по пространственным координатам:

$h_x < |2\mu / u|$, $h_y < |2\mu / v|$, $h_z < |2\nu / w|$ или $\mathrm{Re} \le 2N$,

где $\mathrm{Re} = |V| \cdot l / \mu$ – числа Рейнольдса, l – характерный размер области, $N = \max\{N_x, N_y, N_z\}$.

Дискретные аналоги системы уравнений (5) решаются адаптивным модифицированным попеременно – треугольным методом вариационного типа[8-9].

4. Метод решения сеточных уравнений

Полученные сеточные уравнения можно записать в матричном виде:

$$Ax = f, \qquad (7)$$

где A – линейный, положительно определенный оператор ($A > 0$). Для нахождения решения задачи (7) будем использовать неявный итерационный процесс

$$B\frac{x^{m+1} - x^m}{\tau_{m+1}} + Ax^m = f. \qquad (8)$$

В уравнении (8) m – номер итерации, $\tau > 0$ –итерационный параметр, а B – некоторый обратимый оператор, который называется предобуславливателем или стабилизатором. Обращение оператора B в (8) должно быть существенно проще, чем непосредственное обращение исходного оператора A в (7). При построении B исходили из аддитивного представления оператора A_0 - симметричной части оператора А:

$$A_0 = R_1 + R_2,\ \ R_1 = R_2^*, \tag{9}$$

где $A = A_0 + A_1,\ \ A_0 = A_0^*,\ \ A_1 = -A_1^*$.

Оператор предобуславливатель запишется в следующем виде:

$$B = (D + \omega R_1)D^{-1}(D + \omega R_2),\ D = D^* > 0,\ \ \omega > 0, \tag{10}$$

где D – некоторый оператор.

Соотношения (9)-(10) задают модифицированный попеременно-треугольный метод (МПТМ) решения задачи, если определены операторы R_1, R_2 и указаны способы определения параметров τ_{m+1}, ω и оператора D.

Алгоритм адаптивного модифицированного попеременно – треугольного метода минимальных поправок для расчета сеточных уравнений с несамосопряженным оператором имеет вид:

$$r^m = Ax^m - f,\ \ B(\omega_m)w^m = r^m,\ \ \tilde{\omega}_m = \sqrt{\frac{\left(Dw^m, w^m\right)}{\left(D^{-1}R_2 w^m, R_2 w^m\right)}}, \tag{11}$$

$$s_m{}^2 = 1 - \frac{\left(A_0 w^m, w^m\right)^2}{\left(B^{-1}A_0 w^m, A_0 w^m\right)\left(Bw^m, w^m\right)},\ \ k_m = \frac{\left(B^{-1}A_1 w^m, A_1 w^m\right)}{\left(B^{-1}A_0 w^m, A_0 w^m\right)},\ \ \theta_m = \frac{1 - \sqrt{\frac{s_m{}^2 k_m}{(1+k_m)}}}{1 + k_m\left(1 - s_m{}^2\right)},$$

$$\tau_{m+1} = \theta_m \frac{\left(A_0 w^m, w^m\right)}{\left(B^{-1}A_0 w^m, A_0 w^m\right)},\ \ x^{m+1} = x^m - \tau_{m+1} w^m,\ \ \omega_{m+1} = \tilde{\omega}_m,$$

где r^m - вектор невязки, w^m - вектор поправки, в качестве оператора D используется диагональная часть оператора A.

5. Многопроцессорная вычислительная система ТТИ ЮФУ

Пиковая производительность МВС составляет 18.8 TFlops. МВС включает в себя 8 компьютерных стоек (рис.1). Вычислительное поле многопроцессорной вычислительной системы (МВС) ТТИ ЮФУ построено на базе инфраструктуры HP BladeSystem c-class с интегрированными коммуникационными модулями, системами электропитания и охлаждения. В качестве вычислительных узлов используется 128 однотипных 16-ядерных Blade-серверов HP ProLiant BL685c, каждый из которых оснащен четырьмя 4-ядерными процессорами AMD Opteron 8356 2.3GHz и оперативной памятью в объеме 32ГБ.

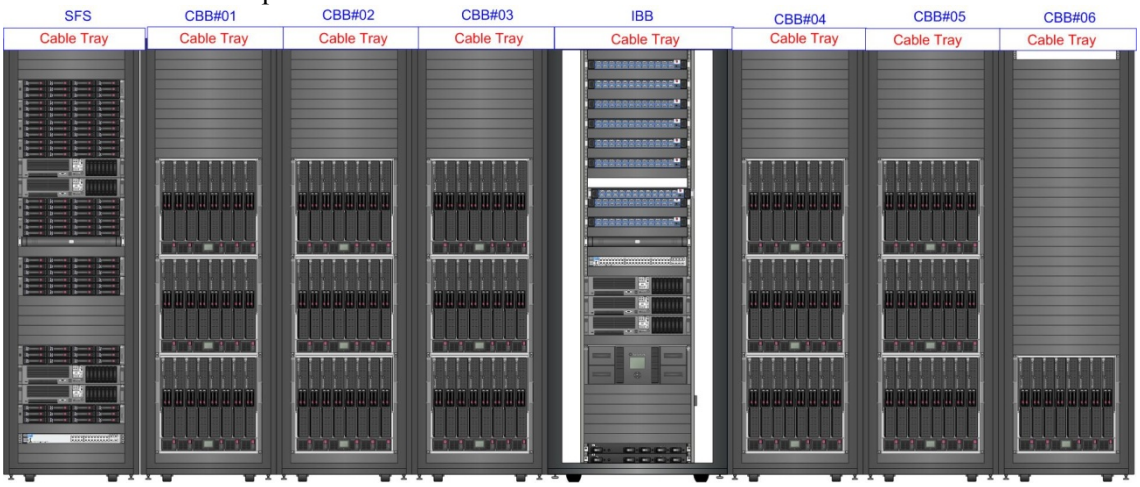


Рис. 1 Многопроцессорная Вычислительная Система

Общее количество вычислительных ядер в комплексе – 2048, суммарный объем оперативной памяти – 4 ТВ. Для управления МВС используется 3 управляющих сервера HP ProLiant DL385G5. Для задач резервного копирования используется библиотека MSL4048.

6. Параллельный вариант алгоритма решения сеточных уравнений

Идея параллельного алгоритма метода решения сеточных уравнений заключается в следующем. После разбиения исходной расчетной области на части по двум координатным направлениям каждый процессор получает свою расчетную область, как показано на рис.2, при этом смежные области перекрываются двумя слоями узлов по направлению, перпендикулярному плоскости разбиения.

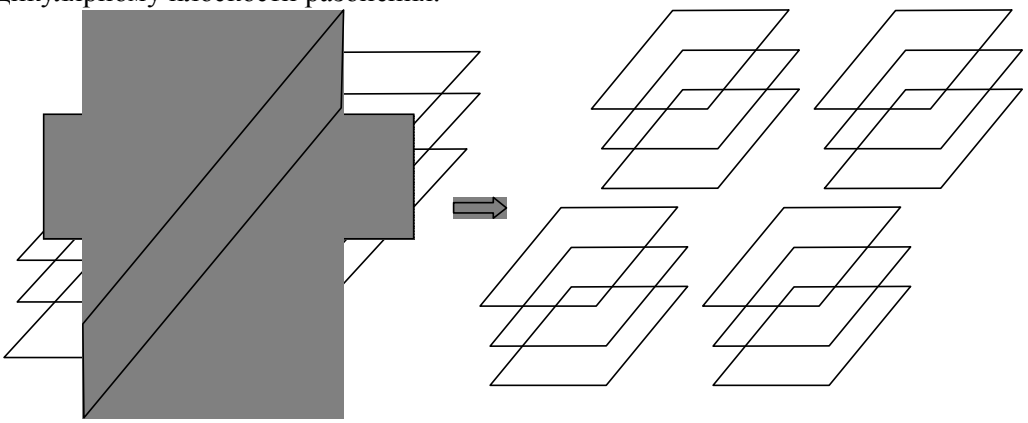


Рис 2. Декомпозиция области.

После того как каждый процессор получит информацию для своей части области, рассчитывается вектор невязки и его равномерная норма. Затем каждый процессор определяет максимальный по модулю элемент вектора невязки и передает его значение каждому процессору. Теперь для вычисления равномерной нормы вектора невязки достаточно на каждом процессоре найти максимальный элемент.

Рассмотрим параллельный алгоритм расчета вектора поправки:

$$(D+\omega_m R_1)D^{-1}(D+\omega_m R_2)w^m = r^m,$$

где R_1 – нижне-треугольная матрица, а R_2 – верхне-треугольная матрица. Для этого решим последовательно системы:

$$(D+\omega_m R_1)y^m = r^m,\ (D+\omega_m R_2)w^m = Dy^m.$$

Вначале вычисляется вектор y^m, при этом расчет начинается в левом нижнем углу. Затем из правого верхнего угла начинается вычисление вектора поправки w^m. Схема расчета вектора y^m изображена на рис. 3.

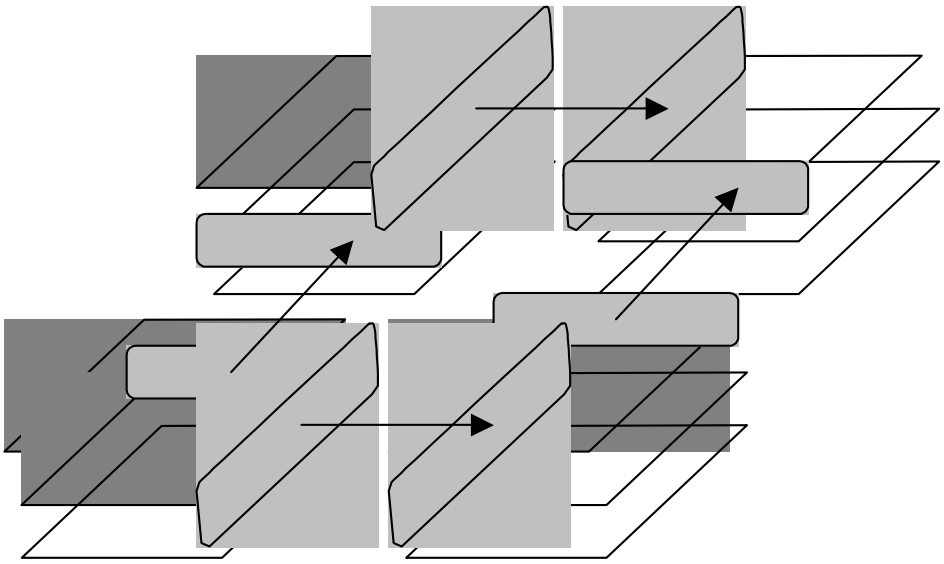


Рис 3. Схема для расчета вектора y^m (Показана передача элементов после расчета двух слоев первым процессором).

На первом шаге вычислений первый процессор обрабатывает верхний слой. Затем осуществляется передача перекрывающихся элементов смежным процессорам. На следующем шаге первый процессор обрабатывает второй слой, а его соседи – первый. Передача элементов после расчета двух слоев первым процессором показана на рис 3. В схеме для расчета вектора y^m только первый процессор не требует дополнительной информации и может независимо от других процессоров вести обработку своей части области, остальные процессоры ждут результатов от предыдущего процессора, пока он не передаст вычисленные значения сеточных функций, для узлов сетки, располагающихся в предшествующих позициях данной строки. Процесс продолжается до тех пор, пока не будут рассчитаны все слои. Аналогичным образом можно решить СЛАУ с верхне-треугольной матрицей для расчета вектора поправки. Далее вычисляются скалярные произведения (11) и выполняется переход на следующий итерационный слой.

Результаты расчета ускорения и эффективности в зависимости от количества процессоров для параллельного варианта адаптивного попеременно-треугольного метода приведены в таблице 1.

Таблица 1. Зависимость ускорения и эффективности от количества процессоров.

Количество процессоров	Время, с.	Ускорение	Эффективность
1	7,490639	1	1
2	4,151767	1,804	0,902
4	2,549591	2,938	0,734
8	1,450203	5,165	0,646
16	0,882420	8,489	0,531
32	0,458085	16,351	0,511
64	0,265781	28,192	0,44
128	0,171535	43,668	0,341

Из таблицы видно, что алгоритм попеременно - треугольного итерационного метода и его параллельная реализация на основе декомпозиции по двум пространственным направления могут эффективно применяться для решения задач гидродинамики при достаточно большом числе вычислителей ($p \le 128$).

На основании данных, полученных экспериментальным путем, были вычислены времена, затраченные на накладные расходы параллельного алгоритма, в том числе:

$t_0 \approx 2 \cdot 10^{-9} c$ - среднее время выполнения полезной операции;

$t_x \approx 5 \cdot 10^{-7} c$ - среднее время отклика (латентность);

$t_n \approx 4 \cdot 10^{-8} c$ - среднее время, затрачиваемое на передачу чисел с плавающей точкой.

Получены теоретические оценки ускорения и эффективности параллельной реализации ПТМ вариационного типа в случае декомпозиции области по одному и двум пространственным направлениям[10]. Алгоритмы, основанные на декомпозиции области по двум направлениям, эффективны для больше количества вычислительных элементов по сравнению с алгоритмами, использующими декомпозицию по одному пространственному направлению, т.к. требуют меньшего объёма передач. В случае равного числа разбиений по двум координатным направлениям теоритические оценки ускорения и эффективности примут вид:

$$A_{уск} = \frac{n}{1 + \frac{2n\left(\sqrt{n}-1\right)}{25t_0}\left(\frac{9t_0}{N_z n} + \frac{N_x + N_y}{N\sqrt{n}}t_n + \frac{t_x}{N}\right) + \frac{2n}{25t_0}\left(t_n\left(\frac{1}{N_x} + \frac{1}{N_y}\right) + \frac{t_x\sqrt{n}}{N_x N_y}\right)},$$

$$E_{эф} = \frac{A_{уск}}{n} = \frac{1}{1 + \frac{2n\left(\sqrt{n}-1\right)}{25t_0}\left(\frac{9t_0}{N_z n} + \frac{N_x + N_y}{N\sqrt{n}}t_n + \frac{t_x}{N}\right) + \frac{2n}{25t_0}\left(t_n\left(\frac{1}{N_x}+\frac{1}{N_y}\right) + \frac{t_x\sqrt{n}}{N_x N_y}\right)},$$

где n - количество вычислительных узлов, N_x, N_y, N_z - размеры расчетной области. Число разбиений по каждому из координатных направлений равно $\sqrt{n}$.

На рис. 4 изображены графики зависимости ускорения от количества процессоров, рассчитанные теоретически и экспериментально.

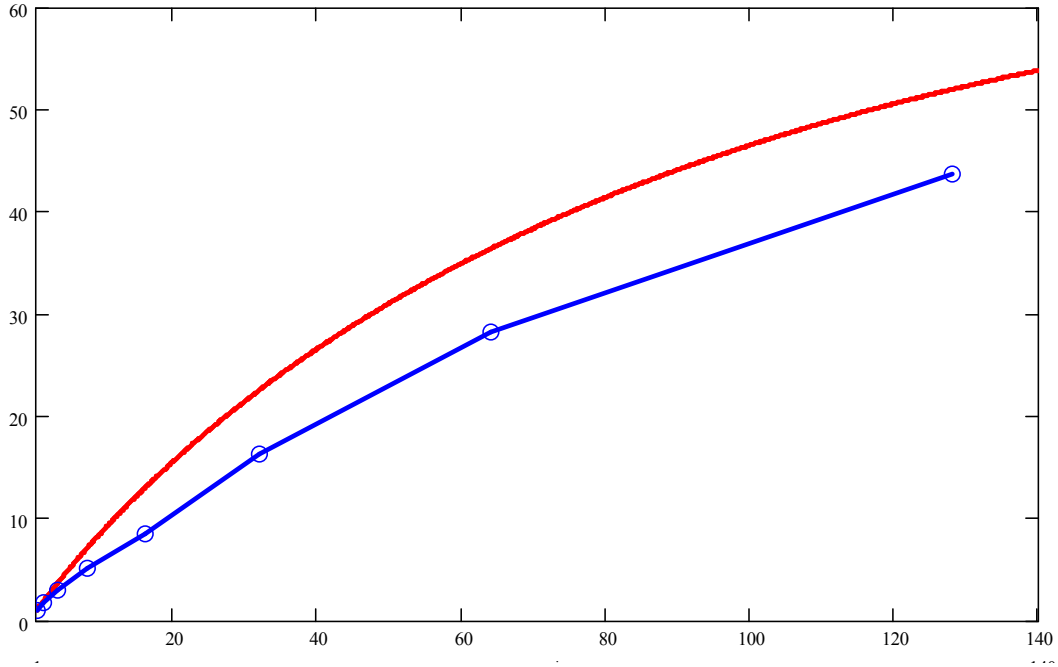


Рис 4. Зависимости ускорения от числа процессоров. Гладкая кривая – теоретическая зависимость, ломаная – экспериментальная.

В теоретических оценках ускорения рассматривается случай модельной задачи с прямоугольной областью. При решении задачи для реального водоема расчетная область имеет сложную форму. При этом реальное ускорение меньше его теоретической оценки. Из рис 4 видно, что обе кривые проходят достаточно близко друг к другу, т.е. зависимость ускорения, полученную при теоретической оценке, можно использовать в качестве оценки сверху для ускорения при параллельной реализации алгоритма ПТМ вариационного типа путем декомпозиции области по двум пространственным направлениям.

7. Результаты численных экспериментов

Численные эксперименты на основе выше описанной математической модели гидродинамических процессов производились на сетке размерностью 351x251x14 для Азовского моря. Построены распределения течений в Азовском море. В таблице 2 представлены данные водного баланса.

Таблица 2. Данные водного баланса в Азовском море.

Гирла[1] Свиное, Кривое и Богдан	+82 м^3/с
Гирло Песчаное	+199 м^3/с
Гирло Мериновое	+105 м^3/с

[1] Ги́рло (укр. происхождения) – название рукава или протоки в дельтах крупных рек, впадающих в Чёрное и Азовское моря.

Гирло Мокрая Кутерьма	+185 $м^3/с$
Гирло Кутерьма	+424 $м^3/с$
Гирла Мертвый Донец и Средняя Кутерьма	+390 $м^3/с$
Кубань	+923 $м^3/с$
Черное море	-1587 $м^3/с$
Сиваш	-115 $м^3/с$
Испарение	-606 $м^3/с$

Приведем результаты численных экспериментов для Азовского моря (баротропные течения) (см. рис. 5 – рис. 6). Палитрой показана интенсивность течения.

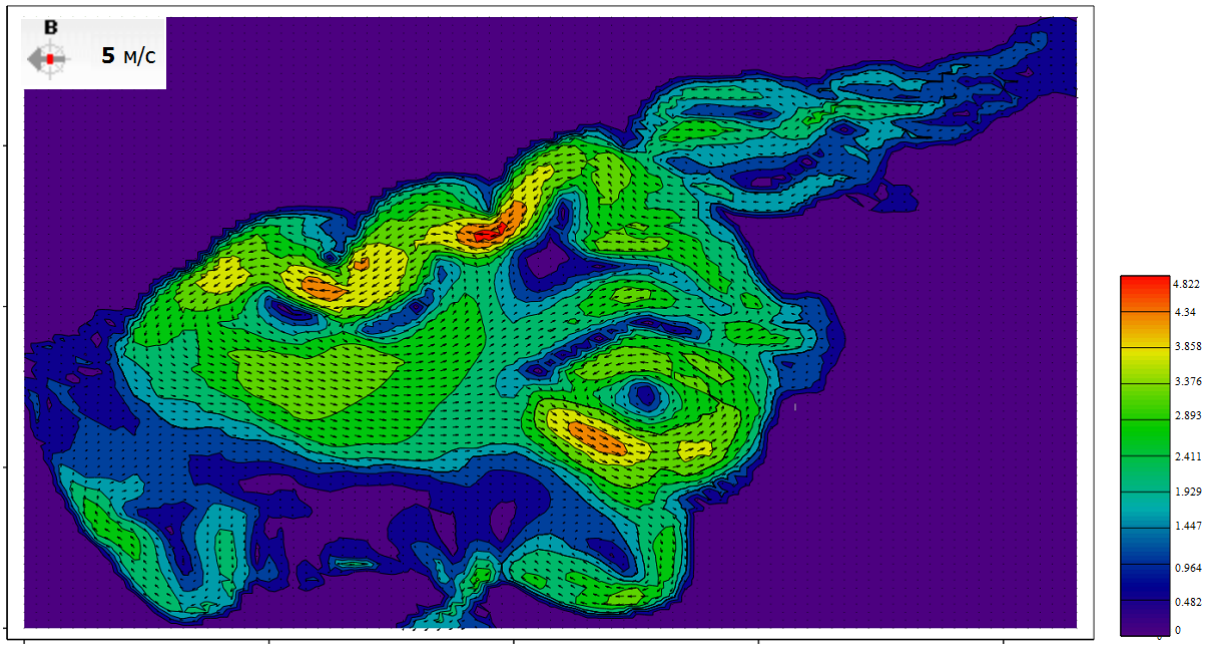


Рис.5. Поле вектора скорости движения водной среды при восточном ветре 5 м/с (баротропные течения)

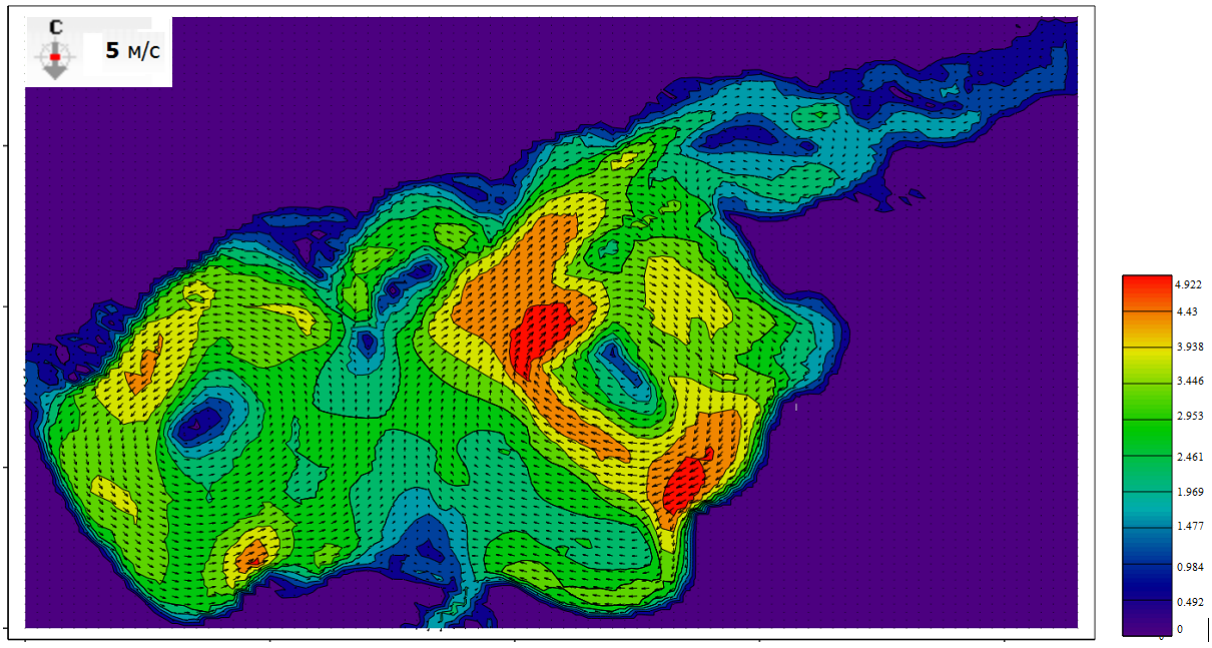


Рис.6. Поле вектора скорости движения водной среды при северном ветре 5 м/с (баротропные течения)

При наличии замкнутого вихревого движения среды значительное количество органических веществ попадает в захват этого района и, опускаясь на дно, образует органический осадок, что приводит к появлению участков анаэробного заражения. На рис. 5 – 6 видно наличие вихревой структуры течения в восточной части Азовского моря, в данном районе вода богата органическими примесями, источниками которых являются реки Дон и Кубань.

Заключение

Работа посвящена построению дискретной математической модели для расчета полей скоростей применительно к прибрежным системам и мелководным водоемам, таким как Азовское море. Отличительными особенностями разрабатываемых алгоритмов являются: высокая производительность, достоверность и точность получаемых результатов. Высокая производительность достигается за счет использования эффективных численных методов решения сеточных уравнений, ориентированных для применения на параллельных вычислительных системах в реальном и ускоренном масштабах времени. Достоверность достигается за счет учета определяющих физических факторов, таких как: сила Кориолиса, турбулентный обмен, сложная геометрия дна и береговой линии, испарение, стоки рек, динамическое перестроение расчетной области, ветровые напряжения и трение о дно, а также за счет учета отклонения значения поля давления от гидростатического приближения. Точность достигается применением подробных расчетных сеток, учитывающих степень «заполненности» расчетных ячеек, а также отсутствием неконсервативных диссипативных слагаемых и нефизичных источников (стоков), возникающих в результате конечно-разностных аппроксимаций. Также в работе показана эффективность алгоритма адаптивного попеременно - треугольного итерационного метода и его параллельной реализации, выполненной на основе декомпозиции области по двум пространственным направлениям, применительно к решению задач гидродинамики мелководных водоемов при достаточно большом количестве вычислителей.

Литература

1. Сухинов А.И., Якушев Е.В. Комплексные океанологические исследования Азовского моря в 28-м рейсе научно-исследовательского судна «Акванавт» // Океанология, 2003, т. 43, №1, с.44-53.
2. Сухинов А.И., Чистяков А.Е., Алексеенко Е.В. Численная реализация трехмерной модели гидродинамики для мелководных водоемов на супервычислительной системе// Математическое моделирование. 2011. Т. 23, № 3. – С.3-21.
3. Чистяков А.Е. Трехмерная модель движения водной среды в Азовском море с учетом транспорта солей и тепла// Известия ЮФУ. Технические науки. – 2009. №8(97). С 75-82.
4. Белоцерковский О.М. Турбулентность: новые подходы - М.: Наука, 2003
5. Белоцерковский О. М., Гущин В. А., Щенников В. В.. Метод расщепления в применении к решению задач динамики вязкой несжимаемой жидкости// Ж. вычисл. матем. и матем. физ., 15:1 (1975). 197–207.
6. Самарский А.А. Теория разностных схем. М. Наука, 1989.
7. Сухинов А.И., Тимофеева Е.Ф. Чистяков А.Е. Построение и исследование дискретной математической модели расчета прибрежных волновых процессов// Известия ЮФУ. Технические науки. – 2011. №8(121). С 22-32.
8. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М. Наука, 1978.
9. Коновалов А.Н. К теории попеременно - треугольного итерационного метода// Сибирский математический журнал. 2002. 43:3. С. 552-572.
10. Чистяков А.Е. Теоретические оценки ускорения и эффективности параллельной реализации ПТМ скорейшего спуска// Известия ЮФУ. Технические науки.– . 2010, №6(107). С 237-249.

Суперкомпьютерное моделирование полупроводниковых квантовых наносистем*

О.А. Ткаченко, В.А. Ткаченко

Институт физики полупроводников им. А.В. Ржанова СО РАН

При исследовании квантовых эффектов и электронного транспорта в наноструктурах требуется многократно решать задачи на больших сетках с варьированием двух и более внешних параметров. Эти вычисления эффективно распараллеливаются. Учет реальной геометрии структур или взаимодействия электронов позволил обнаружить новые физические эффекты: переключение направления тока в Y-разветвителе, флуктуации фазы и температурной зависимости осцилляций Ааронова-Бома в кольцевом интерферометре, появление дополнительной особенности кондактанса в микроконтакте, формирование фрактальных террас падения напряжения и точечное тепловыделение в разупорядоченной решетке антиточек.

1. Введение

В разных областях науки и техники существует множество вычислительных задач, допускающих максимально эффективное распараллеливание, когда однократное решение базовой системы уравнений возможно на однопроцессорном компьютере с достаточно большими оперативной памятью и быстродействием. Полная задача в этом случае требует распределенного решения только из-за обилия значений внешних параметров, т.е. разнообразия начальных, граничных и других условий, входящих в основные уравнения. В настоящее время складывается смешанная технология вычислительных экспериментов, в которых основная часть рутинных расчетов передается кластерным суперЭВМ путем простого распараллеливания, когда каждый процессор независимо от других выполняет серию однотипных заданий. Таким образом, суперкомпьютер становится существенной частью вычислительного эксперимента, тогда как персональный компьютер остается незаменимым для выполнения интерактивных предварительных и завершающих прикладных расчетов. В данной работе мы рассмотрим ряд примеров на эту тему из области, представляющей стык между полупроводниковыми нанотехнологиями и физикой наноструктур.

Основная физика квантовых эффектов и электронного транспорта в полупроводниковых наноустройствах схватывается достаточно простыми универсальными моделями, на основе которых можно получить много интересных результатов [1–4]. При моделировании эффектов, наблюдаемых в эксперименте, приходится многократно выполнять многопараметрические расчеты с решением одномерных, двумерных и трехмерных уравнений Шредингера, Пуассона или Кирхгоффа. Варьирование таких управляющих параметров, как энергия квазичастиц, магнитное поле, химпотенциал, затворное напряжение и температура допускает эффективное распараллеливание. В этих случаях с помощью кластерных суперЭВМ нами получены результаты, которые отчасти представлялись в [5–10] и приводятся здесь в сводном виде вместе с некоторыми новыми результатами.

Общим для изучаемых здесь электронных систем является принципиально одинаковый способ их формирования молекулярно-лучевой эпитаксией и нанолитографией. Подразумевается управляемое введение разрезающих электростатических барьеров в двумерный электронный газ, созданный в глубине гетероструктуры GaAs/GaAlAs. Этот способ ана-

*Работа поддержана программой Президиума РАН «Основы фундаментальных исследований нанотехнологий и наноматериалов» и междисциплинарным интеграционным проектом СО РАН «Математические модели, численные методы и параллельные алгоритмы для решения больших задач СО РАН и их реализация на многопроцессорных суперЭВМ» (2009–2011).

логичен приемам промышленного изготовления сверхбыстродействующих нанотранзисторов и совмещает необходимую технологическую надежность с разнообразием получаемых структур [3, 9]. На пяти примерах будут рассмотрены случаи геометрически простых и весьма сложных разрезов. Все изучаемые здесь электронные системы состоят из малых основных элементов – аксиально-симметричных барьеров (антиточек), а также коротких квазиодномерных сужений в двумерном электронном газе и соединяемых ими субмикронных квантовых точек. Топологически это случаи малых структур из одного [5], трех [6, 7] и шести [8] субмикронных квантовых точечных контактов, а также случай большой квадратной решетки таких контактов [9]. Соответственно, эти 5 структур содержат одну антиточку [7], одну [6, 7], две [2, 8] и 10^5 квантовых точек субмикронного размера [9]. Основным регулятором наполнения сформированных электронных каналов является затворное напряжение.

По мере увеличения размера анализируемых систем будет меняться детальность рассмотрения базовых физических явлений. От тонких эффектов электрон-электронного взаимодействия в одиночном микроконтакте мы перейдем к новым эффектам одночастичной интерференции для промежутка микроконтакт – перемещаемая антиточка, Y-перехода и кольцевого интерферометра и, наконец, опишем моделирование фрактального электронного транспорта в разупорядоченной решетке антиточек.

2. Основные алгоритмы

Электронный транспорт исследуется чаще всего в линейном приближении, когда ток I пропорционален приложенному тянущему напряжению V. Основной величиной, которая в целом характеризует транспорт и структуру, является электрическое сопротивление $R = V/I$, либо кондактанс $G = I/V$.[1] Самые разные структуры субмикронного размера при низкой температуре могут рассматриваться как квантовые волноводы, кондактанс которых дается формулой Ландауэра:

$$G = (2e^2/h) \int \sum_n T_n(E, U) F(E - \mu)\, dE, \tag{1}$$

где $2e^2/h$ – квант кондактанса, e – заряд электрона, h – постоянная Планка, $T_n(E, U)$ – коэффициент прохождения для волн, падающих на наноструктуру из n-ой подзоны широкого подводящего канала, E – полная энергия баллистического электрона, $U(\mathbf{r})$ – эффективный удерживающий потенциал в наноструктуре. Функция $F(E - \mu) = (1/4k_BT)\,\mathrm{sech}^2((E - \mu)/2k_BT)$ учитывает тепловое размытие функции Ферми, μ – химпотенциал, T – температура, k_B – константа Больцмана.

Расчет коэффициента прохождения предполагает решение одночастичного уравнения Шредингера

$$[(\hat{\mathbf{p}} - e\mathbf{A}/c)^2/2m]\psi + (E - U(\mathbf{r}))\psi = 0, \tag{2}$$

где $\hat{\mathbf{p}} = -i\hbar\nabla_{2D}$, $\mathbf{A}$ – вектор потенциал, который учитывает действие перпендикулярного магнитного поля. Обычно при реалистическом моделировании наноструктур эффективный 2D потенциал $U(\mathbf{r})$ вместе с самосогласованным распределением заряда в структуре находится решением трехмерного уравнения Пуассона и далее не меняется. Предполагается, что изменением его формы можно пренебречь [1,6–8] по сравнению с изменением E в уравнении Шредингера.[2]

[1] Возможны два или больше электрических контактов к структуре, так что величины I, V являются двухиндексными. Индексы не пишутся в наиболее важном двухтерминальном случае, когда I, V можно отнести к одним и тем же двум точкам. Ниже будет рассмотрен также трехтерминальный случай (Y-переход).

[2] Иногда все же необходимо учитывать зависимость эффективного потенциала от затворного напряжения и температуры. Этот учет трудно совмещается с моделированием электронного транспорта, поскольку требует самосогласования и в качестве паллиатива может использоваться физически обоснованное задание потенциала $U(\mathbf{r})$ формулами. Далее этот $U(\mathbf{r})$ может переопределяться довольно громоздким численным расчетом, как в случае электрон-электронного взаимодействия в микроконтакте.

В основном для решения задач многоканального квантового рассеяния в двумерном случае применяются разные варианты метода сильной связи, позволяющие сеточную имитацию 2D непрерывных потенциалов. С этой целью нами реализованы программы расчета 2D волновых функций и кондактанса произвольной переходной области в широком двумерном волноводе [1, 2].

Для случаев без магнитного поля использован метод S-матриц [11]. Он основан на предположении постоянства потенциала в продольном направлении на отрезках (x_j, x_{j+1}), допускающем расчет финитного движения по y и матричных элементов перехода $C^{(j)}_{mn} = \int \xi^{(j)}_m \xi^{(j+1)}_n \, dy$, где $\xi^{(j)}_n(y)$ – волновая функция уровня поперечного квантования $E^{(j)}_n$ на участке (x_j, x_{j+1}). Полная волновая функция на этом участке имеет вид:

$$\Psi(x,y) = \sum_{n=1}^{M} (k^{(j)}_n)^{-1/2} \left[a^{(j)}_n e^{ik^{(j)}_n (x-x_j)} + b^{(j)}_n e^{-ik^{(j)}_n (x-x_{j+1})} \right] \xi^{(j)}_n(y) \tag{3}$$

где $k^{(j)}_n = (E - E^{(j)}_n)^{1/2}$ – действительное или мнимое волновое число, отвечающее продольному движению в каждой моде j, а суммирование ограничивается конечным числом мод M. Амплитуды $a^{(j)}$ и $b^{(j)}$ на соседних участках связаны матрицей рассеяния S_j:

$$\begin{pmatrix} a^{(j+1)} \\ \\ b^{(j)} \end{pmatrix} = S_j \begin{pmatrix} a^{(j)} \\ \\ b^{(j+1)} \end{pmatrix}. \tag{4}$$

Полная S-матрица, описывающая рассеяние на всей структуре, является комбинацией матриц S_j [11]:

$$S = S_0 \otimes S_1 \otimes \cdots \otimes S_N = \begin{pmatrix} t & r \\ \\ r' & t' \end{pmatrix}, \tag{5}$$

где t, t' — полные амплитуды прохождения для противоположных направлений, а r, r' — соответственно, амплитуды отражения. Суммарный коэффициент прохождения вычисляется как

$$T_{\text{tot}} = \text{Tr}[tt^\dagger] = \sum_n \sum_m |t_{nm}|^2, \tag{6}$$

где t_{nm} — комплексные амплитуды прохождения с переходом из падающей волны в моде n в прошедшую волну в моде m, и суммирование идет по открытым модам (одномерным подзонам) входного и выходного каналов.

В случае присутствия перпендикулярного магнитного поля использованы другие алгоритмы. Нами разработаны программы, реализующие метод рекурсивных функций Грина для расчета кондактанса [12], а также модификация этого метода для расчета волновых функций двумерного рассеяния [13]. Возможности разработанных программ проверены моделированием квантового транспорта через баллистический кольцевой интерферометр [1, 2, 8] и трехконтактную квантовую точку в Y-переходе [6, 7].

3. Примеры задач и расчеты

3.1. Фриделевские осцилляции в квантовом точечном контакте

Кондактанс субмикронного сужения в двумерном электронном газе (ДЭГ) хорошо описывается одночастичной квантовой механикой. Исключением является экспериментально обнаруженная на первой ступени квантования кондактанса $0.7 \cdot 2e^2/h$ особенность, для которой были предложены десятки объяснений: от простых феноменологических до сложных

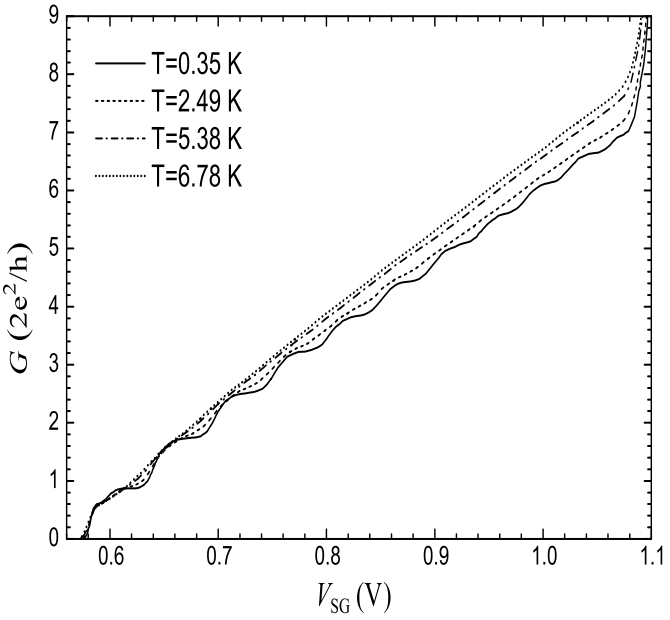


Рис. 1. Экспериментальные зависимости кондактанса микроконтакта G от V_{SG} – напряжения на расщепленном затворе [5]. На первой ступени квантования видна $0.7{\cdot}2e^2/h$ особенность; начиная со второй ступени средний наклон $G(V_{SG})$ с ростом температуры T увеличивается. Другие данные об этой структуре можно найти в [3].

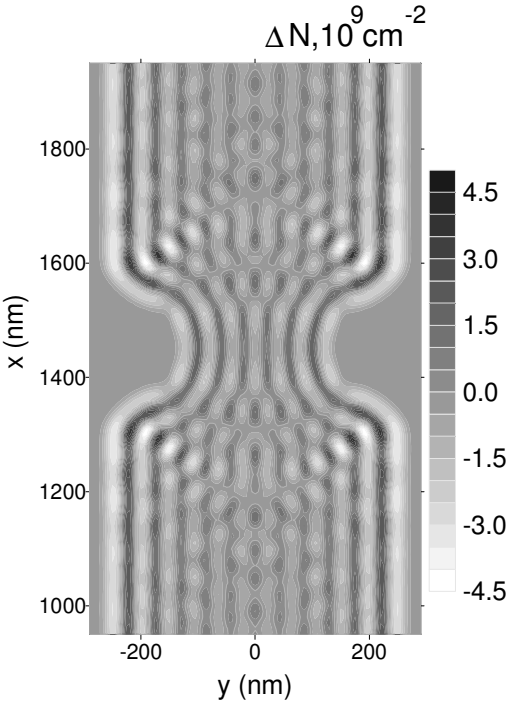


Рис. 2. Разница электронных плотностей $\Delta = n(\mu,T) - n(\mu,T^*)$ в канале с локальным сужением (микроконтактом): $\mu = 3.3$ мэВ, $k_BT = 0.03$ мэВ, $k_BT^* = 0.4$ мэВ.

теоретических и численных. Многомодовый режим казался ясным, но нашлись измерения, которые расходятся с предсказанием одночастичной картины о неподвижных точках кондактанса $N \cdot e^2/h$ при целых $N > 2$. В реальности эти точки с ростом температуры сдвигаются, т.е. средний наклон затворной характеристики увеличивается (рис.1) [5].

Чтобы объяснить температурное поведение кондактанса мы учли межэлектронное взаимодействие поправкой к потенциалу в духе несамосогласованного приближения Хартри-Фока. Такая поправка по форме совпадает с фриделевскими осцилляциями электронной плотности $n(x,y)$ (рис.2), которые находятся по вкладам от всех занятых состояний решением уравнения Шредингера для исходного потенциала $U_0(x,y)$. В многомодовом режиме транспорта можно записать: $\delta U(x,y) = -\alpha\Delta/D$, где $\Delta = n(x,y,\mu,T) - n_0(x,y,\mu)$, где n_0 – плотность со сглаженными фриделевскими осцилляциями, например, $n(x,y,\mu,T^*)$ при достачно большой температуре T^*, $D = m/(\pi\hbar^2)$ – квазиклассическая двумерная плотность состояний, α – безразмерная константа порядка единицы, значение которой не вычисляется в теории, но может быть оценено из сравнения вычисленного кондактанса для потенциала $U_0(x,y) + \delta U(x,y)$ с экспериментом.

Исходный плавный потенциал многомодового канала с сужением $U_0(x,y)$ был задан простыми формулами и являлся грубой идеализацией реальных микроконтактов. Для $U_0(x,y)$ с малым шагом по E (10^{-3} мэВ) вычислены 10^4 двумерных волновых функций на сетке $600 \cdot 140$ узлов с шагом 5 нм по x, y. Эти вычисления, как затратные по времени, выполнялись на многопроцесорных машинах Сибирского суперкомпьютерного центра (СО РАН, Новосибирск) и суперкомпьютерного центра IDRIS (CNRS, France). Интегрированием $|\Psi(x,y,E)|^2$ по E с учетом заполнения состояний, т.е. функции Ферми, определялись электронные плотности для разных химпотенциалов μ и температур.

Наконец, для серий поправленных эффективных потенциалов $U_0(x,y) + \delta U(x,y,\mu,T)$

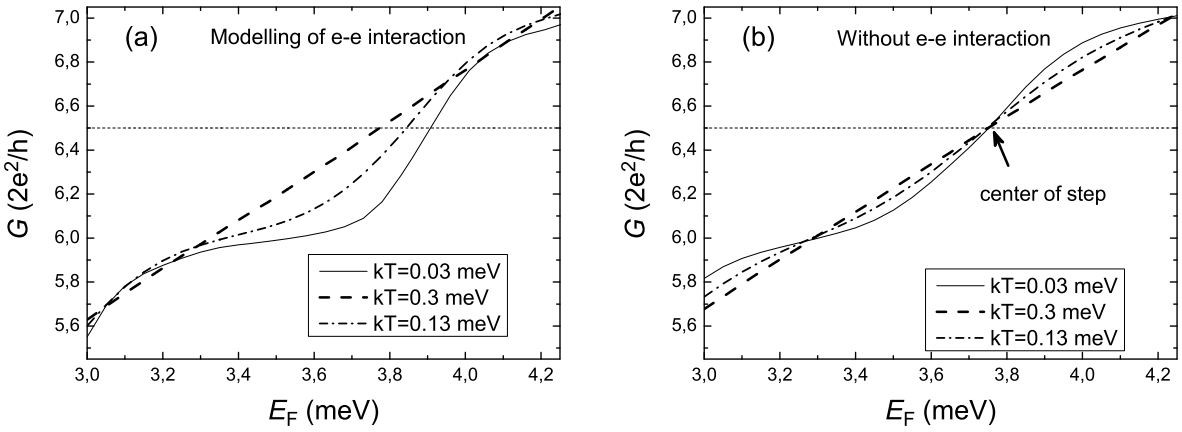


Рис. 3. Вычисленный кондактанс канала с микроконтактом как функция μ при разных T: (a) для $U = U_0(x, y) - \alpha\Delta(T, T^*)/D$, $k_BT^* = 0.4$ meV, $\alpha = 6$; (b) для $U = U_0(x, y)$ [5].

решались задачи двумерного квантового рассеяния со сканированием по энергии, и по формуле (1) вычислялся кондактанс. Найдено, что с учетом фриделевских осцилляций кондактанс сужения ведет себя при изменении μ и T аналогично измеренному: точки кондактанса $N \cdot e^2/h$ сдвигаются в сторону меньшего химпотенциала с ростом температуры. Напротив, если поправку взаимодействия обнулить, то расчет по формуле (1) сохраняет эти точки неподвижными и наблюдается только сглаживание кривых с ростом температуры (рис.3).

Для моделирования 0.7 особенности были выполнены расчеты электронного прохождения через одномерный барьер. Поправка к потенциалу от фриделевских осцилляций находилась по одномерной плотности состояний. Расчеты воспроизводят наблюдаемую температурную зависимость этой особенности (рис.4). Таким образом, электрон-электронное взаимодействие правильно описывает температурные зависимости затворных характеристик в микроконтакте.

3.2. Сканирующая затворная микроскопия канала с сужением

Здесь на примере канала с сужением развивается идея квантового биллиарда с основным элементом в виде антиточки– малой области обеднения в ДЭГ под острием атомно-силового микроскопа, которое действует как перемещаемый затвор, находящийся вместе с гетероструктурой в криостате [7]. Электронная волна, выходящая из сужения в канал, рассеивается на антиточке (рис.5, левая часть). Показанный потенциал получен расчетом трехмерной электростатики устройства и для этого потенциала приведена вычисленная картина плотности вероятности.

Картина рассеяния является сложной и в ней видны частые осцилляции электронной

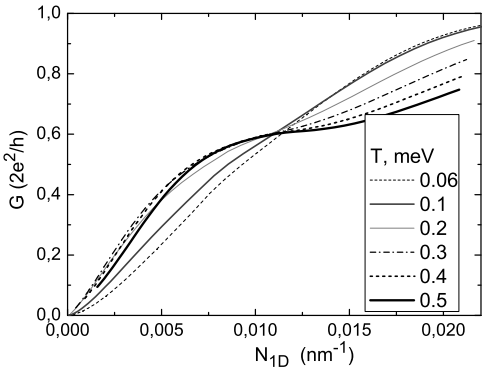


Рис. 4. Вычисленные зависимости коэффициента прохождения баллистического электрона через одномерный потенциальный барьер $U(x) = V_0/ch^2(x/a) - \alpha\delta n(x, \mu, T)\pi\hbar v_F$, содержащий фриделевские осцилляции ($V_0 = 4.8$ мэВ, $a = 100$ nm, $\alpha = 0.2$, $v_F(\mu)$ –скорость Ферми вдали от барьера). Аргументом является электронная плотность в центре барьере, которая в экспериментах с микроконтактом определяется напряжением на расщепленном затворе.

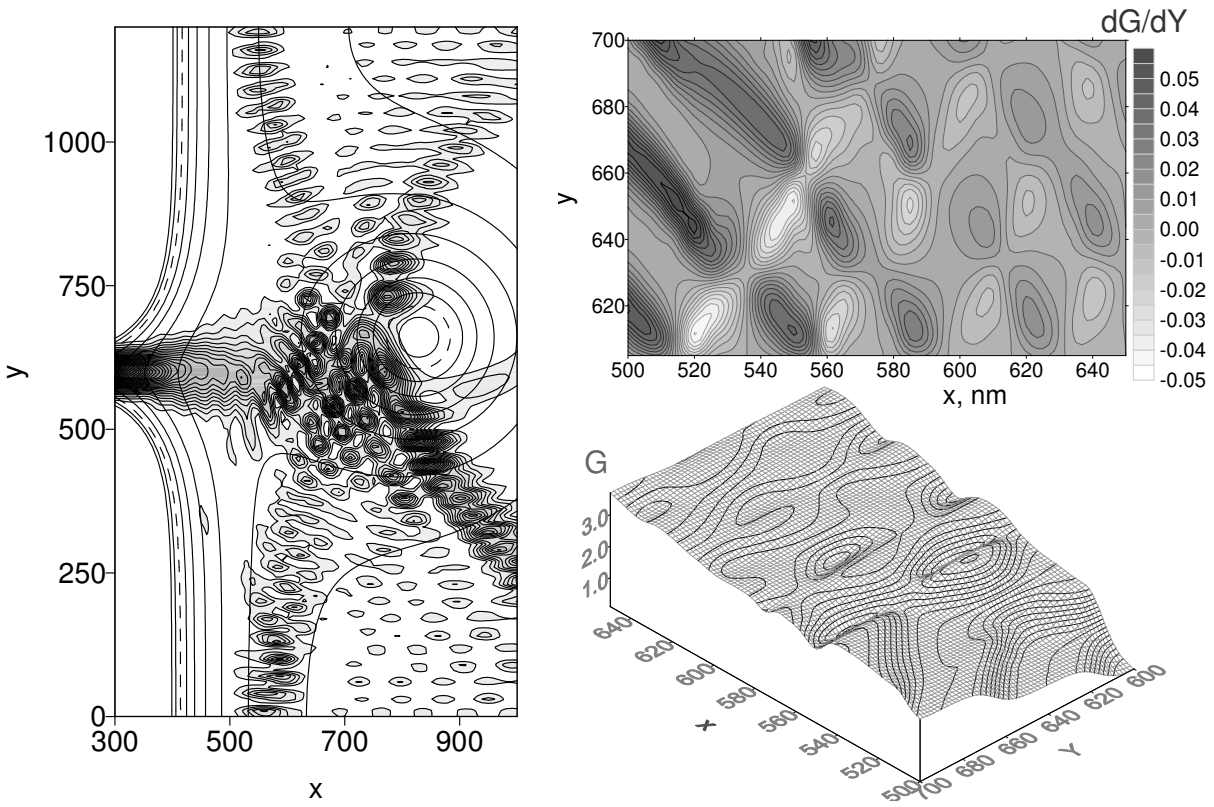


Рис. 5. Слева плотность вероятности и эффективный потенциал (плавные изолинии) для канала с сужением и антиточкой. Координаты центра антиточки $x_t = 830$ nm, $y_t = 660$ nm. Пунктирные изолинии соответствует уровню Ферми. Кондактанс $G = 1.94e^2/h$. Справа внизу кондактанс канала как функция положения антиточки, вверху производная dG/dy

плотности, т.е. эффекты интерференции. Благодаря этим эффектам кондактанс сужения чувствителен к положению антиточки, и построение кондактанса как функции координат острия дает микроскопическое изображение (рис.5, правая часть), которое в свернутом виде содержит информацию об интерференции. Об этом говорит частое чередование минимумов и максимумов производной кондактанса, соответствующее фермиевской длине волны. Эти изображения получены расчетом коэффициента прохождения электронных волн для многих реализаций 2D потенциала, отвечающих перемещению антиточки. Расчет выполнен на кластерной суперЭВМ Zahir центра Idris CRNS (Франция). В данном расчете 2D потенциал брался в виде суммы исходного потенциала в канале с сужением и универсального аксиально-симметричного барьера антиточки. Такая возможность и простая радиальная зависимость потенциала в антиточечном барьере найдены в результате предварительных расчетов трехмерной электростатики устройства, требующих десятков тысяч итераций. Найденное приближение для итогового 2D потенциала избавляет от многократных расчетов 3D электростатики и является весьма надежным, пока острие находится на достаточном расстоянии от сужения. В данном случае не учитывались пространственные флуктуации потенциала, обусловленные случайным расположением заряженных примесей. Заметим, что в эксперименте эффекты интерференции могут быть ослаблены или искажены по сравнению с рассмотренной идеализированной ситуацией, но они уже были зарегистрированы сканирующей затворной микроскопией.

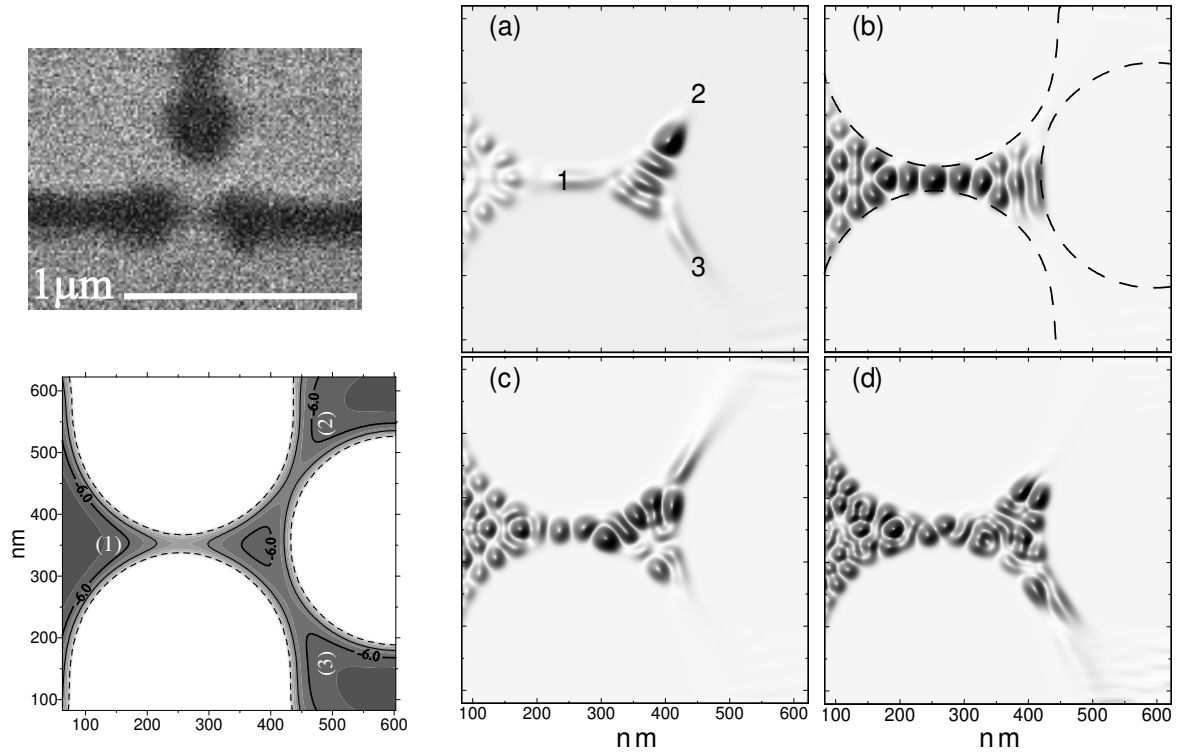


Рис. 6. Слева вверху микрофотография изучаемого устройства до напыления затвора. Видны три антиточки и разрезы к ним – вытравленные (темные) области на поверхности гетероструктуры GaAs/AlGaAs. Внизу вычисленный эффективный потенциал $U(x, y)$ [мэВ] для такой структуры (показана только область $U \leq 0$). Справа распределение плотности вероятности обнаружить баллистический электрон, падающий из резервура 1, вычисленное для четырех значений E_F: (a), (b), (c), (d) – электрон выходит, соответственно, в резервуар 3, 1, 2, 3. Пунктир обозначает границу классически разрешенной области при $E_F = U_2$ [6].

3.3. Интерференция в малой трехконтактной точке Y-перехода

Расчеты трехмерной электростатики показали [1], что развилка одномерных квантовых проволок (Y-переход) является одновременно малой трехвходовой квантовой точкой. Такая одиночная квантовая точка была изготовлена в ИФП СО РАН [3] (рис.6). Известно, что можно управлять направлением тока в Y-переходе с помощью изменения напряжения на двух боковых затворах, антифазно меняющего ширину выходных квантовых проволок. Ток идет преимущественно по более широкому каналу. Это классический эффект. Наши расчеты показывают, что в Y-переходе с небольшой асимметрией (которая в реальных устройствах присутствует всегда) возможен чисто квантовый эффект – управление направлением выхода электрона при изменении уровня Ферми. Степень асимметрии устройства при этом практически не меняется.

Пусть для определенности баллистический электрон попадает в квантовую точку из резервуара 1. Из рисунка 6 видно, что картины интерференции внутри этой точки складываются такими, что при одном значении уровня Ферми E_F электрон выходит в основном в резервуар 2, а при другом – в резервуар 3. По оценке при данном изменении E_F заполняется лишь несколько одночастичных состояний в квантовой точке, т.е. к точке добавляется 8-10 электронов. Из решения уравнения Шредингера вместе с волновыми функциями находятся коэффициенты прохождения между резервуарами, т.е. коэффициенты кондактанса (рис.7, левая часть). Можно видеть, что максимальная степень поляризации $(G_{12}-G_{13})/(G_{12}+G_{13})$ в расчете меняется от -1 до 1.

Заметим, что в эксперименте обнаружен такой же по порядку величины эффект (рис.7,

правая часть). Искать детального согласия теории и измерений не следует, потому что коэффициенты G_{ij} имеют ярко выраженное мезоскопическое поведение из-за интерференции. Это значит, что малейшие изменения геометрии устройства или переключение заряда лишь одного атома во всей наноструктуре меняют форму кривых $G_{ij}(E_F)$, $G_{ij}(V_G)$. Однако сам обнаруженный эффект перенаправлений остается. Таким образом, создание одиночной малой трехконтактной квантовой точки, расчеты и эксперимент позволили обнаружить новое проявление интерференции. Оно является общим для любых типов волн, если размеры слегка асимметричной развилки соизмеримы с длиной волны.

3.4. Мезоскопическое поведение осцилляций Аронова-Бома в малом кольцевом интерферометре

Изготовление малых кольцевых интерферометров с помощью атомного силового микроскопа (АСМ) (локальным анодным окислением) позволило расчетом узнать электронные свойства конкретного экземпляра устройства [3]. По его изображению в АСМ мы восста-

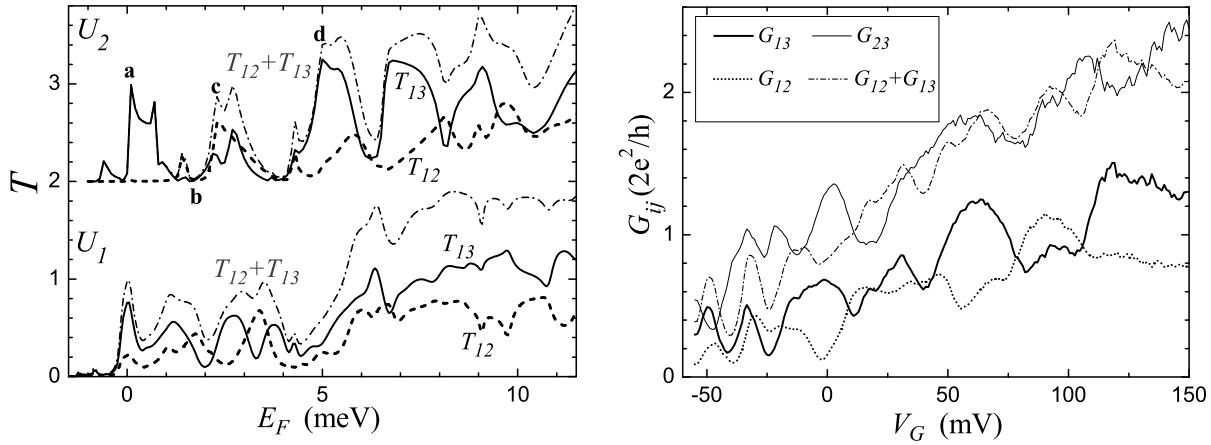


Рис. 7. Слева вычисленная зависимость коэффициентов прохождения от энергии для двух слегка разных реализаций эффективного потенциала U_2 и U_1 (кривые для U_2 сдвинуты вверх для ясности, буквы a—d указывают соответствующие E_F из предыдущего рисунка). Справа зависимость коэффициентов кондактанса от затворного напряжения в эксперименте [6].

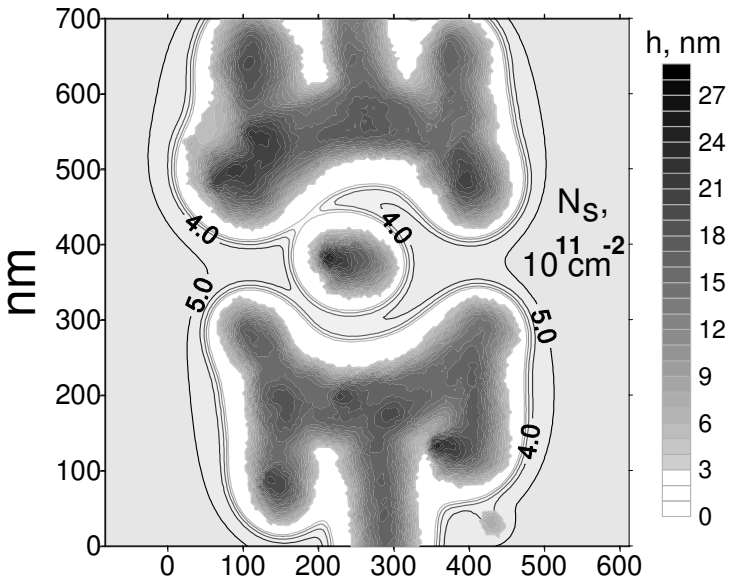


Рис. 8. Карта глубины локального анодного окисления h (показана темным на фоне области обеднения) и вычисленная электронная плотность N_s (изолинии) в случае кольцевого интерферометра, изготовленного с помощью АСМ [3].

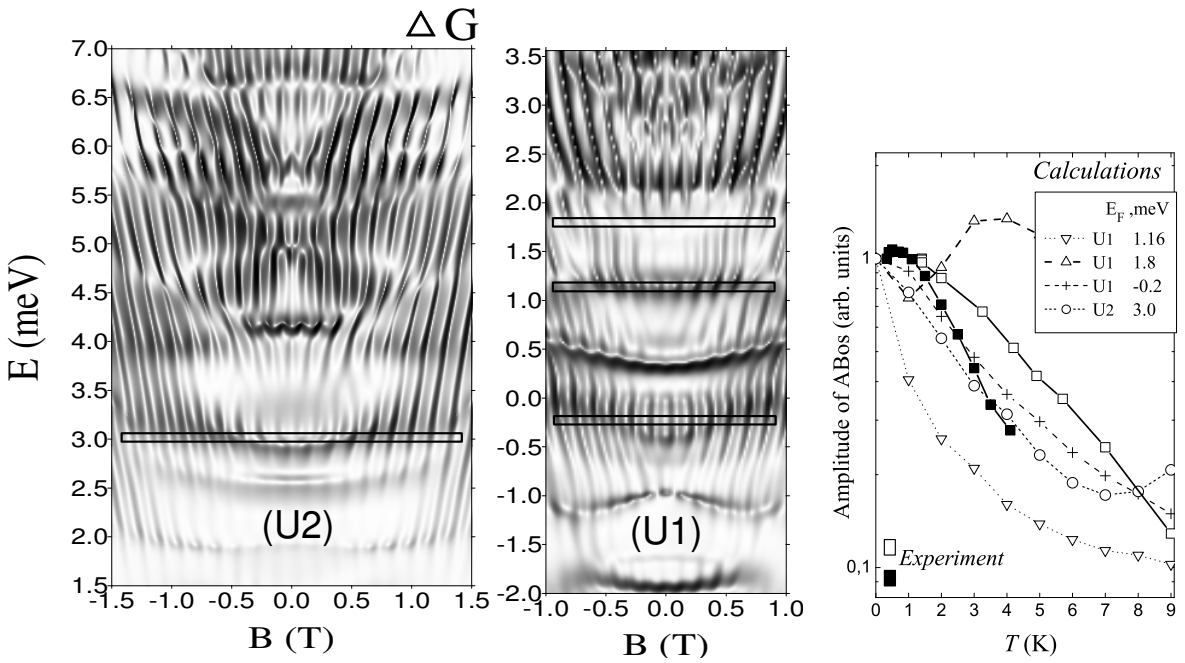


Рис. 9. Слева вычисленные осцилляции Аронова-Бома в пределе нулевой температуры для слегка разных эффективных потенциалов U_2, U_1 в малом кольцевом интерферометре. Максимальное отклонение от фонового кондактанса G составляет $0.4 \cdot e^2/h$. Видны мезоскопические флуктуации амплитуды и фазы осцилляций. Узкими прямоугольниками обозначены E_F, для которых на правой части рисунка показаны вычисленные температурные зависимости амплитуды осцилляций (сплошные линии – эксперимент).

навливаем глубину h для нижней (невидимой) границы анодного окисла (рис.8, темная часть), т.е. форму поверхности полупроводника, на которой ставится граничное условие при решении трехмерной задачи электростатики. Таким образом, находится эффективный потенциал в плоскости двумерного электронного газа, расположенной в 25 нм от исходной поверхности гетероструктуры GaAs/AlGaAs. На рисунке 8 изолиниями показано распределение электронной плотности. Видно, что асимметрия структуры передалась эффективному потенциалу.

Перед нами стояла задача прогнозирования поведения данного интерферометра в широком диапазоне изменения энергии электрона E, магнитного поля B, химпотенциала E_F и температуры T. Для этого необходимо было решать задачу двумерного квантового рассеяния баллистического электрона в найденном потенциале на сетке из 10^4 узлов для 10^5 пар значений (E,B), что потребовало 6 часов счета на 64 процессорах машины Zahir суперкомпьютерного центра IDRIS (France) (рис.9). Видно, что разные реализации потенциала дают собственное сложное поведение амплитуды и, что неожиданно, фазы осцилляций Аронова-Бома (АБ) в пределе нулевой температуры [8].

Имея полный набор состояний, можно было с помощью формулы (1), уже без супер-ЭВМ, проследить за температурным изменением осцилляций АБ. Этим расчетом было обнаружено, что фаза осцилляций стабилизируется при увеличении температуры до $T \sim 1$ K, но изменение E_F дает флуктуации формы температурной зависимости амплитуды осцилляций АБ [8] (рис.9). Нестабильность темпа теплового подавления осцилляций действительно обнаружена в измерениях с данной структурой [10].

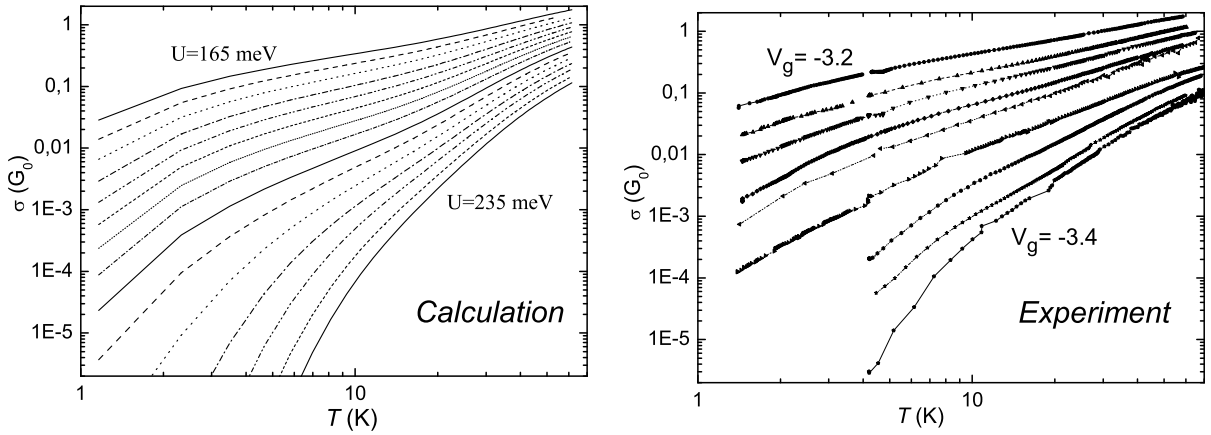


Рис. 10. Вычисленные и измеренные зависимости проводимости решетки антиточек с числом узлов 150 000.

3.5. Фрактальный транспорт в решетке антиточек

Недавно в Институте физики металлов УроРАН была изготовлена и экспериментально исследована разупорядоченная квадратная решетка антиточек–вытравленных углублений на поверхности структуры с квантовой ямой AlGaAs/GaAs/AlGaAs [9].[1] Диаметр областей обеднения в двумерном электронном газе под вытравленными углублениями управлялся напряжением на сплошном металлическом затворе. Чтобы понять поведение этой структуры мы разработали модель, соединяющую описание квантовых явлений в малом масштабе и классического электронного транспорта на больших масштабах.

Двумерный эффективный потенциал задается простыми формулами, как сумма антиточечных аксиально-симметричных потенциалов. Основной параметр — высота барьера U в центре антиточек, – линейно зависит от V_g. Вся структура мысленно разбивается на субмикронные квантовые точки и соединяющие их микроконтакты. Радиальная зависимость антиточечного потенциала и параметр U определяют глубину ямы в электронных озерах и высоту широкого барьера между ними V_{mn} (учтены вариации дистанции между антиточками). Из условий независимости полного числа электронов от T и емкости центра электронного озера от V_g найдено $E_F(U,T)$. При наличии тянущего напряжения квантовым точкам приписываются индивидуальные химпотенциалы, подобно макроскопическим резервуарам баллистических наноструктур. Локальная связь тока через микроконтакт и

[1]Полный размер решетки 300x500 мкм^2, сторона квадрата в элементарной ячейке 1 мкм, диаметр антиточек 0.7 мкм, задаваемый случайный сдвиг антиточек от вершин квадрата не превышал 0.1 мкм.

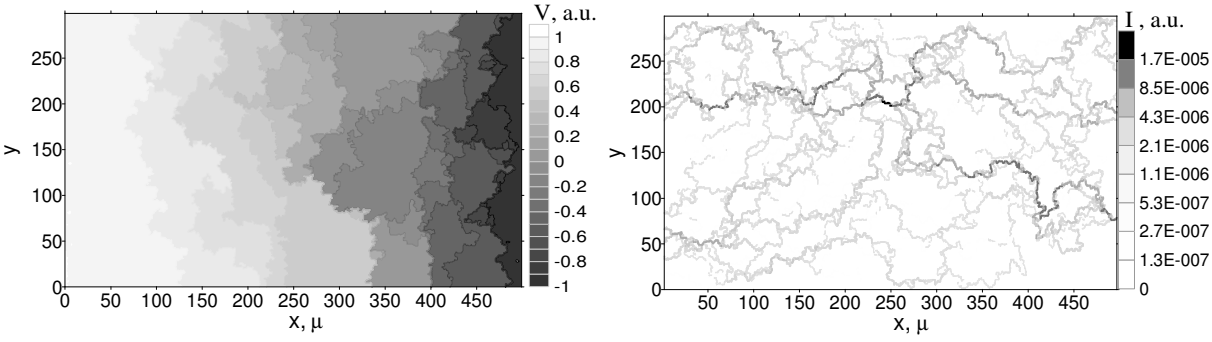


Рис. 11. Пример вычисленных распределений тянущего напряжения и тока в структуре с числом узлов 300×500. Видны террасы напряжения и узкие токовые пути.

напряжения на микроконтакте дается формулой Ландауэра. Полное сопротивление всей решетки вместе с распределением в ней токов и тянущих напряжений вычисляется из законов Кирхгоффа (сумма токов, втекающих в каждую квантовую точку через микроконтакты к четырем соседним точкам, равна нулю). Существенно, что решетка большая (10^5 элементов), а кондактанс микроконтакта зависит от его геометрии, температуры и напряжения на затворе. Поэтому для упрощения расчетов микроконтакты полагаются квазиодномерными, т.е. все вклады T_n в (1) от более высоких подзон выражаются через коэффициент прохождения для нулевой подзоны $T_n(E) = T_0(E - n\hbar\omega, U)$. Профиль потенциала в этом случае есть $U(x) + m\omega^2 y^2/2$. Поверх широкого барьера задается узкий параболический барьер по x с параметрами $\hbar\omega_x$ и $V_0(1 + aT^{-1/2})$ (учет действия примесного флуктуационного потенциала и температурной зависимости высоты барьера из-за эффектов электрон-электронного взаимодействия). Для использованных $U(x)$ выписаны простые формулы для $T_0(E)$, но интеграл (1) берется численно.

Для большой решетки, которая изучалась в эксперименте [9] (150 000 узлов), модель может быть реализована лишь на кластерной суперЭВМ, потому что проводимость решетки зависит от двух непрерывно меняемых внешних параметров T, U. Кроме того, модель содержит около десятка внутренних параметров, значения которых нельзя вычислить из теории, но можно пытаться найти пробными расчетами из сравнения с экспериментом. В итоге этой работы для большой структуры получены зависимости, которые близки к измеренным (рис.10). Особый интерес для изучения электронного транспорта представляет случай малой проводимости $\sigma \sim 10^{-6} G_0$, когда структура близка к порогу перколяции (рис.10). Мы нашли, что распределения тока и напряжения (рис.11) в разупорядоченной решетке антиточек при такой проводимости имеют ярко выраженную фрактальность и значение проводимости определяется кондактансом микроконтакта, оказавшегося на пересечении основной линии тока и линии резкого перепада между соседними террасами в распределении тянущих напряжений.

3.6. Заключение

В работе описаны 5 задач, решенные эффективным распараллеливанием в рамках смешанной технологии вычислительных экспериментов с использованием персональных компьютеров и кластерных суперЭВМ. Для быстрого решения этих задач, возникших недавно на стыке полупроводниковых нанотехнологий и квантовой физики, нами разработаны адекватные упрощенные модели и соответствующие прикладные программы. Расчеты выполнены на машинах суперкомпьютерного центра IDRIS (CNRS, France) и Сибирского суперкомпьютерного центра (СО РАН, Новосибирск).

В перспективе предполагается интеграция высокопроизводительных расчетов кондактанса и реальной трансформации трехмерного потенциала в устройстве при изменении затворных напряжений, магнитного поля и распределения заряда на примесях. Для ряда задач подобная интеграция уже испытана [3, 4]. Дальней целью является реалистическое моделирование рукотворных наноструктурированных электронных систем большой площади, в которых классический и квантовый транспорт переплетаются и требуется разработка новых физических моделей.

Литература

1. Ткаченко О. А., Ткаченко В. А., Бакшеев Д. Г., Квон З. Д., Портал Ж. К. Электростатический потенциал, энергетический спектр и резонансы Фано в кольцевом баллистическом интерферометре на основе гетероперехода AlGaAs/GaAs// Письма в ЖЭТФ 2000. Т. 71. С. 366–371.
2. Ткаченко О. А., Ткаченко В. А., Бакшеев Д. Г. Волновые функции баллистического электрона и отрицательное магнитосопротивление в малом кольцевом

интерферометре// Письма в ЖЭТФ 2004. Т. 79. С. 351–355.

3. Ткаченко О.А., Ткаченко В.А., Квон З.Д., Латышев А.В., Асеев А.Л. Интроскопия квантовых наноэлектронных устройств// Российские нанотехнологии 2010. Т. 5. С. 117–127.
4. Ткаченко В. А., Ткаченко О. А., Квон З. Д., Асеев А. Л. Внедрение компьютерного моделирования в физику наноэлектронных устройств// 1 Всероссийская конференция "Многомасштабное моделирование процессов и структур в нанотехнологиях"Москва. 2008. Тезисы докладов. С.299–300.
5. Tkachenko O.A., Renard V.T., Pyshkin K.S., Gornyi I.V., Dmitriev A.P., Tkachenko V.A., Portal J.-C. Electron-electron interaction in multi-modal quantum point contacts// 15th Int. Symp. Nanostructures: Physics and Technology (Novosibirsk, 2007.) Proceedings. Ioffe Institute. St.Petersburg. 2007, P. 317–318; Renard V. T., Tkachenko O. A., Tkachenko V. A., Ota T., Kumada N., Portal J. C., Hirayama Y. Boundary-Mediated Electron-Electron Interactions in Quantum Point Contacts// Phys. Rev. Lett. 2008. Vol. 100. P. 186801.
6. Tkachenko O. A., Tkachenko V. A., Kvon Z. D., Baksheev D. G., Portal J.-C., Aseev A. L. Steering of electron wave in three-terminal small quantum dot// 13th Int. Symp. Nanostructures: Physics and Technology (St. Petersburg, Russia, 2005.) Proceedings. P. 8–9; Tkachenko O. A., Tkachenko V. A., Kvon Z. D., Aseev A. L., Portal J.-C., Quantum interferential Y-junction switch, Nanotechnology, 2012, Vol. 23. P. 095202.
7. Tkachenko O. A., Tkachenko V. A., Renard V. T., Portal J.-C., Aseev A. L. Scanning Gate microscopy/spectroscopy of quantum channel with constriction: tip voltage controlled electron wave direction in Y-junction// 15th Int. Symp. Nanostructures Physics and Technology (Novosibirsk, 2007). Proceedings. Ioffe Institute. St.Petersburg. 2007, P. 297–298.
8. Tkachenko O. A., Tkachenko V. A., Baksheev D. G., Portal J.-C. Mesoscopical behavior of Aharonov-Bohm effect in small ring interferometer// 13th Intl. Symp. Nanostructures: Physics and Technology (St. Petersburg, 2005.), Proceedings. P. 205–206.
9. Миньков Г. М., Шерстобитов А. А., Ткаченко В. А., Ткаченко О.А. Переход к перколяции и прыжковой проводимости в структурах с двумерным разупорядоченным массивом антиточек// IX Российская конференция по физике полупроводников, Новосибирск-Томск. 2009. Тезисы докладов. С.108.
10. Ткаченко О.А., Ткаченко В.А., Portal J.-C., Квон З.Д., Латышев А.В., Асеев А.Л. Моделирование полупроводниковых квантовых наноустройств//Суперкомпьютерные технологии в науке, образовании и промышленности /Под редакцией: В.А. Садовничего, Г.И. Савина, Вл.В. Воеводина. М: Издательство Московского университета, 2010, С. 55–62; URL: `http://hpc-russia.ru/book2/8.pdf`
11. Cahay M., McLennan M., Datta S. Conductance of an array of elastic scatterers: A scattering-matrix approach// Phys. Rev. B 1988. Vol. 37. P. 10125–10136; Takagaki Y., Ferry D. K. Conductance of quantum waveguides with a rough boundary// J. Phys.: Condens. Matter. 1992. Vol. 4. P. 10421–10432.
12. Ando T. Quantum point contacts in magnetic fields// Phys. Rev. B 1991. Vol. 44. P. 8017–8027.
13. Usuki T., Saito M., Takatsu M., Kiehl R. A., Yokoyama N. Numerical analysis of ballistic-electron transport in magnetic fields by using a quantum point contact and a quantum wire// Phys. Rev. B 1995. Vol. 52. P. 8244–8255.

Параллельное вычисление оценки приближенно оптимальных управлений

О.В. Фесько

Институт Программных Систем им. А.К. Айламазяна РАН

В статье предложен метод расчета априорной оценки, позволяющей судить о качестве приближенного решения, полученного в ходе работы программы улучшения управления для задач оптимизации динамических систем [1]. Метод реализован в виде параллельного алгоритма, являющегося частью программного комплекса оптимизации динамических систем на множествах управлений. Проведено исследование масштабируемости параллельной реализации программы для решения задач химической технологии и биохимической инженерии.

1. Введение

При использовании различных численных методов решения задач оптимального управления крайне важное значение представляет собой возможность получения количественной оценки найденного приближенного управления, которая позволяет судить о точности, близости к точному оптимуму. Такую возможность открывают достаточные условия оптимальности Кротова [2].

В работе [1] рассматривалась задача оптимального управления вида

$$\begin{gathered}\dot{x}(t) = f(t, x(t), u(t)), \quad x(t_I) = x_I, \quad t \in [t_I, t_F], \\ u(t) \in D_u = \{u(t) |\, \underline{u} \le u(t) \le \overline{u}\}, \quad F(x(t_F)) \to \min,\end{gathered} \tag{1}$$

где $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$, $x_i(t), i = \overline{1,n}$ – кусочно-гладкие. Управление $u = (u_1, \ldots, u_p)^T \in \in \mathbb{R}^p$ принадлежит одному из двух классов: кусочно-линейному

$$u(t) = \frac{\left(\left(w^{2i+1} - w^{2i}\right)t - \left(\tau_i w^{2i+1} - \tau_{i+1} w^{2i}\right)\right)}{(\tau_{i+1} - \tau_i)}, \quad t \in [\tau_i, \tau_{i+1}], \tag{2}$$

или кусочно-постоянному $u(t) = w^i,\ t \in [\tau_i, \tau_{i+1}],\ i = \overline{0,m}$, где m — число моментов переключений при разбиении $t_I = \tau_0 < \tau_1 < \tau_2 < \ldots < \tau_{m+1} = t_F$. На параметры управления наложены ограничения типа $w^i \in W = \left\{w^i |\, \underline{u} \le w^i \le \overline{u}\right\}$.

Задача (1) была сведена к задаче условной конечномерной минимизации функции многих переменных $G(w, \tau)$ [3, 4]. Процесс решения полученной задачи состоит в поочередном применении численных алгоритмов: метода Рунге–Кутта для решения задачи Коши и комбинации метода Ньютона с модифицированным методом градиентного спуска для минимизации многоэкстремальной функции $G(w, \tau)$. Требуется количественно оценить полученное при использовании данного подхода приближенное управление.

2. Оценки приближенно оптимального решения

В силу рассматриваемых классов управлений непрерывную задачу (1) сведем к дискретной задаче оптимального управления. Введем множество индексов $K = \{0, 1, \ldots, m+1\}$ для моментов переключений $t_I = \tau_0 < \tau_1 < \tau_2 \ldots < \tau_{m+1} = t_F$, где m — число моментов переключений. В случае *кусочно-постоянного* управления $u(t) = w(k),\ t \in [\tau_k, \tau_{k+1}],\ k \in K$, эквивалентная непрерывной задаче (1) дискретная задача примет вид

$$\begin{gathered}z(k+1) = \mu(k, z(k), w(k)), \quad k \in \{0, 1, \ldots, m+1\}, \\ z(0) = z_0 = x_I, \quad w(k) \in W = \{w(k) |\, \underline{u} \le w(k) \le \overline{u}\}, \\ F(z(m+1)) \to \min,\end{gathered} \tag{3}$$

где функция $\mu\left(k, z(k), w(k)\right)$ — решение задачи Коши

$$\dot{x}(\xi) = f\left(\xi, x(\xi), w(k)\right), \quad \xi \in [\tau_k, \tau_{k+1}], \quad x(\tau_k) = z(k),$$

взятое в точке $\xi = \tau_{k+1}$.

Для случая *кусочно-линейного* управления $u(t) = w^1(k) + w^2(k)t,\ t \in [\tau_k, \tau_{k+1}]$, эквивалентная дискретная задача будет выглядеть как

$$\begin{aligned} &z\left(k+1\right) = \eta\left(k, z(k), w^1(k), w^2(k)\right), \quad k \in \{0, 1, \ldots, m+1\}, \\ &z(0) = z_0 = x_I, \quad w(k) \in W, \quad F\left(z(m+1)\right) \to \min, \end{aligned} \tag{4}$$

где функция $\eta\left(k, z(k), w^1(k), w^2(k)\right)$ — решение задачи Коши

$$\dot{x}(\xi) = f\left(\xi, x(\xi), w^1(k) + w^2(k)\xi\right), \quad \xi \in [\tau_k, \tau_{k+1}], \quad x(\tau_k) = z(k),$$

взятое в точке $\xi = \tau_{k+1}$.

Пусть $\left(\tilde{z}\left(k\right), \tilde{w}\left(k\right)\right)$ — допустимое решение задачи (3). Любой функции $\varphi\left(k, z\right)$ согласно [5] соответствует нижняя граница минимизируемого функционала $F(x(t_F))$ и оценка $\Delta\left(z, w, \varphi\right) \geq 0$ его близости к оптимуму. Запишем оценочную функцию Кротова для задачи (3):

$$\begin{aligned} \Delta\left(\tilde{z}, \tilde{w}, \varphi\right) = F\left(\tilde{z}\left(m+1\right)\right) - \min_{z \in \mathbb{R}^n} \left(F\left(z\right) + \varphi\left(m+1, z\right)\right) + \\ + \sum_{k=1}^{m} \max_{\substack{w \in W, \\ z \in \mathbb{R}^n}} \left(\varphi\left(k+1, \mu\left(k, z, w\right)\right) - \varphi\left(k, z\right)\right) + \max_{w \in W} \varphi\left(1, \mu\left(0, z_0, w\right)\right). \end{aligned} \tag{5}$$

Оценочная функция Кротова для задачи (4) имеет вид:

$$\begin{aligned} \Delta\left(\tilde{z}, \tilde{w}^1, \tilde{w}^2, \varphi\right) = F\left(\tilde{z}\left(m+1\right)\right) - \min_{z \in \mathbb{R}^n} \left(F\left(z\right) + \varphi\left(m+1, z\right)\right) + \\ + \sum_{k=1}^{m} \max_{\substack{w^1, w^2 \in W, \\ z \in \mathbb{R}^n}} \left(\varphi\left(k+1, \eta\left(k, z, w^1, w^2\right)\right) - \varphi\left(k, z\right)\right) + \\ + \max_{w^1, w^2 \in W} \varphi\left(1, \eta\left(0, z_0, w^1, w^2\right)\right), \end{aligned} \tag{6}$$

где $\left(\tilde{z}\left(k\right), \tilde{w}^1(k), \tilde{w}^2(k)\right)$ — допустимое решение задачи.

Если функция $\varphi(k, z)$ удовлетворяет условиям (схема Беллмана [2])

$$\begin{aligned} \varphi\left(k, z\right) = \max_{w^1, w^2 \in W} \varphi\left(k+1, \eta\left(k, z, w^1, w^2\right)\right), \\ \varphi\left(m+1, z\right) = -F(z), \quad k \in \{0, 1, \ldots, m+1\}, \end{aligned} \tag{7}$$

то формулы оценок принимают более простой вид:

$$\Delta\left(\tilde{z}, \tilde{w}, \varphi\right) = F\left(\tilde{z}\left(m+1\right)\right) + \varphi\left(0, z_0\right), \quad \Delta\left(\tilde{z}, \tilde{w}^1, \tilde{w}^2, \varphi\right) = F\left(\tilde{z}\left(m+1\right)\right) + \varphi\left(0, z_0\right) \tag{8}$$

вместо формул (5) и (6) соответственно. Управление, на котором достигается максимум в равенствах (7), может быть выбрано в качестве оптимального решения задач (3) и (4).

3. Описание алгоритма

Параллельный алгоритм поиска начального приближения для решения задачи оптимального управления (1) и вычисления количественной оценки приближенно оптимального управления представлен на схеме 1 и состоит из нескольких блоков.

На начальном этапе формулируется задача оптимального управления: задаются ее параметры, правые части системы дифференциальных уравнений $f(t, x(t), u(t))$ и терминальный функционал качества $F(x(t_F))$ в виде синтаксически правильных выражений языка C++. Далее исполняются следующие блоки:

Блок A: Вычисление начального приближения для ЗОУ.

Шаг 1. Задание параметров: начального состояния системы $z_0 = x_I$; ограничений на управление $\underline{u}, \overline{u}$; шага $\tilde{h}$ для построения расчетной сетки по управлению, моментов переключений τ.

Шаг 2. Построение сетки по управлению $\left\{\underline{u}, \underline{u} + \tilde{h}, \ldots, \overline{u} - \tilde{h}, \overline{u}\right\}$.

Шаг 3. Рекурсивно разрешается цепочка относительно функции Кротова $\varphi(k, z)$ на построенной сетке по управлению.

Шаг 4. Вычисление траекторий $\tilde{z}$, соответствующих найденному оптимальному управлению $\tilde{u}$.

Блок B: Улучшение управления [4].

Шаг 1. Задание начального состояния системы x_I; управления $\tilde{u}$ (найденного в *блоке* ***A***), области поиска, моментов переключений управления τ.

Шаг 2. При данном управлении численно интегрируется система обыкновенных дифференциальных уравнений (метод Рунге-Кутта 4-го порядка или метод Рунге-Кутта-Фельберга с адаптивным шагом).

Шаг 3. Вычисляется значение целевого функционала $G(\tilde{u}, \tau)$.

Шаг 4. Вызывается функция поиска очередного приближения к решению.

Шаг 5. Проверка условий останова. Если условия останова не выполняется, переходим к шагу 2, иначе *блок* ***B*** завершает свою работу.

Блок C: Вычисление оценки приближенно оптимального решения.

Шаг 1. Задание параметров: начального состояния системы $z_0 = x_I$; допустимого управления u^* (найденного в *блоке* ***B***), которое требуется оценить; ограничений на управление $\underline{u}, \overline{u}$; шага $\overline{h} \leq \tilde{h}$ для построения расчетной сетки по управлению, моментов переключений τ.

Шаг 2. Вычисление траекторий z^* и функционала качества $F(z^*)$, соответствующих заданному допустимому управлению.

Шаг 3. Построение сетки по управлению $\left\{u^*, \underline{u}, \underline{u} + \overline{h}, \ldots, \overline{u} - \overline{h}, \overline{u}\right\}$.

Шаг 4. Рекурсивно разрешается цепочка относительно функции Кротова $\varphi(k, z)$ на построенной сетке по управлению.

Шаг 5. Вычисление оценки согласно формулам (8).

Шаг 6. Вывод результатов в виде текстовых файлов и графиков. Завершение работы программы.

Шаги 3 и 4 рекурсивного вычисления функции Кротова $\varphi(k,z)$ блоков $\boldsymbol{A}$ и $\boldsymbol{C}$ соответственно могут выполняться независимо для различных наборов управлений, сгенерированных при построении сетки, за счет чего алгоритм допускает параллельное исполнение.

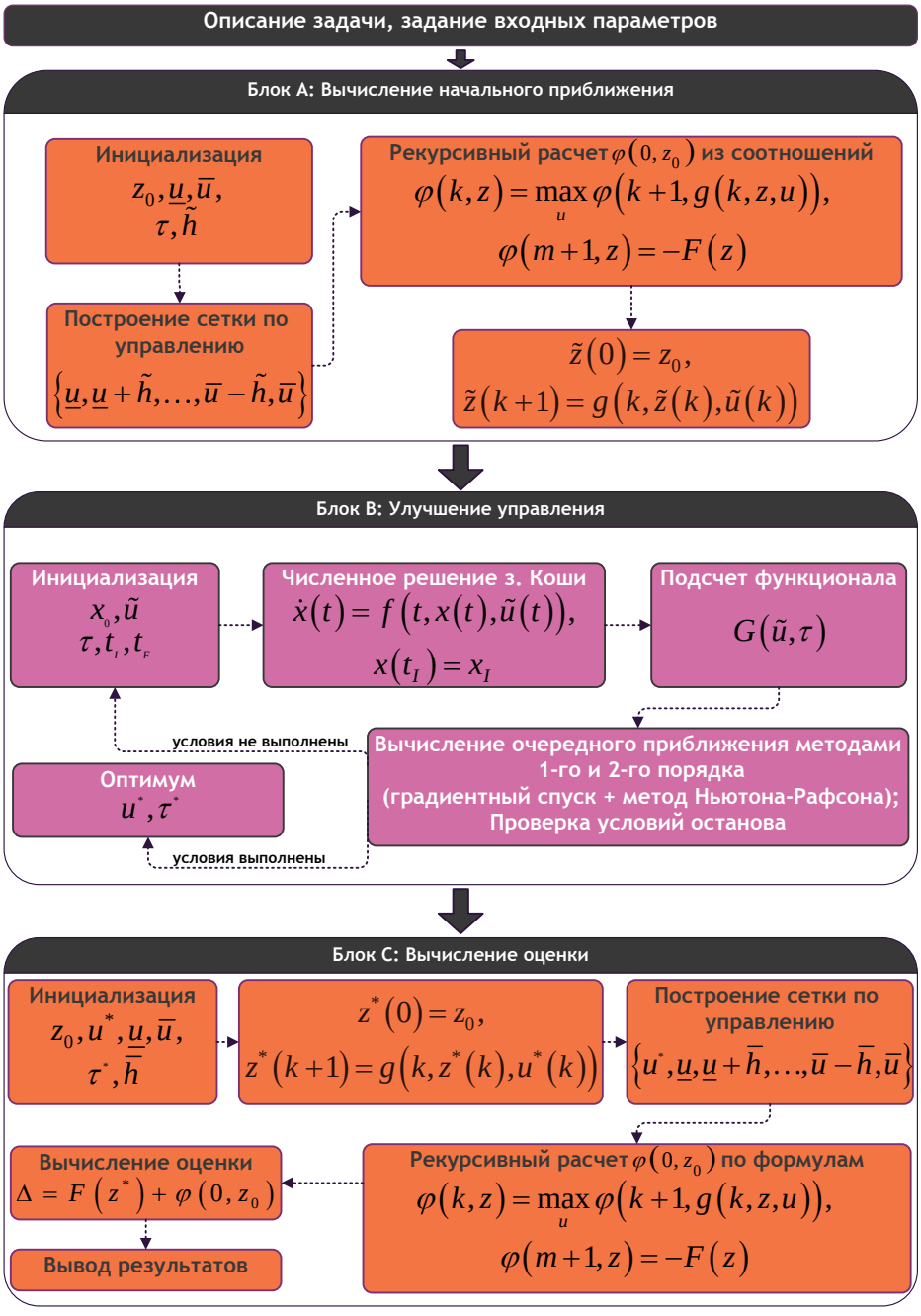


Рис. 1. Вычислительная схема решения поставленной задачи

Распараллеливание вычислительного процесса осуществляется по схеме процессорной фермы. Главный процессор считывает входные данные и формирует сетку по управлению, после чего распределяет наборы управлений между процессорами-подчиненными, на которых производится рекурсивный расчет функции Кротова; после этого главный процессор собирает результаты.

Параллельный алгоритм для решения задачи оптимального управления (1), в основе которого лежит геометрическая декомпозиция расчетной области, описан в работах [3, 4].

4. Вычислительные эксперименты

Программа решения задачи оптимального управления была реализована на языке C++, а в параллельной версии исполнена в среде OpenTS на языке T++ [6]. Преимущество данного подхода заключается в том, что данная система позволяет в динамике выполнять распараллеливание кусков кода программы, планировку вычислений, распределение данных по узлам и пр. без участия пользователя.

Расчеты проводились на высокопроизводительном вычислительном кластере «BLADE» Института Программных Систем РАН, оснащенном 8 вычислительными узлами с двумя процессорами Intel Xeon E5472 (4 ядра по 3.0 Ггц) и 16 ГБ оперативной памяти на каждом узле. Используя предложенный выше подход, решим следующие задачи.

4.1. Задача об оптимизации бифункциональной каталитической смеси

Рассматривается трубчатый реактор, в котором протекает процесс получения бензола из метилциклопентана. Процедура состоит из смешения двух монофункциональных каталитических компонентов, приводящих к реакциям гидрирования и изомеризации. Роль управления u играет массовая доля гидрирующего катализатора (отношение массы компонента гидрогенизации к общей массе катализатора). Характерное время t определяется отношением массы катализатора в конкретном разделе реактора к входному молярному расходу метилциклопентана. На выходе из реактора, финальный момент времени t_F определяется из отношения общей массы катализатора в реакторе к молярному расходу метилциклопентана в реакторе.

Задача оптимизации состоит в отыскании оптимальной каталитической смеси по всей длине реактора с целью максимизации концентрации бензола. То есть необходимо максимизировать функционал

$$F\left(x(t_F)\right) = x_7(t_F) \to \max,$$

где $t_F = 2000$ г $\cdot$ ч/моль.

Имеющие место химические реакции описываются системой дифференциальных уравнений

$$\begin{aligned}
&\dot{x}_1(t) = -k_1 x_1, \quad \dot{x}_2(t) = k_1 x_1 - (k_2 + k_3)x_2 + k_4 x_5,\\
&\dot{x}_3(t) = k_2 x_2, \quad \dot{x}_4(t) = -k_6 x_4 + k_5 x_5,\\
&\dot{x}_5(t) = k_3 x_2 + k_6 x_4 - (k_4 + k_5 + k_8 + k_9)x_5 + k_7 x_6 + k_{10} x_7,\\
&\dot{x}_6(t) = k_8 x_5 - k_7 x_6, \quad \dot{x}_7(t) = k_9 x_5 - k_{10} x_7,
\end{aligned} \tag{9}$$

где константы скорости протекания реакций выражаются кубическими функциями от управления u каталитической смеси $k_i = c_{i0} + c_{i1}u + c_{i2}u^2 + c_{i3}u^3,\ i = \overline{1,10}$. Коэффициенты $c_{ij},\ j = \overline{0,3}$ представлены в таблице 1.

Переменные состояния представляют собой массовые доли химических соединений: при $i = \overline{1,6}$ для метилциклопентана, при $i = 7$ — бензола. Начальное состояние системы задано: $x(0) = (1,0,0,0,0,0,0)^T$. На управление наложено ограничение: $0.6 \le u \le 0.9$.

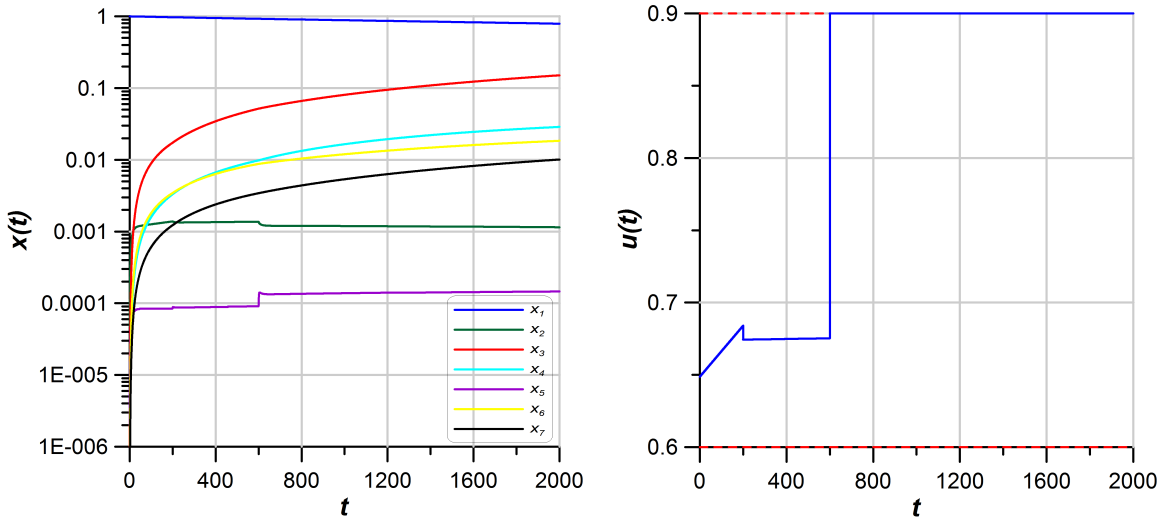
Задача имеет множество локальных максимумов [7]. Необходимо найти глобальный. Управление ищется в классе кусочно-линейных функций с двумя моментами переключений.

При поиске начального кусочно-линейного управления (***блок A*** схемы 1) с шагом по управлению $\tilde{h} = 0.1$ были найдены следующие параметры $\left(w^0, w^1, w^2, w^3, w^4, w^5\right) = (0.6, 0.7, 0.7, 0.7, 0.9, 0.9)$. Значение функционала составило $F = 0.0099729$. На этапе улучшения управления (***блок B***) найдено $\left(w^0, w^1, w^2, w^3, w^4, w^5\right) = (0.648, 0.683,\ 0.674, 0.675, 0.9, 0.9)$. При этом значение функционала $F = 0.0100956$ против $F = 0.0100942$ (см. [7]). При вычислении оценки приближенно оптимального управления (***блок C*** схемы) с шагом по управлению $\overline{h} = 0.05$ получено $\Delta = 0$. На рис. 2(a) и 2(b) изображены оптимальное кусочно-линейное управление и соответствующие ему траектории (на логарифмической шкале). В

Таблица 1. Коэффициенты для констант k_i, определяющих скорость реакций

i	c_{i0}	c_{i1}	c_{i2}	c_{i3}
1	0.2918487e-002	-0.8045787e-002	0.6749947e-002	-0.1416647e-002
2	0.9509977e+001	-0.3500994e+002	0.4283329e+002	-0.1733333e+002
3	0.2682093e+002	-0.9556079e+002	0.1130398e+003	-0.4429997e+002
4	0.2087241e+003	-0.7198052e+003	0.8277466e+003	-0.3166655e+003
5	0.1350005e+001	-0.6850027e+001	0.1216671e+002	-0.6666689e+001
6	0.1921995e-001	-0.7945320e-001	0.1105666e+000	-0.5033333e-001
7	0.1323596e+000	-0.4696255e+000	0.5539323e+000	-0.2166664e+000
8	0.7339981e+001	-0.2527328e+002	0.2993329e+002	-0.1199999e+002
9	-0.3950534e+000	0.1679353e+001	-0.1777829e+001	0.4974987e+000
10	-0.2504665e-004	0.1005854e-001	-0.1986696e-001	0.9833470e-002

таблице 2 приведены результаты времени счета задачи оптимизации бифункциональной каталитической смеси на суперЭВМ.



(a) Траектории (b) Управление

Рис. 2. Оптимальные процессы и управление

Таблица 2. Анализ эффективности параллельной версии программы

Число узлов, p	1	2	3	4	5	6	7
Время, t_p (с)	15473.371	7998.311	5555.059	4284.082	3469.615	3081.801	2609.322
Ускорение, t_1/t_p	1	1.93	2.78	3.61	4.45	5.02	5.93
Эфф-ть, $t_1/t_p/p$	1	0.97	0.92	0.9	0.89	0.84	0.85

4.2. Оптимальное производство белка в биореакторе

Рассмотрим модель по производству секретируемого белка в реакторе периодического действия с подпиткой, представленную в работе [8]. Эта модель использовалась некото-

рыми исследователями при разработке робастных методов решения задач такого характера. Трудности нахождения оптимального управления здесь частично связаны с низкой чувствительностью функционала к управлению. Биореактор описывается системой дифференциальных уравнений

$$\begin{aligned} &\dot{x}_1(t) = g_1(x_2 - x_1) - \frac{u}{x_5}x_1, \quad \dot{x}_2(t) = g_2 x_3 - \frac{u}{x_5}x_2, \\ &\dot{x}_3(t) = g_3 x_3 - \frac{u}{x_5}x_3, \quad \dot{x}_4(t) = -7.3 g_3 x_3 + \frac{u}{x_5}(20 - x_4), \\ &\dot{x}_5(t) = u, \end{aligned} \tag{10}$$

где $g_1 = \dfrac{4.75 g_3}{0.12 + g_3}$, $g_2 = \dfrac{x_4 e^{-5x_4}}{0.1 + x_4}$, $g_3 = \dfrac{21.87 x_4}{(x_4 + 0.4)(x_4 + 62.5)}$, с начальным состоянием $x(0) = (0, 0, 1, 5, 1)^T$. Здесь x_1 — концентрация секретируемого белка `SUC-s2` в культуре (л$^{-1}$), x_2 — общая концентрация белка `SUC-s2` в культуре (л$^{-1}$), x_3 — плотность клеток культуры (г/л), x_4 — уровень глюкозы в культуре (г/л), x_5 — объем культуры (л). Роль управления u играет скорость потока подпитки (л $\cdot$ ч$^{-1}$), $0 \le u \le 2$. Определение оптимальной скорости подпитки в реакторе для получения максимального количества желаемого продукта является очень сложной задачей оптимального управления.

Функционал составлен с целью максимизации количества секретируемого белка `SUC-s2`:

$$F\left(x(t_F)\right) = x_1(t_F)x_5(t_F) \to \max,$$

где $t_F = 15$ ч.

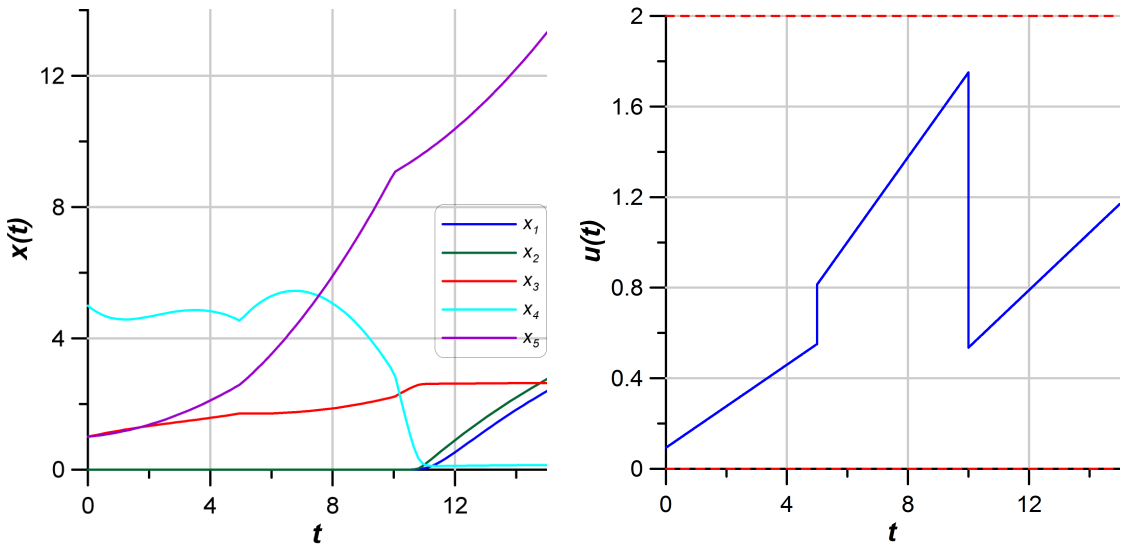
Управление ищется в виде (2) с двумя точками переключений. При поиске начального кусочно-линейного управления (***блок A***) с шагом по управлению $\tilde{h} = 0.25$ были найдены следующие параметры $\left(w^0, w^1, w^2, w^3, w^4, w^5\right) = (0, 0.75, 0.5, 2, 0.5, 1.25)$. Значение терминального функционала — $F = 31.62$. На этапе улучшения управления (***блок B***) найдено $\left(w^0, w^1, w^2, w^3, w^4, w^5\right) = (0.117, 0.526,\ 0.8, 1.74, 0.54, 1.14)$. При этом значение функционала составило $F = 31.82$ против $F = 32.67$ (см. [9], где функция управления искалась в кусочно-постоянном виде с 15 точками переключений). При вычислении оценки приближенно оптимального управления (***блок C***) с шагом по управлению $\overline{h} = 0.2$ получено $\Delta = 0$. На рис. 3(a) и 3(b) изображены оптимальное кусочно-линейное управление и соответствующие ему траектории. В таблице 3 приведены результаты времени счета оптимального производства белка в биореакторе на кластерной установке.

Таблица 3. Анализ эффективности параллельной версии программы

Число узлов, p	1	2	3	4	5	6	7
Время, t_p (с)	2281.76	1170.653	803.972	617.148	510.432	440.973	399.476
Ускорение, t_1/t_p	1	1.95	2.84	3.7	4.47	5.17	5.71
Эфф-ть, $t_1/t_p/p$	1	0.98	0.95	0.93	0.89	0.86	0.82

5. Заключение

В работе предложен параллельный алгоритм вычисления начального приближения для решения задач оптимального управления, а также расчета качества полученного решения в виде количественной оценки на основе достаточных условий оптимальности Кротова. Проведено исследование масштабируемости параллельной программы. Описанный алгоритм — неотъемлемая часть программного комплекса оптимизации динамических систем на множествах управлений простой структуры [3], в котором задействованы возможности современных суперЭВМ, что позволяет, в свою очередь, существенно сократить время вычислений.



(a) Траектории (b) Управление

Рис. 3. Оптимальные процессы и управление

Литература

1. Фесько О.В. Алгоритм поиска кусочно-линейного управления с нефиксированными моментами переключений // Вестник БГУ. Вып. 9: Математика и информатика, Улан-Удэ: Изд-во Бурят. госун-та, 2011. С. 52–56.

2. Кротов В.Ф., Гурман В.И. Методы и задачи оптимального управления. Наука, М., 1973.

3. Фесько О.В. Программный комплекс поиска оптимальных управлений на множествах простой структуры // Параллельные вычислительные технологии (ПаВТ2011): Труды международной научной конференции / Москва. Челябинск: Издательский центр ЮУрГУ, 2011. С. 712.

4. Фесько О.В. Параллельный алгоритм оптимизации динамических систем на множестве кусочно-линейных управлений // Вестник Бурятского государственного университета. Вып. 9: Математика и информатика, Улан-Удэ: Изд-во Бурят. госун-та, 2010. С. 79–87.

5. Трушкова Е.А. Оценка приближенно оптимальных решений на основе преобразований модели объекта // Вестник БГУ. Вып. 9: Математика и информатика, Улан-Удэ: Изд-во Бурят. госун-та, 2011. С. 47–51.

6. Moskovsky A., Roganov V., Abramov S. Parallelism granules aggregation with the T-system // 9th International Conference on Parallel Computing Technologies. LNCS 4671, 2007. P. 293–302.

7. Luus R., Dittrich J., Keil F.J. Multiplicity of solutions in the optimization of a bifunctional catalyst blend in a tubular reactor //Can. J. Chem. Eng. 70. 1992. P. 780–785.

8. Park S., Ramirez W.F. Optimal production of secreted protein in fed-batch reactors //AIChE J. 34. 1988. P. 1550–1558.

9. Luus R. Iterative Dynamic Programming. Boca Raton, Chapman and Hall/CRC 2000. 344 p.

Сортировка массивов коротких последовательностей на GPU для метода сортировки взаимодействий в молекулярной динамике*

Э.С. Фомин

Институт Цитологии и Генетики СО РАН

Необходимость сортировки большого числа коротких массивов с числом элементов в диапазоне $30 \leq n \leq 120$ возникает в молекулярной динамике в методе сортировки взаимодействий. Представлена модификация алгоритма ранговой сортировки для решения данной задачи, реализованная для платформы GP GPU (Tesla C2050). Показано, что предложенная модификация алгоритма в данном диапазоне числа элементов превосходит по эффективности общедоступные для данной платформы библиотечные алгоритмы сортировок.

1. Введение

Расчеты ближних несвязующих взаимодействий в молекулярной динамике (МД) занимают существенную, до 90%, часть от общего времени выполнения [1], что обусловлено следующими факторами:

- по сравнению с расчетами связующих взаимодействий резко (в десятки раз) возрастает число взаимодействий, приходящихся на один атом, что обусловлено необходимостью учета влияния всех $(4\pi/3)\rho r_{cutoff}^3$ атомов, распределенных с плотностью ρ в сфере радиуса, равного расстоянию отсечения взаимодействий r_{cutoff};
- заметно увеличивается неоднородность данных (в сфере взаимодействия оказываются атомы разных типов);
- в потоке данных, собранных из памяти и доставленных на вычислительное устройство, существенно возрастает доля незначимых пар атомов, то есть таких, которые при проверке расстояний приходится отбрасывать;
- существенно возрастает количество промахов кэша, поскольку атомы, хранимые в соседних ячейках памяти компьютера, в силу своего непрерывного движения в процессе моделирования могут существенно разойтись по координатам в пространстве и выйти из области учета взаимодействий.

Таким образом, поток данных, попадающий на вычислительные устройства на этапе расчетов несвязующих взаимодействий, в случае неудачного подбора алгоритмов, отсутствия их адаптации для конкретных вычислительных платформ или, как крайний случай, низкой квалификации пользователя при установке параметров моделирования характеризуется низкой плотностью «полезных» данных, что неизбежно сказывается на эффективности расчетов.

Высокую эффективность расчетов в молекулярной динамики могут обеспечить только адаптивные схемы организации вычислительного процесса, которые позволяют существенно увеличить вероятность нахождения нужных данных в потоке за счет подстраивания схемы вычислений под текущее состояние моделируемой системы и платформу выполнения.

*Данная работа поддержана грантами СО РАН: интеграционные проекты №39, №47, №130, а также Министерством образования и науки РФ (Госконтракты №П857, №07.514.11.4011). В работе использованы ресурсы Суперкомпьютерного комплекса МГУ «ГрафИТ!» в рамках конкурса «Эффективное использование GPU-ускорителей при решении больших задач», проводимого компанией «Т-Платформы».

В настоящее время такие адаптивные схемы организации вычислений используют метод связанных ячеек (LC - linked cells) [2] для поиска ближайших соседей, метод сортировки взаимодействий (IS - interaction sorting) [3] для ограничения доли незначимых пар атомов, метод Верлет таблицы (VT - Verlet table) [4] для хранения временно взаимодействующих пар, методы переодического переупорядочения атомов (LCR - linked cell based reordering) [5] для минимизации промахов кэша и автоматическую, «на лету», подстройку параметров расчетной схемы [8] под максимальную эффективность путем контролирования времени работы различных шагов алгоритма. Последние способы дополнительно позволяют избегать деградации выполнения, то есть замедления расчетов, в длительной молекулярной динамике.

Сущность вышеупомянутых методов заключается в следующем:

- *LC - метод связанных ячеек.* Для поиска ближайших соседей используется разделение пространства моделирования на пространственные ячейки с длиной ребра большей или равной радиусу отсечения взаимодействий r_{cutoff}. Для каждой ячейки составляется список соседних к ней ячеек (связанные ячейки). Все атомы системы распределяются по ячейкам согласно своим координатам. Поиск соседей для любого атома выполняется в ячейке самого атома, а также в связанных с ней ячейках.

- *IS - метод сортировки взаимодействий.* Метод связанных ячеек приводит к излишним расходам по оценке расстояний между парами атомов. Действительно, в ячейке с ребром равным r_{cutoff} и в 26 связанных с нею ячейках находится $N_1 = 27\rho r^3_{cutoff}$ атомов, при этом в пределах r_{cutoff} находится всего $N_2 = (4\pi/3)\rho r^3_{cutoff}$ атомов. Таким образом, во всех связанных ячейках доля взаимодействующих атомов достаточно мала $(N_2/N_1)\times 100\% \sim 16\%$. С целью избежать излишних проверок расстояний атомы сортируются согласно их координатам вдоль оси, соединяющей любые две соседние ячейки. Имея отсортированные последовательности атомов двух соседних ячеек при расчетах расстояний, возможно отбрасывать концы последовательностей, если расстояние между проверяемыми атомами превосходит r_{cutoff}.

- *VT - метод Верлет таблицы.* Найденные пары взаимодействующих атомов сохраняются в таблице, в которой для каждого атома хранится список атомов находящихся на расстоянии, не превышающей $r_{cutoff} + r_{skin}$, где r_{skin} - страховочная область. Регулируя ширину страховочной области можно подобрать оптимальное соотношение между временем расчета взаимодействий и частотой перестройки таблицы. Необходимость перестройки таблицы возникает всякий раз, когда какой-либо атом, в результате своего движения перешел пределы страховочной области r_{skin}.

- *LCR - метод переупорядочения атомов.* Все атомы и их таблицы параметров взаимодействий пересортировываются в памяти, согласно текущим координатам таким образом, чтобы атомы близкие в пространстве были близки по адресам памяти. Такая пересортировка выполняется через заданный интервал модельного времени системы, который пропорционален подвижности молекул системы.

В любой адаптивной схеме для организации необходимого порядка обработки данных интенсивно используются различного вида сортировки. Именно благодаря сортировкам атомов по пространственным ячейкам снижается вычислительная сложность поиска ближайших соседей с $O(N^2)$ до $O(N)$ (метод LC [2], комбинированный метод LC + VT [9]), существенно увеличивается доля взаимодействующих пар в потоке данных от $\sim 16\%$ до $\sim 60\%$ (метод IS [3]) и обеспечивается поддержка пространственной и временной локальности данных повышая вероятность нахождения данных в кэше с 90.9% (без упорядочения) до 99.4% на процессорах стандартной архитектуры x86 (метод LCR [5]).

Любая сортировка данных требует соответствующих накладных расходов, и может быть использована только, если выгода от ее использования превышает потери. Сортиров-

ка взаимодействий увеличивает общую производительность молекулярной динамики для реализаций, основанных на широкораспространенном комбинированном методе LC + VT до 10% (за счет ускорения этапа построения таблицы более чем в два в половиной раза) [6]. Сортировка взаимодействий более чем в два раза увеличивает общую производительность динамики для реализаций, основанных на оригинальном методе LC [7], что позволяет последнему при расчетах плотных систем с высокой мобильностью атомов превзойти по эффективности комбинированный метод CL + VT. Относительные потери на сортировку в обоих подходах не превышают 5-10% для соответствующих этапов алгоритма для последовательных версий и увеличиваются до 15% для векторизованных SSE версий [8]. Увеличение относительной доли затрат на алгоритмы сортировки в векторизованом коде обусловлено невозможностью их эффективной векторизации.

Метод сортировки взаимодействий, обладая доказанной эффективностью для стандарной архитектуры x86, не гарантированно даст эффект при его использовании в программах молекулярной динамики, выполняющихся на GPU. Наиболее существенной проблемой является необходимость многочисленных сортировок данных, и далеко не очевидно, что GPU позволит обеспечить необходимую эффективность. Основные особенности алгоритма сортировки взаимодействий, которые необходимо учитывать при портировании, следующие:

- Сортировке подвергаются массивы с весьма небольшим числом элементов в них. Действительно, среднее число молекул воды на любую пространственную ячейку размером $10 \times 10 \times 10$ Å^3 при нормальной плотности не превышает ~ 30 (полное число атомов ~ 90).

- Общий объем необходимых сортировок значителен, то есть число небольших по ~ 30 элементов последовательностей велико. Действительно, общее число определяется числом ячеек, на которые разбита пространственная область $[N_x, N_y, N_z]$ и числом пар, которые образует любая ячейка с соседями, то есть 26. Таким образом, полное число массивов для сортировки равно $26N_xN_yN_z$, что для среднего расчета достигает величины в несколько десятков тысяч. При этом такие сортировки требуется выполнять на каждом шаге МД.

Итого, для обеспечения эффективности молекулярной динамики на аппаратуре, позволяющей «мелкозернистый» параллелизм (векторные операции с большим числом компонент, либо огромное число легких потоков, как на GPU) необходимо построение алгоритма сортировки большого объема коротких последовательностей. В данной работе анализируется возможность применения алгоритма ранговой сортировки на платформе GPU для решения этой задачи. Выбор данного алгоритма обусловлен тем, что наряду с тем, что он имеет модификации с вычислительной сложностью $O(N \times logN)$, такой же как у любых прочих эффективных алгоритмов сортировки, он имеет «соблазнительную» теоретическую возможность выполняться за время O(1). А когда есть выбор - допустить простаивание потоков или бросить их все на выполнение задачи пусть и неэффективным $O(N^2)$ образом, результат выбора очевиден.

2. Ранговая сортировка последовательности

Алгоритмы ранговой сортировки [10] основаны на вычислении рангов элементов и их упорядочении в соответствии с полученным рангом. Ранги элементов могут быть определены любым способом, однако в практике зачастую достаточным является следующая формула вычисления ранга:

$$rank(s_i) = \sum_j rank(R(s_i, s_j))$$

где ранжируются результаты $R(s_i, s_j)$ операций попарного сравнения элементов и выполняется суммирование полученных рангов. В общем случае, в зависимости от выбора операции

сравнения, значение ранга может принимать как целочисленное, так и вещественное значение. Заметим, что для наиболее естественного выбора в качестве операции сравнения R операции «<» (меньше) значение ранга имеет булевский тип, который транслируется в целочисленный, согласно преобразованию $true \to 1$ и $false \to 0$. Если операция R между элементами неопределенна, то ее ранг принимается равный нулю. Поскольку для получения ранга элемента при таком выборе функции ранжирования требуется попарное сравнение всех элементов, то вычислительная сложность таких алгоритмов равна $O(N^2)$, где N - число элементов в сортируемой последовательности. Такая высокая вычислительная сложность означает неприменимость этих алгоритмов в практических расчетах систем с огромным числом элементов $N >> 1$.

В случае, если операция сравнения элементов последовательности является транзитивной, то исчезает необходимость сравнения всех $O(N^2)$ пар. В этом случае возможно использование подхода «разделяй и властвуй», что позволяет для алгоритмов ранговой сортировки строить схемы с вычислительной сложностью $O(N \times logN)$, как и для стандартных алгоритмов быстрой сортировки, типа $qsort$ или $merge$ [10]. Примером подобного алгоритма является алгоритм, описанный в [11] и реализованный для платформы GPU.

Однако при использовании вычислительной платформы с «идеальным» параллелизмом, алгоритмы ранговой сортировки превосходят по скорости вычислений стандартные алгоритмы, поскольку имеют временную сложность $O(1)$ [12]. Здесь под «идеальной» параллельной системой понимается система, которая может обеспечить сколь угодно большое число реально одновременно выполняемых потоков и поддерживает на машинном уровне параллельные операции для векторов, такие как scan, reduce, transposition [13]. Причем важным условием является то, что данные параллельные операции выполняются за такое же время, как и любая скалярная операция, например за один такт. Для такой идеальной параллельной системы алгоритм ранговой сортировки выполняется всего за 3 операции:

1. расчет рангов операции сравнения с помощью N^2 потоков для всех пар элементов множества и формирование матрицы, в которой на пересечении k-строки с m-столбцом находится результат сравнения элементов a_k и a_m;

2. выполнение операции reduce (суммирование) над всеми рядами полученной матрицы с формированием вектора, содержащего ранги всех элементов последовательности;

3. выполнение операции transpose для переупорядочения элементов последовательности в соответствии с вектором рангов.

Поскольку любая реальная параллельная вычислительная система является приближением к «идеальной», то достижение вычислительной сложности $O(1)$ невозможно. Причины этого очевидны, поскольку:

1. реальная параллельная система может обеспечить большое, но ограниченное число параллельных потоков,

2. даже при поддержке на машинном уровне параллельных операций для векторов, вряд ли реализация параллельных операций, требующих суммирования, таких как scan и reduce, будет выполняться за то же время, что и скалярные операции.

Первое возражение не является существенным, поскольку для задачи сортировки N элементов достаточно N^2 потоков. Например, для задачи сортировки коротких последовательностей длиной не более 32 элементов достаточно иметь 1024 потока, что уже могут обеспечить GPU устройства с compute capability версии 2.0 и выше [14]. Второе возражение существенно, поскольку в настоящее время даже наиболее востребованные параллельные операции, типа scan и reduce, реализованы с помощью библиотечных функций, например библиотеки CUDA C SDK [16], и включают циклы по элементам обрабатываемых последовательностей.

Таким образом, хотя теоретическая временная сложность алгоритма равна $O(1)$, ожидаемая временная сложность реализации алгоритма, как описано ниже, может быть равной $O(log^2 N)$.

В данной работе выполнена реализация алгоритмов ранговой сортировки для платформы Tesla C2050 с помощью технологии CUDA. Выполненно сравнение эффективности разработанного алгоритма с эффективностью наиболее широко используемых параллельных алгоритмов сортировки, которые имеют временную сложность $O(log^2 N)$, таких как $mergeSort$, $bitonicSort$, $odd-even$, входящими в пакет CUDA SDK [16]. Сравнение выполнено как для диапазона $1 \leq n \leq 32$ элемента, в котором теоретическая временная сложность равна $O(1)$ и для диапазона $32 \leq n \leq 1024$, где теоретическая временная сложность пропорциональна $O(N)$.

2.1. Сортировка последовательности целых чисел без дубликатов

Временная сложность $O(1)$ для сортировки последовательности чисел достигается при условии, что число доступных потоков выполнения не менее чем N^2, где N - число элементов в последовательности. Такое число потоков необходимо, чтобы за один шаг можно было получить таблицу сравнения $R(s_i, s_j)$ для всех пар элементов s_i и s_j.

Ранг операции сравнения выбирается как $rank(s_i, s_j) = (bool)(s_i > s_j)$. В матрицу сравнения записываются значения целые числа $\{0, 1\}$ представляющие булевские значения $\{false, true\}$. Ранг элемента рассчитывается как сумма значений в рядах матрицы $rank(s_i) = \sum_j rank(s_i, s_j)$.

Если некоторая последовательность S упорядочена, $s_0 \leq s_1 \leq .. \leq s_{n-1}$, то для любого ее элемента s_k выполняется: $s_k > s_0(true), s_k > s1(true), ..s_k > s_{k-1}(true), s_k > s_k(false), s_k > s_{k+1}(false), s_k > s_{n-1}(false)$. То есть, полное число парных сравнений данного элемента со всеми остальными элементами последовательности, которое имеет значение равное true равно индексу элемента в данной последовательности. То есть, выполняется формула $\sum_i rank(s_k, s_i) = k$. Значение ранга элемента не изменится, если последовательность будет переупорядочена произвольным образом, поскольку это приведет к перестановке элементов в сумме, но не к изменению результата, поскольку результат операция сложения коммутативен. Таким образом, ранг элемента в последовательности не зависит от упорядочения последовательности. Этот факт позволяет легко находить для любого элемента неупорядоченной последовательности его позицию в упорядоченной последовательности. Для этого достаточно просуммировать все значения парных сравнений данного элемента со всеми остальными элементами.

Таким образом, возвращаясь к алгоритму, чтобы получить позицию любого элемента в отсортированной последовательности необходимо суммировать все значения в рядах матрицы сравнений. Для идеальной параллельной машины такое суммирование выполняется за одну параллельную операцию, типа scan или reduce. Для реальной машины подобная параллельная операция реализуются либо программными средствами, либо на машинном уровне. В обоих случаях эффективный алгоритм такого суммирования требует $log^2 N$ шагов. Поскольку все суммирование по рядам может выполняться независимо и число рядов меньше числа потоков $N < P$, то суммарное время получения рангов всех элементов не изменяется и равно $log^2 N$.

Последним этапом данного алгоритма является восстановление правильного порядка элементов последовательности. Это также может быть сделано за один шаг, поскольку знание индекса элементов позволяет делать их запись независимо друг от друга. Итого, полное число шагов, необходимых алгоритму равно 3 для идеальной параллельной машины, либо для реальной параллельной машины определяется шагом суммирования, то есть пропорционально $log^2(N)$.

Алгоритм является эффективным только в случае, когда число доступных потоков подчиняется условию $P \geq N^2$. Если оно нарушается, то невозможно будет получить полную

таблицу парных сравнений за один шаг, что в свою очередь в худшую сторону изменит эффективность алгоритма. Действительно, если допустить, что число потоков является недостаточным, чтобы обработать таблицу за один шаг, то есть выполняется $P < N^2$, но достаточно, чтобы обработать за один шаг хотя бы один ряд целиком, то есть $P > N$, то таблица парных сравнений должна обрабатываться частями. Полная высота обрабатываемой части таблицы (при условии ее ширины N) равна $k = P/N$, а число частей таблицы равно $m = N/k$, или после подстановки значения k величина m равна N^2/P. Учитывая, что число шагов для нахождения позиций элементов не меняется $log^2(N)$, получаем, что полное число шагов равно $(N^2/P) \times log^2(N)$. Таким образом, эффективность данного алгоритма при числе потоков P, удовлетворяющем условию $N \leq P \leq N^2$, находится в диапазоне $[N \times log^2(N), log^2(N)]$.

2.2. Сортировка последовательности целых чисел с дубликатами

Наличие дубликатов в последовательности приводит к неработоспособности вышеприведенного алгоритма. Это происходит по той причине, что все дубликаты имеют один и тот же ранг. Как результат, запись элементов исходной последовательности по позициям, определяемым рангами, приводит к тому, что в конечной последовательности появляются позиции, в которые вообще не производится запись. То есть, в пересортированной последовательности появляются элементы, значение которых не определено. Легко понять, что позиции с неопределенными элементами появляются сразу за элементом, имеющим дубликаты, и их число равно n_d, где n_d - число дубликатов (см. рис. 1). Под числом дубликатов здесь и далее будем понимать полное число элементов, совпадающих по значению, без включения исходного элемента. Дополнительно ограничимся также и тем предположением, что в последовательности находятся дубликаты только с одним значением. Последнее условие никак не ограничивает корректность нижеприведенных рассуждений, которые верны и в случае дубликатов с различными значениями, однако это упрощает описание.

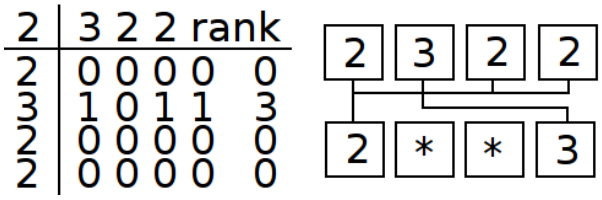


Рис. 1. Пример таблицы сравнений для последовательности целых с дубликатами (слева) и результирующая последовательность после обработки алгоритмом (справа)

Идея, позволяющая разрешить проблему с дубликатами, заключается в том, чтобы найти такое преобразование матрицы сравнений $U \to U'$, которое с одной стороны изменит конечные ранги дубликатов, упорядочив их в области $[rank(s), rank(s) + n_d)$, где $rank(s)$ - ранг дубликата s и n_d - число дубликатов, и с другой стороны не изменит ранги элементов без дубликатов.

Сделаем следующее модификацию определения ранга сравнения, определив его как $rank_{new}(s_i, s_j) = s_i - s_j$. Очевидно, что операция $bool(x) = 1, x > 0|0, x \leq 0$ приводит модифицированный ранг $rank_{new}$ к значению исходного ранга $bool(rank_{new}(s_i, s_j)) = rank(s_i, s_j)$, то есть при таком расширении не потеряна информация об упорядочении элементов. Операции $bool(x)$ имеет следующее свойство: $bool(x+n) = bool(x)$, для всех $|x| > n$, где n - целое положительное число, а операция $|x| = x, x > 0| - x, x < 0$ есть операция взятия абсолютного значения. Таким образом, матрица сравнений допускает добавление любого числа n, к элементу s_{ij} в любой позиции, тем самым изменяя модифицированный ранг, но без изменения значений исходных рангов элементов при условии, что число $n < |s_{ij}|$. Поскольку для дубликатов $|s_{ij}| = 0$, то для них при таком преобразовании ранг меняется всегда. Разное по-

ведение рангов для элементов с дубликатами и без них позволяет построить преобразование матрицы сравнения, котором позволяет изменить ранги дубликатов с сохранением прежних рангов у всех остальных элементов. Легко проверить, что матрица T, состоящая из одних единиц и с нулями на главной диагонали обладает требуемым свойством. Действительно:

- Для элементов без дубликатов вне главной диагонали возможны два варианта $s_{ij} < 0$ и $s_{ij} > 0$. Добавление 1 к целому отрицательному числу даст либо отрицательное значение, либо нуль, то есть не изменит значение функции *bool*. Если $s_{ij} > 0$, то значение функции *bool* также не меняется при увеличении s_{ij}.

- Для элементов с дубликатами $s_{ij} = 0$. При добавлении 1 значение функции *bool* меняется: $bool(0) = 0$ и $bool(1) = 1$.

К сожалению, преобразование $U = U + T$, изменяет ранги всех дубликатов одинаково. Это изменение равно $rank(s) \to rank(s) + n_d$, где n_d - число дубликатов. Действительно, в ряду i ранги сравнений меняются только у дубликатов (за исключением лежащего на главной диагонали), а так как изменение равно 1 и их число n_d, то ранг элемента меняется на n_d. Это не то, что хотелось бы от алгоритма. От алгоритма требуется, чтобы ранг первого дубликата не менялся, второго изменился на 1, третьего на 2 и т.д.

Чтобы обеспечить различное изменение ранга дубликатов, заметим, что допустимы дополнительные модификации матрицы T, сохраняющие ранги элементов без дубликатов, а именно удаление единиц с любой позиции матрицы T. Действительно, если позиция с которой удаляется 1 соответствует позиции, не связанной с дубликатами, то ранг элемента на ней вообще не изменяется при замене 1 на 0. Однако такие модификации матрицы T для позиций, связанных с дубликатами, приводят уменьшению ранга $rank(s) \to rank(s) - 1$. Последнее свойство позволяет построить матрицу T, которая обладает нужными свойствами. Для этого нужно оставить один ряд неизменным, затем убрать одну единичку из любого оставшегося ряда матрицы T, затем - две единички из любого из оставшихся рядов и т.д. Порядок удаления единички в ряду и порядок рядов несущественен. Таким образом, возможно получить $(N-1)! \sum(1/k!)$ матриц T. Поскольку любая из таких матриц годится, выберем наиболее простую из них, а именно ту, которая получается, если первый ряд не менять, из второго убрать единичку слева от главной диагонали, из второго — две единички слева от главной диагонали и т.д., то есть получить верхнетреугольную матрицу.

Полный алгоритм выглядит следующим образом:

1. Составляем матрицу сравнений U с использованием ранга $rank(s_i, s_j) = s_i - s_j$.

2. Делаем преобразование $U \to U + T$, где T - верхнетреугольная матрица, состоящая из единиц, с нулями на главной диагонали.

3. Выполняем операцию *bool* над каждым элементом матрицы U, то есть $U \to bool(U)$.

4. Вычисляем ранг каждого элемента исходной последовательности S по алгоритму, описанному в разделе 2.1.

2.3. Сортировка последовательности вещественных чисел без дубликатов

Выполняется по алгоритму, описанному в разделе 2.1.

2.4. Сортировка последовательности вещественных чисел с дубликатами

1. Рассчитываем ранги элементов по алгоритму, описанному в разделе 2.1. Полученная последовательность рангов $I = i_0, i_1, .., i_{n-1}$ является неупорядоченной последовательностью целых чисел с дубликатами.

2. Для полученной последовательности I рассчитываем ранги I' по алгоритму, описанному в разделе 2.2.

3. Пересортировываем исходную последовательность согласно рангам I', полученным на шаге 2.

3. Детали реализации

Алгоритм ранговой сортировки реализован на языке C++ с использованием CUDA Toolkit 3.2 на базе программы MOLKERN [15]. Реализованы три версии алгоритма «1», «lgN» и «N»:

- «1» версия, которая использует максимальное доступное число потоков ядра (1024) таким образом, чтобы достичь временной сложности близкой $O(1)$ на последовательностях до 32 элементов;
- «lgN» базовая версия, построенная на подходе «разделяй и властвуй» по аналогии с [11], с вычислительной сложностью $O(log^2 N)$ для обработки последовательностей до 1024 элементов;
- «N» дополнительная версия, построенная на полном парном сравнении всех элементов последовательностей с числом используемых потоков равном числу элементов последовательности, то есть с вычислительной сложностью $O(N)$ в исследуемом диапазоне.

Дополнительная версия использована только для сравнительной демонстрации временной сложности. Все версии алгоритма обрабатывают последовательности с длиной равной $2k$, где k - целое число. Операции reduce и transform были реализованы программным образом, при этом все данные для них размещались в разделяемой памяти. Так что для последовательностей длиной менее 32 элементов временная сложность операции reduce в лучшем случае составляла $O(log^2 N)$, а операции transform - $O(1)$. В худшем случае для операции transform, поскольку для произвольных последовательностей нельзя гарантировать какой либо определенный порядок доступа к памяти, временная сложность могла составлять $O(N)$, по причине наличия конфликтов блоков при обращении к разделяемой памяти.

Время выполнения сортировки измерялось функциями тайминга, которые предоставляются SDK. Поскольку время измеряется на процессоре общего назначения и включает в себя время загрузки/выгрузки ядра, то при замерах ядро принудительно выполнялось многократно (не менее 1000 раз), тем самым относительный вклад от времени загрузки/-выгрузки ядра падал в 1000 раз.

Для использованных примеров сортировки был написан выполняемый на ядре параллельный генератор случайных чисел. Был использован алгоритм генерации случайных чисел Парка-Миллера [17]. Для уничтожения нежелательных корреляций между сериями случайных чисел, генерируемых разными потоками, на ядро передавался массив различных простых чисел для инициализации генераторов. Размерность массива совпадала с максимально возможным числом потоков на ядре.

Расчеты выполнялись на процессоре Tesla C2050. Данный процессор может обеспечить 1024 потоков на ядро. Таким образом, «1» версия алгоритма может показать время близкое к константному только для последовательностей с числом элементов не выше 32. При большем числе элементов в последовательностях его временная сложность является квадратичной $O(N^2)$. Дополнительная версия «N» алгоритма обладает линейной временной сложностью $O(N)$ вплоть до числа элементов в последовательности не превышающем максимально возможного числа потоков на ядро. При большем числе элементов данная версия обладает также квадратичной временной сложностью $O(N^2)$.

В реализации использовались следующие модификации, отличающие ее от алгоритма [11]:

- Внесено отображение вещественных чисел (координаты молекул в ячейках) на короткие целые.
- Полученное целое значение записывается в верхние 2 байта целого числа, а номер элемента в последовательности записывается в нижние 2 байта. Данный прием ликвидирует косвенность обращения к данным при сортировке, и позволяет избежать проблемы с сортировкой дубликатов [12], которые возникают в случае отображения разных вещественных чисел на одинаковые целые.
- Используется двух-этапный подход (создается план сортировки по базисным атомам молекулы и план сортировки используется для всех остальных атомов тех же молекул).

Выполненная реализация алгоритма оптимизирована для случая длины последовательностей в пределах 256 элементов, для последовательностей длиной менее 32 элементов дополнительно исключена синхронизация между варпами. Тестирование выполнялось на коротких последовательностях элементов, длиной не более 256 элементов. Выполнялось сравнение полученного алгоритма со стандартными алгоритмами сортировок из библиотеки CUDA SDK.

4. Результаты расчетов

Скорость работы разработанного алгоритма и эффективность полученной реализации сравнивалась с эффективностью работы различных сортирующих алгоритмов, предоставленных CUDA SDK. Эти алгоритмы включали в себя сортировку слиянием, radix сортировку и сортирующие сети (битональную и четно-нечетную) [16]. Результаты сравнения приведены на рис. 2. Такое сравнение позволяет оценить качество реализации алгоритма ранговой сортировки и соответствие оценок и реальных значений теоретической сложности.

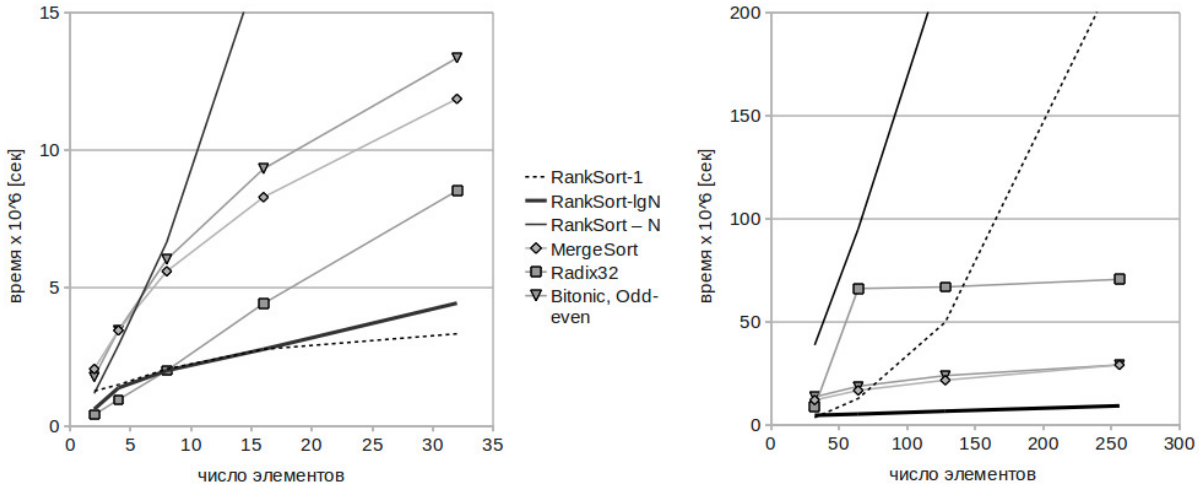


Рис. 2. Время выполнения сортировки коротких последовательностей для различных методов: (слева) число элементов до 32, (справа) число элементов до 256

Сравнение времени работы сделано для трех вариантов «1», «lgN» и «N» алгоритма ранговой сортировки. Вариант «1» соответствовал реализации, в которой использовались все доступные в ядре потоки, то есть 256 для Tesla C2050. Данный вариант эмулирует счет с временной сложностью $O(1)$. Вариант «N» является дополнительным, в нем число используемых потоков соответствовало числу сортируемых элементов, он эмулирует вычисления с временной сложностью $O(N)$. И вариант «lgN» соответствует реализации, в которой используется свойство транзитивности элементов и подход «разделяй и властвуй». Его временная

сложность равна $O(log^2(N))$. Сравнение сделано для сортировки последовательностей длиной до 256 элементов и состоящей из вещественных чисел одинарной точности $float$.

На рис. 2 видно, что варианты «N» и «lgN» показывают временную сложность совпадающую с теоретической. Вариант «1» при длине последовательности до 32 элементов, то есть до той длины, когда все возможные 1024 потока могут быть использованы для расчета матрицы сравнений за «один» шаг, имеет наилучшее время выполнения, хотя его временная сложность все же выше константной. При большем числе потоков данный вариант показывает квадратичное зависимость от числа элементов, но в любом случае, в рассматриваемой области он оказывается эффективней варианта «N», хотя второй имеет линейную сложность. При обработке последовательности длиной 256 элементов оба варианта «1» и «N» имеют равную производительность (данная точка не захватывается графиками). В целом, вплоть до ~ 100 элементов время расчетов вариантом «1» алгоритма ранговой сортировки сравнимо со временем работы прочих популярных алгоритмов сортировки.

Несмотря на то, что алгоритм с «константной» временной сложностью показал наилучшие результаты в области малых длин последовательностей, эффект является незначительным, чтобы быть полезным для практических расчетов на данном этапе развития GPU устройств. Оптимальные, «ровные» и практически полезные результаты в заданной области длин последовательностей демонстрирует «lgN» вариант алгоритма, построенный на подходе «разделяй и властвуй» и имеющий вычислительную сложность $O(\lg^2 N)$.

Графики показывают также, что качество реализации алгоритма соответствует общедоступным библиотечным реализациям алгоритмов сортировки. Полученная реализация алгоритма ранговой сортировки (вариант «lgN») при сортировке коротких последовательностей превосходит почти втрое рассмотренные библиотечные алгоритмы. Исходя из того, что «lgN» вариант ранговой сортировки по построению полностью аналогичен алгоритму MergeSort [11], разница в производительности связана только с адаптацией кода под специальный случай коротких последовательностей.

5. Заключение

В работе проанализирована возможность портирования на платформу GPU алгоритма сортировки взаимодействий, использующегося в расчетах ближних невалентных взаимодействий в молекулярной динамике. Показано, что для данного алгоритма возможно преодоление его «узких мест», ограничивающих область его применимости, и связанных с сортировкой большого объема коротких последовательностей пар взаимодействующих атомов. Разработана модификация алгоритма параллельной ранговой сортировки, специально адаптированая под случай последовательностей с длиной, не превышающей 256 элементов, требуемой методом. Проанализирована эффективность разных версий предложеного алгоритма, имеющих различную вычислительную и временную сложность. Выполнено сравнение эффективности разработанного алгоритма с реализациями алгоритмов MergeSort, RadixSort, BitonicSort и OddEvenSort, представленными в библиотеке CUDA SDK, и в диапазоне длины последовательностей, не превышающем 256 элементов, показана существенно (~ 3 раза) большая эффективность предложенного алгоритма.

Данная работа находится в контексте исследований алгоритмов молекулярной динамики, обеспечивающих предельно возможную эффективность данного метода. Следующими этапами данной работы является сранительная оценка эффективности использования данного алгоритма в рамках различных современных модификаций методов Верлет таблицы и связанных ячеек, и, что наиболее интересно, в рамках новой варианта Верлет подхода, специально предназначенного для выполнения в параллельной окружении, метода локальных Верлет таблиц [18].

Литература

1. URL: http://www.gromacs.org/tests
2. Quentrec B., Brot C. New methods for searching for neighbours in molecular dynamics computations. J. Comput. Phys. 1973, 13, 430-432.
3. Gonnet P. A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. J. Comp. Chem. 2007, 28 (2), 570-573.
4. Verlet L. Computer Experiments on Classical Fluids. Phys Rev 1967, 159, 98-103.
5. Meloni S., Rosati M. Efficient particle labeling in atomistic simulations J. Chem. Phys. 2007, 126, 121102.
6. Cui Z.W., Sun Y., Qu J. M. The neighbor list algorithm for a parallelepiped box in molecular dynamics simulations. Chinese Sci Bull, 2009, 54(9), 1463-1469.
7. Фомин Э.С. Сравнение метода Верлет таблицы и метода связанных ячеек для последовательной, векторизованной и многопоточной реализаций. Вычислительные методы и программирование, 2010, Т.11, 299-305.
8. Fomin E.S. Consideration of Data Load Time on Modern Processors for the Verlet Table and Linked Cell Algorithms. J. Comp. Chem., 2011, Volume 32, Issue 7, 1386-1399.
9. Yao Z., Wang J.-S., Liu G.-R., Cheng M. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. Computer Physics Communications 2004, 161, 27-35.
10. Knuth D. E. The Art of Computer Programming. Vol. 3. Sorting and Searching, Addison-Wesley, 1973.
11. Satish N., Harris M., Garland M. Designing efficient sorting algorithms for manycore GPUs. Parallel and Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on 23-29 May 2009, 1-10.
12. Louri A., Hatch J.A. and Na J. A Constant-Time Parallel Sorting Algorithm and Its Optical Implementation. EEE Micro, 60-71.
13. Blelloch G. E. Scans as primitive parallel operations. Computers, IEEE Transactions on, 1989, 38(11), 1526 - 1538.
14. NVIDIA CUDA Programming Guide 3.1, 2010, ©2006-2010 NVIDIA Corporation.
15. Фомин Э.С., Алемасов Н.А., Чирцов А.С., Фомин А.Э. Библиотека программных компонент MOLKERN для построения программ молекулярного моделирования. Биофизика, 51, 7, 2006, 110-113.
16. NVIDIA CUDA Toolkit 3.2, 2010, ©2006-2010 NVIDIA Corporation.
17. Park S.K. and Miller K.W. Random Number Generators: Good Ones are Hard to Find. Communications of the ACM, 1988, 31 (10), 1192-1201.
18. Gonnet P. Pairwise verlet lists: Combining cell lists and verlet lists to improve memory locality and parallelism. J. Comp. Chem. 2012, 33(1), 76-81.

О развитии двух параллельных алгоритмов численного моделирования взаимодействия излучения с веществом*

М.А.Чащин, Л.И. Рубина, О.Н. Ульянов

Институт математики и механики УрО РАН, Екатеринбург, Россия

Представлена методика численного моделирования взаимодействия ионизирующего излучения с веществом для ряда физических и математических постановок задачи радиационного переноса в смесях веществ. Методика предполагает расчет вклада каждой учитываемой спектральной линии; вычисление населенностей уровней в соответствии с уравнениями кинетики населенностей; нахождение интенсивности излучения в соответствии со спектральным уравнением переноса с коэффициентами, вычисляемыми по населенностям уровней и электронной температуре, которая удовлетворяет уравнению энергобаланса. Решение задачи в достаточно полных постановках сопряжено с большими объемами вычислений и требует применения параллельных технологий. Авторы разрабатывают методику, основанную на двух различных методах решения задачи и на развитии двух параллельных алгоритмов. Предложенные алгоритмы и разработанные программы для суперкомпьютеров позволили проводить моделирование переноса излучения в плоских слоях вещества с учетом до тысячи спектральных линий за приемлемое время с приемлемой точностью. Приводятся результаты численных расчетов.

1. Введение

Перенос излучения и процессы, связанные с переносом излучения в однородных и неоднородных средах, вызывают большой интерес исследователей. Численное моделирование и исследование этих процессов используются как при изучении многих явлений природы, так и при создании новой техники [1-4]. Детальное изучение процесса переноса излучения и связанных с ним процессов — непростая задача, требующая учета многих тонких эффектов и приводящая к проведению большого объема вычислений, к обработке больших многомерных массивов информации.

Разработкой методов, алгоритмов и программ для численного моделирования радиационного переноса занимаются многие исследователи (например, [5]).

Авторы на протяжении ряда лет проводили работы в указанном направлении, основанные на реализации двух развиваемых ими методов (МАПИ и МПЛЧ) численного решения задачи. Метод МАПИ (Метод Аналитического Представления Излучения) использует явное представление интенсивности излучения, метод МПЛЧ (Метод Полиномов Лагранжа-Чебышева) основан на использовании интерполяционных полиномов Лагранжа по узлам Чебышева. Методика разрабатывается для случая взаимодействия ионизирующего излучения с веществом, состоящим из водородоподобных и гелиоподобных ионов, а также ионов, у которых, на внешней оболочке отсутствуют электроны.

На первом этапе изучалась такая физическая модель явления, которую адекватно представляет случай, когда рассматриваемые вещества имеют доплеровские профили излучения и поглощения спектральных линий. В качестве математической модели была взята система уравнений, включающая систему уравнений кинетики населенностей уровней и систему стационарных интегро-дифференциальных уравнений, состоящую из уравнения переноса излучения в

* Работа выполнена в рамках программы межрегиональных и межведомственных фундаментальных исследований УрО РАН (проект 12-С-1-1001) и программы фундаментальных исследований Президиума РАН "Информационные, управляющие и интеллектуальные технологии и системы" (проект 12-П-1-1023) при поддержке УрО РАН.

«непрерывном спектре» и уравнений переноса излучения в каждой из учитываемых спектральных линий с соответствующими граничными условиями [6]. Пересечением спектральных линий пренебрегалось, энергобаланс не учитывался. При наличии доплеровских контуров было возможно не учитывать пересечения резонансных линий. Первое предположение — отсутствие пересечения спектральных линий вполне естественно, что и показали дальнейшие исследования. Со вторым — не учетом энергобаланса дело обстоит, очевидно, не так. Но, как показали дальнейшие исследования, в некоторых случаях можно изучать явление, пренебрегая энергобалансом [7,8], хотя, конечно, более полные модели должны содержать уравнение энергобаланса для электронной температуры. Была разработана методика для численного моделирования процессов переноса излучения в неоднородном плоском слое смеси веществ, содержащем до десяти подслоев, имеющих разные плотности и температуры, при наличии до ста резонансных линий за приемлемое время с приемлемой точностью [9-11]. Развитие методики состояло как в развитии оригинальных, так и в подборе известных, но отвечающих особенностям задачи математических и вычислительных методов решения задачи, совершенствовании и модификации численных алгоритмов и реализующих их программ.

Благодаря использованию параллельных технологий и все более мощных суперкомпьютеров удалось решать задачи в достаточно полных постановках. В настоящее время возможно рассчитывать интенсивность излучения, учитывая не несколько десятков, а сотни линий. Слой, через который проходит излучение, может содержать до десяти подслоев, отличающихся как составом вещества, так и плотностью и электронной температурой или электронная температура в слое меняется согласно уравнению энергобаланса.

В данной статье представлены результаты развития методики при переходе к физическим моделям с фойгтовскими профилями излучения и поглощения в спектральных линиях. Этот класс задач с физической точки зрения является существенно более содержательным, чем ранее решаемые задачи. С математической и вычислительной — значительно более сложный. Переход к таким моделям потребовал пересмотреть математическую модель. Ранее в случае доплеровских контуров было возможно не учитывать пересечение резонансных линий. Рассмотрение моделей с фойгтовскими профилями сделало этот учет необходимым, что привело к отказу от раздельного вычисления интенсивности излучения в линиях и в непрерывном спектре и, следовательно, к разработке нового подхода к решению единого интегро-дифференциального уравнения переноса излучения.

Кроме того, на этом этапе развития методики, применения новых параллельных технологий и современных суперкомпьютеров была поставлена задача оценки влияния учета энергобаланса на решение задачи в различных постановках. Для более полного описания физического процесса, в математической модели температуру требуется определять, используя информацию о спектре излучения в среде. Таким образом, при решении задачи необходимо совместно с решением уравнения переноса излучения (УП) и системы уравнений кинетики (УК), описывающих долю ионов веществ смеси, пребывающих в определенном состоянии, которое называется населенностью уровня, решать уравнение энергобаланса (УБ) для электронной температуры вещества. Результаты развития методики представлены в работе.

2. Постановка задачи

Рассматривается стационарный случай задачи переноса излучения в плоском слое смеси веществ. Слой конечной толщины L может состоять из отдельных подслоев $D_m (1 \le m \le M, M \le 10)$. В каждом подслое смесь может обладать своими характеристиками (состав, плотность). Смесь состоит из Γ компонент, γ – номер компоненты. Атомы каждой компоненты могут быть в нескольких состояниях с разной степенью возбуждения и ионизации. Каждое состояние определяется парой индексов (Ii), где I — степень ионизации, i — номер (дискретного) возбужденного состояния для данной степени ионизации.

Относительная доля ионов γ - го вещества, находящегося в состоянии (Ii), обозначается $c_{iI}^{(\gamma)}(x\})$ и называется населенностью этого состояния. Все населенности для каждой конкрет-

ной компоненты вещества можно считать некой векторной величиной $\vec{c}^{(\gamma)}(x) = (c_1^{(\gamma)}(x), c_2^{(\gamma)}(x), ..., c_r^{(\gamma)}\{x\})$. Они удовлетворяют системе уравнений кинетики (УК).

(УК) $W^{(\gamma)}\vec{c}^{(\gamma)} = \vec{b}^{(\gamma)}$

Элементы кинетических матриц $W^{(\gamma)}(x)$ и компоненты вектора $b^{(\gamma)}(x)$ зависят от средних по углу и энергии интенсивностей излучения в каждой точке сетки по пространственной переменной. Кроме того, имеется условие нормировки $\sum_{iI} c_{iI}^{(\gamma)} = 1$.

Вещества в смеси связаны через поле излучения. Интенсивность излучения зависит от векторов населенностей для каждого вещества в смеси. Если $I(x, \mu, \varepsilon, Te)$ — интенсивность излучения вдоль некоторого луча в точке среды x с энергией ε и электронной температурой Te, то в среде выполняется уравнение переноса излучения (μ — косинус угла между направлением луча и осью x ,ось направлена перпендикулярно поверхности слоя).

(УП) $$\mu \frac{dI}{dx} + \kappa I = S$$

Здесь $\kappa = \kappa(\mu, \varepsilon, Te, c_{jJ}^{(\gamma)})$ — коэффициент поглощения, $S = S(\mu, \varepsilon, Te, c_{jJ}^{(\gamma)})$ — функция источника, $\mu \in [-1,1] : x \in (0, L]$, если $\mu > 0 : x \in [0L)$, если $\mu < 0, \varepsilon \in [-\infty, \infty]$.

Будем рассматривать случай, когда $Te = Te(x)$ и удовлетворяет уравнению энергобаланса

(УБ) $$-\frac{d}{dx}(\kappa_0 Te^n \frac{d\ Te}{dx}) = 2\pi \int_{-1}^{+1} d\mu \int_{-\infty}^{+\infty} [kI - S] d\varepsilon$$

Здесь κ_0 — коэффициент, являющийся функцией эффективного заряда атомного ядра смеси, n — показатель молярной теплоемкости.

Ранее для доплеровского контура излучения рассматривалось уравнение энергобаланса, когда $\kappa_0 = 0$. В настоящей работе приведены результаты решения системы уравнений (УК, УП, УБ) для случая фойгтовского профиля излучения и произвольного коэффициента $\kappa_0 \neq 0$.

3. Метод решения уравнения энергобаланса

В уравнении энергобаланса (УБ) сделаем замену, полагая $Q = Te^{(n+1)}$. Для определения функции Q получаем уравнение

(УБ 1) $$\frac{d^2 Q}{dx^2} = -[2\pi(n+1)/\kappa_0] \int_{-1}^{+1} d\mu \int_{-\infty}^{+\infty} (kI - S) d\varepsilon$$

Сравнивая (УП) и (УБ 1), получаем

(УБ 2) $$\frac{dQ}{dx} = [2\pi(n+1)/\kappa_0] \int_{-1}^{+1} \mu d\mu \int_{-\infty}^{+\infty} I d\varepsilon\ , \qquad (\kappa_0 \neq 0)$$

Решение уравнения (УБ 2) может быть сведено к последовательному вычислению трех интегралов

(И) $J_T(x,\varepsilon)=\int_{-1}^{+1} I\mu d\mu$, $Int(x)=\int_{-\infty}^{+\infty} J_T d\varepsilon$, $Q(x_{i+1})=[2\pi(n+1)/\kappa_0]\int_{x_i}^{x_{i+1}} Int(x)dx$,

$(i=0,1,...,l)$,

если известно $Te(x_0)$.

$Te(x_{i+1})=Q^{1/\{n+1\}}(x_{i+1})$. $\{x_0,x_1,....,x_l\}$ — сетка по пространственной переменной.

В методике МАПИ для вычисления интеграла $J_T(x,\varepsilon)=\int_{-1}^{+1} I\mu d\mu$ отрезок [-1,+1] разбивается на 5 частей: $[-1,\mu_1]$, $[\mu_1,\mu_2]$, $[\mu_2,\mu_3]$, $[\mu_3,\mu_4]$, $[\mu_5,+1]$. Точки μ_1 и μ_5 совпадают, как правило, с точками разрыва заданной начальной интенсивности излучения. Отрезок $[\mu_2,\mu_3]$ содержит точку $\mu=0$. Здесь интенсивность излучения имеет свои особенности, связанные с тем, что точка $\mu=0$ - особая точка уравнения переноса. Требуется достаточно частая сетка, чтобы учесть особенности поведения подынтегральной функции на этом участке. Для вычисления интегралов на каждой части отрезка используется полином Лежандра своей специальным образом выбранной степени.

Для вычисления несобственного интеграла $Int(x)=\int_{-\infty}^{+\infty} J_T d\varepsilon$ используется метод Лобатто, позволяющий варьировать пределы интегрирования и достигать в расчете заданной высокой гарантированной точности на каждом интервале, предварительно полученном после выделения точек разрыва подынтегральной функции.

Для вычисления значений $Q(x_{i+1})$ в узлах сетки по пространственной переменной также используются полиномы Лежандра.

В методике МПЛЧ все интегралы вычисляются построением полиномов Лагранжа специально выбранной степени с узлами Чебышева. Совпадение результатов расчетов по двум значительно отличающимся алгоритмам расчета принимается за критерий истины.

4. О развитии методики МПЛЧ

Метод МПЛЧ (Метод Полиномов Лагранжа-Чебышева) существует в нескольких модификациях.

Программа foigt_ncom39_8 осуществляет расчеты вариантов с фойгтовскими профилями для смесей, содержащих до четырех компонент в неоднородных слоях (10 слоев). Слои могут отличаться по плотности, по температуре (постоянной) и по составу вещества, максимальное число линий — 39.

Программа bal480sl рассчитывает варианты с фойгтовскими профилями, смеси могут содержать до 4 компонент, неоднородные слои (10 слоев) по плотности и составу вещества, решается уравнение энергобаланса. Размерности массивов по пространственной и по энергетической переменным могут изменяться от варианта к варианту. Выбор размерности зависит от того, какую задачу надо решать (либо много слоев, либо много линий, максимально 39 линий и 10 слоев). Ограничение на количество точек в сетке по пространственной переменной (480 точек на все подслои и 120 точек на 1 слое).

Для задач с учетом нескольких десятков линий предложен и реализован в программных кодах новый высокоэффективный алгоритм, на порядок уменьшено время расчета одного варианта (сравните времена счета варианта на одинаковом числе процессоров в таблицах 1 и 2).

В таблицах 1 и 2 представлены результаты расчета одинакового варианта: 2 линии, 2 компоненты вещества в слое, фойгтовский профиль. Результат, полученный по старой методике (таблица 1, 2008г.). Результат, полученный по усовершенствованной программе метода МПЛЧ (таблица 2, 2010г.).

Таблица 1. (2008г.)

N	T(N) (сек.)	H(N)	Q(N)
2	14.7	1.00	100%
4	7.9	1.86	93%
8	5.3	2.77	69%
16	4.0	3.68	45%
32	3.1	4.74	29%

Таблица 2. (2010г)

N	T(N) (сек.)	H(N)	Q(N)
1	6.12	1	100%
16	0.39	15.7	98%
32	0.26	23.5	73%
64	0.20	30.6	47%
128	0.16	38.3	30%
151	0.15	40.8	27%

Здесь N — число процессоров, T(N) — время счета на данном числе процессоров (сек.), H(N) — ускорение при использовании данного числа процессоров, Q(N)=100%T(1)/{N*T(N)} — эффективность параллельной реализации

Разработанная параллельная программа foigt_ncom_oc3 позволяет проводить численное моделирование радиационного переноса в смесях веществ с учетом нескольких сотен спектральных линий (до1000), (см. рис.1).

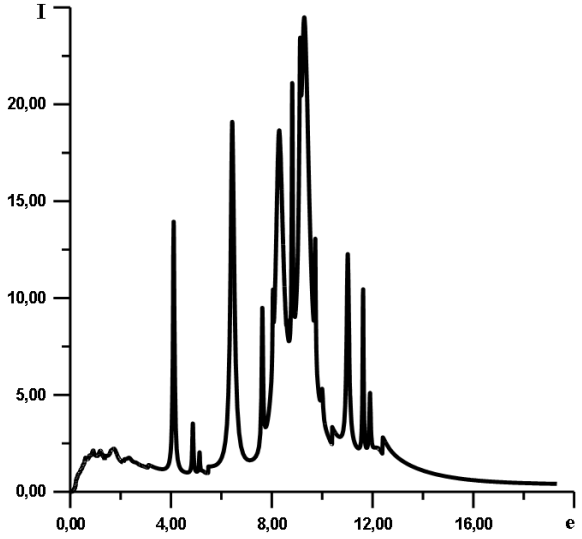


Рис.1. Интенсивность излучения при μ =0.48, x=0.4, плотность 0.03, температура Te =1.5, смесь из 4-х компонент, 988 резонансных линий

Учет при численном моделировании значительного количества спектральных линий является весьма актуальным, поскольку моделирование с учетом большого числа спектральных линий наиболее адекватно отражает физическую сущность явления. Решение таких задач сопряжено с очень большими объемами вычислений, которые под силу только мощным современным компьютерам. Более того, считается общепринятым, что для большинства задач и в обозримом будущем решение полного уравнения переноса слишком дорогостояще как по времени, так и по памяти. Современные базы данных содержат сотни тысяч линий. Поэтому в других подходах при численном моделировании, как правило, используются различные физические и математические приближения, в частности многогрупповой метод и т.п.

Но развитие параллельных технологий, использование современных суперкомпьютеров, разработка новых математических методов и вычислительных подходов позволяют надеяться на продвижение в решении таких задач в достаточно полных постановках (метод МПЛЧ).

Представляется, что численная методика метода МПЛЧ соответствует мировому уровню [5,12-14].

5. Параллельная программа МПЛЧ

Параллельная программа по методике МПЛЧ разработана в системе DVM (Distributed Virtual Memory) или (Distributed Virtual Machine). Распараллеливание программы ведется по угловой переменной. Все циклы по угловой переменной, (а также и данные (массивы)), автоматически распределяются на заданное число процессоров. Программа легко масштабируется. Возможно задание любого числа процессоров от 1 до N_μ.

В связи с разработкой программы с фойгтовским профилем излучения и поглощения и использовании новой математической модели (единое уравнение переноса), параллельная программа по методике МПЛЧ существенно изменилась. Ранее в распараллеливании участвовало три подпрограммы. Это подпрограмма решения уравнения кинетики (DCWR), подпрограмма решения уравнения переноса в непрерывном спектре (DKDUC) и подпрограмма решения уравнения переноса в дискретном спектре (DKDUK). При этом подпрограмма решения уравнения переноса в дискретном спектре решалась для каждой линии отдельно. Цикл по линиям был распараллелен на произвольное (*1<=p<=L, L* — число линий) число процессоров. При этом был реализован так называемый "параллелизм задач". Внутри подпрограммы DKDUK было вставлено распараллеливание циклов по угловой переменной. В новой программе по методике МПЛЧ отсутствует "параллелизм задач", т.е. подпрограмма DKDUK решает уравнение переноса для всех линий, а также все необходимые интегралы. При этом у нас решается единое уравнение переноса в непрерывном и дискретном спектре. Также подпрограмма DKDUK решает уравнение энергобаланса. Таким образом, распараллеливается только подпрограмма DKDUK по угловой переменной. Подпрограмма DKDUC совсем отсутствует. Модификация программы по методике МПЛЧ привела к 10-ти кратному ускорению программы.

Для распараллеливания данных в подпрограмме DKDUK система DVM использует модель параллелизма по данным (МПД) [15].

При работе с удаленными данными модель параллелизма по данным преобразуется в модель передачи сообщений.

В модели передачи сообщений каждый процесс имеет собственное локальное адресное пространство. Обработка общих данных и синхронизация осуществляется посредством передачи сообщений. В нашем случае с процессора на процессор (каждый каждому) передаются недостающие части интегралов по угловой переменной. Передача с процессора на процессор интеграла намного экономичнее, чем передача интенсивности излучения.

6. О развитии методики МАПИ

Методика МАПИ (Метод Аналитического Представления Излучения) существует в нескольких модификациях: МАПИ_ЯКОБИ, МАПИ_Л, МАПИ_2Л, МАПИ_3Л.

В МАПИ_ЯКОБИ для вычисления интегралов используются разные полиномы (Эрмита-Чебышева, Якоби, Лагерра, Лежандра). Выбор того или иного полинома зависит от асимптотики подынтегральной функции. Естественно, что выбор степени каждого полинома при изменении задачи требует отдельной работы.

В модификации МАПИ_Л все интегралы по энергетической переменной ε в линиях вычисляются с использованием метода Лобатто, который позволяет задавать точность вычисления интеграла, варьировать размер области интегрирования при расчете интегралов с бесконечными пределами. Выбор сетки интегрирования в данном методе зависит только от значений подынтегральной функции на сетке и итоговой точности расчета интеграла, заданной для метода. В нашей задаче подынтегральная функция зависит от параметра μ (угловой переменной). Сетка для каждого значения параметра μ строится своя, число точек в сетке для разных значений параметра может значительно отличаться, если поведение подынтегральной функции разное при разных значениях параметра. В нашей задаче поведение интенсивности излучения при малых значениях угловой переменной более сложное, чем при других значениях этого параметра. Соответственно сетка при малых значениях μ содержит больше точек, чем при других значениях этого параметра.

Следует заметить, что подынтегральные функции при решении нашей задачи негладкие. Они имеют разрывы и “пики”, поэтому предварительно область интегрирования разбивается на участки, где функция достаточно гладкая, что позволяет вычислять интегралы с высокой точностью. Если заданная высокая точность не может быть достигнута, то это, как правило, является указанием на то, что какая-то особенность функции не выделена, что иногда обнаруживается при счете новых задач.

В модификации МАПИ_2Л метод Лобатто используется для вычисления интегралов по энергии не только в линиях, но и в непрерывном спектре. Разработка этого комплекса программ связана с тем, что при переходе к фойгтовскому профилю излучения и проведению расчетов для широкого диапазона постоянных электронных температур оказалось, что результаты расчета населенностей уровней по двум методикам не совпадают. Изучение причины расхождений результатов показало, что результат вычисления одного из несобственных интегралов в непрерывном спектре был разным при расчете по методу МПЛЧ с использованием полиномов Лагранжа и по методу МАПИ с использованием полиномов Лагерра. Расчет ”спорного” интеграла по методу Лобатто с гарантированной точностью позволил разрешить “спорный” вопрос (пункт 8). Были установлены необходимые для требуемой точности вычислений границы интегрирования, что позволило получить совпадение результатов расчета населенностей уровней по обеим методикам.

В модификации МАПИ_3Л метод Лобатто используется для вычисления интегралов по энергетической переменной всюду, в том числе при вычислении интегралов, необходимых при решении уравнения энергобаланса. Применение этого комплекса программ оправдано при решении новых задач, использующих уравнение энергобаланса, когда поведение функций значительно меняется. Следует заметить, что у нас нет тестовых вариантов расчетов. Критерий истины один - совпадение результатов при расчете по разным методикам. Если результаты не совпадают, то большую роль в установлении истины играет возможность расчетов на сетках с любой заданной точностью, не связанных с какими-то определенными полиномами. Такие возможности заложены в комплексе МАПИ_3Л.

Программа МПЛЧ ориентирована на сокращение времени счета варианта, но этот метод при переходе на новые задачи требует дополнительной настройки. Для получения быстрого результата нужна гарантия, что полученный результат правильный. Такую гарантию дает “настройка” с использованием программы МАПИ_3Л. Существуют понятия: “методический счет” и “серийный счет”. Комплекс МПЛЧ способен проводить серийный счет, а комплексы МАПИ_Л, МАПИ_2Л, МАПИ_3Л — методические расчеты. Они требуют на несколько порядков большего времени для расчета варианта, чем МПЛЧ или достаточно большого числа процессоров, но зато дают картину процесса с заданной гарантированной точностью.

В комплексе МАПИ нет ограничений на выбор сетки по пространственной переменной. Она может задаваться любой, в том числе по узлам полиномов Чебышева, как в методе МПЛЧ. Это создает дополнительные удобства при сравнении результатов, полученных по разным методикам.

7. Параллельный комплекс МАПИ_3Л

В комплексе МАПИ_3Л для распараллеливания используется MPI- технологии параллельного программирования. Как было отмечено ранее [7], комплекс МАПИ разбит на задачи, каждая из которых распараллеливается отдельно по тем параметрам, которые используются в данной задаче. Внутреннее наполнение таких задач зависит от постановки глобальной задачи (ГЗ), но параметры, по которым производится распараллеливание, сохраняются. Иногда меняется диапазон их изменения, если этого требует (ГЗ). Так задача A_{ijkl} вычисляет средние по энергии от интенсивности излучения в дискретном спектре. Она распараллелена по числу точек i в сетке по пространственной переменной (i <241), по числу компонент j в смеси веществ(j <5), по числу линий k в дискретном спектре (k <40), по числу точек l в сетке по угловой переменной (l <96). В реальных расчетах заложенное в этой задаче полное распараллеливание удается осуществить только тогда, когда диапазон изменения параметров в расчетном варианте достаточно мал.

В сложных вариантах на одном из предоставленных p процессоров вычисляется $(i \times j \times k \times l)/p$ точек задачи.

Другая задача C_{ij}, например, вычисляет интегральные характеристики, осуществляющие связь кинетической матрицы с интенсивностью излучения вне линий (в непрерывном спектре), населенности уровней и температуру на сетке по пространственной переменной. Она распараллелена по числу i точек в сетке по пространственной переменной и по числу j компонент в смеси веществ. Следует заметить, что большая эффективность распараллеливания достигается благодаря тому, что вычисление многих величин повторяется на каждом из используемых процессоров, а передаются с процессора на процессор только сравнительно небольшие массивы переменных. Число используемых процессоров p может быть задано любым. Естественно, что в сложных вариантах, чем больше процессоров используется в счете, тем меньше времени требуется на расчет варианта.

8. Изучение влияния точности вычисления интеграла $\int_{\varepsilon_{\min}}^{\infty} \varepsilon^2 \exp(-\varepsilon/Te)\sigma(\varepsilon/\varepsilon_{Ji})d\varepsilon$ на поведение населенностей уровней

Выписанный выше интеграл используется при вычислении компонент матрицы в уравнении кинетики. Для расчета данного интеграла в программе МАПИ_ЯКОБИ использовались полиномы Лагерра вычисления несобственных интегралов, а в программе МПЛЧ — полиномы Лагранжа. В методе МПЛЧ пределы интегрирования задаются конечными (верхний предел интегрирования выбирался R=30). Когда рассчитывались варианты с невысокой постоянной электронной температурой, итоговые результаты расчетов населенностей и интенсивностей излучения по обеим методикам совпадали (что принималось за критерий истины). При расчете варианта, в котором была задана электронная температура $Te = 10$, обнаружились значительные расхождения в результатах расчетов населенностей по двум разным методикам (рис.2, кривая 1 получена по методу МПЛЧ, кривая 2 получена по программе МАПИ_ЯКОБИ). Анализ расчетов показал, что результаты расчета интенсивности излучения одинаковы. В кинетической матрице компоненты, зависящие от интенсивности излучения, совпадали в результате расчетов по обеим методикам. Отличия обнаружились при вычислении выписанного в заглавии пункта несобственного интеграла. Причина могла быть в неточности счета как по методике МАПИ_ЯКОБИ, так и по методике МПЛЧ. Возможно, неправильно выбрана степень используемого полинома. Потребовался такой метод вычисления, который не связан со степенью полинома. Так возник комплекс МАПИ_2Л, который позволил провести расчеты несобственного интеграла с разной заданной точностью и с разными пределами интегрирования и установить, когда увеличение верхней границы интегрирования R не влияет на величину интеграла, что привело к уточнению счета данного интеграла по методу МПЛЧ. В результате расчеты по двум методикам совпали (рис.2, R=1000, линия 2.).

На рис. 2. изображена населенность уровня c_1 для двух значений верхней границы вышеуказанного интеграла в методе МПЛЧ((1) — R=30 и (2) — R=1000), вещество — цинк. При R=1000 графики населенности, сосчитанные по разным методикам (МАПИ_2Л, МАПИ_ЯКОБИ, МПЛЧ) совпали.

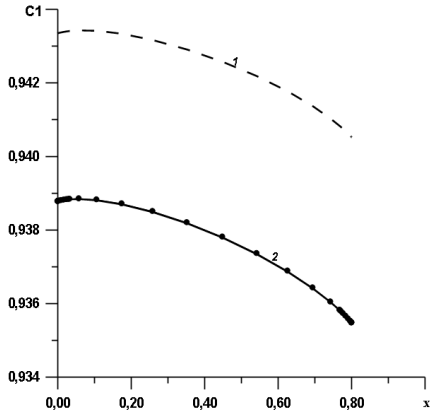


Рис.2

9. Результаты численного моделирования

• Разработка методики на основе подхода МАПИ (комплекс программ МАПИ_3Л) позволила решать задачу с заданной (гарантированной) точностью, что дает возможность определить необходимые параметры и сетки, оценить точность вычислений по «быстрой» методике, основанной на методе МПЛЧ. Сравнительные расчеты по методам МПЛЧ и МАПИ приведены на рис. (2,3). Рис.2 описан выше. На рис.3 представлены результаты расчета варианта для слоя, состоящего из двух подслоев, содержащих вещество кальций ($n = 25{,}9$; $k_0 = 0{,}5$). В одном подслое ($0 \le x \le 0{,}4$) плотность вещества равна 0,03; в другом подслое ($0{,}4 \le x \le 0{,}8$) плотность вещества равна 0,1. Учитывалось 2 резонансные линии. На рис.3(а) — населенность c_1, сосчитанная по методике МАПИ (сплошная линия) и по методике МПЛЧ (точки). На рис. 3(б) — электронная температура, полученная по двум методикам для данного варианта.

• Зависимость электронной температуры и населенностей уровня от плотности вещества в слое представлена на рис.4(а) и рис.4(б). Наблюдается незначительное уменьшение электронной температуры в слое при увеличении плотности вещества.

• Влияние на интенсивность излучения учета уравнения энергобаланса представлено на рис.5. Изучение влияния показало, что качественно картина не меняется в случае переменной электронной температуры. Наблюдается лишь количественное изменение интенсивности излучения.

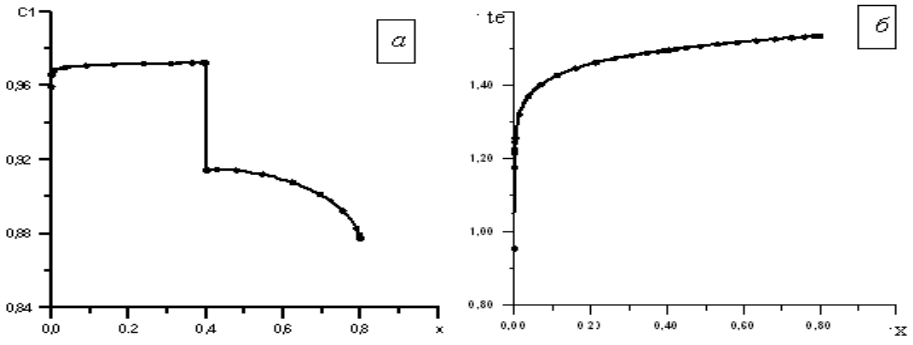


Рис. 3

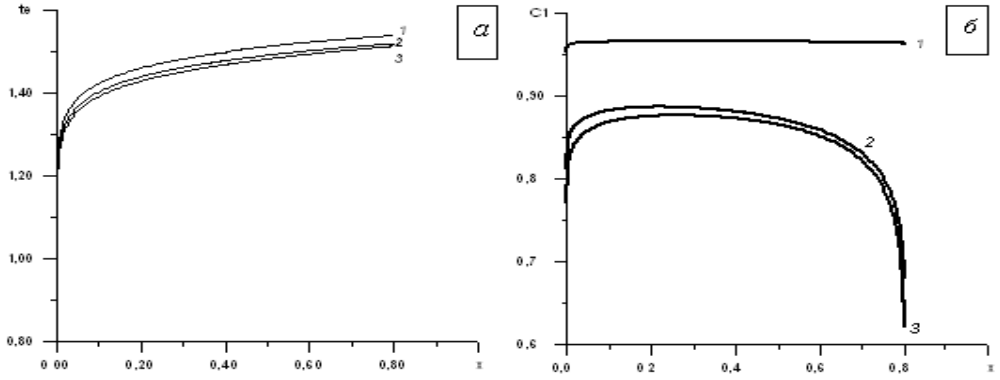


Рис.4

(а) Электронная температура $Te = Te(x)$, $n = 25.9$, $k_0 = 0.5$ 1 — плотность 0.03, 2 — плотность 0.25, 3 — плотность 0.35.

(б) Населенности уровней для $Te = Te(x)$, $n = 25.9$, $k_0 = 0.5$ 1 — плотность 0.03, 2 — плотность 0.25, 3 — плотность 0.35

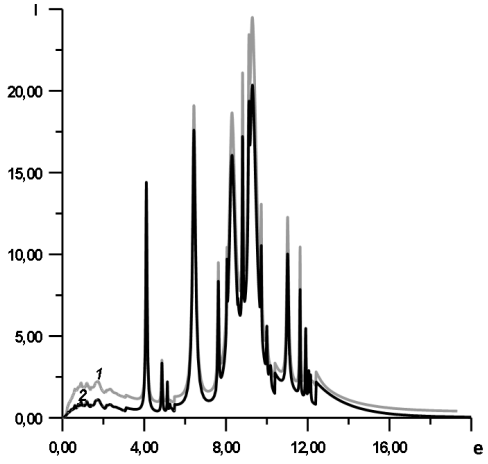


Рис.5

Интенсивность излучения при: μ =0.48, x=0.5, плотность — 0.03, смесь из 4 компонент вещества, 156 линий. 1 — интенсивность излучения для температуры Te =1.5.

2 — интенсивность излучения для температуры $Te = Te(x), n = 25.9, \kappa_0 = 0.5$.

10. Заключение

В статье представлены результаты развития методики численного моделирования взаимодействия ионизирующего излучения с веществом. Рассматриваемая модель предполагает, что изучается случай вещества, имеющего фойгтовские профили излучения и поглощения, энергобаланс учитывается. Методика предполагает расчет вклада каждой учитываемой спектральной линии; вычисление населенностей уровней в соответствии с уравнениями кинетики населенностей; нахождение интенсивности излучения в соответствии со спектральным уравнением переноса с коэффициентами, вычисляемыми по населенностям уровней и электронной температуре, которая удовлетворяет уравнению энергобаланса. Методика основана на двух методах решения задачи и, соответственно, на разработке двух реализующих эти методы параллельных алгоритмов.

Основными результатами являются:

- разработаны параллельные алгоритмы и программы, реализующие методы МАПИ и МПЛЧ для новой математической постановки задачи, отражающей новую (и более полную) физическую модель;
- разработан алгоритм решения уравнения энергобаланса (пункт 3);
- создан комплекс программ для решения задачи переноса излучения в плоском слое смеси веществ с учетом нескольких сотен линий (рис. 1);
- разработаны новые алгоритмы и программы МПЛЧ, позволившие на порядок сократить время расчетов по сравнению с ранее применявшимся параллельным алгоритмом МПЛЧ;
- достигнуто приемлемое время расчетов достаточно сложных и физически содержательных задач;
- получена возможность проведения серийных инженерных и научных расчетов;
- проведены серии расчетов на суперкомпьютерах URAN (ИММ УрО РАН, г. Екатеринбург) и МВС-100К (МСЦ РАН, г. Москва), в том числе для определения электронной температуры в однородных и неоднородных средах;
- проведено сравнение результатов, полученных по методикам МАПИ и МПЛЧ, установлены допустимые пределы изменения параметров задачи.

Целями дальнейшего развития методики возможно (в силу потребностей практического применения) могут быть:

1. модификация разработанных (для новой математической постановки задачи) алгоритмов и программ для случая доплеровских профилей излучения и поглощения;
2. адаптация параллельных алгоритмов и кодов развиваемой методики для проведения расчетов на гибридных вычислительных системах;
3. распространение методики на случай подвижной среды;
4. рассмотрение цилиндрического слоя, а также 2-х и 3-х мерного случая;
5. качественное (на 2-3 порядка) увеличение количества учитываемых резонансных линий;
6. включение разрабатываемых кодов (с учетом достижения третьей, четвертой и пятой цели) в комплексы программ для расчета тех практических задач физики сплошных сред, в которых рассматриваемое взаимодействие излучения с веществом является лишь одним из элементов постановки задачи.

Представляется, что первые три из указанных целей вполне достижимы в рамках развиваемой методики при современном развитии параллельных технологий и суперкомпьютерных вычислительных систем, достижение четвертой цели требует развития новых (или, во всяком случае, существенного изменения в методике) математических и вычислительных подходов, пятая (а поэтому и шестая) цели могут быть достигнуты лишь при использовании нового поколения суперкомпьютеров.

Литература

1. Михалас Д. Звездные атмосферы. Т. 1,2. М.: Мир, 1982.
2. Никифоров А.Ф., Новиков В.Г., *Уваров В.Б.* Квантово-статистические модели высокотемпературной плазмы и методы расчета росселандовых пробегов и уравнений состояния. М.: Физико-математическая литература, 2000.
3. Сушкевич Т.А. Математические модели переноса излучения. М. БИНОМ. Лаборатория знаний, 2006.
4. Gonzalez M., Audit E. Numerical treatment of radiative transfer // Astrophysics and Space Science **298**: 357-362, 2005.
5. Manual of the FLYCHK code (National Institute of Standards and Technology, USA) // *http://*nlte.nist.gov/FLY/Doc/Manual_FLYCHK_Nov08.pdf

6. Леликова Е.Ф., Рубина Л.И., Ульянов О.Н., Чащин М.А Параллельные вычислительные технологии в задаче о переносе радиационного излучения // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. М. 2002. Вып.3. С. 3-13.
7. Леликова Е.Ф., Рубина Л.И., Ульянов О.Н., Чащин М.А Параллельные вычисления в задачах, возникающих при математическом моделировании переноса излучения // Автоматика и телемеханика, 2007, № 5. С.126-140.
8. Леликова Е.Ф., Ульянов О.Н., Рубина Л.И., Чащин М.А. Параллельные вычисления в задачах, возникающих при математическом моделировании переноса излучения // В кн.: Энциклопедия низкотемпературной плазмы (Серия "Б", том VII-1 «Математическое моделирование в низкотемпературной плазме». Часть 3. Под ред. ч.к. РАН Ю.П. Попова. Гл. редактор энциклопедической серии академик В.Е. Фортов.). М.: ЯНУС-К, 2008. 698 с. С.514-522.
9. Чащин М.А., Л.И. Рубина, О.Н. Ульянов “Развитие алгоритмов для задач моделирования радиационного переноса”. Тезисы докладов Всероссийской школы молодых исследователей и V Всероссийской конференции “Актуальные проблемы прикладной математики и механики”, посвященной памяти академика А.Ф. Сидорова (Абрау-Дюрсо, 13-18 сентября 2010 г). Екатеринбург: УрО РАН, 2010. С. 86.
10. Чащин М.А., Е.Ф. Леликова, Л.И. Рубина, О.Н. Ульянов ”Численное моделирование переноса излучения в смесях веществ”. Тезисы международной конференции “X Забабахинские научные чтения”, РФЯЦ-ВНИИТФ, 2010г. Снежинск. С.272-273.
11. Чащин М.А., Е.Ф. Леликова, Л.И. Рубина, О.Н. Ульянов “Параллельные вычислительные технологии в задаче о переносе излучения”, Параллельные вычислительные технологии (ПаВТ’2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2010. (стр. 692). ISBN 978-5-696-03987-9 Электронное издание.].
12. Bowen C.; Lee R. W.; Ralchenko Yu. Comparing plasma population kinetics codes: Review of the NLTE-3 Kinetics Workshop // Journal of quantitative spectroscopy & radiative transfer 2006, vol. 99, no1-3, pp. 102-119.
13. The HITRAN Database // www.hitran.com
14. Rothman L.S. The HITRAN 2008 molecular spectroscopic database *// Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 110, pp. 533-572 (2009).
15. http://www.keldysh.ru/dvm

Параллельная реализация каталитической реакции ($CO + O_2 \rightarrow CO_2$) / Pt_{110} с помощью асинхронного клеточного автомата

А. Е. Шарифулина[1]

Институт вычислительной математики и математической геофизики СО РАН

В статье описывается клеточно-автоматная (КА) модель реакции окисления монооксида углерода (*CO*) на поверхности платины. Даётся формальное определение асинхронного КА и приводятся результаты моделирования каталитической реакции. Для достижения высокой эффективности распараллеливания асинхронный КА преобразуется в блочно-синхронный. Проводится сравнительный анализ основных характеристик моделирования, полученных для асинхронного и блочно-синхронного КА. И на основе полученных результатов делается вывод о приемлемой точности аппроксимации асинхронного режима блочно-синхронным для класса задач "реакция – диффузия". В статье представлены результаты распараллеливания блочно-синхронного КА и приведены оценки эффективности параллельной реализации.

1. Введение

Сегодня компьютерное моделирование является основным средством изучения явлений в большинстве наук: физике, химии, биологии, экономике, социологии. Традиционные методы моделирования, основанные на решении дифференциальных уравнений, недостаточно эффективны для описания распределенных динамических систем таких как популяции животных, человеческие сообщества, химические и физические процессы на микро-уровне. Эти явления существенно нелинейны и диссипативны, обладают способностью к самоорганизации и самовоспроизведению. Решение сложных систем дифференциальных уравнений с частными производными, использующихся для описания таких явлений, сопряжено со значительными математическими трудностями. Кроме того, пространственно распределенные, неоднородные системы, далёкие от состояния равновесия, невозможно описать в терминах дифференциальных уравнений. Сложное поведение таких нелинейных динамических систем в большинстве случаев можно определить простыми локальными правилами, описывающими явления на микро-уровне. Такие представления явлений легко выражаются в терминах клеточных автоматов.

Клеточный автомат (КА) представляет собой множество связанных по входам и выходам одинаковых конечных автоматов (клеток) с простыми детерминированными или вероятностными правилами переходов, вычисляющими новые состояния в зависимости от значений соседних клеток [1, 2]. КА наиболее эффективны для описания фазовых и бифуркационных переходов, где важно учитывать флуктуации, где коллективное поведение системы определяется локальным поведением составляющих ее элементов, когда система является неоднородной и представляется затруднительным определение каких-либо усредненных величин, способных адекватно отражать ее состояние в целом [3].

Одним из примеров таких систем являются реакции гетерогенного катализа, использующиеся сегодня во многих областях человеческой деятельности. Классической реакцией гетерогенного катализа является окисление монооксида углерода (*CO*) на металлах платиновой группы (Pt, Pd). Изучение механизма протекания каталитических процессов на Pt и Pd важно с точки зрения фундаментальной науки, так как особенностью этих реакций является возникновение в неравновесных условиях таких критических явлений, как множественность стационарных состояний, кинетические фазовые переходы, автоколебания, хаос, гистерезис [4]. Кроме того, окислительно-восстановительные реакции на металлах платиновой группы имеют важное практическое применение, т.к. они являются основой каталитических преобразователей, использующихся для очистки выхлопных газов.

Каталитические реакции представляют собой открытые нелинейные системы, поведение которых определяются взаимодействием частиц на атомно-молекулярном уровне, влиянием

температуры, влиянием процессов переноса вещества и тепла на скорость химического превращения. Асинхронный вероятностный клеточный автомат, известный ещё как кинетический метод Монте-Карло, позволяет отобразить нелинейность каталитических процессов непосредственным образом, моделируя взаимодействие реальных атомов и молекул с помощью дискретных правил переходов.

Для изучения пространственно-временной динамики каталитических реакций необходимо моделировать взаимодействие большого количества молекул ($\approx 10^{20}$) в течение длительного периода времени ($\approx 10^{10}$ итераций). Следовательно, решение таких задач требует использования эффективных алгоритмов распараллеливания. Проблема эффективного распараллеливания асинхронных КА с вероятностными правилами переходов на сегодняшний день до конца не решена. В работах [5, 6] предложен метод достижения высокой эффективности распараллеливания путём аппроксимации асинхронного КА блочно-синхронным КА. Блочно-синхронный режим работы нарушает стохастичность моделируемого процесса, поэтому эквивалентность эволюций асинхронного и блочно-синхронного КА не может быть доказана в общем случае.

Целью работы является сравнение эволюций асинхронного КА, моделирующего реакцию окисления *CO* на поверхности Pt, и блочно-синхронного; параллельная реализация блочно-синхронного КА и анализ эффективности распараллеливания.

Во втором разделе статьи приведены уравнения реакции, описывающие каталитическое окисление *CO* на Pt, представлена клеточно-автоматная модель реакции и результаты КА-моделирования. В третьем разделе описывается метод преобразования асинхронного режима работы КА в блочно-синхронный и проводится сравнительный анализ их эволюций. Четвёртый раздел посвящен параллельной реализации блочно-синхронного КА, приводятся алгоритм и оценки эффективности распараллеливания.

2. Моделирование каталитической реакция окисления CO на Pt

2.1 Описание механизм реакции

Экспериментальные и теоретические исследования, проведённые в [4, 7], показали, что в ходе реакции окисления происходит периодическая реконструкция структуры поверхности платины из гексагональной в кубическую ($hex \leftrightarrow 1\times1$). Реконструкция поверхности вызвана изменением каталитических свойств поверхности под воздействием адсорбированного *CO*. Детальный механизм реакции, учитывающий перестройку и изменение каталитических свойств поверхности, описан в [8]:

$$
\begin{array}{lrcll}
s_1: & CO_{gas} + * & \xrightarrow{p_1} & CO_{ads} & \text{– адсорбция } CO; \\
s_2: & CO_{ads}^{hex} & \xrightarrow{p_2} & CO_{gas} + *_{hex} & \text{– десорбция } CO \text{ с активного центра } *_{hex}; \\
s_3: & CO_{ads}^{1\times1} & \xrightarrow{p_3} & CO_{gas} + *_{1\times1} & \text{– десорбция } CO \text{ с активного центра } *_{1\times1}; \\
s_4: & 4CO_{ads} & \xrightarrow{p_4} & 4CO_{ads}^{1\times1} & \text{– переход поверхности под воздействием } CO_{ads} \text{ в состояние } 1\times1; \\
s_5: & *_{1\times1} & \xrightarrow{p_5} & *_{hex} & \text{– переход поверхности из состояния } 1\times1 \text{ в } hex; \\
s_6: & O_{2(gas)} + 2*_{1\times1} & \xrightarrow{p_6} & 2O_{ads}^{1\times1} & \text{– адсорбция кислорода на активные центры } *_{1\times1}; \\
s_7: & O_{2(gas)} + 2*_{hex} & \xrightarrow{p_7} & 2O_{ads}^{hex} & \text{– адсорбция кислорода на активные центры } *_{hex}; \\
s_8: & CO_{ads} + * & \xrightarrow{p_8} & *+ CO_{ads} & \text{– диффузия } CO_{ads} \text{ по поверхности}; \\
s_9: & CO_{ads} + O_{ads} & \rightarrow & CO_{2(gas)} + 2* & \text{– реакция между } CO_{ads} \text{ и } O_{ads}.
\end{array}
\qquad (1)
$$

Символ «$*_{hex}$» обозначает свободный активный центр поверхности с гексагональной структурой, «$*_{1\times1}$» – свободный активный центр поверхности с кубической структурой, а символ «*» – это свободный активный центр с любой структурой *hex* или 1×1. Активный центр,

– это атом поверхности катализатора, на котором адсорбируются молекулы. На одном центре может адсорбироваться только одна молекула, и адсорбция на одном центре не влияет на возможность адсорбции на соседние.

Согласно алгоритму, предложенному в [8], активные центры поверхности катализатора выбираются случайным образом, для выбранного центра с вероятностью p_i выбирается одна из элементарных стадий s_i, i = 1,...,9 приведенных выше (1). Вероятность выбора всех стадий, кроме s_9, вычисляется по формуле:

$$p_i = k_i / \sum_{l=1}^{8} k_l , \qquad (2)$$

где k_i – константа скорости i-ой стадии, причём $k_8 = M_{diff} \sum_{l=1}^{7} k_l$, где M_{diff} – параметр интенсивности диффузии. Стадия s_9 реализуется сразу же после выбора одной из следующих стадий: адсорбции CO (s_1), адсорбции O (s_6, s_7), диффузии (s_8). Это связано с тем, что взаимодействие между молекулами CO_{ads} и O_{ads}, оказавшихся на соседних активных центрах, происходит мгновенно.

Известно, что в условиях, далеких от равновесных, реакция окисления CO на поверхности платиновых металлов может сопровождаться появлением таких критических явлений как автоколебания, подвижные волны, хаос. Колебательный характер реакции был обнаружен в вычислительных экспериментах [7, 8] с помощью метода Монте-Карло при следующем наборе констант скоростей элементарных стадий:

$$k_1=14{,}7,\ k_2=4,\ k_3=0{,}03,\ k_4=3,\ k_5=2,\ k_6=56,\ k_7=0{,}056,\ M_{diff}=50. \qquad (3)$$

Колебания концентраций реагентов обусловлены обратимым фазовым переходом поверхности платины из состояния *hex* в 1×1, различная адсорбционная активность *hex* и 1×1 - поверхности является причиной периодической смены покрытий $CO_{ads} \leftrightarrow O_{ads}$

2.2 Клеточно-автоматная модель реакции окисления CO на Pt

Механизм реакции, описанный в разделе 2.1, реализуется с помощью асинхронного КА. Поверхности катализатора соответствует клеточный массива, молекулы и атомы, участвующие в реакции, – это состояния клеток. Элементарные стадии s_i описываются вероятностными правилам переходов КА. Асинхронный режим работы соответствует случайному выбору активных центров поверхности катализатора.

Асинхронный клеточный автомат определяется тремя понятиями [9]:

$$\aleph_\alpha = \langle A, X, \Theta \rangle ,$$

A – это алфавит состояний клеток, X – множество имен клеток, Θ – локальный оператор.

Алфавит состояний выбран в соответствии с реагентами, участвующими в реакции:

$$A = \{ *_{1\times1}, *_{hex}, CO_{ads}^{1\times1}, CO_{ads}^{hex}, O_{ads}^{1\times1}, O_{ads}^{hex} \}. \qquad (4)$$

Символы $*_{1\times1}$ и $*_{hex}$ обозначают свободный активный центр поверхности с гексагональной и кубической структурой соответственно; $CO_{ads}^{1\times1}$, CO_{ads}^{hex} – это молекула монооксида углерода, адсорбированная на 1×1 и *hex* поверхности; $O_{ads}^{1\times1}$, O_{ads}^{hex} – молекула кислорода, адсорбированная на 1×1 и *hex* поверхности соответственно.

Множество имён $X=\{(i, j): i=1..M_i, j=1..M_j\}$ определяется координатами клеток в дискретном пространстве, соответствующем поверхности катализатора. Клеткой называется пара $(u, (i, j))$, где $u \in A$ – это состояние клетки, $(i, j) \in X$ – имя клетки. На множестве имен вводятся именующие функции $\varphi(i, j): X \to X$, определяющие имена соседних клеток для клетки (i, j). Конечное множество именующих функций называется шаблоном соседства $T(i, j)$, ставящим в соответствие каждой клетке массива множество её соседей [10].

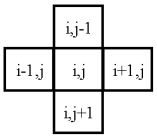
В исследуемой модели используются следующие шаблоны соседства (**Рис. 1**):

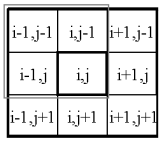
$$
\begin{aligned}
&T_1(i,j) = \{\varphi_0(i,j)\};\\
&T_5(i,j) = \{\varphi_0(i,j), \varphi_1(i,j), ...\varphi_4(i,j)\};\\
&T_9(i,j) = \{\varphi_0(i,j), \varphi_1(i,j), ...\varphi_8(i,j)\};\\
&T_{13}(i,j) = \{\varphi_0(i,j), \varphi_1(i,j), ...\varphi_{12}(i,j)\},
\end{aligned}
\tag{5}
$$

где именующие функции имеют следующий вид:

$$
\{\varphi_0(i,j), \varphi_1(i,j), .., \varphi_{12}(i,j)\} = \{(i,j), (i,j-1), (i+1,j), (i,j+1), (i-1,j), (i-1,j-1),
$$
$$
(i,j-2), (i+1,j-1), (i+1,j+1), (i-1,j+1), (i,j-2), (i+2,j), (i,j+2), (i-2,j)\}.
$$

$T_1(i,j) =$ 

$T_5(i,j) =$ 

$T_9(i,j) =$ 

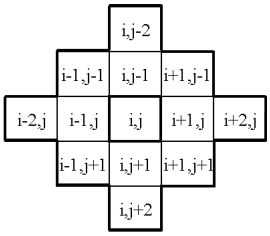
$T_{13}(i,j) =$ 

Рис. 1. Шаблоны соседства, использующиеся в КА-модели реакции окисления.

Локальный оператор $\Theta(i, j)$ определяет правила изменения состояний клеток в соответствии с уравнениями реакции и является сложной композицией подстановок и их суперпозиций:

$$
\Theta(i,j) = \{\theta_{(1,9)}, \theta_2, \theta_3, \theta_4, \theta_5, \theta_{(6,9)}, \theta_{(7,9)}, \theta_{(8,9)}\}, \tag{6}
$$

где θ_l, $l \in \{2, 3, 4, 5\}$ – подстановки, соответствующие стадиям реакции окисления, а $\theta_{(l,9)} = \theta_9(\theta_l)$, $l = 1,6,7,8$ – суперпозиции двух подстановок θ_l и θ_9. Подстановки θ_l и суперпозиции $\theta_{(l,9)}$ выбираются с вероятностью p_l, вычисляемой по формуле (2).

В КА-модели реакции окисления каждой элементарной стадии s_l соответствует подстановка θ_l. Например, адсорбция монооксида углерода на свободный активный центр (s_1) описывается подстановкой θ_1:

$$
\theta_1(i,j): \ \{(*,(i,j))\} \xrightarrow{p_1} \{(CO_{ads},(i,j))\} \tag{7}
$$

Отсутствие индексов 1×1 и *hex* означает, что подстановка применяется независимо от состояния поверхности и не изменяет её структуру.

Подстановки и суперпозиции изменяют состояние клеток в зависимости от состояний соседних клеток, принадлежащих соответствующему шаблону моделирования. Например, для применения подстановки θ_6 необходимо, чтобы две соседние клетки находились в состоянии $*_{1\times1}$:

$$
\theta_6(i,j): \ \{(*_{1\times1},(i,j))(*_{1\times1},\varphi_k(i,j))\} \xrightarrow{p_6} \{(O_{ads}^{1\times1},(i,j))(O_{ads}^{1\times1},\varphi_k(i,j))\}, \quad k = 1,2,3,4 \tag{8}
$$

Подстановки θ_l, $l \in \{1,2,3,5\}$ применяются к одной клетке, поэтому для них используется шаблон $T_1(i, j)$. Подстановка θ_4 применяется к блоку, состоящему из четырёх соседних клеток, блок выбирается случайным образом по шаблону $T_9(i, j)$. Для применения остальных подстановок θ_l, $l \in \{6,7,8,9\}$ требуются состояния двух клеток: (i, j) и одной из четырёх соседних клеток $\varphi_k(i, j)$, k =1,2,3,4. Соседняя для (i, j) клетка $\varphi_k(i, j)$ выбирается по шаблону $T_5(i, j)$ с вероятностью 0,25.

Суперпозиция подстановок $\theta_{(l,9)} = \theta_9(\theta_l)$, $l = 1,6,7,8$ предполагает применение подстановки θ_9 к результату выполнения θ_l, l = 1,6,7,8. Необходимость использования суперпозиции связана с особенностями реализации стадии s_9. При применении суперпозиции $\theta_{(l,9)} = \theta_9(\theta_l)$, $l = 6,7,8$ сначала к клетке с именем (i, j) применяется подстановка θ_l, которая по шаблону $T_5(i, j)$ выбирает одну из четырёх соседних клеток $\varphi_k(i, j)$, k =1,2,3,4. Сразу же после применения θ_l к выбранным клеткам (i, j) и $\varphi_k(i, j)$ применяется подстановка θ_9, которая также по шаблону $T_5(i, j)$ выбирает одну из четырёх соседних клеток. В результате объединения

шаблонов получаем, что при применении суперпозиции необходимо использовать шаблон $T_{13}(i,j)$:

$$T_{13}(i,j) = \bigcup_{k=1}^{4} T_5(\varphi_k(i,j)) \quad (9)$$

При применении суперпозиции $\theta_{(1,9)} = \theta_9(\theta_1)$ достаточно использования шаблона $T_5(i, j)$, т.к. подстановка θ_1 изменяет состояние одной клетки, и затем к этой же клетке применяется подстановка θ_9, которая по шаблону $T_5(i, j)$ выбирает одну из четырёх соседних клеток $\varphi_k(i, j)$.

Асинхронный режим функционирования КА предполагает, что локальный оператор $\Theta(i, j)$ применяется по очереди к случайно выбранным клеткам массива, сразу же изменяя их состояния. Далее асинхронный КА с определенными выше алфавитом A, множеством имён X и локальным оператором $\Theta(i, j)$ будем обозначать символом KA_α.

Реагенты, адсорбированные на поверхности катализатора, постоянно диффундируют, тогда как остальные процессы (адсорбция, десорбция, реконструкция поверхности) происходят намного реже, поэтому константа скорости диффузии s_8 значительно выше, чем константы скорости остальных стадий. В КА-модели это реализуется с помощью увеличения вероятности выбора подстановки, моделирующей диффузию, в M_{diff} раз по сравнению с суммой вероятностей остальных подстановок. В соответствии с [7, 8] значение M_{diff} выбирается в диапазоне 50 ÷ 100.

Время в КА-модели дискретно, весь процесс КА-моделирования разбивается на итерации. При моделировании реакции окисления итерация принимается равной $M_i \cdot M_j \cdot M_{diff}$ применениям локального оператора $\Theta(i, j)$ к случайно выбранным клеткам массива. За итерацию клеточный массив Ω переходит из одного глобального состояния $\Omega(t)$ в другое $\Omega(t+1)$, где t – номер итерации. Последовательность $\Sigma(\Omega)=\Omega(0),...,\Omega(t),\Omega(t+1),...,\Omega(t_{fin})$, полученная в результате итеративного функционирования КА, называется эволюцией, $\Omega(0)$ – исходное состояние клеточного массива, $\Omega(t)$ – состояние массива на t-ой итерации, t_{fin} – число итераций [10].

2.3 Результаты КА-моделирования

Компьютерное моделирование реакции окисления с помощью последовательной реализации KA_α выполнялось на клеточном массиве размером $M_i \times M_j$ = 200×200 клеток с периодическими граничными условиями. Вероятности применения подстановок θ_l вычисляются по формуле (2) для констант скорости k_l (3): $p_1 = 3{,}613 \cdot 10^{-3}$, $p_2 = 9{,}83 \cdot 10^{-4}$, $p_3 = 7{,}37 \cdot 10^{-6}$, $p_4 = 7{,}37 \cdot 10^{-4}$, $p_5 = 4{,}92 \cdot 10^{-4}$, $p_6 = 1{,}3762 \cdot 10^{-2}$, $p_7 = 1{,}38 \cdot 10^{-5}$, $p_8 = 9{,}80392 \cdot 10^{-1}$. Параметр диффузии $M_{diff} = 50$. В исходном состоянии все клетки массива находятся в состоянии $*_{hex}$.

В качестве характеристик КА-моделирования реакции окисления выбраны следующие величины:

- концентрации реагентов, адсорбированных на поверхности катализатора: $n(O_{ads})$, $n(CO_{ads})$;
- скорость образования CO_2: $v(CO_2)$;
- доли поверхности с кубической и гексагональной структурой: $f(1\times1)$, $f(hex)$.

Концентрация адсорбированных на поверхности катализатора веществ вычисляется после каждой итерации как отношение количества клеток (N), находящихся в состоянии, соответствующем данному реагенту, к размеру клеточного массива ($M_i \cdot M_j$):

$$n(CO_{ads}) = \frac{N(CO_{ads}^{1\times1}) + N(CO_{ads}^{hex})}{M_i \cdot M_j} \quad (10)$$

$$n(O_{ads}) = \frac{N(O_{ads}^{1\times1}) + N(O_{ads}^{hex})}{M_i \cdot M_j} \quad (11)$$

Скорость образования CO_2 вычисляется как отношение числа применений подстановки θ_9, произошедших за итерацию, к размеру клеточного массива:

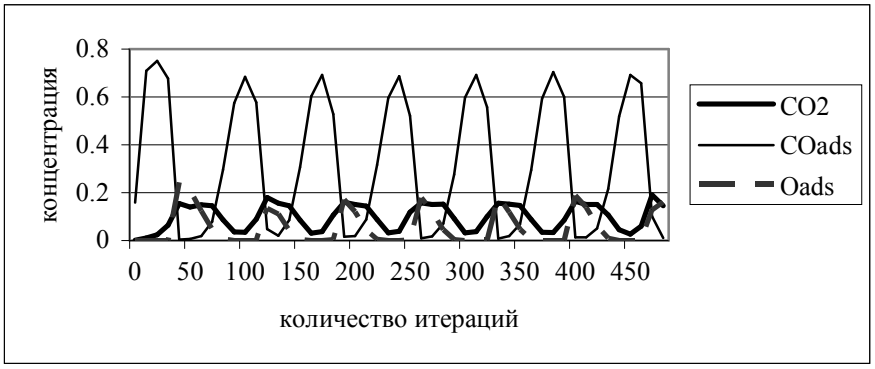
$$v(CO_2) = \frac{N(CO_{ads} + O_{ads})}{M_i \cdot M_j} \tag{12}$$

Доля 1×1 и *hex* поверхности вычисляется после каждой итерации по следующим формулам:

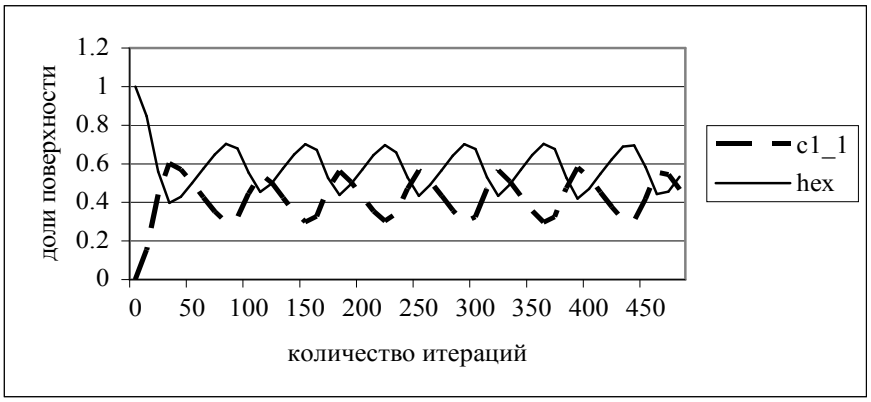
$$f(1\times1) = \frac{N(*_{1\times1}) + N(CO_{ads}^{1\times1}) + N(O_{ads}^{1\times1})}{M_i \cdot M_j} \tag{13}$$

$$f(hex) = \frac{N(*_{hex}) + N(CO_{ads}^{hex}) + N(O_{ads}^{hex})}{M_i \cdot M_j} \tag{14}$$

В результате КА-моделирования реакции окисления получены колебания концентраций реагентов $n(O_{ads})$, $n(CO_{ads})$, скорости образования CO_2 – $v(CO_2)$ и доли поверхности с кубической и гексагональной структурой $f(1\times1)$, $f(hex)$ (**Рис. 2**).



a)



b)

Рис. 2. Характер колебаний в реакции окисления *CO* на Pt:
a) концентрация CO_{ads}, O_{ads} и скорость образования CO_2; b) доли 1×1 и *hex* поверхности.

Колебания концентраций реагентов в реакции окисления *CO* наблюдаются только при определённых значениях констант скорости реакции. С помощью исследования эволюции KA_α построена бифуркационная диаграмма в пространстве констант скорости адсорбции кислорода $k_6 \in [0;10^5]$ и монооксида углерода $k_1 \in [0;200]$ (**Рис. 3**). Точками обозначены значения, полученный в результате КА-моделирования, пунктирная линия построена с помощью линейной интерполяции.

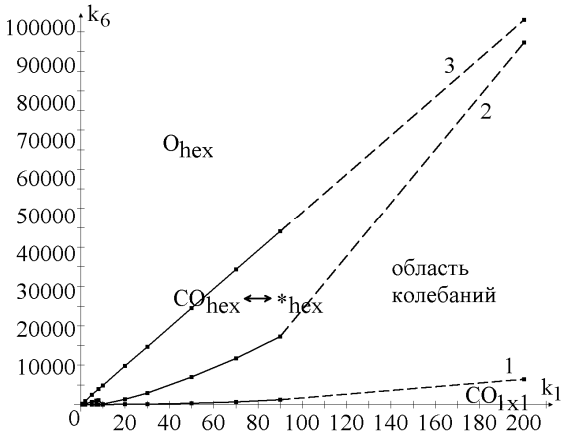


Рис. 3. Бифуркационная диаграмма реакции окисления *CO* на Pt.

В зависимости от значений констант k_6 и k_1 обнаружено наличие трёх различных режимов протекания реакции:

- два равновесных состояния – поверхность катализатора покрывается $CO_{1\times1}$ (ниже границы 1) либо O_{hex} (выше границы 3);
- режим колебаний концентраций реагентов и скорости образования CO_2 (область колебаний между кривыми 1 и 2)
- периодическая смена $CO^{hex}_{ads} \leftrightarrow *_{hex}$, при которой доля 1×1 поверхности мала, и адсорбция кислорода практически не происходит (область между кривыми 2 и 3).

3. Блочно-синхронный режим функционирования КА

3.1 Преобразование асинхронного КА в блочно-синхронный

Изучение пространственно-временной динамики реакции окисления *CO* требуют проведения вычислительных экспериментов с использованием клеточных массивов размером $M_i \times M_j$ = 8000×8000 клеток в течение 10^6 итераций. Решение таких задач на однопроцессорной машине займёт несколько месяцев, поэтому необходимо использовать эффективные алгоритмы распараллеливания. Эффективное распараллеливание асинхронных КА является трудновыполнимой задачей, так как при асинхронном режиме межпроцессорный обмен данными приходится выполнять после изменения каждой граничной клетки. Поэтому асинхронный режим работы изменяется на блочно-синхронный, который вводит частичную синхронизацию режима функционирования, не нарушая при этом условия корректности, т.е. состояние клетки одновременно не могут изменить разные подстановки.

Преобразование асинхронного КА $KA_\alpha = \langle A, X, \Theta \rangle$ в блочно-синхронный $KA_\beta = \langle A, X, \Theta \rangle$ выполняется следующим образом [5].

1) На множестве имён X определяется шаблон, называемый блоком $B(i, j)$. Для выполнения условия корректности блок должен включать в себя все шаблоны соседства (4):

$$T_1(i,j) \subset T_5(i,j) \subset T_9(i,j) \subset T_{13}(i,j) \subseteq B(i,j) \quad \Rightarrow \quad B(i,j) = T_{13}(i,j),\ |B(i,j)| = 13. \qquad (15)$$

Блок $B(i, j)$ определяет на множестве имён множество разбиений $\Pi = \{X_1, X_2, ..., X_{13}\}$ таких, что для всех X_k, $k = 1,...,|B(i,j)|$ выполняются соотношения:

$$|X_k| = \frac{|X|}{|B(i,j)|}, \qquad \bigcup_{k=1}^{|B(i,j)|} X_k = X, \qquad X_k \cap X_l = \varnothing \quad \forall\ k,l = 1,...,|B(i,j)|; \qquad (16)$$

$$\bigcup_{(i,j)\in X_k} B(i,j) = X, \qquad B(i,j) \cap B(k,l) = \varnothing \quad \forall (i,j),(k,l) \in X_k. \qquad (17)$$

2) Итерация разбивается на $|B(i,j)| = 13$ этапов. На каждом этапе локальный оператор $\Theta(i,j)$ применяется ко всем клеткам выбранного случайным образом разбиения X_k. Порядок выбора клеток внутри разбиения X_k не важен, т.к. условие (15) гарантирует, что локальный оператор применяется к разным блокам и шаблоны применения подстановок не пересекаются.

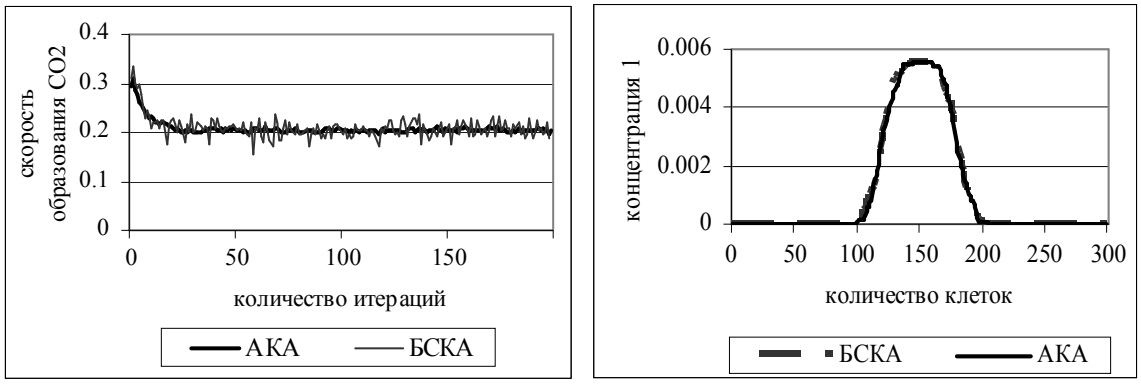
3.2 Сравнение асинхронного и блочно-синхронного режимов работы КА

При блочно-синхронном режиме работы на каждом этапе локальный оператор применяется только к клеткам одного разбиения X_k, что существенно уменьшает стохастичность выбора клеток. Эволюция блочно-синхронного КА для некоторых классов задач не совпадает с эволюцией асинхронного КА, например, для КА со взвешенными шаблонами, вычисляющих новые состояния в зависимости от взвешенной суммы состояний соседних клеток. При асинхронном и блочно-синхронном режимах КА со взвешенными шаблонами для некоторых последовательностей случайных чисел, определяющих выбор клеток, формируются различные устойчивые структуры (**Рис. 4**).

a) b)

Рис. 4. Устойчивые состояния КА со взвешенными шаблонами при одинаковых начальных значениях и различных режимах функционирования: a) асинхронный; b) блочно-синхронный.

В общем случае невозможно доказать эквивалентность эволюций асинхронного и блочно-синхронного КА, каждая задача требует проведения отдельного анализа результатов КА-моделирования. Вычислительные эксперименты для различных моделей класса "реакция – диффузия" подтверждают возможность использования блочно-синхронного преобразования для этого класса задач. Например, при реализации модели Ziff-Gulari-Barshad (ZGB) с помощью асинхронного и блочно-синхронного КА значения скорости образования CO_2, концентраций CO_{ads} и O_{ads} очень близки.



a) b)

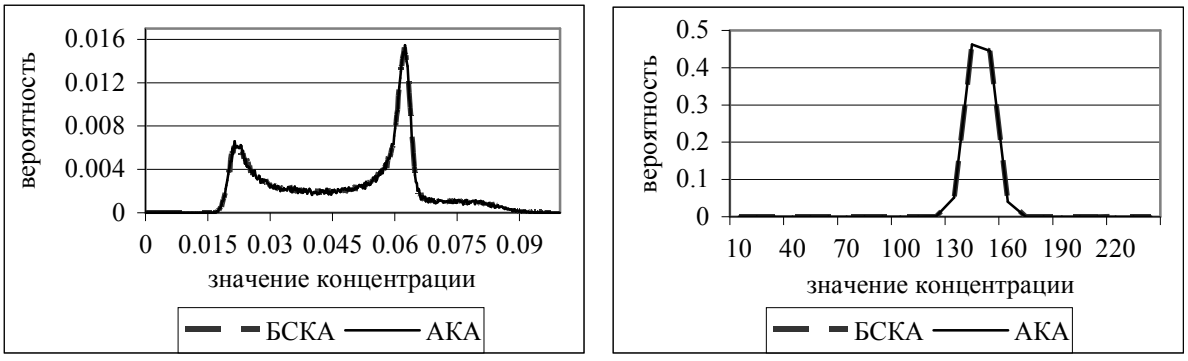
Рис. 5. Сравнение асинхронного и блочно-синхронного режимов моделирования задач класса "реакция-диффузия": a) скорость образования CO_2 в КА-модели ZGB ; b) концентрация единиц в КА-модели диффузии.

На **Рис. 5a** представлены изменения скорости образования CO_2 при вероятности адсорбции CO 0,5. Уровень клеточно-автоматного шума для значений, полученных с помощью блочно-синхронного КА при $B(i, j) = T_9(i, j)$, находится в пределах допустимой величины и при увеличении размера блока уменьшается. Ещё одним примером КА-модели, допускающей преобразование асинхронного режима в блочно-синхронный, является диффузия. При моделировании диффузии с помощью асинхронного и блочно-синхронного КА осреднённые значения концентрации единиц совпадают (**Рис. 5b**).

Для проверки возможности применения блочно-синхронного преобразования в случае КА-моделирования реакции окисления CO на Pt (1), сравнивались результаты вычислительных экспериментов, выполненных при $M_i \times M_j$ = 200×200 клеток, $t_{fin}=10^5$ итераций и для значений констант скорости (3). В качестве параметров сравнения выбраны следующие характеристики:

- распределение вероятностей $n(O_{ads})$, $n(CO_{ads})$, $\nu(CO_2)$, $f(1\times1)$, $f(hex)$ и периодов колебаний T;
- математическое ожидание и дисперсия $n(O_{ads})$, $n(CO_{ads})$, $\nu(CO_2)$, $f(1\times1)$, $f(hex)$, T и доверительные интервалы для математического ожидания и дисперсии;
- бифуркационная диаграмма реакции окисления.

Анализ результатов моделирования показал, что значения распределений вероятностей $n(O_{ads})$, $n(CO_{ads})$, $\nu(CO_2)$, $f(1\times1)$, $f(hex)$, T, полученные с помощью KA_α и KA_β, очень близки. На **Рис. 6** представлено распределение вероятностей $\nu(CO_2)$ и периодов колебаний. Среднеквадратичные разности распределений вероятностей этих характеристик для асинхронного и блочно-синхронного КА не превосходят 10^{-4}. Например, среднеквадратичные разности распределений вероятностей $\nu(CO_2)$ и периодов T составляют $E(\nu(CO_2)) = 5.9 \cdot 10^{-5}$ и $E(T) = 7.1 \cdot 10^{-5}$.



a) b)

Рис. 6. Сравнение значений распределения вероятностей, вычисленных с помощью KA_α и KA_β: a) скорость образования CO_2; b) периоды колебаний.

Математическое ожидание ($M\xi$), дисперсия ($D\xi$) и доверительные интервалы для математического ожидания $I_{M\xi}$ и дисперсии $I_{D\xi}$, вычисленные для скорости образования CO_2 и периодов колебаний, приведены в таблице 1. Доверительные интервалы рассчитывались с доверительной вероятностью γ=0.95. Из таблицы видно, что значения числовых характеристик для концентраций адсорбированных реагентов, скорости образования CO_2 и доли 1×1 и hex поверхности, полученных в результате моделирования с помощью асинхронного и блочно-синхронного КА, отличаются незначительно.

Бифуркационная диаграмма, построенная в результате анализа эволюции блочно-синхронного КА в пространстве констант скорости адсорбции кислорода k_6 и монооксида углерода k_1, совпадает с диаграммой, вычисленной с помощью асинхронного КА (**Рис. 3**). Это подтверждает, что KA_α и KA_β демонстрируют одинаковый характер поведения реакции окислении.

Таблица 1. Статистические характеристики $v(CO_2)$ и T, полученные с помощью $КА_\alpha$ и $КА_\beta$.

Характеристики	$M\xi$	$D\xi$	$I_{M\xi}$	$I_{D\xi}$
$v(CO_2)^{АКА}$	0.048841	0.000317	(0.048805; 0.048876)	(0.000316; 0.000318)
$v(CO_2)^{БСКА}$	0.048826	0.000316	(0.048791; 0.048862)	(0.000315; 0.000317)
$T^{АКА}$	14.490580	0.461572	(14.485427;14.495732)	(0.456621;0.466523)
$T^{БСКА}$	14.469609	0.538590	(14.464047; 14.475171)	(0.532817; 0.544363)

Полученные результаты свидетельствуют о совпадении эволюций асинхронного и блочно-синхронного КА.

4. Результаты распараллеливания блочно-синхронного КА

Распараллеливание блочно-синхронного КА заключается в разделении клеточного массива $\Omega(A, X)$ на домены $|Dom| = \frac{|X|}{n}$, которые распределяются между n процессами. Каждый процесс вычисляет новые значения клеток своего домена и пересылает граничные значения соседним процессам. Обмен граничными значениями выполняется в конце каждого этапа, т.к. условие (15) гарантирует, что при применении локального оператора $\Theta(i, j)$ к клеткам выбранного разбиения X_k не понадобятся значения состояний клеток, вычисленных на текущем этапе [5]. Объём пересылаемых данных составляет $2 \cdot P_{Dom}$ байт, где P_{Dom} – периметр домена.

Распараллеливание блочно-синхронного КА выполнялось на суперкомпьютере «МВС-100К» (МСЦ РАН) с использованием библиотеки MPI. При распараллеливании используется гибридная модель MPI+OpenMP. На каждом вычислительном узле запускается 4 MPI-процесса, в каждом из которых запускается по 2 потока. Для эффективного использования ресурсов вычислительного модуля потоки явным образом назначаются на ядра, объединённые общей кэш-памятью. Результаты распараллеливания клеточного массива размером $|X|$=8000×8000 представлены в таблице 2. В качестве характеристик распараллеливания рассматриваются: T_n – время вычислений с использованием n процессов, $S(n) = \frac{T_1}{T_n}$ – ускорение и $Q(n) = \frac{T_1}{T_n \cdot n}$ – эффективность распараллеливания.

Таблица 2. Характеристики распараллеливания блочно-синхронного КА

n	1	4	16	32	64	128
T_n, s	266,68	67,62	18,15	9,45	5,06	2,85
$S(n)$	1	3,94	14,69	28,36	52,69	93,73
$Q(n)$	1	0,99	0,92	0,89	0,82	0,73
$\|Dom\|$	$2{,}56 \cdot 10^8$	$6{,}4 \cdot 10^7$	$1{,}6 \cdot 10^7$	$8 \cdot 10^6$	$4 \cdot 10^6$	$2 \cdot 10^6$

Данные приведенные в таблице показывают, что при использовании до 128 процессов эффективность $Q(n)$ выше 80%, при дальнейшем увеличении числа процессов эффективность распараллеливания падает. Это связано с возрастанием накладных расходов на обеспечение обмена данными между вычислительными узлами и недостаточной загрузкой ядер. Для достижения высокой эффективности размер домена $|Dom|$ должен превышать $2{,}04 \cdot 10^6$ клеток.

5. Заключение

В работе реализован асинхронный клеточный автомат, моделирующий реакцию окисления *CO* на поверхности Pt. Построенная КА-модель демонстрирует колебания скорости реакции, концентраций веществ, адсорбирующихся на поверхности катализатора, и долей 1×1 и *hex*

поверхности. Колебания сопровождаются различными волновыми процессами на моделируемой поверхности.

Для достижения высокой эффективности распараллеливания выполнено преобразование асинхронного КА в блочно-синхронный. Для проверки эквивалентности эволюций KA_α и KA_β сравнивались значения распределения вероятностей концентраций и периодов колебаний, математическое ожидание и дисперсия распределения вероятностей, доверительные интервалы и бифуркационные диаграммы реакции окисления. Статистические характеристики, полученные в результате моделирования с помощью асинхронного и блочно-синхронного КА, отличаются незначительно, что свидетельствуют о применимости преобразования KA_α в KA_β для реакции окисления *CO*.

Выполнено распараллеливание блочно-синхронного КА, моделирующего реакцию окисления *CO* на Pt. Анализ характеристик распараллеливания при $|X|$=8000×8000 клеток показал, что для достижения высокой эффективности размер домена $|Dom|$ должен превышать $2{,}04 \cdot 10^6$ клеток.

Литература

1. Wolfram S. «New Kind of Science» // Wolfram Media, Inc. 2002 // http://www.wolframscience.com/
2. Тоффоли Т., Марголус Н. Машины клеточных автоматов. М.: Мир, 1991. - С. 269.
3. Ванаг В.К. Исследование пространственно распределенных динамических систем методами вероятностного клеточного автомата // Успехи физических наук. Обзоры актуальных проблем, 1999. - Т. 169. - № 5. - С. 481-505.
4. Ronald Imbihl and Gerhard Ertl. Oscillatory Kinetics in Heterogeneous Catalysis // Chemical Reviews, 1995. - Vol. 95, No. 3. – P. 697-733.
5. Бандман О.Л. Параллельная реализация клеточно-автоматных алгоритмов моделирования пространственной динамики // Сибирский журнал вычислительной математики. 2007. - № 4. – С. 345-361.
6. Nedea S.V., Lukkien J.J., Jansen A.P.J., Hilbers P.A.J. Methods for parallel simulations of surface reactions // arXiv:physics/0209017. – V. 1. - 2002.
7. Elokhin V. I., Latkin E. I., Matveev A. V., and Gorodetskii V. V. Application of statistical lattice models to the analysis of oscillatory and autowave processes in the reaction of carbon monoxide oxidation over platinum and palladium surfaces // Kinetics and Catalysis. – 2003. - V. 44. - №. 5. - P. 692-700.
8. Latkin E.I., Elokhin V.I., Gorodetskii V.V. Monte Carlo model of oscillatory CO oxidation having regard to the change of catalytic properties due to the adsorbate-induced Pt(100) structural transformation // Journal of Molecular Catalysis A: Chemical. - 2001. -V. 166. - P. 23–30.
9. Бандман О.Л. Клеточно-автоматное моделирование диффузионно-реакционных процессов // Автометрия. - 2003. -Т. 39. - № 3. - С. 1-16.
10. Бандман О.Л. Клеточно-автоматные модели пространственной динамики. // Системная информатика - Методы и модели современного программирования. -2006, №10. – С. 59 – 113.

Intel MKL Poisson Library for scalable and efficient solution of elliptic problems with separable variables

A. Kalinkin, A. Kuzmin

Intel Corporation

Intel®MKL Poisson Library is a collection of routines that combine discrete Fourier transforms and LU decomposition to solve Laplace, Poisson, and Helmholtz mesh problems with separable variables. The routines provide an approximate solution of some two- and three-dimensional problems with an arbitrary combination of boundary conditions, Dirichlet, Neumann, or periodic, in Cartesian or spherical coordinate systems. Intel MKL®Poisson Library is optimized for modern Intel multi-core processors and demonstrates excellent performance and scalability compared to similar solvers.

1. Introduction

Among elliptic boundary value problems, the class of problems with separable variables can be solved fast and directly. Elliptic problems with separable variables usually assume that the computational domains are simple e.g., parallelepiped or sphere, and constant coefficients [1]. This kind of problems can serve to generate preconditioners in iterative procedures for more complex methods. They can also be used in low-accuracy models similar to the ones used in Numerical Weather Simulations.

Intel® MKL Poisson Library enables solving Laplace, Poisson, and Helmholtz mesh problems with separable variables. The routines provide an approximate solution of some two- and three-dimensional problems with an arbitrary combination of boundary conditions, Dirichlet, Neumann, or periodic, in Cartesian or spherical coordinate systems.

The problems with the separable variables can be suboptimal in the sense that they require slightly more arithmetic operations (up to logarithmic factor) than the number of unknowns to compute the solution to the problem. This statement is true if, for example, the sizes of the discretized problems are powers of 2.

Computational Mathematics suggests that we take into consideration not only arithmetic operations, but also the cost of memory operations as well. Modern computers can perform computations at a very high speed, while lacking the ability to deliver data to the computational units. Keeping in mind that a memory operation can easily be dozen to hundred times slower than an arithmetic one, a computationally optimal algorithm could compute the solution slower than memory optimal algorithm.

The recent developments in processor industry resulted in multicore processors become standard processors not only for powerful clusters, but also for home computers and laptops. Therefore, the algorithm can also be non-optimal from the parallelization point of view. Optimality here can be understood in terms of the number of synchronization points and/or in terms of the amount of data that needs to be transferred between different cores.

In summary, the modern computational algorithm should focus on 3 key aspects, namely, parallelization, memory operations, and arithmetic costs.

This work extends previously published paper [2] with new routines and performance comparison.

The purpose of this article is to demonstrate on a simple 3D problem with separable variables that taking into account modern model the solution can be computed efficiently and fast. This would also help developers to compute solution to the problem with separable variables really negligible with respect to other computations. To complete our goal, we will evaluate software provided by Intel Corporation. In particular, we focus on the comparison (where possible) of NAG* SMP Library provided at [3] and Intel® MKL provided at [4].

2. Problem Statement

We are going to use the following notation for boundaries of a parallelepipedal domain $a_x < x < b_x$, $a_y < y < b_y$, $a_z < z < b_z$ in Cartesian space:

$bd\_a_x = \{x = a_x, a_y \le y \le b_y, a_z \le z \le b_z\}$, $bd\_b_x = \{x = b_x, a_y \le y \le b_y, a_z \le z \le b_z\}$

$bd\_a_y = \{a_x \le x \le b_x, y = a_y, a_z \le z \le b_z\}$, $bd\_b_y = \{a_x \le x \le b_x, y = b_y, a_z \le z \le b_z\}$

$bd\_a_z = \{a_x \le x \le b_x, a_y \le y \le b_y, z = a_z\}$, $bd\_b_z = \{a_x \le x \le b_x, a_y \le y \le b_y, z = b_z\}$.

The wildcard "*" may stand for any of the symbols a_x, b_x, a_y, b_y, a_z, b_z so that bd_* denotes any of the above boundaries.

The 2D Helmholtz problem is to find an approximate solution of the Helmholtz equation $-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} + qu = f(x, y, z),\ \ q = const \ge 0$, in a parallelepiped, that is, a domain $a_x < x < b_x$, $a_y < y < b_y$, $a_z < z < b_z$ with one of the following boundary conditions on each boundary bd_*:

- The Dirichlet boundary condition: $u(x, y, z) = G(x, y, z)$
- The Neumann boundary condition: $\frac{\partial u}{\partial n}(x, y, z) = g(x, y, z)$, where

$n = -x$ on bd_a_x, $n = x$ on bd_b_x, $n = -y$ on bd_a_y, $n = y$ on bd_b_y, and $n = -z$ on bd_a_z, $n = z$ on bd_b_z.

- The periodic boundary condition: $u(a_x, y, z) = u(b_x, y, z)$, $u(x, a_y, z) = u(x, b_y, z)$ or $u(x, y, a_z) = u(x, y, b_z)$

We can see that the Poisson problem can be obtained from the Helmholtz problem by setting the Helmholtz coefficient q to zero. The Laplace problem can be obtained by setting the right-hand side f to zero in addition to Helmholtz coefficient.

To find an approximate solution for 3D problems, a uniform mesh is built in the parallelepipedal domain:

$$x_i = a_x + ih_x,\ x_j = x(j)$$
$$i = 0,..,n_x,\ \ h_x = \frac{b_x - a_x}{n_x},$$
$$a_x = x(0) < .. < x(j) < .. < x(n_j) = b_x$$

It is possible to use the standard five-point finite difference approximation on this mesh to compute the approximation to the solution:

We assume that the values of the approximate solution are computed in the mesh points (x_i, y_i, z_i), provided the values of the right-hand side *f(x, y, z)* at these points are given and the values of the appropriate boundary functions *G(x, y, z)* and/or *g(x, y, z)* in the mesh points laying on the boundary of the rectangular domain are known.

Discrete Fourier Transform (DFT) computations are highly dependent on the dimension. For powers of 2, the DFT requires the least possible number of operations, while for the primes the number of opera-

tions reaches its maximum value. We consider only dimensions that are powers of 2 as the difficult test case with a high data movement to operations ratio.

3. Single precision performance

We first look at single precision computations that are of value for Numerical Weather Simulation problems and consider the Poisson Library (PL) from Intel® MKL. PL does computations in four steps by consecutive calls to the s_init_helmholtz_2d, s_commit_helmholtz_2d, s_helmholtz_2d, and free_helmholtz_2d routines. We measure the total time spent in computations.

We consider the homogeneous Dirichlet problem with an exact solution $u(x, y, z) = \sin \pi x \sin \pi y \sin \pi z$ on the rectangular domain 0<x<1, 0<y<1, 0<z<1 as our test case.

All test cases were compiled with Intel® Fortran compilers (version 11.1). We ran each piece of code 10 times in a loop and then selected the best time out of the ten collected. Time measurements were completed using the dsecnd routine from Intel® MKL. We also used the Poisson Library from Intel® MKL 10.3 Update 7.

Figure 1 shows scalability of Intel® MKL PL routines in 3D Cartesian case. We measure computational time spent in four PL routines for different regular mesh sizes starting from 32x32x32 and ending with 512x512x512 and with different numbers of OMP threads.

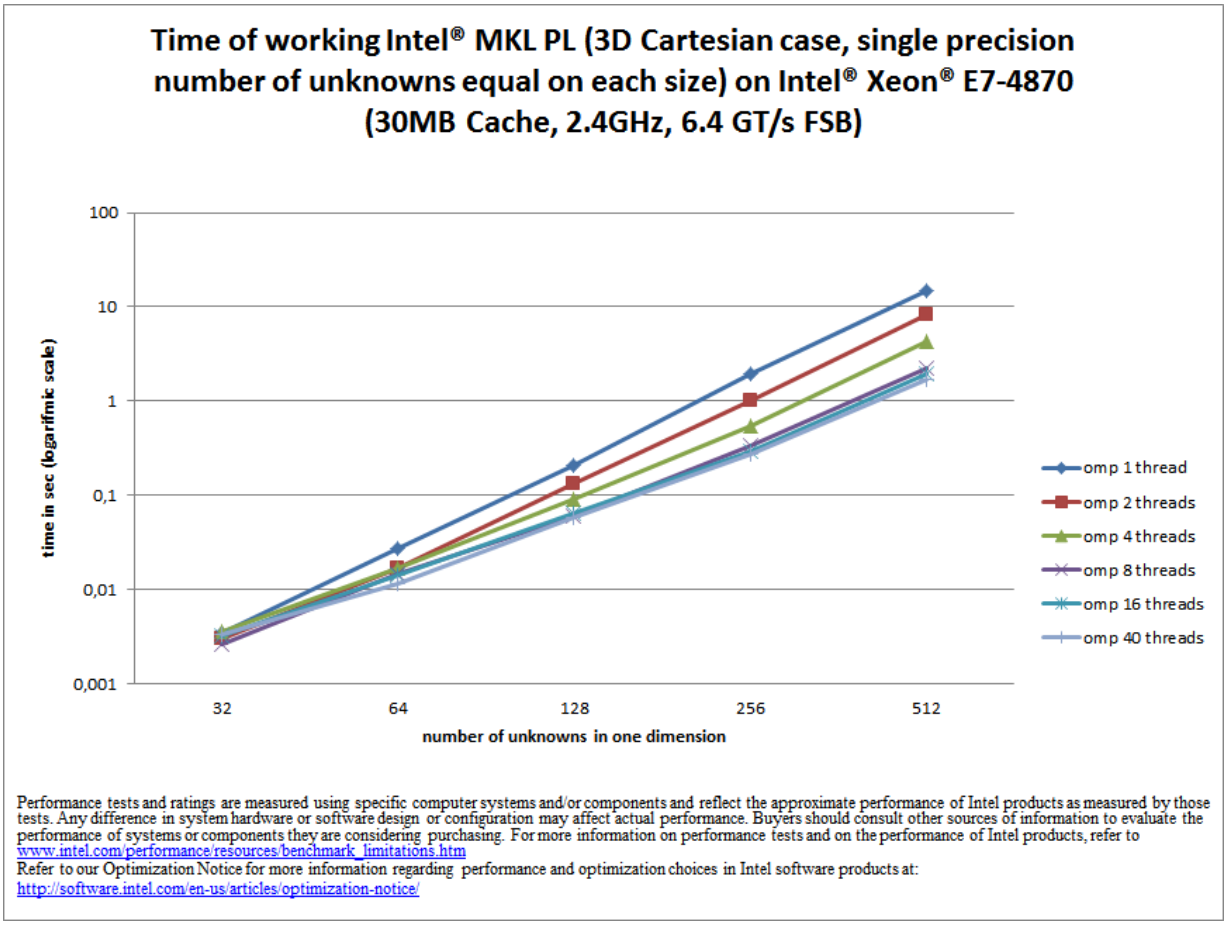


Figure 1. Scalability of Intel® MKL PL (3D Cartesian case) (single precision)

4. Double precision performance

We next look at double precision computations that are of value for preconditioning of elliptic problems with slightly varying coefficients and we consider the D03FAF routine from NAG* SMP Library and the Poisson Library from Intel® MKL. The D03FAF routine is able to compute the solution to the Helmholtz problem in a Cartesian coordinate system in a single step. PL does computations in four steps by consecutive calls to the double precision routines d_init_helmholtz_2d, d_commit_helmholtz_2d, d_helmholtz_2d, and free_helmholtz_2d. For fairness, we measured the total time spent in computations for both software libraries.

We consider the same homogeneous Dirichlet problem with the same exact solution $u(x, y, z) = \sin \pi x \sin \pi y \sin \pi z$ on the same rectangular domain 0<x<1, 0<y<1, 0<z<1 as our test case.

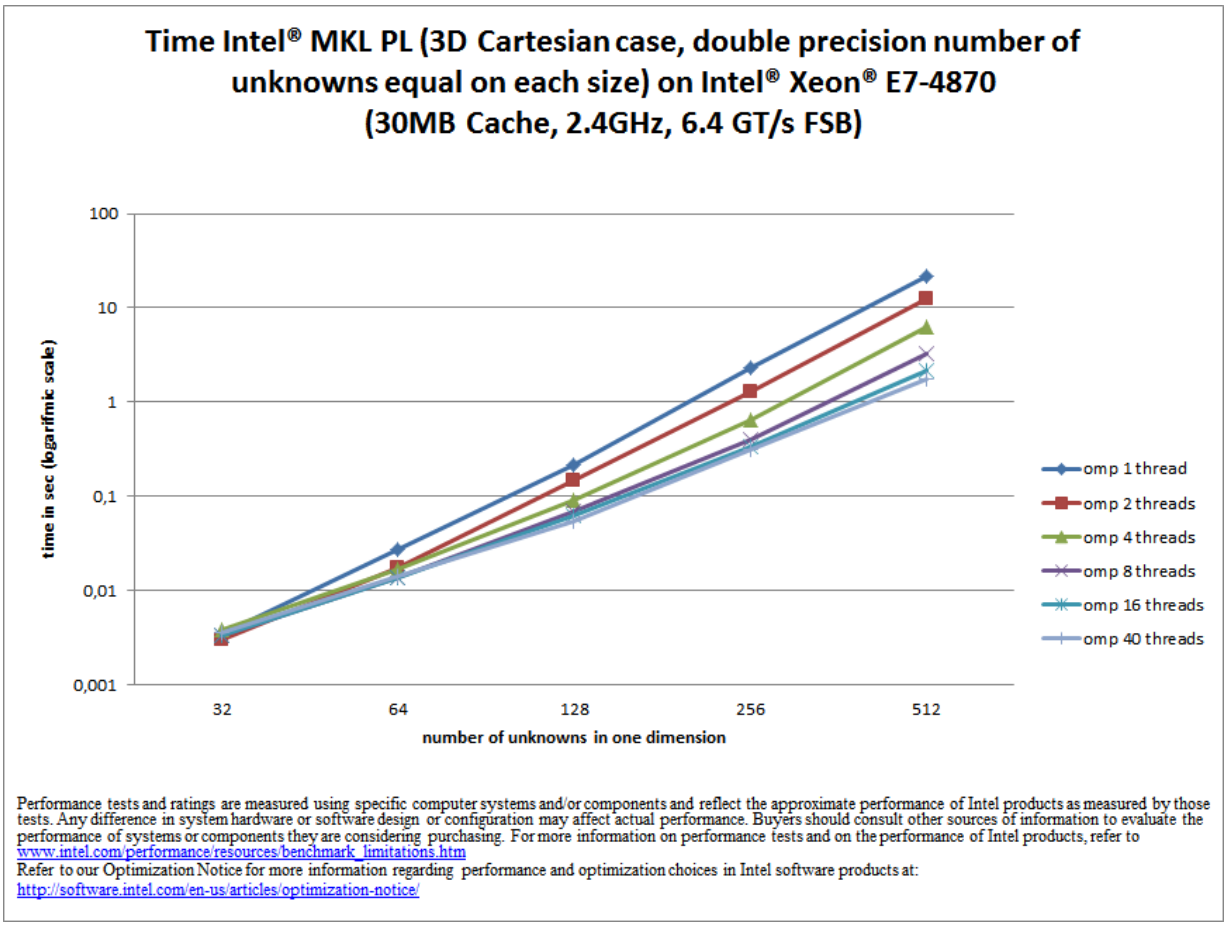


Figure 2. Scalability of Intel® MKL PL (3D Cartesian case) (double precision)

Figure 2 shows scalability of Intel® MKL PL routines in 3D Cartesian case in double precision arithmetic. Figure 3 shows the ratio of computational time spent in the D03FAF routine and computational time spent in four PL routines for different regular mesh sizes starting from 32 and ending with 512 mesh points in each dimension. When the ratio curve is above 1, PL Helmholtz 2D solver is faster than D03FAF.

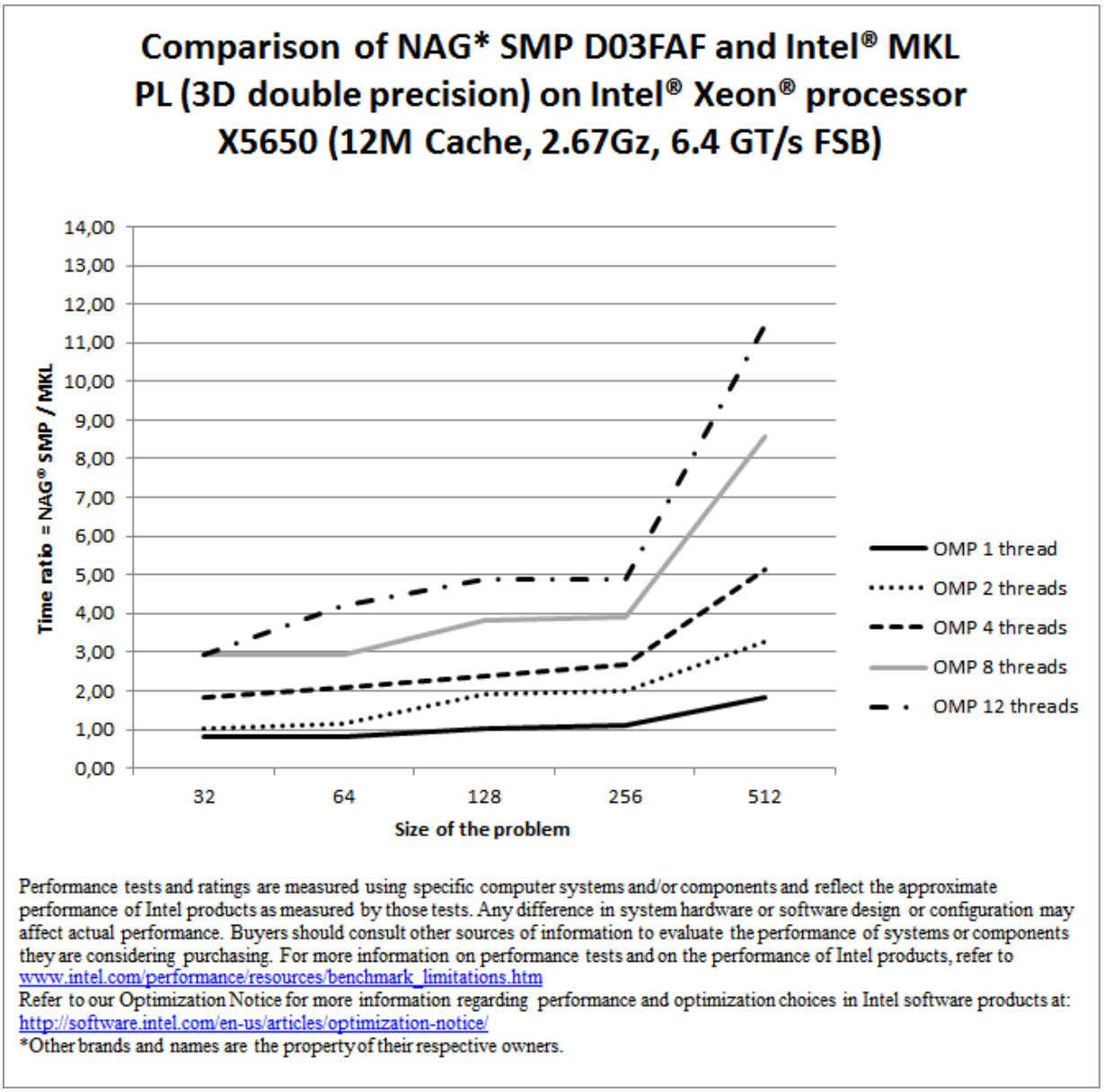


Figure 3. Comparison of NAG* **SMP** D03FAF and Intel® MKL PL (3D double precision)

From Figure 3, it can be seen that even on a single thread, the PL routines can be roughly two times faster than the D03FAF routine. When threaded, the advantage grows up to 11 times on 12 threads. It is also clear that PL has heavier interface that results in essentially slower performance for small problems (sizes up to 100). We think that small size problems are not of great interest for HPC area, so the lower performance of PL in this range should not be considered as a problem. However, PL can regain some performance in the case when the solution of problems different in right-hand side only as it can do pre- (init and commit) and post-computational (free) step only once unlike the D03FAF routine.

5. Conclusions

We have investigated the performance of software available for solving 3D Helmholtz problems with separable variables in parallelepipedal domains. We found that the implementation of the solver from

NAG SMP Library shows good serial performance for small size problems (up to 64 mesh points in one dimension). The optimized implementation of Intel® MKL PL shows better performance and scalability on larger problem sizes. Performance gains are up to 10 times for some dimensions. Both libraries produce solutions with the same level of accuracy.

References

1. A. A. Samarskii and E. S. Nikolaev, Methods of Solution of Grid Problems, Nauka, Moscow (1978) (in Russian)

2. A. A. Kalinkin, Y.M. Laevsky, S.V. Gololobov, 2D Fast Poisson Solver for High-Performance Computing, Parallel Computing Technologies, Lecture Notes in Computer Science 2009, Vol. 5698/2009

3. NAG® SMP Library, http://www.nag.co.uk/numeric/fl/fsdescription.asp

4. Intel® Math Kernel Library, http://www.intel.com/software/products/mkl

Анализ средств визуального программирования параллельных вычислений*

В.Л. Авербух[1], М.О. Бахтерев[1]

ИММ УрО РАН, г. Екатеринбург[1]

В статье рассматривается история развития и потенциал использования средств визуального программирования параллельных вычислений. Анализ систем, разработанных, начиная с 80-ых годов прошлого века, позволяет выявить стоящие перед разработчиками проблемы и предложить некоторые методы их решения.

1. Введение

Наметившаяся в 70-ых годах XX века массовая компьютеризация потребовала резкого увеличения объема программного продукта, роста производительности труда программистов и, конечно, роста числа программистов. С появившимися в эти годы методиками визуального программирования связывались значительные надежды на облегчение обучения, увеличение производительности труда программистов, повышение надежности программного продукта и уменьшение сложности процесса программирования. Еще большие надежды возлагались на эти методики в связи с параллельным программированием, которое заведомо считается более сложным, чем традиционное последовательное. Работы в области визуального программирования параллельных вычислений начались, практически, одновременно с созданием первых сред визуального программирования последовательных программ. В 90-ых годах отмечался особенно большой интерес к данному направлению. Исследования и опытные разработки продолжались и в последующие годы. Мы рассматривали состояние дел в данной области в работе [1], опубликованной в 2003 году. В этой работе мы еще раз проанализируем итоги развития визуального программирования параллельных вычислений за почти четыре десятилетия, что поможет выявить перспективы его развития и возможности создания эффективных сред проектирования и разработки.

2. Визуальное программирование

Термин “Визуальное программирование” подразумевает использование визуальных выражений (графов, схем, диаграмм, пиктограмм и т.п.) в процессе программирования, что дает возможность специфицировать программу в двумерном или трехмерном виде.

Традиционно визуальные языки программирования подразделяются на иконические языки, использующие иконы (пиктограммы) для представления объектов, операций и функций, и диаграмматические языки, основанные на использовании схем и диаграмм, так или иначе описывающие программные процессы. В отдельный класс выделялись языки, ориентированные на использование таблиц, формуляров и бланков.

Как уже указывалось, при появлении визуального программирования в 80-ых годах на него возлагались большие надежды в плане упрощения процесса программирования. Считалось, что визуальный способ описания программы более доступен мышлению начинающего программиста, так как картинка отражает реальный мир, тогда как текст лишь указывает на объекты реального мира. Кроме того, многомерность графики может увеличить информативность по сравнению с одномерным потоком текста за счет использования, например, формы, размера, цвета, текстуры, направления или расстояния. То есть визуальный метод описания был призван понизить уровень абстракции представления алгоритмов. Важным представлялось то, что возникала возможность перейти при программировании от мышления в рамках виртуальной (Фор-

* Работа выполнена при поддержке программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности", а также проекта 12-П-1-1034 УрО РАН.

тран, Алгол, etc.) машины к мышлению более обозримыми и компактными визуальными образами. Предполагалось активное использование визуальных метафор на базе какой-либо естественной образности, а также схем различного типа (конечных автоматов, графов потоков данных, диаграмм состояний, сетей Петри).

Первые реализации сред визуального программирования были основаны на так называемой *исполняемой графике*, когда имеется возможность описывать алгоритмы в визуальной форме и исполнять визуальные программы непосредственно на компьютере без перевода в текст на «нормальном» языке. Отсюда вытекает идея о том, что визуальные языки должны иметь графические представления для всех элементов программы, как для управляющих конструкций, так и для структур данных. Казалось, что за счет использования визуальных методик проектирования программ (например, диаграмм Нэсси-Шнейдермана) можно будет вообще отказаться от этапа кодирования. Так как визуальное представление дает возможность анимировать (оживить) изображения соответствующих программных конструкций, то с помощью анимационной графики предполагалось разрешить проблемы, зачастую возникавшие у начинающих программистов, когда они не могли соотнести статичный текст программы и тот процесс, который этим текстом генерировался.

Значительная часть реализаций сред визуального программирования в 80-ых и 90-ых годах базировалась на использование диаграмматических языков. Диаграмматические языки характеризуются строго определенным, формализованным словарем, состоящим из сравнительно небольшого количества элементов. Также, как правило, строго определен пространственный синтаксис - расположение элементов схем в пространстве и относительно друг друга. Многие системы проводили размещение элементов схем автоматически. Работа с системами программирования на базе диаграмматических языков, как правило, идет по общей схеме - пользователь, применяя систему меню, указывает на графические шаблоны - элементы диаграмм и размещает их в рабочем поле. После этого происходит заполнение соответствующих текстовых полей шаблонов. Системы программирования на базе диаграмматических языков по существу используют гибридные - текстово-графические языки.

Существует много примеров использования в качестве базовой метафоры блок-схем и диаграмм Несси-Шнейдермана. Эти диаграммы базируются на представлении потока управления программ. Были широко распространены визуальные языки, описывающие в том или ином виде потоки данных. Простая структура графов облегчает модульное проектирование и дает возможность применять схемы на всех уровнях описания проектируемых систем. Нет необходимости применять специальные языковые средства увязки отдельных модулей. Примеры визуальных языков на базе конечных автоматов, сетей Петри, HIPO-диаграмм встречались реже. В 90-ые годы, на следующем этапе развития визуальных языков, появились реализации на базе UML диаграмм.

В диаграмматических системах, основанных на представлении потока управления, обычно отсутствуют средства графического представления данных программы. Все, что касается данных должно вводиться в обычном текстовом виде, и все эти системы нуждаются в обширном объеме текстового ввода. Многие системы требуют на каком-то этапе детализировать фрагменты программы, для чего необходимо делать вставки на обычном текстовом языке программирования. Использование графов потоков данных влечет аналогичные проблемы - отсутствие высокоуровневых управляющих структур, приводящее к усложнению и запутыванию диаграмм, отсутствие средств поддержки нетривиальных структур данных и т.д. Кроме этого при применении графов потоков данных единственный способ передачи семантической информации - это использование имен узлов и дуг. У систем, основанных на других схемных метафорах обычно сильно ограничена область применения. Для разрешения проблем, возникающих при использовании диаграмм потоков данных и потоков управления, применялись смешанные диаграмматические метафоры, объединяющие, например, диаграммы потоков данных и сети Петри. Но имели место лишь единичные примеры таких решений.

Реализации систем на базе иконических языков часто базируются на расширенных моделях потоков данных. В этом случае в узлах графа помещается картинка, представляющая, как правило, заранее написанную функцию обработки данных. Существуют интересные примеры иконических языков с использованием естественной образности, достаточно точно отображающей смысл той или иной функции.

Графы потоков данных, также как и графы потоков управления, достаточно легко анимируются. Однако систем исполняемой графики с анимацией немного. Существуют примеры разработки в рамках идей “исполняемой графики” визуальных вариантов “обычных” языков, включая языки функционального программирования, для которых не просто подобрать адекватные методики визуализации основных понятий.

Диаграмматические языки оказались наиболее востребованным типом визуальных языков программирования. Именно диаграмматическими языками на базе потоков данных являются визуальные языки, встроенные в ряд математических пакетов. Сравнительно свежим примером классического диаграмматического языка на базе потоков данных является Microsoft Visual Programming Language (VPL). Можно вспомнить, что большую роль играют визуальные средства на базе диаграмматической образности в таких инструментальных средах программирования как Visual BASIC, Delphi и пр. Правда, в данном случае под визуальным программированием понимается не совсем то (или совсем не то), что первоначально подразумевалось в 80-ых годах. Теперь это не системы исполняемой графики, которые казались целью развития визуальных программных сред в 80-ых годах. В известных средах визуального программирования основную роль играет не визуальное задание структуры потока управления или потока данных, а задание структуры интерактивного поведения прикладной программы. Опыт создания иконических языков программирования сыграл свою роль при проектировании систем графического (иконического) интерфейса.

3. Визуальные языки параллельного программирования

Первые реализации визуальных языков параллельного программирования появились, практически, сразу же после появления последовательных аналогов в 80-ые годы и даже чуть ранее.

Попытки визуального задания параллелизма на первых порах ограничивались тем, что участки визуального представления программы (графа потока данных или схемы, отображающей графы других типов), которые, по мнению программиста, можно распараллелить, выделялись каким-либо способом (обычно двойной или жирной линией). В ранних средах такого типа не было явной визуальной поддержки сущностей параллельного программирования, связанных с посылками сообщений между процессами или их порождения/уничтожения. На следующем этапе кроме подобных подсказок компилятору в языки вводились описания достаточно сложных конструкций параллелизма. В визуальных языках параллельного программирования при задании последовательной части программы использовались традиционные диаграмматические методики описания потока управления. В ряде языков использовались наборы простых пиктограмм, представляющих различные программные конструкции того или иного языка программирования. (Например, VISO - визуальный вариант языка Occam [2].)

В ходе развития визуальных параллельных языков программирования выявилась также тенденция к созданию “концептуальных систем”, когда разработчиками выдвигалась определенная концепция описания параллелизма и соответственно параллельного программирования, для которых подбиралась своя методика визуализации. Была поставлена задача разработки систем визуального программирования, в которых весь цикл разработки (непосредственное программирование, отладка правильности и настройка производительности) будет вестись в рамках одной ментальной модели. Можно даже сказать, что при проектировании подобных сред визуального программирования важным является формирование самой ментальной модели функционирования параллельных программ. Другой вопрос - решены ли эти задачи и даже решаемы ли они в принципе.

Методики непосредственного создания программ и визуального представления в подобных средах, несмотря на использование различных подходов к заданию параллелизма, похожи. Программист создает общую схему параллельной (или распределенной) программы, а затем конкретизирует ее путем задания конкретных деталей на отдельных участках. Образность почти всех систем - диаграмматическая. Набор графических эле-

ментов традиционно невелик и, в целом, упрощен, даже, если и используются какие-либо пиктограммы. Анимация для описания динамики процессов не используется.

Среди ранних “концептуальных” решений обратим внимание на появившиеся в начале 90-ых годов языки CODE [3] и Phred [4].

На языке CODE программа представляла собой граф потоков данных решения данной задачи, состоящий из узлов-процессов (разноцветных окружностей с именем), в которых проводятся вычисления и дуг (стандартных стрелок), соединяющих порты этих узлов. На следующем этапе узлы аннотируются (описываются в текстовом виде) порты ввода и вывода, а также задаются достаточно сложные правила запуска узлов, содержащие условия, по которым будут работать данные узлы.

При проектировании языка Phred особое внимание уделялось проблеме *детерминированности* в параллельном программировании. Недетерминированные программы могут содержать трудно обнаруживаемые ошибки. Если система программирования может предупреждать программиста о потенциальной недетерминированности, программист имеет больше шансов избежать западни, связанной с “гонкой сообщений” параллельной программы. Язык Phred рассматривается в связи с проблемами восприятия параллелизма вычислений. Его называют *координационным языком*, в противоположность обычным языкам, описывающим только вычисления.

Программа на языке Phred состоит из *потока управления*, *потока данных* и *множества интерпретаций узлов* (то есть процедур, запускаемых при достижении узла). Визуальная компонента языка Phred обеспечивает изображение потоков управления и данных, создаваемых непосредственно при помощи графического редактора. Интерпретации узлов представляются спецификациями процедур. Дуги в графе потока управления представляют дизъюнктивные и конъюнктивные потоки управления. Таким образом, граф потока управления языка Phred обеспечивает представление последовательных потоков, переключение управления, параллелизм и синхронизацию.

Важным примером концепций описания параллелизма может служить язык Visper [5]. Существенным в проекте является введенное понятие графа взаимодействия процессов - *Process communication graph*. (Это понятие, включая язык PCG, используется также в ряде родственных разработок визуальных языков.) Граф взаимодействия процессов комбинирует графы потока управления и потока данных в единый визуальный формализм. Насколько можно понять из публикаций, в рамках многолетних исследований и разработок рассматривались различные идеи по визуализации процесса параллельного программирования. Поэтому какого-то канонического варианта язык не существует. По нашему мнению важной идеей в Visper'е является идея использования методик отображения параллельности, которые ранее использовались на этапах отладки и настройки эффективности параллельной программы. В основу этих методик положены (несколько модернизированные) понятия *карты параллельности* и *диаграммы пространства-времени.*

Карта параллельности отображает возможную параллельность задачи. Это одновременно структура данных для перезапуска (переигровки) потока управления и графический метод представления параллельных процессов. Карта показывает историю проработки процессов в виде потока событий на временной шкале.

Динамика выполнения параллельной программы в диаграммах пространства-времени представляется как поток событий в двумерном пространстве, где одна ось показывает время, а вторая - реальные отдельные процессоры. При модернизации этих понятий в рамках Visper'а временная ось переосмысливается как ось потока управления, а процессорная ось расширяется за счет введения понятия группы. Граф взаимодействия процессов определяется на трехмерном пространстве программирования. По оси ***X*** показываются процессоры и группы процессоров, а также потоки данных, которыми они обмениваются. Ось ***Y*** определяет поток управления программы, а на оси ***Z*** показано количество процессоров.

Существует еще целый ряд подходов к созданию визуальных параллельных языков, реализованных уже в первое десятилетие XXI века.

В языке Vorlon [6] реализована модель параллельного исполнения потока объектов (*parallel object-flow execution model*). Эта модель объединяет объектно-ориентированную модель и модель потока данных. Таким образом, существует возможность обеспечить как задание паралле-

лизма, так и всей сложности конкретной прикладной задачи. При этом такие аспекты параллелизма, как синхронизация, обеспечиваются за счет конструкций, поддерживающих поток данных, а задание типов и их взаимосвязи дает возможность описывать все многообразие прикладной области.

Параллельный граф потока объектов состоит из узлов и дуг. Узлы представляют некоторую форму вычисляемого элемента, а дуги описывают взаимозависимости потоков управления и возможные маршруты передачи параметров между узлами.

Также с идеями потока данных и объектно-ориентированного программирования связан интересный проект языка HiPPO (High Performance Parallel Object-oriented) [7]. Рассматривается модернизация графа потока данных, который в данном случае является потоком ссылок на объекты.

VisualGOP [8], еще одна реализация среды параллельного программирования, была осуществлена с использованием графово-ориентированной модели программирования (*graph-oriented programming model*). Основа среды - язык GOP, на котором путем создания логического графа описывается логическая структура параллельных или распределенных программ. Этот граф служит для представления коммуникативных связей и синхронизации между отдельными программами в распределенной среде выполнения. VisualGOP не зависит ни от конкретных языков программирования, на которых написаны эти программы, ни от платформ, на которых программы функционируют.

Важным и активно разрабатываемым направлением в визуальных языках параллельных вычислений являются диаграмматические языки, базирующиеся на парадигме посылки сообщений между процессами. Работа в средах программирования, разработанных на базе этих языков, во многом напоминала работу в других средах визуального программирования - пошаговая детализация программы, представленной в визуальном виде; использование гибридного подхода - и графика и текстовые описания.

Наибольший интерес в этой связи представляет по нашему мнению язык Grapnel [9], в котором достаточно полно (на момент его создания в середине 90-ых годов) отражены достижения, как в параллельном, так и в визуальном программировании. В частности в языке Grapnel реализована визуальная поддержка механизма динамического порождения и уничтожения процессов и возможность использования предопределенных шаблонов топологий процессов.

При решении многих прикладных задач процессы размещаются на базе определенных топологий, таких как линейный массив, кольцо, сетка, дерево, etc. Механизм использования стандартных топологий служит эффективным средством создания динамически расширяемых распределенных программ, то есть таких программ, размерность топологии которых становится известно только во время работы программ. Тем самым использование шаблонов предполагает масштабируемость параллельной программы. В системе Grapnel также проявилась тенденция в развитии визуальных языков параллельного программирования, связанная с тем, что в визуальном виде представляются лишь те части программы, которые имеют отношение к описанию взаимодействия между процессами. Остальные (сравнительно простые для программиста) фрагменты, относящиеся к последовательным частям программы, а также декларативные фрагменты задаются в текстовом виде.Термин “Визуальное программирование” подразумевает использование визуальных выражений (графов, схем, диаграмм, пиктограмм и т.п.) в процессе программирования, что дает возможность специфицировать программу в двумерном или трехмерном виде.

4. Проблемы диаграмматичности

Обзор средств визуального программирования параллельных вычислений показывает, что в большинстве своём словарь их языков основан на диаграммах или схемах различных типов. Чем характеризуются эти языки? Прежде всего, ограниченным набором возможных визуальных элементов языка с жестко определенными значениями и правилами размещения на экране. У языков, использующих потоки управления, в общем-то, нет какого-то смысла, дополнительного по отношению к традиционным текстовым языкам программирования. В диаграмматических системах, основанных на представлении потока управления, обычно отсутствуют средства

графического представления данных программы. То, что касается данных зачастую должно вводиться в обычном текстовом виде. Все эти системы нуждаются в обширном объеме текстового ввода. Использование графов потоков данных влечет аналогичные проблемы - отсутствие высокоуровневых управляющих структур, приводящее к усложнению и запутыванию диаграмм, отсутствие средств поддержки нетривиальных структур данных и т.д. Как уже указывалось, при применении графов потоков данных единственный способ передачи семантической информации - это использование имен узлов и дуг, а у систем, основанных на других схемных метафорах сильно ограничена область применения.

По нашему мнению рано делать какие-либо заключения о пригодности предлагаемых в “концептуальных” средах идей параллельного программирования. Что же касается методик непосредственного создания программ и визуального представления в подобных средах, то все они, несмотря на использование различных подходов к заданию параллелизма, стандартны. Программист создает общую схему параллельной (или распределенной) программы, а затем конкретизирует ее путем задания конкретных деталей на отдельных участках. Образность - диаграмматическая. Набор графических элементов традиционно невелик и, в целом, упрощен, даже, если и используются какие-либо пиктограммы (иконы). Анимация для описания динамики процессов не используется.

Отметим, что ограниченность “диаграмматического” словаря часто не позволяет решать проблемы, проявляющиеся при разработке средств визуального представления в связи с развитием современных языков программирования.

Динамические языки и компилируемые языки последнего поколения развивались, среди прочих направлений, и в направлении развития средств описания процедур доступа к элементам данных. Так, они предоставляют программистам возможность использовать в операторных выражениях обращения к динамическим и ассоциативным массивам, к спискам, к элементам разбиения строк по регулярным выражениям, в некоторых случаях (Ruby) даже к записям в таблицах баз данных.

Обратим внимание на важную для многих диаграмматических языков особенность - исключение из структуры программы механизмов адресации данных, вместо которых предлагается абстракцию *стрелки* (->, <-), связывающей выходы одних операторов (или иных программных конструкций) с входами других. *Стрелка*, однако, не является метафорой доступа к элементу структуры данных задачи, так как не предполагает формирования значения вне связи между двумя операторами, что не соответствует одному из основных свойств данных - существовать и тогда, когда действия над ними не производятся. То есть, в большинстве визуальных языков программист должен иметь дело, хоть и с элементарными, но неявными доступами к значениям (даже не к данным). Развитых средств описания ссылок на результаты вычисления в таких языках нет, и единственной такой ссылкой является узел в орграфе, представляющий оператор, который изображается в виде некоторого трёхмерного или чаще двумерного объекта, например, некоторого многоугольника с исходящей из него линией, символизирующей результат, к которой подключаются входящие в другие узлы-операторы дуги.

Несмотря на то, что при таком подходе существует возможность независимо определять операнды для операторов, которые в этих системах обычно являются n-арными, внесение локальных изменений в программу может потребовать глобального редактирования связей в её графическом представлении. Например, тогда, когда значение, получаемое в одной группе операторов нужно обработать совместно со значением из другой, пространственно отдалённой.

На языке с явным доступом к данным (особенно императивном) в такой ситуации можно было бы обойтись несколькими локальными исправлениями текстового представления программы: сохранить требуемые значения в структуре данных программы, добавив несколько выражений в разнесённые по коду группы операторов, а потом обратится к ним, тогда, когда понадобится их совместная обработка.

Отметим, что иная ситуация возникает у иконических языков, основанных на использовании потоков данных. (Примером подобного языка является визуальный язык HI-Visual, лежащий в основе системы автоматизации конторского труда и основанный на популярной метафоре рабочего стола [10].) В таких языках сначала нужно выбрать группу объектов (операндов), а затем операцию над ними. То есть язык ориентирован на явное указание данных, над которыми нужно выполнить операцию.

Конечно, на популярность и широту применения тех или иных средств программирования влияют многие факторы, но наличие в инструменте программирования развитых средств связи операторной части выражений с данными делает его более удобным в использовании. Эти средства могут играть важную роль в упрощении параллельного программирования.

Во второй половине 90-ых годов был предпринят ряд попыток выхода из “диаграмматического” тупика и ввода в процесс программирования динамики.

Нами был предложен проект визуального языка, использующего некоторый аналог популярных в статистической графике диаграмм Гантта (Gantt charts), традиционно применяемых в системах отладки производительности параллельных программ. Диаграммы Гантта используют полосы различной длины и типа для показа продолжительности того или иного состояния процесса. В нашем случае диаграммы изображались не в виде прямой линии, а в виде цветного двумерного и трехмерного кольца. Был реализован макет двумерного варианта визуального языка параллельного программирования. Пользователь, используя такие понятия, как процесс, обмен, работа, ожидание, канал и др., и задавая времена работы и ожидания, мог определять основные характеристики процессов в специальных окнах. Затем визуально описываются коммуникационные связи между процессами. Уже на ранних этапах описания процесса могло начинаться динамическое отображение смоделированных возможных вариантов его работы в виде модифицированных диаграмм Гантта. Дополнительные описания процесса, как на внутреннем уровне, так и на уровне взаимодействия должны изменять картину, отображающую его деятельность. (Предложенные идеи были близки к идеям, описанным в проекте системы T-Model [11].) Была также сделана попытка создать трехмерный вариант этого языка с использованием размещаемых в пространстве трехмерных аналогов диаграмм Гантта. Несмотря на недостаток серьезной базовой концепции программирования, идеи использования динамики и трехмерности (не привязанной к графово-диаграмматическим канонам) представляются достаточно плодотворными и могут использоваться в дальнейших разработках.

Существуют примеры разработок систем визуализации параллельных вычислений, в которых делается попытка визуализации, как параллелизма, так и динамики обработки данных. (См. среду на базе языка VIM Language [12].) Здесь предусматривается прямое отображение визуальных спецификаций непосредственно в программу конкретного процессора. Визуальные образы должны представлять высокоуровневые математические объекты, например, параметризированную матрицу и вектор при описании методов решения задач линейной алгебры.

Визуальный интерфейс, задающий начальные значения для прикладной вычислительной системы, можно рассматривать как специализированный визуальный язык параллельного программирования. Так, в рамках проекта ASSY [13], который направлен на создание метасистемы, поддерживающей разработку проблемно-ориентированных систем программирования, реализованы средства для решения задачи о взаимодействиях потоков разреженной плазмы. Плазма представляется набором довольно большого числа модельных частиц, которые характеризуются набором параметров, таких как заряд, координаты, скорость и т.п. Область решения разбивается равномерной прямоугольной сеткой на ячейки. С каждой частицей связывается ячейка, в которой частица в данный момент находится. В определенных методом точках сетки вычисляются сеточные функции, описывающие, в частности, действующие в ней силы. Программирование заключается в задании вычислительных функций внутри каждой ячейки, которые определяют и состояние соседних ячеек. Пользователь может отслеживать ход программирования, наблюдая собранное из “кирпичей”-ячеек пространство моделирования вместе с заданными вычислениями. Распараллеливание осуществляется компилятором автоматически.

В подобных средах осуществляется визуализация только лишь основных сущностей программирования, в данном конкретном случае являющимися высокоуровневыми понятиями определенного вычислительного метода. В каком-то смысле можно говорить о реализации визуального интерфейса к вычислительным математическим пакетам.

5. Восприятие визуальных языков

Следует указать также на проблемы, возникающие при визуальном представлении и соответственно адекватном восприятии данных программы и ее динамики.

При проектировании визуальных сред для параллельного программирования (также как и для отладки правильности и производительности программ) необходимо обратить серьезное внимание на проблемы, связанные с восприятием и интерпретацией визуальных образов. Опишем те из них, которые уже как-то прояснились после изучения и анализа существующих систем, а также на основе собственного опыта исследований и экспериментальных разработок.

Визуализация реальных параллельных программ приводит к громоздким и зачастую не интерпретируемым изображениям. Каким бы большим не был экран, объем визуальных данных, необходимых для представления серьезной параллельной или распределенной программы, будет превосходить его возможности. Практика показывает, что даже не очень большое усложнение структуры программы приводит к запутанным картинкам, напоминающим по сложности интерпретации пазлы.

Мы рассматривали различные методы решения этой проблемы. Например, в средах визуального программирования активно используются приемы семантического зуминга, позволяющие “сворачивать” и “разворачивать” визуальные блоки, отображающие отдельные части программы. Можно использовать идеи “бесконечного экрана” и/или “полета” над визуальным пространством или внутри его [14]. Нами также были реализованы пилотные версии программ с имитацией полета над плоским и сферическим информационным пространством и с возможностью погружения в него. Трехмерные методики размещения объектов параллельных программ использовались нами с середины 90-ых годов. Такие методики предполагают использование тех или иных метафор визуализации параллельных вычислений, в частности, метафоры комнаты. Нами предпринимался ряд попыток реализации метафоры комнаты в макетных системах визуального программирования. В одной из таких систем диаграммы и пиктограммы, размещенные на стенах, представляли управляющие конструкции и данные программной функции. Связи между конструкциями показывалась в пространстве, а не на плоскости, как в традиционных двумерных диаграмматических и иконических системах визуального программирования. Вся программа в этом случае представлялась зданием, а интересующая пользователя комната-функция вытягивалась из здания подобно блоку при блочном строительстве. Однако и в этом случае разработчик, используя в той или иной мере визуализированный инструментарий, должен был создавать программы в традиционном, по сути, текстовом режиме [15].

Методики визуализации на базе виртуальной и расширенной реальности также могут применяться как при создании, так и при отладке параллельных программ. Именно они должны обеспечить наиболее эффективное использование трехмерности и динамики. Однако, кроме непосредственного восприятия, серьезной проблемой остается и адекватная интерпретация чрезвычайно больших объемов сложноструктурированной информации, а также проблема порождения сложных визуальных текстов.

6. Заключение

Следует отметить, что постепенно интерес к созданию визуальных языков параллельного программирования ослабевает. Новые разработки (данные - на начало десятых годов XXI столетия) появляются все реже, хотя говорить о полном прекращении активности в этой области исследований нельзя.

Таким образом, оказывается, что развитие средств визуального программирования параллельных вычислений в своем развитии столкнулось с целым рядом проблем, касающихся как принципиальных вопросов визуального описания сущностей современных языков программирования, так и восприятием больших объемов визуальной информации. Представляется, что решения возникающих проблем следует искать на путях создания принципиально новых методик параллельного программирования, включающих в себя возможность использования метафор разработки и визуализации, способных поддерживать адекватные ментальные модели.

Литература

1. Авербух В.Л., Байдалин А.Ю., Разработка средств визуализации программного обеспечения параллельных вычислений. Визуальное программирование и визуальная отладка параллельных программ. // ВАНТ, сер. Математическое моделирование физических процессов, 2003, вып. 4., с. 68-80.
2. Al-Mulhem M.S. Concurrent programming in VISO // Concurrency: Pract. Exper. 2000; 12, pp. 281-288.
3. Newton P. A Graphical Retargetable Parallel Programming Environment and Its Efficient Implementation // Technical Report TR93-28, Dept. of Computer Sciences, Univ. of Texas at Austin, 1993.
4. Beguelin A.L., Nutt G.J. Visual parallel programming and determinacy: A language specification, an analysis technique, and a programming tool. Journal of Parallel and Distributed Computing, 22(2), August 1994, pp. 235-250.
5. Stankovic N., Zhang K. A distributed parallel programming framework // IEEE Transactions on Software Engineering. Vol. 28. No 5. May 2002, pp. 478-493.
6. Webber J., Lee P.A. Visual, Object-Oriented Development of Parallel Applications // Journal of Visual Languages & Computing Vol. 12, Issue 2, pp. 145-161.
7. Lee P.A. Phillips C., Watson P. Final Report: High Performance (Parallel) Object-Oriented Software Systems (HiPPO) // http://www.parallelism.cs.ncl.ac.uk/projects/hippo/FinalReport.pdf
8. Chan F., Cao J., Chan A.T. S., Zhang K. Visual programming support for graph-oriented parallel/distributed processing // Softw. Pract. Exper. 2005; 35, pp. 1409–1439.
9. Kacsuk P . Dozsa G . Fadgyas T . Designing parallel programs by the graphical language GRAPNEL / Microprocess . And Microprogramm. 1996, V. 41, 8-9, pp. 625 - 643.
10. Hirakawa M., Tanaka M., Ichicawa T. An Iconic Programming System, HI-VISUAL // IEEE Transactions on Software Engineering. Vol. 16 No 10 (October 1990), pp.1178-1184.
11. Самофалов В.В., Коновалов А.В. Т-модель: система наглядных представлений параллельных процессов в транспьютерных сетях // Алгоритмы и программные средства параллельных вычислений. Екатеринбург. ИММ УрО РАН. 1995. Стр. 157-169.
12. Mirenkov N. VIM Language Paradigm // Proceeding of CONPAR-94 - VAPP VI International Conference on Parallel and Vector Processing. Jahannes Kepler University of Linz. Austria. September 6--8 1994. (Lecture Notes in Computer Science) Springer-Verlag. Berlin. 1994. pp. 569-580.
13. Вшивков В.А., Краева М.А., Малышкин В.Э. Параллельная реализация метода частиц // Программирование. 1997, N 2. Стр. 39-51.
14. Флягина Т.А., Авербух В.Л. Новые подходы к проектированию видов отображения и методов интерфейса для систем визуализации программного обеспечения параллельных вычислений // XI Международный семинар Супервычисления и математическое моделирование. Тезисы. Саров. РФЯЦ ВНИИЭФ. 2009. Стр. 106-107.
15. Авербух В.Л., Байдалин А.Ю., Исмагилов Д.Р., Казанцев А.Ю., Тимошпольский С.П., Использование трехмерных метафор визуализации // 14-я Международная Конференция по Компьютерной Графике и Зрению ГрафиКон'2004 6-10 Сентября 2004 Москва, Россия. Труды Конференции. МГУ им. М.В. Ломоносова. Стр. 295-298.

Hoplang — язык обработки потоков данных мониторинга *

А.В. Адинец[1,2], С.А. Жуматий[1], Д.А. Никитенко[1]

[1]НИВЦ МГУ им. М.В.Ломоносова
[2]Объединённый институт ядерных исследований

При анализе работы больших вычислительных систем необходимо обрабатывать огромный объём данных мониторинга — загрузка процессоров, сетей, памяти, аппаратные счётчики, датчики температуры, и многое другое. Традиционные реляционные СУБД не в состоянии обеспечить обработку такого объёма информации. NoSQL-БД, системы MapReduce и языки запросов к ним масштабируются для обработки больших объёмов данных, но у них есть существенные недостатки. В частности, это непрозрачность взаимодействия с БД, отсутствие поддержки вложенных циклов и запросов, а также невозможность выполнения фильтрации по предикатам в самой БД. Hoplang разработан для анализа потоков данных мониторинга, и стремится избежать перечисленных выше недостатков. Его основные цели — абстракция языка от формы хранения данных, масштабируемость, поддержка распределённых запросов к БД и распределённой обработки данных, лёгкость расширения. Общая концепция языка близка к модели MapReduce, но обеспечивает большую гибкость и прозрачность в обработке потоков данных.

1. Введение

Современные суперкомпьютеры для большинства пользователей и даже администраторов представляют собой “чёрные ящики”. Насколько эффективно работают задачи на таких больших установках не знает практически никто. Для того, чтобы оценить производительность и эффективность только программы требуются специальные средства — профилировщики, трассировщики, средства анализа. Запуск с помощью этих средств часто помогает оптимизировать конкретную программу, но увы, их невозможно применять к каждой запущенной программе, т.к. они вносят очень существенные накладные расходы. А уж об эффективности использования установки в целом даже не идёт речи — таких средств практически нет.

С другой стороны, для общей оценки эффективности работы как отдельной задачи, так и вычислительной установки в целом, вполне достаточно собирать традиционные данные мониторинга, такие как загрузка процессора, интенсивность использования дисков, подкачки, сети, оперативной памяти и т.п. Анализ даже таких данных способен дать качественно новую картину использования суперкомпьютера и даже картину работы отдельной задачи.

Решение разбивается на две проблемы — сбор данных мониторинга и их анализ. Для больших установок объём данных мониторинга становится очень большим и эти проблемы могут помешать нормально работе самих вычислителей. Для сбора данных существует достаточно много средств, многие из них поставляются в открытых исходных кодах и являются расширяемыми. В качестве примеров можно привести Ganglia, Zabbix, Nagios, Clustrx, а также многие другие. В данной работе мы не будем на них останавливаться, заметим лишь, что в наших условиях эффективным выходом может быть распределение данных по физически разным базам данных.

И хотя средств мониторинга существует много, средств для решения второй проблемы — обработка и анализ полученных данных — существует не так уж много. Как уже было упомянуто, данные могут быть распределены на несколько разных физических баз, сам

*Работа выполнена в рамках гос. контракта 07.514.12.4001, совместного проекта Евросоюза и России с кодовым названием HOPSA

состав баз данных тоже может быть разным. Например, данные по задачам хранятся в MySQL, данные по пользователям — в LDAP, а данные мониторинга загрузки — в нереляционной БД, например Cassandra. Схемы хранения данных также могут быть различны и в разных условиях оптимальными могут оказаться разные схемы.

Надо учесть, что и объём данных мониторинга тысяч и десятков тысяч (а близкой перспективе и миллионов) вычислительных узлов — это десятки и сотни гигабайт информации в сутки, по самым оптимистическим прогнозам. С такими потоками данных способны справиться далеко не все реляционные СУБД. Многие NoSQL-базы это могут, но сильно пригрывают в гибкости хранения, да и не всегда все данные можно хранить в одной базе, даже распределённой. Например, задача записи в базу сразу на нескольких серверах, т. к. такой объём просто невозможно собрать и передать через один сервер, разрешима далеко не во всех базах.

Таким образом "традиционное" решение — одна БД и набор SQL- или NoSQL-запросов в данном случае плохо применим.

Эта статья построена следующим образом. В разделе 2 мы описываем нашу текущую инфраструктуру, её недостатки и предъявляемые к создаваемому языку требования. Далее, в разделе 3 описываются основные конструкции языка, а в разделе 4 — возможности его параллельной реализации. Наконец, в разделе 6 приведены итоги и планы дальнейшей работы.

2. Существующие решения и необходимость Hoplang

Для решения задачи обработки больших объёмов данных мониторинга суперкомпьютерных комплексов нами первоначально был использован подход с использованием одной из наиболее распространённых технологий массовой обработки данных — технологии Hadoop [1], свободной реализации концепции MapReduce.

Нами была выбрана высокопроизводительная база данных Cassandra [2], а для формулирования запросов обработки данных — язык PIG [3]. Такая комбинация позволила обрабатывать реальные данные с таких суперкомпьютеров как "Чебышёв" и "Ломоносов". Однако, были выявлены существенные недостатки такого подхода: недостаточная производительность и низкая переносимость.

Недостаточная производительность обусловлена вовсе не низкой скоростью работы какого-то компонента построенного комплекса, а высокими накладными расходами. Процесс компиляции PIG-запроса в реальную java-программу, загрузка и запуск полученной java-программы занимают зачастую больше времени, чем собственно вычисления.

Низкая переносимость связана с особенностями языка PIG. Данный язык требует явного указания источника данных, поддерживает очень ограниченный набор баз данных и ограничен в возможностях. Несмотря на то, что все компоненты построенного комплекса являются opensource и допускают модификацию кода, необходимые изменения потребовали бы кардинальных изменений исходных программных продуктов.

Для решения этой проблемы нами был предложен язык обработки данных Hoplang, ориентированный именно на обработку данных мониторинга. Под словами "язык обработки данных" мы понимаем не только описание синтаксиса, но и концепции обработки данных. Несмотря на то, что язык ориентирован на конкретную задачу, он покрывает достаточно широкий класс задач и может быть использован для многих видов массовых вычислений.

Основные концепции языка: данные представляются в виде именованных или промежуточных потоков, программа на Hoplang задаёт только схему (алгоритм) обработки данных. Это значит, что в программе на Hoplang нельзя задать адрес базы данных или имя таблицы, столбца. Также в программе нельзя указать какие части должны быть распределены по разным вычислительным узлам, какие выполнены совместно, как именно должен быть выполнен запрос к БД, и т.п.

Всё, что касается источников данных, возлагается на “драйвер” базы данных. Это компонент языка, который может быть подключен к интерпретатору, одновременно может быть подключено множество драйверов. Каждый драйвер позволяет осуществлять доступ к базе данных конкретного типа — Cassandra, MySQL, MongoDB [4], CSV-файл и т.п. Драйвер должен осуществлять перевод обращения из программы на Hoplang к источнику данных в запрос на языке базы данных, по возможности, осуществляя его оптимизацию. Кроме того, драйвер должен выполнять преобразования представления данных, хранящихся в БД, ко внутреннему представлению Hoplang.

Например, драйвер может переместить условия выборки из программы на Hoplang в запрос SQL, или “склеить” два вложенных обращения в один сложный запрос к БД. Отображение имён потоков данных на конкретные базы данных и их внутренние схемы задаётся администратором в конфигурационном файле и при работе программы компилятор Hoplang определяет какой драйвер должен обработать какое обращение к потоку данных.

Интерфейс для написания собственных драйверов БД открыт, поэтому добавить поддержку нового типа базы данных не составляет большой сложности. Новый драйвер достаточно расположить в специальном, каталоге и он будет автоматически использован компилятором. На данный момент драйвер может быть написан только на языке ruby [5].

Сам язык построен так, чтобы основные секции обработки данных были независимы или могли обрабатывать данные в режиме конвейера — принимая и отдавая потоки данных. Язык поддерживает агрегатные функции, выполнение которых оптимизируется автоматически. При возможности выполнить их средствами базы данных соответствующий драйвер попытается это сделать. Язык предусматривает и написание собственных агрегаторов, но в этом случае поддержка со стороны базы данных будет ограниченной.

3. Основные конструкции языка

Hoplang является языком со слабой динамической типизацией. На этапе выполнения имеется строгое различие лишь между простыми значениями, кортежами и потоками. Элементами одного потока могут быть либо кортежи, либо простые значения. Из простых типов в настоящее время поддерживается строковый и целочисленный типы, вещественный тип может быть добавлен в будущем, если в нём возникнет необходимость. Типы в программе явно не объявляются. Типы значений, читаемых из БД, определяются динамически на основании схемы БД. В качестве оптимизации в будущем планируется использовать вывод типов при компиляции запроса.

Структура программы Hoplang представляет собой последовательность операторов, как в обычных процедурных языках. Не допускается указание двух операторов в строке, конец оператора определяется концом строки или комментарием. Комментарии в Hoplang начинаются с символа `#` и оканчиваются концом строки. Многострочных комментариев в Hoplang нет.

Переменные Hoplang делятся на *потоковые* и *скалярные*. Потоковые переменные представляют собой именованный поток передачи данных. Запись в такую переменную по сути эквивалентна передаче данных в поток, а чтение — чтению очередного элемента данных из потока. Чтения и запись в потоковые переменные осуществляется только специальными операторами — хопстансами.

Скалярные переменные — или скалярные значения, или кортежи с именованными полями. В такой переменной может храниться, например, загрузка процессора вычислительного узла в виде пользовательского времени, системного времени, времени простоя, времени ввода-выводи и отметки времени. К полям переменной, которая является кортежем, можно обращаться через точку по именам, например `myvar.f1`.

Скалярные переменные могут быть объявлены в любом месте программы оператором `var`, за которым следует имя переменной. Как в большинстве современных языков, пере-

менная имеет область видимости — тот блок, в котором она объявлена.

Hoplang поддерживает основные управляющие конструкции: `if ... else ... end` (рис. 1) и `while ... end` (рис. 2). Конструкции `else` и `elif` опциональны, `elif` может повторяться любое число раз.

```
if cond
...
elif cond
...
else
...
end
```

Рис. 1. Вид оператора `if`

```
while cond
...
end
```

Рис. 2. Вид оператора `while`

Операции над переменными, допускаемые в Hoplang: числовые, строковые, сравнения и логические. Числовые: `+, -, *, /`. Поддерживаются круглые скобки для изменения приоритетов операций. Из строковых операций поддерживается конкатенация — оператор `&`. Операции сравнения: `==, !=, <, >, <=, >=`, возвращают результат логического типа. Логические операции `and, or, xor` поддерживаются для значений логического типа. Для переменных, которые являются кортежами, операции могут выполняться только над их полями, например, как на рис. 3. Присваивания могут осуществляться как для поля переменной, так и для переменной целиком. Во втором случае справа от оператора присваивания должны стоять переменная, либо набор выражений, предварённых именами — рис. 4.

```
a.val = x.counter + y.cpus
```

Рис. 3. Доступ к полям переменных

```
v = name1 => expr1, ... , nameN => exprN
```

Рис. 4. Присваивание полям переменной

Хопстансы — конструкции Hoplang, позволяющие работать с потоками данных. Хопстансы не могут выполняться внутри блоков `if` или `while`. Допускаются вложенные хопстансы. Основной хопстанс Hoplang — хопстанс `each`. Хопстанс читает заданный поток данных, выбирая нужные данные по условию, и выполняет обработку. При окончании потока выполняется специальная “финальная” секция хопстанса.

Общий вид хопстанса `each` представлен на Рис 5. Здесь `stream1` — имя входного потока (потоковая переменная). Это имя должно быть описано в файле конфигурации и может указывать как на “живой поток” данных от системы мониторинга, так и на базу данных. `stream2` — имя выходного потока. Его может использовать любой следующий по тексту программы хопстанс. `v` — переменная хопстанса, в ней будет значение очередной порции данных, полученных из входного потока. Эта переменная объявляется автоматически и

доступна только в теле хопстанса и секции `final`. `cond` — условие выборки из потока. Конструкции `where` и `final` могут отсутствовать.

```
stream2 = each v in stream1 where cond
...
final
...
end
```

Рис. 5. Вид оператора `each`

Важный оператор, который работает внутри хопстанса — оператор `yield`. Этот оператор записывает свои аргументы в выходной поток хопстанса. Общий вид оператора `yield` представлен на рис. 6. Второй вариант возвращает в поток все текущие поля переменной `myvar` под их именами. В случае, если выражение expr является полем переменной, то часть `name =>` может быть опущена и имя поля в выходном потоке будет совпадать с именем поля переменной. Если в качестве аргумента выступает агрегатор, то имя также можно опустить и по умолчанию оно будет совпадать с именем агрегатора.

```
yield name1 => expr1, ..., nameN => exprN
yield myvar.cpu_load
yield myvar
```

Рис. 6. Виды оператора `yield`

Простой пример подсчёта суммы и количества чётных элементов в поле 'v' в потоке на рис. 7. Тело хопстанса пустое, так как все операции тут выполняются агрегаторами `sum` и `count` в секции `final`. Встроенные агрегаторы, допустимые в секции final: `min, max, count, sum, avg, top, bottom`.

```
out = each v in mystream where v.f % 2 == 0
final
  yield sum(v.f), count(v)
end
```

Рис. 7. Пример подсчёта суммы

Ещё один пример, когда поток данных преобразуется в новый поток представлен на Рис. 8. Здесь мы суммируем времена, затраченные на систему, ввод-вывод и прочее в поле потока под именем `other`, а время пользователя и время простоя оставляем как есть. В выходной поток попадают только те элементы потока, где время пользователя больше 10%.

```
out = each v in cpudata where v.user > 10
  yield other => v.sys + v.wio + v.nice,
    idle => v.idle, user => v.user
end
```

Рис. 8. Пример обработки потока

Другой хопстанс — `seq`. Он предназначен для обработки данных из промежуточного потока (обычно полученного от другого хопстанса). В этом хопстансе нельзя обращаться к внешним источникам данных. Вид хопстанса представлен на на рис. 9. Секция `final` опциональна.

```
out = seq v in stream1
...
final
...
end
```

Рис. 9. Вид хопстанса seq

Следующий хопстанс — `group`. Он аналогичен конструкции group by в SQL, то есть производит группировку данных по заданному выражению. В отличие от each, секция `final` вызывается для каждой группы. Общий вид хопстанса представлен на рис. 10.

```
stream1 = group v in stream2 by cond
...
final
...
end
```

Рис. 10. Вид хопстанса `group`

Следующий хопстанс — `sort`. Этот хопстанс производит сортировку потока. В отличии от рассмотренных ранее, он не имеет ни тела, ни секции `final`. Общий вид хопстанса представлен на рис. 11.

```
stream1 = sort v in stream2 by cond
```

Рис. 11. Вид хопстанса `sort`

Ещё один хопстанс — `scan`. Он производит одну операцию над всеми элементами потока. Общий вид представлен на рис. 12. Этот хопстанс, как и `sort`, не имеет тела и секции `final`.

```
stream2 = scan stream1 postfix by OPERATION on field
stream2 = scan stream1 prefix by OPERATION on field
```

Рис. 12. Вид оператора `scan`

Хопстансы `top` и `bottom` — эквивалентны сортировке потока и выборке заданного числа наибольших или наименьших элементов по заданному критерию. Общий вид этих хопстансов представлен на рис. 13. Хопстансы возвращают первые или последние N элементов потока, отсортированных по выражению `expr`.

```
stream2 = top N stream1 by expr
stream2 = bottom N stream1 by expr
```

Рис. 13. Вид операторов `top` и `bottom`

Допускается чтение ровно одного элемента потока. Это должен быть поток, описанный в конфигурации, а не порождённый в программе. Иными словами, допускается чтение одной строки из базы данных. Общий вид такого оператора представлен на рис. 14. Если под условие попадает несколько элементов, будет возвращён только один, какой именно — не специфицируется.

```
myvar = get dbvarname field1, ... fieldN where cond
```

Рис. 14. Вид оператора `get`

4. Возможности параллельной обработки

Выше были описаны синтаксические конструкции языка. Теперь рассмотрим как осуществляется параллельная обработка данных в рамках Hoplang. Первый параллельный принцип, применяющийся в Hoplang — *конвейерность*. Любые два хопстанса могут передавать данные в режиме конвейера через поточную переменную. В простейшей реализации разные хопстансы работают в разных нитях на одном компьютере, в общем случае — в разных процессах на разных компьютерах.

Следующий принцип реализации параллелилизма — *распараллеливание доступа к БД*. Один и тот же источник может храниться в нескольких базах данных. Например, данные мониторинга от десяти тысяч узлов эффективнее распределить по нескольким физическим БД. Или данные о пользователях двух кластеров могут храниться в двух БД.В случае, если данные распределены по разным БД, хопстанс автоматически разделяется на несколько соответствующих хопстансов. По окончании работы всех хопстансов, их результаты при необходимости объединяются.

Распределение вычислений может быть произведено компилятором Hoplang и в случае, если предполагаемый объём вычислений от одной БД слишком велик. Это эвристическая оценка и её работа зависит от реализации компилятора Hoplang. В этом случае, запрос к БД разбивается на несколько, и на каждый полученный запрос создаётся свой экземпляр хопстанса.

Для обеспечения корректности работы и снижения накладных расходов при параллельной работе, в Hoplang введены ограничения на доступ к переменным. Любая переменная доступна на чтение только в своей области видимости, а на запись — только на уровне своего хопстанса. Это значит, что во вложенном хопстансе доступ на запись переменных из объемлющего хопстанса невозможен.

Если в теле хопстанса объявлена переменная, и она используется в секции `final`, то этот хопстанс не может быть распределён на несколько, так как нет возможности обеспечить корректность его работы. Данная проблема будет решена в следующей версии языка, в которой будет добавлена возможность задать пользовательский код для агрегации данных из распараллеленных хопстансов.

5. Текущее состояние

В настоящее время реализована первая версия интерпретатора Hoplang, для реализации использовался язык Ruby. Поддерживаются полноценный синтаксис выражений, драйверы для чтения данных из формата CSV и из БД Cassandra, а также хопстанс `each` и его последовательная версия `seq`. Для драйвера БД Cassandra поддерживается проталкивание предикатов фильтрации в `where` в запрос к БД. Для этого требуется, чтобы предикат был конъюнкцией простых условий, т.е. условий вида <поле> <операция сравнения> <константа>. При этом операция сравнения не может быть `!=`, должно быть хотя бы одно сравнение на равенство, и для всех фильтруемых полей в БД Cassandra должны быть созданы индексы. Нам кажется, что это будет наиболее часто используемый случай проталкивания индексов в запрос к БД.

Для демонстрации возможностей текущей реализации мы провели сравнение между Hoplang и нынешней версии системы запросов к данным мониторинга кластера, использующей связку Pig + Hadoop. Запрос: по пользователю требуется получить среднее число ЦПУ по его задачам, и число самих задач. Код запроса на языке Hoplang представлен на

рис. 15, а код на языке Pig приведён на рис. 16. Сразу бросается в глаза разница в читаемости кода. На Pig более запроса занимает код для чтения данных из БД Cassandra. Реализация БД Cassandra в Pig такова, что данные читаются в нетипизированном виде, и для преобразования его в типизированный вид необходимо выполнить дополнительные манипуляции. Кроме того, при чтении из другой БД в Pig придётся изменить код запроса, в то время как в Hoplang достаточно поменять файл конфигурации. Более того, после реализации в Hoplang полноценного синтаксиса конвейеров в стиле оболочек типа sh и полноценных агрегаторов запрос удастся сократить до одной строчки.

```
ts = each t in tasks where t.user == 'serdyuk'
 yield key => t.key, ncpus => t.ncpus
end
avgs = seq t in out3
 var ncpus, n
 ncpus = ncpus + t.ncpus
 n = n + 1
final
 yield ncpus, n, ncpus / n
end
```

Рис. 15. Код запроса на языке Hoplang для получения среднего числа процессоров за всё время работы кластера по задачам пользователя 'serdyuk'

```
REGISTER hopsa-udfs.jar;

%default cf 'tasks_gra'
%default user 'serdyuk'
%default resdir '/does/not/exist'

tsr = LOAD 'cassandra://hopsa/$cf' USING CassandraStorage() AS (
  k:bytearray, va:{t:(n:bytearray, v:bytearray)}); /* $ */
ts1 = FOREACH tsr {
  u = FILTER va BY n=='user';
  nc = FILTER va BY n=='ncpus';
  GENERATE k AS key, FLATTEN(u.v) AS user, FLATTEN(nc.v) AS ncpus;
};
ts = FOREACH ts1 GENERATE key, user, (long)ncpus AS ncpus;
tsf = FILTER ts BY user == '$user'; /* $ */
gu = GROUP tsf BY user;
timeu = FOREACH gu GENERATE group AS user, AVG(tsf.ncpus) AS
  avgncpus, COUNT(tsf) AS ntasks;
STORE timeu INTO '$resdir' USING PigStorage('\t'); /* $ */
```

Рис. 16. Код запроса на языке Pig для получения среднего числа процессоров за всё время работы кластера по задачам пользователя 'serdyuk'

С использованием Hoplang запрос исполнялся 2.03 секунды, а с использованием Pig — 12.35 сек. Экономия достигается прежде всего за счёт проталкивания запроса в БД. Hoplang-версия использует индексы в БД Cassandra, и поэтому обрабатывает намного меньше данных, в то время как Pig-версия вынуждена сканировать всю БД, после чего извлекать из неё данные по конкретному пользователю. Разумеется, по мере роста объёма собранных

данных, Hoplang-версия будет масштабироваться значительно лучше.

6. Дальнейшее развитие

Hoplang уже реализует множество возможностей для обработки больших объёмов данных. Однако, в ближайших планах у нас стоит его усовершенствование и расширение. Среди основных целей стоит отметить:

- реализация функций, как скалярных, обрабатывающих только одно скалярное значение, так и функций-хопстансов, которые могут выступать самостоятельной ступенью конвейера
- добавление использования пользовательского кода для объединения результатов параллельных хопстансов (редукторы),
- объединение последовательных хопстансов в цепочку, с удалением имён промежуточных “связующих” поточных переменных

Литература

1. Tom White Hadoop: The Definitive Guide O’Reilly, 2009. 501 p.
2. Apache Cassandra DB official cite URL: http://cassandra.apache.org/ (дата обращения: 01.11.2011)
3. Apache PIG Latin official cite URL: http://pig.apache.org/ (дата обращения: 01.11.2011)
4. MondoDB official cite URL: http://www.mongodb.org/ (дата обращения: 01.11.2011)
5. Д. Флэнаган, Ю. Мацумото Язык программирования Ruby Издательство Питер, 2011 496 стр.

Алгоритмы решения обратных геофизических задач на многопроцессорных вычислительных системах *

Е.Н. Акимова[1], Д.В. Белоусов[1], В.Е. Мисилов[2]

Институт математики и механики УрО РАН[1], Уральский федеральный университет[2]

Для решения обратных задач гравиметрии предложены эффективные устойчивые параллельные алгоритмы на основе итерационных методов градиентного типа. Для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки построены параллельные методы матричной прогонки, квадратного корня и метод сопряженных градиентов с предобуславливателем. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel с использованием новых вычислительных технологий. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

1. Введение

Важнейшими задачами исследования структуры земной коры являются обратные задачи гравиметрии: задача гравиметрии о нахождении переменной плотности в слое [1] и структурная задача гравиметрии о восстановлении поверхности раздела между средами [2]. Задачи гравиметрии описываются линейными и нелинейными интегральными уравнениями Фредгольма первого рода, т.е. являются существенно некорректными задачами. При разработке методов решения задач используются идеи итеративной регуляризации [3]. После дискретизации с использованием итерационных процессов задачи сводятся к системам линейных алгебраических уравнений (СЛАУ) с плохо обусловленными заполненными матрицами большой размерности (несколько сотен тысяч). Необходимость повышения точности результатов решения задач, в частности использование более мелких сеток, существенно увеличивает время вычислений.

Важнейшими задачами исследования неоднородности земной коры являются задачи электроразведки. Одним из известных методов электроразведки является метод вертикального электрического зондирования (ВЭЗ). После использования конечно-разностной аппроксимации задача ВЭЗ сводится к решению СЛАУ с блочно-трехдиагональной матрицей [4]. Другой важной задачей электроразведки является задача бокового каротажного зондирования (БКЗ). В результате интерпретации данных каротажа получают значение удельного электрического сопротивления пласта, близкое к истинному. В работе [5] показано, что после использования конечно-разностной аппроксимации задача БКЗ сводится к решению СЛАУ с блочно-трехдиагональной матрицей большой размерности.

Одним из путей уменьшения времени расчетов и повышения эффективности решения геофизических задач является распараллеливание алгоритмов и использование многопроцессорных вычислительных систем (МВС). В научно-исследовательских институтах и университетах России распространены массивно-параллельные суперкомпьютеры кластерного типа с распределенной памятью. В Институте математики и механики УрО РАН (г. Екатеринбург) установлены МВС-1000/17ЕК, МВС-ИММ и суперкомпьютер «Уран» (http://parallel.uran.ru/node/6), которые успешно используются при решении прикладных задач.

В настоящее время в мире для решения прикладных задач наметилась тенденция к использованию в качестве вычислительных систем многоядерных гибридных вычислителей с графическими процессорами (видеокартами). По сравнению с суперкомпьютерами МВС гибридные вычислительные системы на основе графических процессоров представляют собой более деше-

* Работа выполнена при поддержке УрО РАН в рамках Программы фундаментальных исследований Президиума РАН № 18.

вую многопроцессорную технику с низким энергопотреблением. В Институте математики и механики УрО РАН установлен гибридный вычислительный кластер на основе видеоускорителей NVIDIA Tesla (URL: http://parallel.uran.ru/news).

В данной работе для решения линейной обратной задачи гравиметрии о нахождении плотности в слое итерационными методами градиентного типа и нелинейной обратной задачи гравиметрии о восстановлении поверхности раздела между средами с помощью итеративно регуляризованного метода Ньютона, а также для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки построены эффективные параллельные прямые и итерационные алгоритмы. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel. Проведено исследование эффективности и оптимизация параллельных алгоритмов для решения геофизических задач на гибридных вычислительных системах. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

2. Методы решения обратных задач гравиметрии

2.1 Задача о нахождении плотности в слое

Рассматривается обратная задача гравиметрии о нахождении переменной плотности $\sigma = \sigma(x, y)$ в слое $\Pi 1 = \{(x, y, z) \in R^3 : (x, y) \in D,\ H_1(x, y) \le z \le H_2(x, y)\}$ по гравитационным данным, измеренным на площади $D = \{(x, y) \in R^2 : a \le x \le b,\ c \le y \le d\}$ земной поверхности ($H_1, H_2 -$ константы для горизонтального слоя). Используется априорная информация об отсутствии аномалий плотности вне слоя с границами $H_1 = H_1(x, y)$ и $H_2 = H_2(x, y)$ такими, что $H_1 < H_2\ \forall (x, y)$, и выполняется условие $H_i(x, y) \underset{\substack{x\to\pm\infty \\ y\to\pm\infty}}{\to} h_i = const$. При этом предполагается, что распределение плотности $\sigma(x, y)$ внутри слоя не зависит от z (ось z направлена вниз). Задача нахождения неизвестной плотности сводится к решению линейного двумерного интегрального уравнения Фредгольма первого рода [1]

$$A\sigma \equiv f \int_a^b \int_c^d \left\{ \frac{1}{\left[(x-x')^2 + (y-y')^2 + H_1^2(x', y') \right]^{1/2}} - \frac{1}{\left[(x-x')^2 + (y-y')^2 + H_2^2(x', y') \right]^{1/2}} \right\} \sigma(x', y')\, dx'dy' = \Delta g(x, y), \tag{1}$$

где $f -$ гравитационная постоянная, $\Delta g(x, y) -$ гравитационный эффект, порождаемый источниками в горизонтальном или криволинейном слое.

Предварительная обработка гравитационных данных, связанная с выделением аномального поля, выполняется по методике, предложенной П.С. Мартышко и И.Л. Пруткиным [6].

После дискретизации уравнения на сетке, где задана $\Delta g(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам задача (1) сводится к решению системы линейных алгебраических уравнений (СЛАУ) либо с симметричной положительно определенной матрицей (горизонтальный слой), либо с несимметричной матрицей (криволинейный слой). Так как уравнение (1) относится к классу некорректно поставленных задач, то СЛАУ, возникающее в результате дискретизации уравнения, является плохо обусловленной и преобразуется к виду (схема Лаврентьева)

$$(A + \alpha E)z = b, \tag{2}$$

где $\alpha -$ параметр регуляризации.

В случае криволинейного слоя исходная матрица СЛАУ несимметрична, поэтому система предварительно преобразуется к виду (схема Тихонова)

$$(A^T A + \alpha' E)z = A^T b. \tag{3}$$

где A^T – транспонированная матрица, α' – параметр регуляризации.

Для решения систем уравнений (2) и (3) используются итерационные методы градиентного типа: метод простой итерации (МПИ)

$$z^{k+1} = z^k - \frac{1}{\lambda_{\max}}[(A + \alpha E)z^k - b], \tag{4}$$

где $\lambda_{\max}$ – максимальное собственное значение матрицы $A + \alpha E$ (симметричный случай);
метод минимальных невязок (ММН)

$$z^{k+1} = z^k - \frac{(A(Az^k - b), Az^k - b)}{\left\|A(Az^k - b)\right\|^2}(Az^k - b); \tag{5}$$

метод минимальной ошибки (ММО)

$$z^{k+1} = z^k - \frac{\left\|Az^k - b\right\|^2}{\left\|A^T(Az^k - b)\right\|^2} A^T(Az^k - b); \tag{6}$$

и метод наискорейшего спуска (МНС)

$$z^{k+1} = z^k - \frac{\left\|A^T Az^k - A^T b\right\|^2}{\left\|A(A^T Az^k - A^T b)\right\|^2} A^T(Az^k - b). \tag{7}$$

Для методов (5), (6) и (7) в регуляризованном варианте матрица A заменяется на $A + \alpha E$. Условием останова итерационных процессов является «останов по невязке»: $\left\|Az^k - b\right\| / \left\|b\right\| < \varepsilon$.

Для решения линейной обратной задачи гравиметрии о восстановлении плотности в слое устойчивые итерационные методы градиентного типа численно реализованы на многопроцессорном комплексе МВС-ИММ с помощью библиотеки MPI [7] и графических процессорах NVIDIA с помощью технологии CUDA [8].

Распараллеливание итерационных методов градиентного типа основано на разбиении матрицы A горизонтальными полосами на m блоков, а вектора решения z и вектора правой части b СЛАУ на m частей так, что $n = m \times L$, где n – размерность системы уравнений, m – число процессоров, L – число строк матрицы в блоке (рис.1). На текущей итерации каждый из m процессоров вычисляет свою часть вектора решения. В случае умножения матрицы A на вектор z каждый из m процессоров умножает свою часть строк матрицы A на вектор z. В случае матричного умножения $A^T A$ каждый из m процессоров умножает свою часть строк транспонированной матрицы A^T на всю матрицу A. Host-процессор (ведущий) отвечает за пересылки данных и также вычисляет свою часть вектора решения.

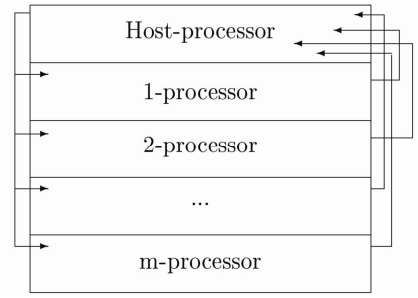


Рис. 1. Схема распределения данных по процессорам

Распараллеливание и численная реализация итеративно регуляризованного МПИ на видеоускорителях NVIDIA при решении линейной задачи гравиметрии описаны в работе [9]. В данной работе эти принципы распространяются на распараллеливание и реализацию на видеоускорителях NVIDIA итерационных методов ММН, ММО и МНС.

Для оптимизации работы с памятью при вычислениях используются два приема.

1. Для сеток не очень большой размерности (размер СЛАУ 12100×12100), когда данные входят в память видеокарты, матрица A порядка N и вектор Z размерности N расширяются до размерности M и дополняются нулями таким образом, чтобы M было кратно числу блоков. Размер блока BLOCK_SIZE (threads) выбирается кратным 16, поскольку в одном блоке группируются до 512 потоков. Тогда количество блоков вычисляется по формуле: blocks=M/BLOCK_SIZE. Вычисления производятся без выгрузки данных в память Host-процессора. Данные находятся только в памяти видеокарты.

2. Для сеток довольно большой размерности (размер СЛАУ 40000×40000), когда данные не входят в память видеокарты, наилучшим по быстродействию оказывается метод вычисления элементов матрицы A «на лету», т.е. вычисление значения элемента матрицы происходит в момент обращения к этому элементу без сохранения его в память видеокарты. Это позволяет существенно снизить количество обращений к памяти видеокарты и заметно ускорить процесс вычислений по сравнению с хранением матрицы A в памяти Host-процессора и порционной загрузкой в видеоускоритель для вычислений.

2.2 Задача о нахождении поверхности раздела между средами

Рассматривается трехмерная структурная обратная задача гравиметрии о восстановлении поверхности раздела между средами по известному скачку плотности и гравитационному полю, измеренному на некоторой площади земной поверхности.

Предполагается, что нижнее полупространство состоит из слоев постоянной плотности, разделенных искомыми поверхностями S_i. В предположении, что гравитационная аномалия создана отклонением искомой поверхности S от горизонтальной плоскости $z = H$ (ось z направлена вниз), в декартовой системе координат функция $z = z(x, y)$, описывающая искомую поверхность раздела, удовлетворяет нелинейному двумерному интегральному уравнению Фредгольма первого рода

$$A[z] \equiv f\Delta\sigma \int_a^b \int_c^d \left\{ \frac{1}{\left[(x-x')^2 + (y-y')^2 + z^2(x',y') \right]^{1/2}} - \frac{1}{\left[(x-x')^2 + (y-y')^2 + H^2 \right]^{1/2}} \right\} dx'dy' = G(x,y), \qquad (8)$$

где f – гравитационная постоянная, $\Delta\sigma$ – скачок плотности на границе раздела сред, $G(x, y)$ – аномальное гравитационное поле, $z = H$ – асимптотическая плоскость для данной границы раздела. Задача гравиметрии (4) является существенно некорректной задачей.

После дискретизации уравнения (8) на сетке $n = M \times N$, где задана $G(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам имеем систему нелинейных уравнений

$$A_n[z] = F_n. \qquad (9)$$

Для решения системы уравнений (9) используется итеративно регуляризованный метод Ньютона [10]

$$z^{k+1} = z^k - \left[A_n'\left(z^k\right) + \alpha_k I \right]^{-1} \left[A_n\left(z^k\right) + \alpha_k z^k - F_n \right]. \qquad (10)$$

Здесь $A_n(z^k)$ и F_n – конечномерные аппроксимации интегрального оператора и правой части в уравнении (3), $A_n'(z^k)$ – производная оператора A в точке z^k, I – единичный оператор, α_k – последовательность положительных параметров регуляризации.

Нахождение очередного приближения метода Ньютона z^{k+1} по найденному z^k сводится к решению СЛАУ

$$A_n{}^k z^{k+1} = F_n{}^k, \tag{11}$$

где $A_n{}^k = A_n'\left(z^k\right) + \alpha_k I$ – плохо обусловленная несимметричная заполненная $n \times n$ матрица, $F_n{}^k = A_n{}^k z^k - \left(A_n\left(z^k\right) + \alpha_k z^k - F_n\right)$ – вектор размерности n.

Предварительно система уравнений (11) приводится к виду

$$B^k z^{k+1} \equiv \left[(A_n^k)^T A_n^k + \alpha'_k I\right] z^{k+1} = (A_n^k)^T F_n^k \equiv b, \tag{12}$$

где $(A_n^k)^T$ – транспонированная матрица, α'_k – параметры регуляризации.

На каждом шаге метода Ньютона для решения СЛАУ (12) с симметричной положительно определенной матрицей используются итерационные методы градиентного типа (4) – (7) и метод сопряженных градиентов (МСГ) в регуляризованном варианте

$$z^{k+1} = z^k - \gamma_k (B^k z^k - b) + \beta_k (z^k - z^{k-1}), \tag{13}$$

где γ_k и β_k вычисляются по известным формулам [11]. Условием останова итерационного процесса МСГ является следующее: $\left\|B^k z^k - b\right\| / \left\|b\right\| < \varepsilon$.

Численная реализация и распараллеливание метода Ньютона с использованием градиентных методов для решения нелинейной обратной задачи гравиметрии о нахождении поверхности раздела между средами выполнены на МВС-ИММ. В ближайшее время предполагается реализация метода Ньютона на графических процессорах NVIDIA.

3. Параллельные алгоритмы решения СЛАУ с блочно-трехдиагональными матрицами

Как упоминалось во введении, задачи электроразведки (ВЭЗ и БКЗ) после конечно-разностной аппроксимации [4,5] сводятся к решению СЛАУ с блочно-трехдиагональными матрицами, представленными на рис.2.

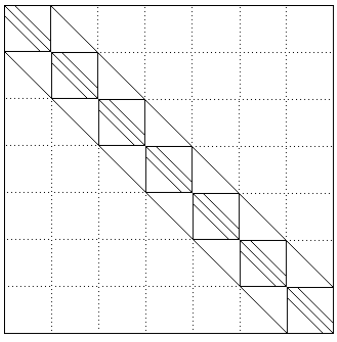


Рис. 2. Вид матрицы СЛАУ

Для решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки используются параллельные алгоритмы матричной прогонки (ПАМП), квадратного корня (ПАКК) и метод сопряженных градиентов с предобуславливателем (ПМСГ) в случае решения СЛАУ с симметричной положительно-определенной матрицей.

Параллельные алгоритмы реализованы на графических процессорах NVIDIA с помощью технологии CUDA и многоядерном процессоре Intel с помощью технологии OpenMP [7].

3.1 Параллельный алгоритм матричной прогонки

Рассмотрим систему уравнений с блочно-трехдиагональными матрицами общего вида

$$\begin{cases} C_0\overline{Y}_0 - B_0\overline{Y}_1 = \overline{F}_0, & i = 0 \\ -A_i\overline{Y}_{i-1} + C_i\overline{Y}_i - B_i\overline{Y}_{i+1} = \overline{F}_i, & i = 1,...,N-1 \\ -A_N\overline{Y}_{N-1} + C_N\overline{Y}_N = \overline{F}_N, & i = N, \end{cases} \tag{14}$$

где $\overline{Y}_i$ – искомые векторы размерности *n*, $\overline{F}_i$ – заданные векторы размерности *n*, A_i, B_i, C_i – квадратные матрицы порядка *n*.

В работе [12] для решения СЛАУ (14) предложен параллельный алгоритм матричной прогонки. Основная идея параллельного алгоритма заключается в следующем. Исходную область *P* (прямоугольник) разобьем на *L* подобластей вертикальными линиями так, что $N = L \times M$.
В качестве параметрических неизвестных выберем векторы $\overline{Y}_K$, $K = 0, M, ..., N$, связывающие неизвестные на сетке по вертикали. Относительно $\overline{Y}_K$ строится редуцированная система уравнений меньшей размерности по сравнению с исходной, которая решается классическим методом матричной прогонки [13]. После нахождения векторов-параметров $\overline{Y}_K$ остальные искомые неизвестные выражаются через параметрические неизвестные и находятся в каждой подобласти *L* независимо.

3.2 Метод сопряженных градиентов с предобуславливателем

Для решения СЛАУ (14) можно использовать эффективный метод сопряженных градиентов с предобуславливателем. Введение предобуславливания применяется с целью ускорения сходимости итерационного процесса и состоит в том, что исходная система уравнений $Ax = b$ заменяется на систему

$$C^{-1}Ax = C^{-1}b, \tag{15}$$

для которой итерационный метод сходится существенно быстрее.

Условием выбора предобуславливателя C является условие

$$cond(\tilde{A}) << cond(A), \quad cond(\tilde{A}) = \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}, \quad cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{16}$$

где $cond(A)$ и $cond(\tilde{A})$ – числа обусловленности матриц A и $\tilde{A}$; $\lambda_{\max}$, $\tilde{\lambda}_{\max}$ и $\lambda_{\min}, \tilde{\lambda}_{\min}$ – наибольшее и наименьшее собственные значения матриц A и $\tilde{A}$, соответственно.

Для системы уравнений (15) метод сопряженных градиентов с предобуславливателем C имеет следующий вид

$$\begin{gathered} r^0 = b - Ax^0, \qquad p^0 = C^{-1}r^0, \qquad z^0 = p^0, \\ x^{k+1} = x^k + \alpha_k p^k, \quad \alpha_k = \frac{(r^k, z^k)}{(Ap^k, p^k)}, \quad r^{k+1} = r^k - \alpha_k Ap^k, \\ z^{k+1} = C^{-1}r^{k+1}, \quad p^{k+1} = z^{k+1} + \beta_k p^k, \quad \beta_k = \frac{(r^{k+1}, z^{k+1})}{(r^k, z^k)}. \end{gathered} \tag{17}$$

Условием останова итерационного процесса МСГ с предобуславливателем является «останов по невязке».

Распараллеливание МСГ аналогично распараллеливанию других итерационных методов градиентного типа (см. рис. 1).

3.3 Метод квадратного корня

Одним из быстрых методов решения СЛАУ с симметричной положительно определенной матрицей является метод квадратного корня [11]. Метод основан на разложении симметричной матрицы A в произведение $A = S^T S$, где S – верхняя треугольная матрица с положительными элементами на главной диагонали, S^T – транспонированная. Метод состоит в последовательном решении двух систем уравнений с треугольными матрицами

$$S^T y = b, \quad Sz = y. \tag{18}$$

Решения систем уравнений (18) находятся по рекуррентным формулам

$$\begin{cases} y_1 = b_1 / s_{11}, \\ y_i = \dfrac{b_i - \sum\limits_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2,3,....,n; \end{cases} \quad \begin{cases} z_n = y_n / s_{nn}, \\ z_i = \dfrac{y_i - \sum\limits_{k=i+1}^{n} s_{ik} z_k}{s_{ii}}, \ i = n-1, n-2, ..., 1. \end{cases} \tag{19}$$

Основная идея распараллеливания метода квадратного корня для многопроцессорного вычислителя с общей памятью основана на параллельном вычислении элементов s_{ij}, $j = i,...,n$, каждой $i-$ой строки матрицы S. Строка с номером i разбивается на m частей так, что $n - i = m \times L_i$, где $i-$номер строки, $n-$размерность системы уравнений, $m-$число процессоров, L_i-число элементов строки, вычисляемых каждым процессором (рис. 3).

1 – процессор	2 – процессор	…	m – процессор

Рис. 3. Разбиение $i-$ой строки по процессорам

4. Специализированный Веб-портал решения задач на многопроцессорных вычислителях

Параллельные алгоритмы решения обратной задачи гравиметрии о восстановления плотности в слое и структурной задачи гравиметрии о восстановлении поверхности раздела между средами, а также параллельные алгоритмы решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях», установленный в Отделе некорректных задач анализа и приложений Института математики и механики УрО РАН.

Изначально Веб-портал был предназначен для запуска программ решения задач гравиметрии на многопроцессорном комплексе МВС-1000/17ЕК через Веб-интерфейс [14]. В настоящее время на Веб-портале предусмотрен запуск программ для решения задач на МВС-ИММ, гибридном вычислительном кластере NVIDIA Tesla, установленном в ИММ УрО РАН и гибридной вычислительной системе (ГВС) NVIDIA GeForce, установленной на кафедре ВМ и УМФ ИРИТ-РтФ УрФУ (рис.4). МВС-ИММ состоит из 14 2-х процессорных 2-х ядерных модулей AMD Opteron 64 bit (2.6 Ггц), интерфейса GbitEthernet и 112 Гб оперативной памяти. Кластер NVIDIA Tesla включает 20 вычислительных узлов, имеющих 8 GPU Tesla S2050, 50 Гб ОЗУ и 2 шестиядерных CPU. ГВС представляет собой 4-ядерном процессор Intel Core I7-950 с графическими процессорами NVIDIA GeForce GTX 480.

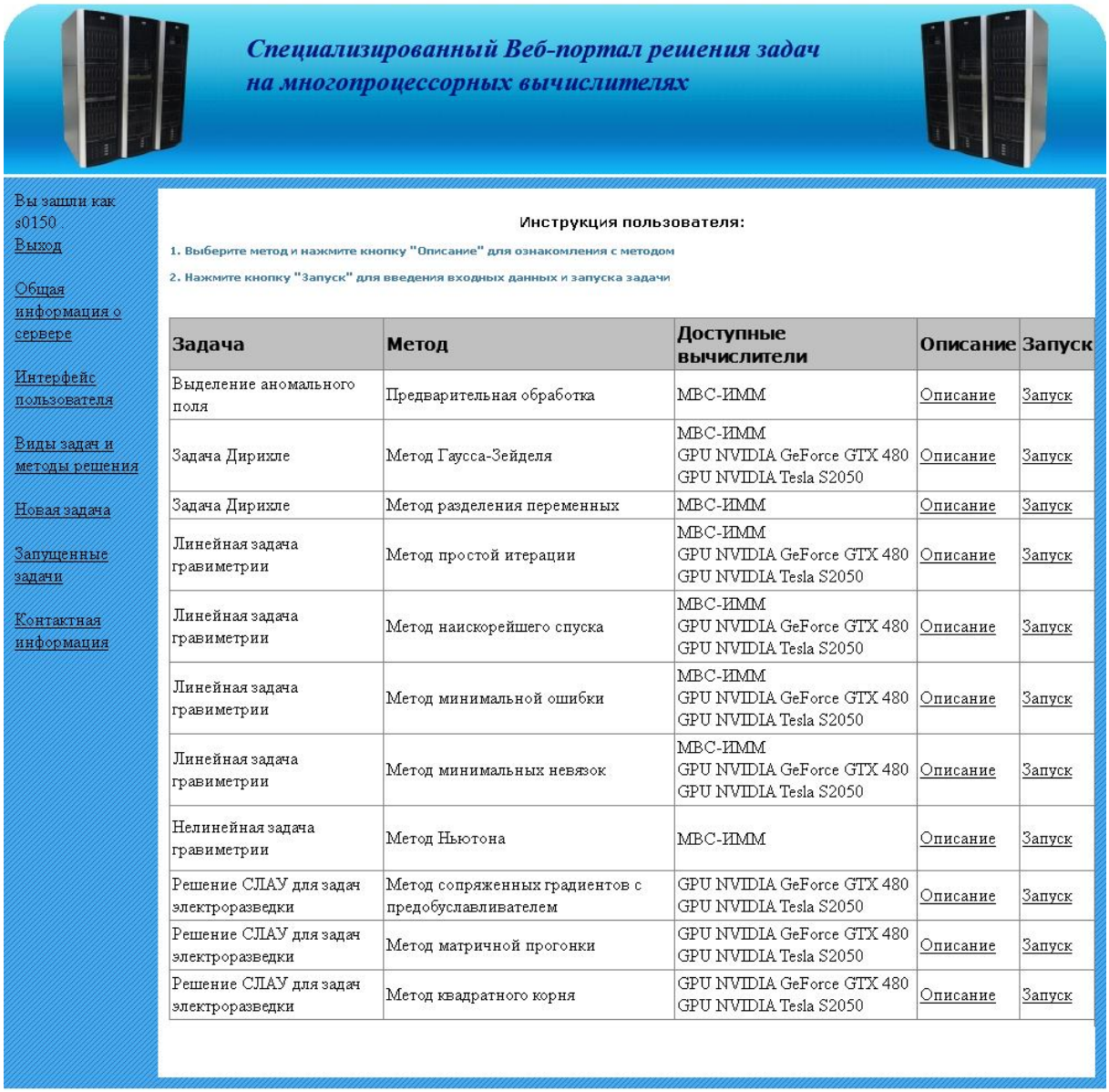


Рис. 4. Список задач, доступных для решения

4.1 Общая характеристика

Специализированный Веб-портал предназначен для запуска программ решения задач гравиметрии (выделение аномального поля, нахождения плотности в слое, восстановления поверхности раздела между средами) и решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки на многопроцессорном комплексе МВС-ИММ, вычислительном кластере NVIDIA Tesla и системе ГВС NVIDIA GeForce. Веб-портал предоставляет возможность пользователю через Веб-интерфейс выбирать тип вычислителя с указанием числа процессорных узлов, вид задачи и метод ее решения, загружать входные данные, получать выходные данные и графическое изображение результатов решения с помощью графического пакета Surfer. Для каждой задачи выводится время счета.

Веб-портал состоит из трех основных частей (рис. 5): HTTP-сервер IIS (Internet Information Services – информационные службы Интернета), на котором установлено Веб-приложение; база данных SQL Server 2000, в которой хранятся все задачи пользователей с входными и выходными данными; служба, выполняющая загрузку данных, запуск задач на многопроцессорные вычислители различных типов, просмотр состояния задачи и загрузку результатов завершившихся задач на Веб-портал.

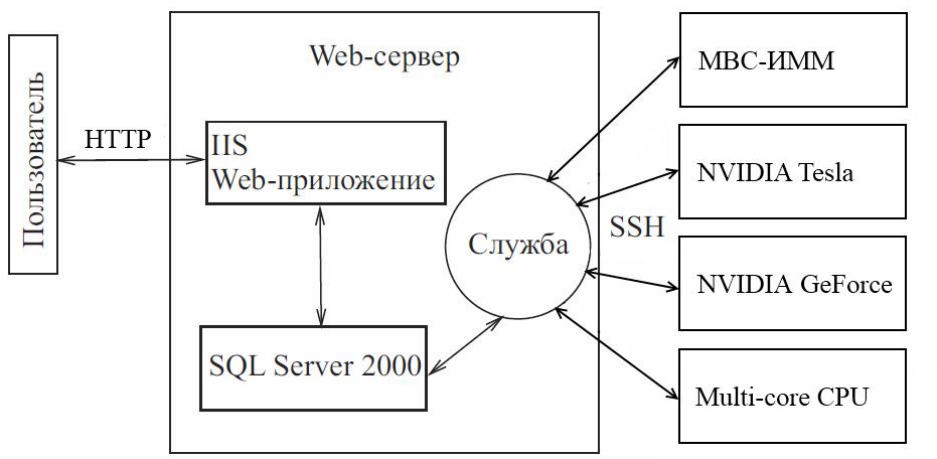


Рис. 5. Архитектура Веб-портала

4.2 Веб-приложение

В настоящее время к исходному Веб-приложению (см. [14]) добавлено следующее.

1. Дополнена система аутентификации. При регистрации на Веб-портале происходит проверка одноименной учетной записи на МВС-ИММ. При ее отсутствии или несовпадении паролей регистрация отменяется. Реализовано SSL-шифрование трафика.

2. Предусмотрена возможность взаимодействия с разными вычислительными устройствами. Веб-портал позволяет запускать методы решения задач на многопроцессорных вычислителях различного типа, поддерживающих связь по **ssh**, передачу файлы по **scp** и запуск программ через планировщик **mpirun** или **sbatch**.

3. Модифицированы службы, отвечающие за прием и передачу файлов и запуск задач на вычислителях различного типа. В интерфейс запуска задачи добавлен выбор вычислителя, на котором эта задача будет запущена. В интерфейсе просмотра запущенных задач добавлено отображение вычислителя, на котором была запущена выбранная задача. Добавлена возможность удаления запущенной задачи из списка задач.

4. Добавлено управление загруженными файлами данных для вычислений, а именно, возможность запуска другой задачи с этими же данными с целью снижения нагрузки на сервер (перекачки файлов). Расширены наборы параметров задачи. Например, для решения линейных задач гравиметрии имеются две возможности – введение двух констант (границ) для горизонтального слоя и загрузка двух файлов (границ) для криволинейного слоя.

5. Подключено управление созданием изображений: пользователь может выбрать, из каких входных и выходных файлов строить изображения, которые будут видны на странице с конкретной решенной задачей. Для задач гравиметрии строится трехмерная поверхность и линии уровня с помощью программы Scripter пакета Surfer.

6. Изменен дизайн страниц, добавлены подробные описания задач и методов. Реализована система управления контентом, а именно, возможность редактирования на сайте описания методов решения задач и добавления новых.

5. Результаты численных экспериментов

5.1 Решение задач гравиметрии с реальными данными

Для восточной части Урала был обработан массив гравитационных данных, измеренный на площади S, имеющей размеры 59.4×144 км^2. Эта площадь пространственно совпадает с зоной Буткинской аномалии векового хода, ограничивающей Урал с востока. Зона представляет собой субмеридиональную аномалию электропроводности земной коры. Аномальное гравитационное поле предоставлено сотрудниками Института геофизики УрО РАН (г. Екатеринбург).

Для изучения природы аномалии по реальным наблюденным данным решены задача 1 о нахождении плотности в горизонтальном слое между глубинами $H_1 = 10$ км и $H_2 = 20$ км для области S и задача 2 о восстановлении поверхности раздела между средами. При этом шаги сетки $\Delta x = 0.594$ км и $\Delta y = 1.44$ км, гравитационная постоянная $f = 6.67 \cdot 10^{-8}$ см^3/г·с^2. Расстояние до асимптотической плоскости составляло $H = 15$ км. Скачок плотности принимался равным $\Delta\sigma = 0.2$ г/см^3. После дискретизации исходных уравнений на сетке задачи свелись к СЛАУ с симметричной (задача 1) и несимметричной (задача 2) матрицами 10000×10000. Для решения задачи 1 использовался параллельный итеративно регуляризованный ММН с параметром регуляризации $\alpha = 0.001$. Для решения задачи 2 использовался итеративно регуляризованный метод Ньютона, на каждом шаге которого применялся параллельный МСГ.

Задача 1 решена на МВС-ИММ, NVIDIA Tesla и ГВС GeForce. Задача 2 решена на МВС-ИММ. На рис. 6 изображено распределение плотности в слое, восстановленной по аномальному полю для области S. На рис. 7 изображена восстановленная поверхность раздела.

Интерпретация результатов проведена сотрудниками ИГФ УрО РАН (г. Екатеринбург). В результате интерпретации выделен протяженный субмеридиональный блок земной коры пониженной плотности (0.2─0.3 г/см^3) (см. [15]).

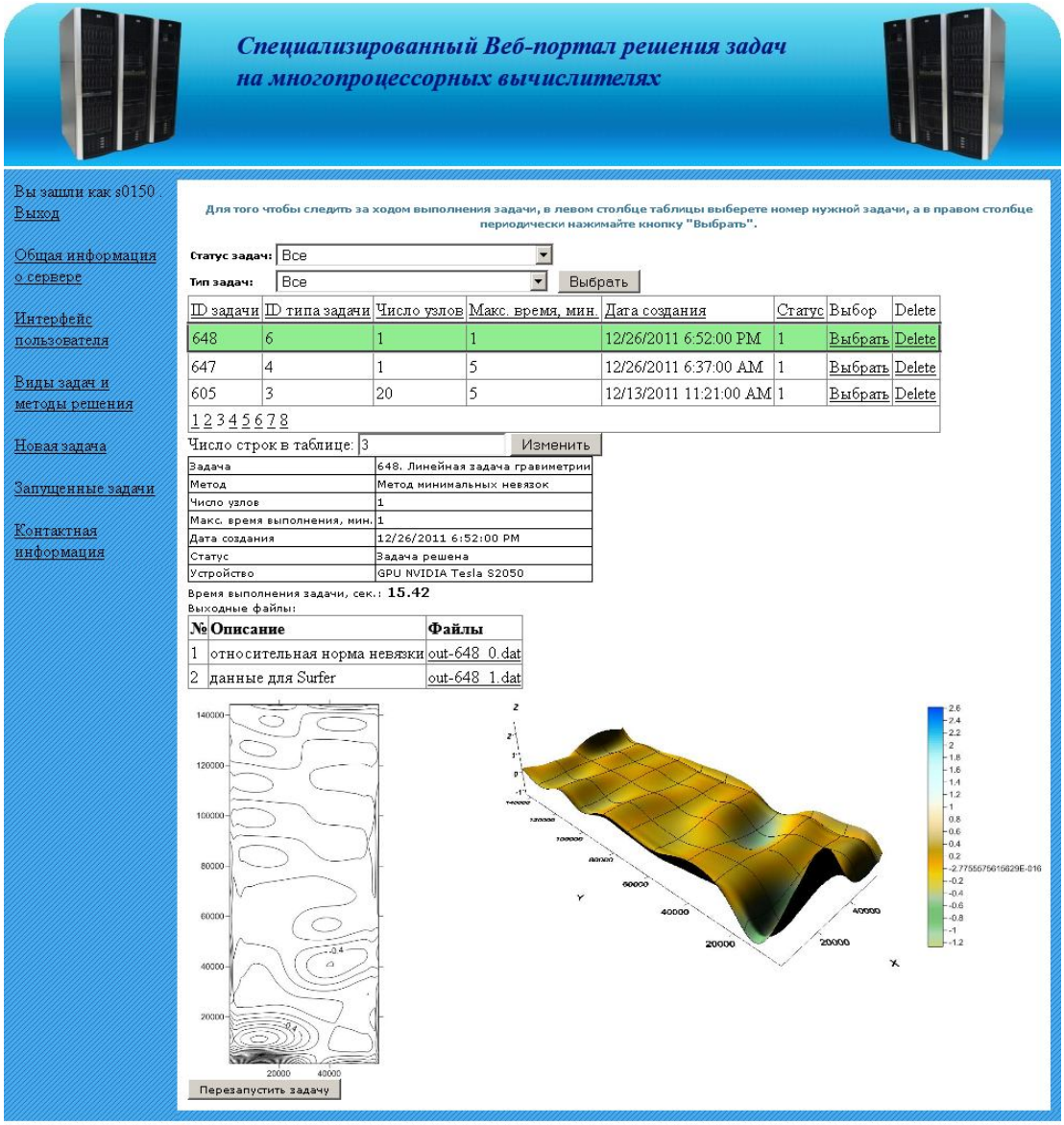


Рис. 6. Решение линейной задачи гравиметрии

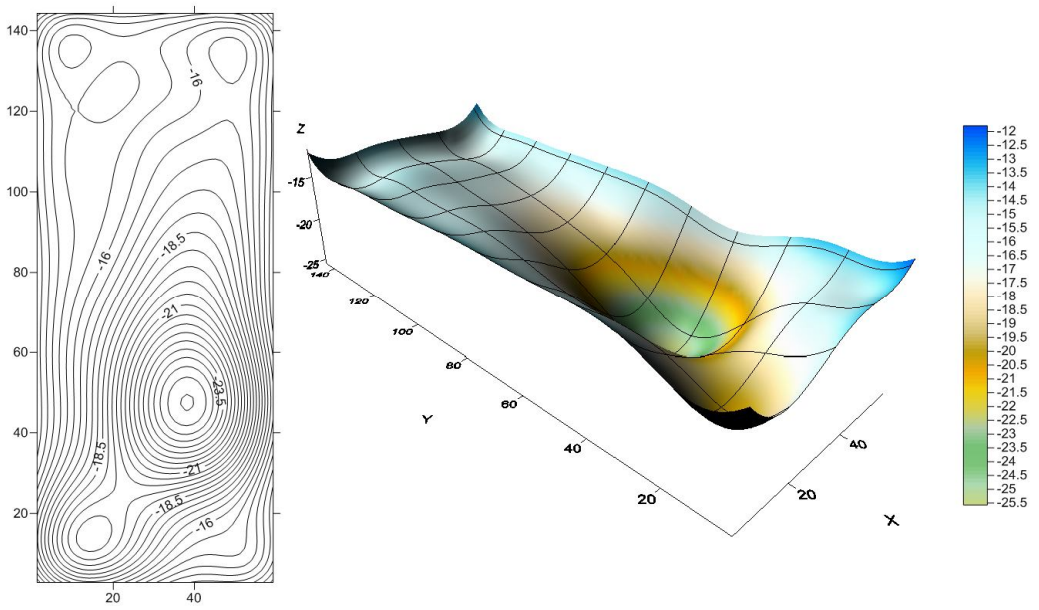


Рис. 7. Решение нелинейной задачи гравиметрии

В табл. 1 приводятся времена решения линейной задачи гравиметрии при $\|Az^k - b\| / \|b\| \approx 0.011$ (749 итераций) на МВС-ИММ и ГВС NVIDIA GeForce GTX 480. Для решения задачи на МВС-ИММ использовалась технологии MPI, для решения задачи на многоядерном процессоре Intel использовалась технология OpenMP, для решения задачи на видеоускорителе GeForce использовалась технология CUDA.

Для сравнения времени счета решения задачи введем коэффициенты ускорения и эффективности параллельных алгоритмов

$$S_m = T_1 / T_m, \quad E_m = S_m / m, \quad S = T_1 / T_2,$$

где T_m – время выполнения параллельного алгоритма на МВС–ИММ либо на многоядерном процессоре с числом процессоров или ядер m $(m > 1)$, T_1 – время выполнения последовательного алгоритма на одном процессоре либо на одном ядре, T_2 – время решения задачи на видеоускорителе.

Таблица 1. Времена решения линейной задачи гравиметрии

Вычислитель	Время T_m, мин.	Ускорение S_m либо S	Эффективность E_m
Intel Core I7 (1 ядро)	7.73	–	–
Intel Core I7 (2 ядро)	4.05	1.91	0.96
Intel Core I7 (4 ядро)	2.1	3.68	0.92
NVIDIA GeForce GTX 480	0.2	38.7	–
МВС-ИММ (1 проц)	24.33	–	–
МВС-ИММ (2 проц)	12.18	1.99	0.99
МВС-ИММ (4 проц)	6.10	3.99	0.99
МВС-ИММ (10 проц)	2.46	9.89	0.99
МВС-ИММ (20 проц)	1.24	19.6	0.98
МВС-ИММ (50 проц)	0.5	48.7	0.97

Результаты вычислений показывают, что использование метода Ньютона и итерационных методов градиентного типа при решении обратных задач гравиметрии позволяют получать корректные решения и определять аномальные плотностные параметры изучаемых глубинных зон земной коры. Применение параллельных алгоритмов при решении задач гравиметрии уменьшает время счета.

5.2 Решение задачи о нахождении распределения потенциала

С помощью параллельного алгоритма матричной прогонки, предобусловленного метода сопряженных градиентов и метода квадратного корня решена задача о нахождении распределения потенциала в проводящей среде с известным квази-модельным решением.

Исходные данные и квази-модельное решение задачи предоставлены лабораторией скважинной геофизики Института нефтегазовой геологии и геофизики СО РАН (г. Новосибирск).

После дискретизации задача сводится к решению СЛАУ с плохо обусловленной симметричной положительно-определенной блочно-трехдиагональной матрицей вида размерности 76136×76136 с квадратными блоками порядка 248.

Приближенное решение задачи сравнивалось с модельным решением с помощью вычисления относительной погрешности

$$\sigma = \left\| \overline{Y}^{M} - \overline{Y}^{\Pi} \right\| / \left\| \overline{Y}^{M} \right\|,$$

где $\overline{Y}^{M}$ – модельное решение задачи, $\overline{Y}^{\Pi}$ – приближенное решение задачи.

Условие $\sigma < \varepsilon$ выбиралось в качестве критерия останова итерационного ПМСГ при решении задачи. Предварительно находилось число обусловленности исходной матрицы A

$$cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \approx 1.3 \cdot 10^{11}, \quad \lambda_{\max} = 1.4 \cdot 10^{6}, \quad \lambda_{\min} = 1.1 \cdot 10^{-5} > 0,$$

где $\lambda_{\max}$ и $\lambda_{\min}$ – наибольшее и наименьшее собственные значения исходной матрицы.

В случае решения задачи предобусловленным ПМСГ с целью проверки условия (16) находилось число обусловленности матрицы $\tilde{A}$

$$cond(\tilde{A}) = \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}} \approx 4.1 \cdot 10^{9} < cond(A).$$

Задача решена с помощью параллельного метода сопряженных градиентов с предобуславливателем, параллельного алгоритма матричной прогонки и параллельного метода квадратного корня при $\sigma_{ПМСГ} \approx 10^{-7}$, $\sigma_{ПАМП} \approx 2 \cdot 10^{-7}$, $\sigma_{ПМКК} \approx 6 \cdot 10^{-7}$. На рис. 8 представлено численное решение задачи.

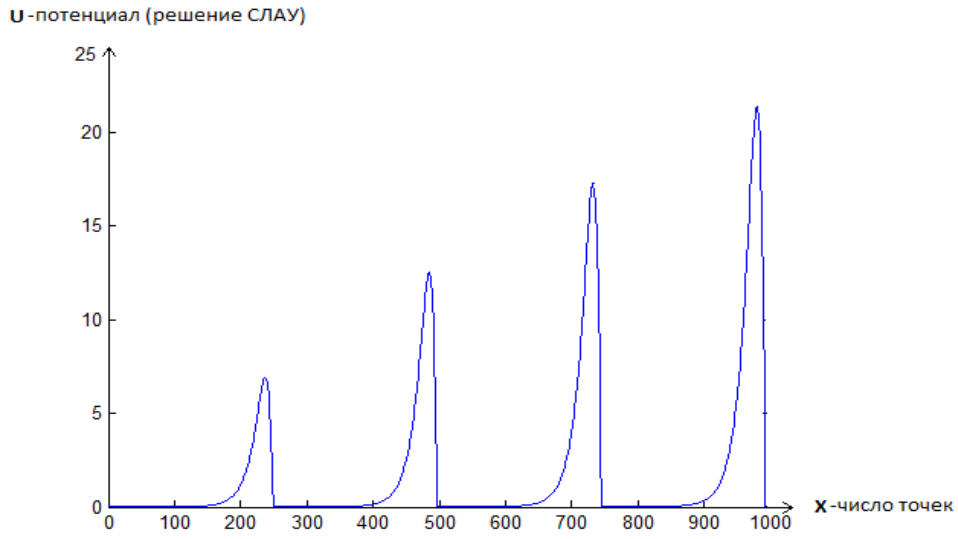


Рис. 8. Численное решение задачи

В табл. 2 приведены времена счета решения задачи на ГВС, установленной в Отделе некорректных задач анализа и приложений ИММ УрО РАН. Система состоит из 4-х ядерного процессора Intel Core I5-750 и видеоускорителя NVIDIA GeForce GTX 285. Предполагается включить данную ГВС в Веб-портал.

Заметим, что время решения задачи с помощью МСГ без предобуславливателя на одном ядре Intel Core I5-750 при $\sigma_{МСГ} = 10^{-3}$ составило 55 мин., что существенно превышает времена решения задачи, представленные в табл. 2.

Таблица 2. Времена решения задачи с квази-модельными данными

Метод	Вычислитель	T_m, сек.(Windows API)	T_m, сек. (OpenMP)
ПМСГ	Intel Core I5 (одно ядро)	57	21
	Intel Core I5 (два ядра)	46	16
	Intel Core I5 (четыре ядра)	36	14
	NVIDIA GeForce GTX 285	–	14
ПАМП	Intel Core I5 (одно ядро)	52	21
	Intel Core I5 (два ядра)	28	18
	Intel Core I5 (четыре ядра)	16	14
	NVIDIA GeForce GTX 285	–	10
ПАКК	Intel Core I5 (одно ядро)	12	7.4
	Intel Core I5 (два ядра)	9	4.6
	Intel Core I5 (три ядра)	10	3.8
	Intel Core I5 (четыре ядра)	12	4.2
	NVIDIA GeForce GTX 285	–	3

Вначале распараллеливание методов ПМСГ, ПАМП и ПАКК для многоядерного процессора Intel с общей памятью проводилось средствами создания потоков операционной системы (ОС) с использованием средств разработки Windows API [16]. Для параллельного выполнения блока вычислений программы создавались потоки (Threads), каждый из потоков выполнялся на «логическом процессоре» ОС, вычисляя свою порцию данных. В конце каждого блока вычислений производилась барьерная синхронизация потоков. Для оптимизации и уменьшения времени счета была использована технология OpenMP. Средствами библиотеки функций OpenMP с использованием специальных директив компилятора проведено автоматическое распараллеливание циклов. Интервал размера L переменной цикла i разбивался на m частей. Каждый поток процесса вычислял свою p -ую часть данных, где $p = L/m$ (рис. 3).

Для уменьшения времени решения задачи также использовалась технология NVIDIA CUDA. Существенное отличие выполнения программы с использованием CUDA от выполнения программы с помощью OpenMP состоит в том, что программа на CUDA выполняется «в тысячи потоков». Такие программы используют массивно-параллельный принцип программирования. При разработке алгоритмов были перенесены на GPU только «ресурсоемкие» векторно-матричные операции. Принцип работы состоял в следующем. Основной поток программы выполнялся на CPU, данные из оперативной памяти загружались на GPU, где проводились расчеты. Далее результаты расчетов выгружались в оперативную память. Очень часто распараллеливание на GPU не является эффективным ввиду образования «узкого горлышка» пропускной способности памяти.

По времени счета наиболее быстрым является метод ПАКК. Время решения СЛАУ 76136×76136 на 4-ядерном процессоре Intel и видеокарте составляет несколько секунд.

Результаты вычислений показывают, что использование параллельных методов ПАМП, ПАКК и ПМСГ с предобуславливателем позволяет достаточно быстро решать задачи с плохо обусловленными матрицами на многопроцессорных вычислителях и можно рекомендовать данные методы для решения задач электроразведки.

6. Заключение

Для решения линейной обратной задачи гравиметрии о нахождении плотности в слое итерационными методами градиентного типа и нелинейной обратной задачи гравиметрии о восстановлении поверхности раздела между средами с помощью итеративно регуляризованного метода Ньютона, а также для решения СЛАУ с блочно-трехдиагональными матрицами примени-

тельно к задачам электроразведки построены параллельные прямые и итерационные алгоритмы. Алгоритмы реализованы на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном процессоре Intel. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях». Решены задачи с квази-модельными и реальными данными.

Результаты вычислений показывают, что использование метода Ньютона и параллельных методов градиентного типа при решении обратных задач гравиметрии позволяют получать корректные решения и определять аномальные плотностные параметры изучаемых глубинных зон земной коры. Использование параллельных алгоритмов решения СЛАУ с блочно-трехдиагональными матрицами применительно к задачам электроразведки позволяет достаточно быстро решать задачи с плохо обусловленными матрицами на многопроцессорных вычислителях.

Литература

1. Мартышко П.С., Кокшаров Д.Е. Об определении плотности в слоистой среде по гравитационным данным // Геофизический журнал. 2005. Т. 27. № 4. С. 678–684.
2. Нумеров Б.В. Интерпретация гравитационных наблюдений в случае одной контактной поверхности // ДАН СССР. 1930. № 21. С. 569–574.
3. Васин В.В., Еремин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН. 2005.
4. Тихонов А.Н., Самарский А.А. Уравнения математической физики. М.: Наука, 1966.
5. Дашевский Ю.А., Суродина И.В., Эпов М.И. Квазитрехмерное математическое моделирование диаграмм неосесимметричных зондов постоянного тока в анизотропных разрезах // Сиб. ЖИМ. 2002. Т. 5. №3 (11). С. 76-91.
6. Мартышко П.С., Пруткин И.Л. Технология разделения источников гравитационного поля по глубине // Геофизический журнал. 2003. Т. 25. № 3. С. 159–68.
7. Воеводин Вл.В. Технологии параллельного программирования. URL: http://parallel.ru/ (дата обращения: 06.02.2012).
8. Берилло А. NVIDIA CUDA – неграфические вычисления на графических процессорах. URL: http://www.ixbt.com/video3/cuda-1.shtml (дата обращения: 06.02.2012).
9. Акимова Е.Н., Белоусов Д.В. Распараллеливание алгоритмов решения линейной обратной задачи гравиметрии на МВС-1000 и графических процессорах // Вестник ННГУ. 2010. № 5. Ч. 1. С. 193–200.
10. Bakushinsky A., Goncharsky A. Ill-Posed Problems: Theory and Applications. London: Kluwer Akad. Publ. 1994.
11. Фаддеев В.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Гос. издат. физ.-мат. литературы. 1963.
12. Акимова Е.Н. Распараллеливание алгоритма матричной прогонки // Математическое моделирование. 1994. Т. 6. № 9. С. 61-67.
13. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978.
14. Акимова Е.Н., Гемайдинов Д.В. Параллельные алгоритмы решения обратной задачи гравиметрии и организация удаленного взаимодействия между МВС-1000 и пользователем // Вычислительные методы и программирование. 2008. Т. 9. № 1. С. 133–144.
15. Мартышко П.С., Васин В.В., Акимова Е.Н., Пьянков В.А. О комплексной интерпретации гравитационных и магнитовариационных данных (на примере Башкирского предуралья) // Геофизика. 2011. № 4. С. 30-36.
16. Методика разработки многопоточных приложений: принципы и практическая реализация. URL: http://www.rsdn.ru/article/baseserv/RUThreadingMethodology.xml (дата обр. 06.02.2012).

Адаптация технологии параллельных вычислений к задачам корпоративных информационных систем. Сложности практического применения

П.А. Баркетов, В.И. Сердюк

ЗАО «Софтпоинт» (Спонсор ПаВТ 2012)

В статье описываются возможности применения параллельных вычислений для решения задач в информационной системе любой организации. Уникальность ее состоит в том, что автор отталкивается от практики применения параллельных вычислений в реальных системах.

На сегодняшний день разработано множество технологий параллельных вычислений, но внедрение в реальные массовые информационные системы сопряжено с множеством сложностей. В данной статье автор раскрывает некоторые из них

1. Введение

В современном мире любая крупная организация имеет в своем распоряжении корпоративные информационные системы (далее – КИС), охватывающие все аспекты ее хозяйственной деятельности. Наиболее известные КИС на российском рынке – SAP, Microsoft Dynamics, 1С и другие. Для упрощения излагаемого материала, в статье мы будем рассматривать КИС на базе наиболее распространенной платформы 1С:Предприятие, подразумевая под ней все возможные КИС, так как подходы к применению параллельных вычислений при решении задач в различных системах на самом деле схожи.

2. Предпосылки к применению параллельных вычислений в решении задач КИС

Итак, наиболее популярная архитектура информационной системы имеет 3 уровня:

- сервер баз данных;
- сервер приложения;
- клиентские приложения.

На следующем рисунке отражена типовая архитектура современных КИС (см. на Рис.1).

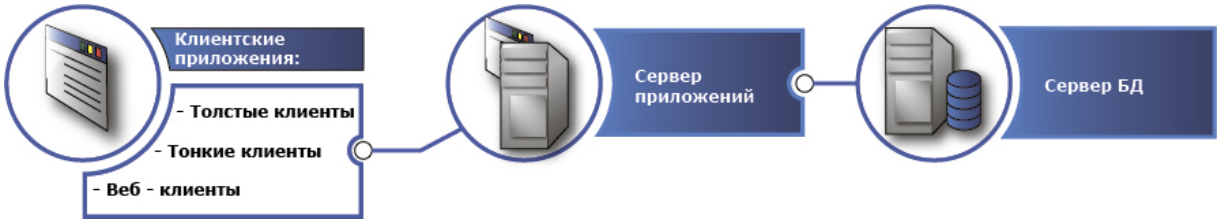


Рис. 1. Типовая архитектура современных КИС

В крупных КИС количество клиентских подключений может быть от 300 до нескольких тысяч, как в штатном режиме, так и в пиковые моменты работы. Нагрузка на оборудование в пиковые моменты увеличивается в несколько раз, в сравнении с штатным режимом. При этом, увеличиваются длительности основных типовых операций, выполняемых в системе. В определенный момент КИС может быть полностью не работоспособна. Для обработки большого потока информации компании приобретают дополнительные аппаратные ресурсы, сотни организаций разрабатывают решения для ускорения того или иного отдельного компонента системы (к примеру, oracle real application cluster). Но, есть ли уверенность, что закупка новых аппаратных или программных ресурсов даст требуемый результат? Для ответа на этот вопрос разберем несколько подходов, отраженных на следующем рисунке (см. на Рис.2):

Рис. 2. Типовой подход к решению проблем производительности

При вариантах «загрузить до определенных пределов» и «рассматривается покупка/добавление новых аппаратных ресурсов» рассматривается возможность параллелизма. Но на практике, в большинство проблем КИС, описанных в варианте «загрузить до определенных пределов», решается подходом «рассматривается покупка/добавление новых аппаратных ресурсов». При этом, ни в том, ни в другом случае не используются параллельные вычисления (но причины тому, мы намеренно не будем рассматривать в данной статье).

3. Описание подходов использования параллельных вычислений в решение задач и практические примеры

Далее, речь пойдет о нескольких популярных подходах к реализации и применению параллельных вычислений.

Предлагаются два подхода к реализации параллельных вычислений в существующих КИС:

3.1 Реализация механизма параллельных вычислений средствами бизнес – логики

Преимущества:

- Уменьшение длительности выполнения операций системы.
- Предсказуемый уровень потребления ресурсов и масштабирования системы, в случае отсутствия конкуренции за общие ресурсы.

Сложности:

- Не все операции КИС поддаются распараллеливанию. Исходя из этого, большое время тратиться на анализ логики работы.
- Трудоемкость практической реализации. Так как основные прикладные языки программирования не приспособлены для параллельного программирования, то возникают сложности в практической реализации.
- Потребуются работы по адаптации механизма распараллеливания к привычным интерфейсным формам пользователей.

Архитектура решения: (см. Рис.3)

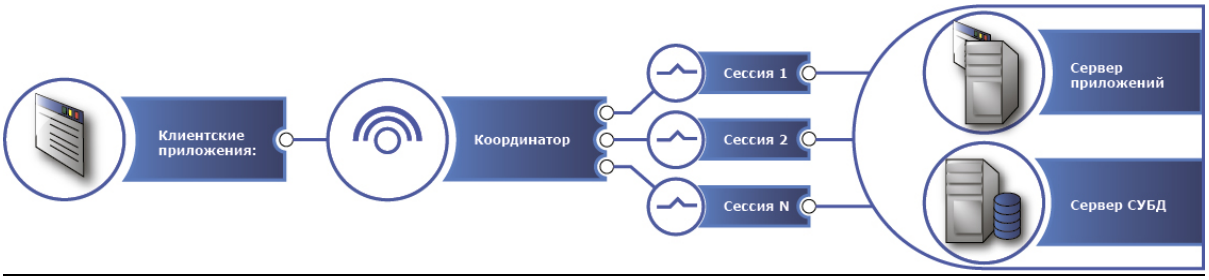


Рис. 3. Архитектура решения для реализации механизма параллельных вычислений средствами бизнес - логики

Примечание к архитектуре решения:
- Клиентское приложение – интерфейсное приложение, через которое пользователь взаимодействует с КИС.
- Координатор – модуль, осуществляющий анализ входных данных клиентского приложения и распределение задач между параллельными сессиями.
- Сессии – экземпляры клиентских приложений для параллельного выполнения.
- Сервер приложения, сервер СУБД – основные звенья КИС.

Где применимо:
- Параллельный расчет заработной платы для сотрудников.
- Распараллеливание регламентных процедур по закрытию различных вида учетов (бухгалтерского, налогового).
- Отчетность.

Реальные примеры реализации:

Предыстория:
Крупная розничная сеть магазинов по всей территории СНГ. Необходим ежемесячный расчет заработной платы для 55 000 сотрудников. Исторически сложилось, что расчет заработной платы ведется на системе 1С7.7 и MSSQL 2005. Написана достаточно сложная логика расчета для каждого сотрудника, применяются различные мотивационные схемы.
Расчет для всех сотрудников занимал более 2 суток, причем, ежегодно прогнозируется прирост персонала по количеству на 30% и более. После первичного анализа производительности КИС на этапе расчета заработной платы установлено, что загрузка MSSQL по основным ресурсам (CPU, Disk) не более 5%, а узким местом является расчет, выполняемый на стороне «клиента». После анализа алгоритмов выявлено, что расчет заработной платы каждого сотрудника является независимым друг от друга, поэтому целесообразно для ускорения этого процесса использовать подход, описанный выше.

Реализация:
На прикладном языке 1С7.7 написан координатор, который разделяет всех сотрудников на группы и раздает задания отдельным сеансам 1С. Таким образом, каждый отдельный сеанс проводит расчет заработной платы части сотрудников. Далее, рассчитанные данные со всех сеансов 1С консолидируются и отображаются для пользователя КИС, как и раньше. Таким образом, мы распараллеливаем расчетный алгоритм и получаем значительное ускорение.

Сложности:
На первый взгляд можно предположить, что расчет ускориться пропорционально количеству сессий 1С7.7. В идеальной замкнутой системе это именно так. Но, на практике необходимо обязательно анализировать и другие параметры, такие как: сетевое окружение,

свободная оперативная память, эффективность использования кеша MSSQL, логика блокировочного механизма. Без анализа «доступной» производительности этих ресурсов планирование реальной эффективности распараллеливания расчетов невозможно либо будет очень неточной.
В частности, для системы 1С7.7 и MSSQL одним из узких мест распараллеливания при архитектуре «Терминал – сервер СУБД» будет являться сетевое взаимодействие. Это обусловлено тем, что операции 1С формируют огромное количество (более 1000/секунду) запросов к серверу БД.

Результаты:
Изначально, расчет заработной платы занимал 52 часа. С внедренным механизмом распараллеливания удалось сократить этот показатель на порядок.
В следующей таблице (см. Таб.1) представлена зависимость между количеством сессий для распараллеливания операций расчета и соответствующим ему времени выполнения операции расчета:

Таблица 1. Зависимость между количеством сессий для распараллеливания операции и времени выполнения операции

Количество сессий 1С для распараллеливания операций расчета, шт.	Время выполнения операции расчета, часы.
5	12,48
20	3,9
100	0,78
200	0,69

Мы видим, что распараллеливание операции расчета только 5 сессий позволило сократить общее время расчета почти в 5 раз. Для наглядности продемонстрируем зависимости времени расчета от количества распараллеливаемых сессий 1С в виде следующего графика (см. Рис.4):

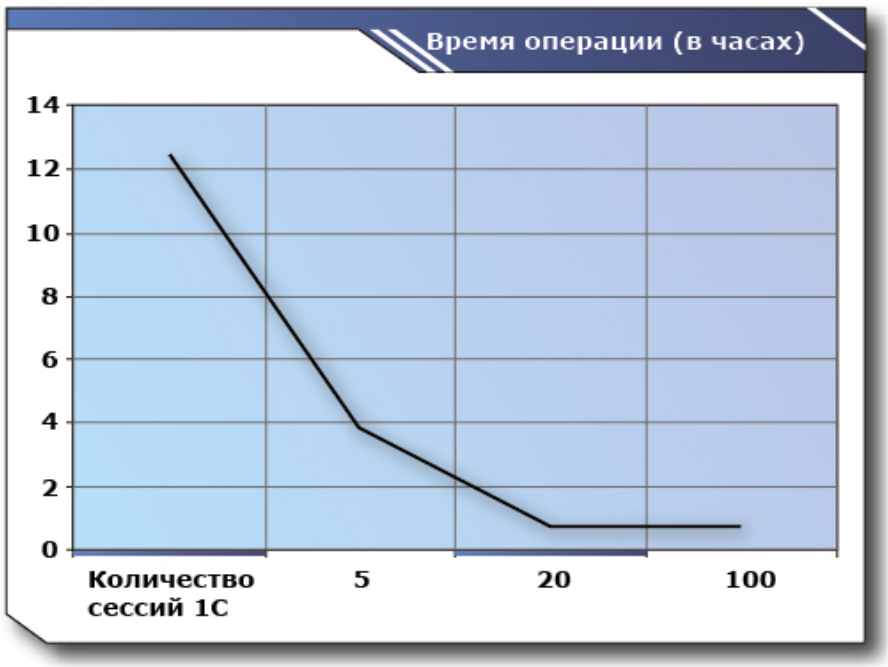


Рис. 4. Зависимость времени расчета от количества распараллеливаемых сессий 1С

Обратите внимание, что динамика ускорения операции в 1С не линейна и замедляется при большом количестве сессий 1С. Не исключено, что длительность операции может

ухудшиться из-за появления новых узких мест, обусловленных большим количеством сессий, например, эскалация блокировок на MS SQL, нехватка оперативной памяти и свопирование.

3.2 Реализация механизма параллельных вычислений системными средствами

Преимущества:

- Независимость от определенной КИС. Позволяет использовать технологию распараллеливания для других КИС.
- Сравнительно простой процесс внедрения. Не обязательно разбираться в логике работы КИС.

Сложности:

- Техническая реализация в используемых КИС.
- Балансировка нагрузки.
- Требуемый уровень отказоустойчивости.

Архитектура решения: (см. Рис.5)

Рис. 5. Архитектура решения для реализации механизма параллельных вычислений системными средствами

Примечание к архитектуре решения:
Координатор – механизм интеллектуального распределения запросов между серверами СУБД.

Где применимо:
В любых КИС, где процентное соотношение операций на запись значительно меньше, чем операций на чтение.

Реальные примеры реализации:

Предыстория:
Крупный холдинг ведет учет финансов по всем своим подразделениям в единой системе 1С8.2 и MSSQL 2008 R2. Пользователей не устраивает скорость основных операций в информационной системе (от 10 – 60 секунд на выполнение). При этом, простои в работе КИС прямо пропорциональны финансовым потерям организации.
Учитывая тот факт, что количество пользователей постоянно увеличивается, требуется разработать механизм ускорения оперативной деятельности и создать запас прочности по производительности в периоды большой нагрузки (например, в период сдачи отчетности).
Были проведены исследования производительности и сделаны следующие выводы:

- Нет ярко выраженных узких мест на сервере БД (загрузка в среднем 30-40% по всем ресурсам).
- Каждая операция пользователя формирует большой набор простейших запросов к MSSQL 2008 R2 (1 000 и более), поэтому основным узким местом является работа с сетевым интерфейсом.
- Прослеживается деградация скорости работы пользователей при большом количестве одновременных операций (подтверждает предыдущее утверждение).
- Распараллеливание каждого запроса на уровне MS SQL, учитывая их количество и малую длительность, неэффективно.

Рассматриваемые решения:
- Отказоустойчивый кластер для MSSQL. Решение частично решает проблему надежности, но не решает проблему производительности.
- Механизм распараллеливания расчетов на уровне прикладной логики. В отличие от предыдущего примера, нет возможности разделить логику на независимые части, большое многообразие операций. Учитывая, что операции выполняются до 1 минуты – этот вариант очень трудоемкий и неэффективный.
- Оптимизация запросов. Но при этом, количество запросов огромное количество и они в отдельности оптимальны, поэтому, рассмотрение данного подхода не целесообразно.

Предложенное решение:
Создание координатора на уровне драйвера доступа к БД MSSQL 2008 R2, который бы позволил осуществить интеллектуальную балансировку нагрузки между большим количеством экземпляров MSSQL 2008 R2. Учитывая, что процентное соотношение запросов на чтение и изменение составляет 99% к 1% соответственно, есть возможность отправлять все запросы на запись всем экземплярам MSSQL 2008 R2, а на чтение – наименее загруженным.

Сложности:
- Внедрение данного координатора в работающую систему 1С8.2.
- Контроль целостности БД на различных серверах MSSQL 2008 R2.
- Реализация интеллектуального анализа запросов к БД (чтение/запись).
- Реализация надежного механизма отработки всех нестандартных ситуаций и ошибок.

Результаты:
Ускорение достигается за счет распараллеливания запросов на чтение на сервере MS SQL. Плюс к этому, балансировать запросы можно не только с учетом загруженности серверов, но и, например, с учетом бизнес–логики (запросы, относящиеся к бухгалтерии, выполняются на одном сервере MSSQL 2008 R2, к операционной деятельности – на другом). Это дает возможности более эффективного использования кеша MS SQL 2008 R2. Преимущества данного механизма очевидны.
Количественное ускорение механизма – от 20 до 50% для 2-3 серверов БД. Для кластера с большим количеством серверов этот показатель может изменяться в лучшую сторону.
Развитие данного подхода – использование альтернативных TCP\IP механизмов передачи данных между серверами, что позволит сократить временные затраты на передачу данных.
Пример работающего механизма – решение компании Dolphin «IX PCI EXPRESS GEN2».

В данной статье не рассматривалась тема параллелизма на стороне сервера СУБД. Практически на всех современных СУБД присутствует этот механизм, но решение на использование должно быть взвешенное и эффект проверен. Как правило, для OLTP систем рекомендуется отключать параллелизм, а в случае с OLAP системами – включать. Также требуется учитывать аппаратные возможности, например, количество процессорных ядер.

4. Общие выводы

Итак, на основании вышесказанного заключаем, что существует множество подходов к распараллеливанию операций в современных КИС, реализуемых различными механизмами. Но, на практике их использование ограничено. Обычно это обусловлено:

- недостаточной осведомленностью о современных подходах к распараллеливанию;
- невозможностью либо значительной сложностью интеграции с текущей аппаратной и программной инфраструктурой КИС;
- необходимостью простого обслуживания;
- большой стоимостью внедрения.

Основной вывод заключается в том, что универсального механизма распараллеливания задач для КИС, ни на аппаратном, ни на программном уровне, на данный момент нет. Но в тоже время можно предложить универсальный подход (правило) на аудит любых задач в КИС в плане реализации механизма распараллеливания. Но эта тема уже отдельной статьи.

Литература

1. Воеводин В.В., Вл.В.Воеводин. Параллельные вычисления.,2004 (ISBN 5-94157-160-7).
2. Селище Н., Администрирование системы 1С:Предприятие 8.2,2012 (978-5-459-00657-5)

Моделирование плазмы методом частиц в ячейках на гетерогенных кластерных системах*

С.И. Бастраков[1], А.А. Гоносков[2], Р.В. Донченко[1],
Е.С. Ефименко[2], А.С. Малышев[1], И.Б. Мееров[1]

Нижегородский государственный университет им. Н.И. Лобачевского[1],
Институт прикладной физики РАН[2]

Ставится задача разработки программного обеспечения для моделирования плазмы на гетерогенных кластерных системах. Для решения задачи используется широко распространенный метод частиц в ячейках (Particle-in-Cell, PIC). Предлагается подход к распараллеливанию вычислительной схемы метода для кластерных систем. Рассматриваются особенности реализации метода для графических процессоров. Приводятся результаты вычислительных экспериментов, характеризующие эффективность и масштабируемость текущей реализации. Формулируются выводы и планы по дальнейшему развитию.

1. Введение

Одной из востребованных в настоящее время областей численного моделирования физических процессов является моделирование динамики плазмы и взаимодействия мощных лазерных импульсов с различными мишенями. В числе важных приложений можно выделить создание компактных источников для адронной терапии при лечении онкологических заболеваний, создание фабрик короткоживущих изотопов для биоимиджинга, разработку приборов для исследования внутримолекулярных и внутриатомных процессов.

Часто, в связи с высокой степенью нелинейности и геометрической сложностью задачи, исследование динамики плазменных структур основывается на моделировании плазмы методом частиц в ячейках (Particle-in-Cell, PIC) [1]. Основная специфика метода заключается в одновременной обработке принципиально разнородных массивов данных, содержащих информацию о считающихся непрерывными координатах и скоростях заряженных частиц плазмы, и об электромагнитном поле, определенном на пространственной сетке. Отсутствие однозначных путей упорядочивания обращений к памяти обуславливает сложность эффективной программной реализации, как для классических кластерных вычислительных систем, так и для гетерогенных систем с использованием графических процессоров. Применение метода для решения прикладных задач часто требует использования суперкомпьютерных технологий – известны задачи, требующие моделирования динамики $\sim 10^9$ и более частиц в пространстве, представленном $\sim 10^8$ ячеек.

Целью настоящей работы является представление текущих результатов, полученных в рамках проекта, направленного на поиск и реализацию наиболее эффективных подходов к параллельному моделированию плазмы методом частиц в ячейках на гетерогенных кластерных системах.

2. Постановка задачи и метод решения

Моделирование плазмы методом частиц в ячейках [1] основано на математической модели, в рамках которой плазма представляется ансамблем отрицательно заряженных электронов и положительно заряженных ионов, создающих и двигающихся под действием электромагнитных полей. Движение заряженных частиц описывается в рамках классических релятивистских

* Работа выполнена в лаборатории «Информационные технологии» ВМК ННГУ при поддержке ФЦП «Научные и научно-педагогические кадры инновационной России», госконтракт № 02.740.11.0839.

уравнений движения. Эволюция электромагнитного поля описывается в рамках уравнений Максвелла, в которые также входит векторное поле плотности тока, создаваемое частицами.

Используется классическая схема метода частиц в ячейках, каждая итерация вычислительного цикла состоит из 4 этапов: взвешивание токов, интегрирование уравнений поля, интерполяция полей, интегрирование уравнений движения частиц, и соответствует одному шагу по времени. Взвешивание токов состоит в определении сеточных значений плотности тока по известным положениям и скоростям частиц. Каждая частица вносит вклад в значение плотности тока только в 8-ми ближайших к ней узлах сетки; для получения итоговых значений вклады всех частиц суммируются [1]. Далее выполняется шаг численного интегрирования уравнений Максвелла по времени, определяются новые сеточные значения электрического и магнитного поля; используется метод FDTD [2]. На следующем этапе производится линейная интерполяция сеточных значений электрического и магнитного поля для точек нахождения частиц; для получения каждого значения используются 8 ближайших сеточных значений. Интерполированные значения полей используются при интегрировании уравнений движения частиц методом Boris [1].

3. Реализация для гетерогенных кластерных систем

Параллельная реализация разрабатывается для использования на гетерогенных кластерных системах. Рассматривается случай, когда узлы кластера содержат центральные и графические процессоры. Используется следующая схема работы:

1. Распределение задачи по узлам вычислительного кластера осуществляется при помощи интерфейса MPI.
2. На каждом из узлов кластера запускается несколько MPI-процессов. Каждый процесс использует для вычислений либо одно или несколько ядер центральных процессоров, либо один из графических процессоров (или других устройств).
3. Распараллеливание на ядра центральных процессоров внутри узла производится при помощи технологии OpenMP. Программирование графических процессоров выполнено при помощи технологии OpenCL.

3.1 Декомпозиция задачи

Декомпозиция задачи осуществляется по территориальному принципу: расчетная область разбивается на подобласти (домены), операции над которыми выполняются параллельно разными MPI-процессами. Процесс, на котором производятся операции над определенным доменом, хранит данные об электромагнитных полях и данные о частицах, находящихся в соответствующей части физического пространства. Данные, относящиеся к узлам, попадающим на границы между двумя или более доменами, хранятся во всех вычислительных процессах, обрабатывающих такие домены. Кроме того, процесс хранит данные о магнитных полях в центрах ячеек, граничащих с обрабатываемым им доменом.

Операции параллельной обработки внутренних частей доменов выполняются полностью аналогично последовательной версии. Для поддержания актуальности данных во всех доменах используются обмены данными о токах, полях и частицах. Схема организации обменов устроена следующим образом (рис. 1).

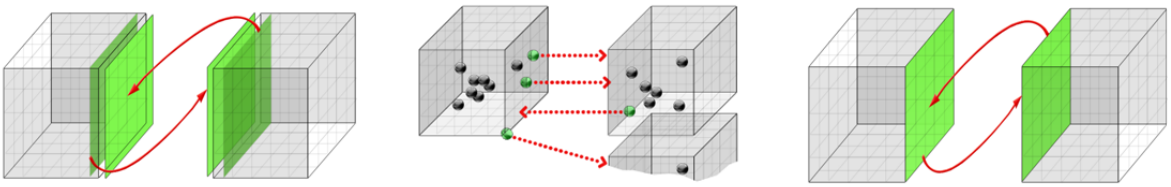


Рис. 1. Схема организации обменов (поля, частицы, токи)

При обменах токами и полями каждый домен взаимодействует с 6 соседями, при обменах частицами – с 26 соседями (в случае, когда частица покидает пределы домена, она попадает в

один из 26 соседних доменов). Таким образом, все обмены данными осуществляются между процессами, обрабатывающими соседние домены.

3.2 Особенности реализации для GPU

В данном разделе будет использоваться терминология OpenCL. Эксперименты проводились с использованием GPU NVIDIA, приведем соответствие терминологии OpenCL и NVIDIA CUDA: локальная память в OpenCL соответствует разделяемой (shared) в CUDA, элемент работы и группа работ – потоку/нити (thread) и блоку потоков/нитей (thread block).

Данные вычислительных экспериментов показывают, что основное время работы распределяется между этапами интегрирования уравнений движения частиц и взвешивания токов. Оба этапа оперируют с частицами, поэтому ключевым является способ представления и обработки данных о частицах на GPU. Важной оптимизацией при работе с глобальной памятью GPU является обеспечение коалесинга (coalescing) [5], в данном случае это обозначает обработку соседних по расположению в памяти частиц соседними (по индексам) элементами работы в одной группе работ.

Традиционным способом упорядочивания обращений к памяти при реализации метода частиц в ячейках (как на CPU, так и на GPU) является группировка частиц по ячейкам. Естественным решением при реализации на GPU является обработка частиц в одной ячейке отдельной группой работ. Возможной проблемой является недостаточная вычислительная нагрузка на группу работ – для ряда задач, в том числе и целевых задач разрабатываемого ПО, среднее количество частиц в ячейке составляет десятки, в то время как для эффективной загрузки современных GPU желательный размер группы работ составляет не менее нескольких сотен [5].

Для решения данной проблемы используется подход, близкий к предложенному в [4]. Соседние ячейки группируются в суперячейки, все суперячейки имеют равный размер, кратный размеру ячейки (например, 2 ячейки по каждой оси). Частицы сгруппированы в памяти по суперячейкам, все частицы суперячейки обрабатываются одной группой работ с интенсивным использованием локальной памяти. При интегрировании уравнений движения частиц локальная память используется как ручной кэш для загрузки из глобальной памяти значений полей, необходимых для интерполяции, при взвешивании токов локальная память используется для накопления промежуточных результатов. Благодаря этому обеспечивается высокая загрузка устройства и эффективный паттерн доступа к памяти GPU.

Отметим, что количество разделяемой памяти в современных GPU NVIDIA (до 48 КБ) накладывает существенные ограничения на размер суперячейки и размер группы работ: например, при использовании группы работы размера 256 в двойной точности, размер суперячейки не может превышать 2 ячейки по каждой оси. Вероятно, использование групп работ и суперячеек большего размера (при наличии большего количества памяти) позволило бы еще более эффективно задействовать GPU.

4. Результаты экспериментов

Для анализа корректности реализации метода частиц в ячейках был использован набор тестовых задач, соответствующие результаты приведены в [3]. В данном разделе приводятся результаты экспериментов, связанные с оценкой эффективности и масштабируемости реализации для традиционных и гетерогенных кластерных систем.

Вычислительные эксперименты проводились на следующих системах:

1. МВС-100К в МСЦ РАН (на каждом узле 2 CPU Intel Xeon E5450, 8 ГБ RAM, Infiniband DDR, CentOS 5.6 x86_64).
2. Akka в High Performance Computing Center North (на каждом узле 2 CPU Intel Xeon L5420, 16 ГБ RAM, Infiniband 4x, CentOS 5.6 x86_64).
3. Кластер ННГУ (на каждом узле 2 CPU Intel Xeon L5630, 2 GPU NVIDIA Tesla X2070, 24 ГБ RAM, Infiniband QDR, Windows Server 2008 HPC Edition SP2 x64).
4. Суперкомпьютер «Ломоносов» в НИВЦ МГУ (на каждом узле 2 CPU Intel Xeon E5630, 24 ГБ RAM, 2x NVIDIA Tesla X2070, Infiniband QDR, Clustrx CNL v1.1 x86_64).

В серии экспериментов по анализу масштабируемости рассматривалась тестовая задача моделирования Ленгмюровских колебаний плазмы с использованием 503 миллионов частиц и 17 миллионов ячеек пространственной сетки, двойная точность. На каждое ядро создавался отдельный MPI-процесс. Результаты для систем МВС-100К и Akka приведены на рис. 2. При использовании *2048 ядер* на рассматриваемых системах достигается *эффективность 71% и 85% относительно 512 ядер*.

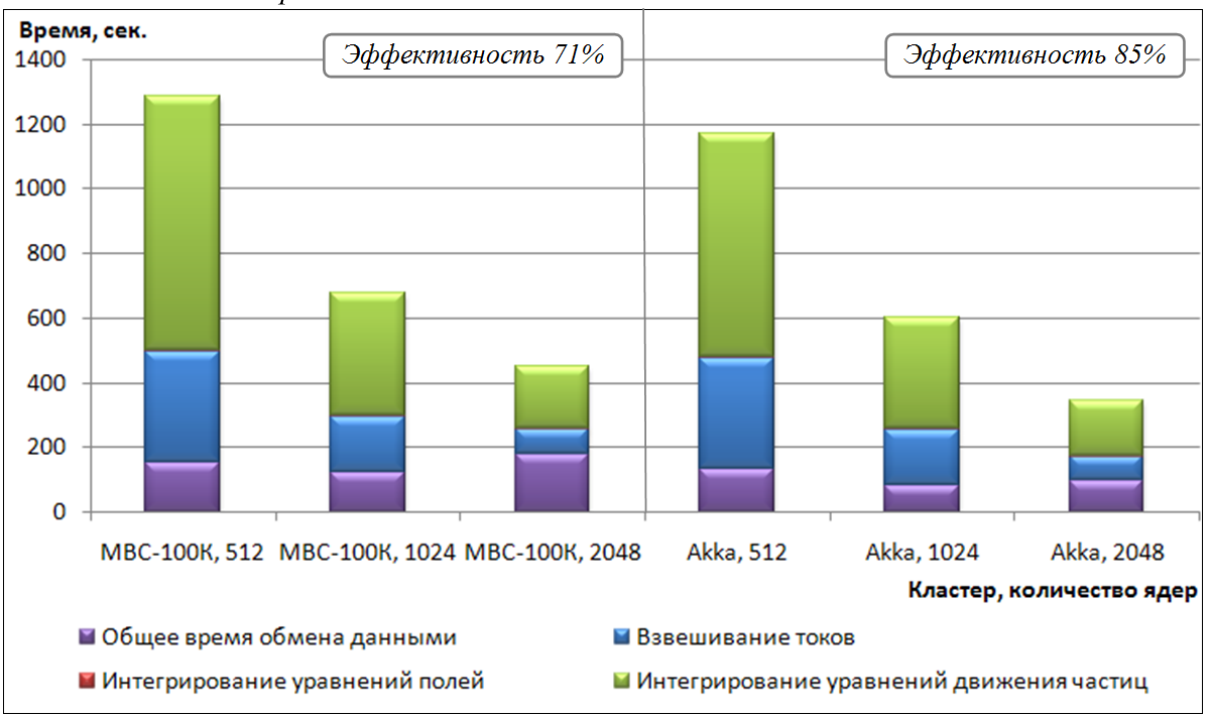


Рис. 2. Масштабируемость на кластерных системах

В серии экспериментов по сравнению производительности CPU и GPU на одном узле решалась задача моделирования Ленгмюровских колебаний плазмы с использованием 983 тысяч частиц, 33 тысяч ячеек пространственной сетки. Результаты для одинарной и двойной точности, полученные на кластере ННГУ, приведены на рис. 3, времена обменов между оперативной памятью и памятью GPU включены во времена соответствующих этапов вычислений. В двойной точности при использовании 1 GPU достигает ускорения около *3,1* раз относительно версии с использованием 8 ядер CPU, при использовании 2 GPU – около *4,7* раз. В одинарной точности ускорения при использовании 1 и 2 GPU относительно 8 ядер CPU составляют *4,3* и *5,8* раз соответственно.

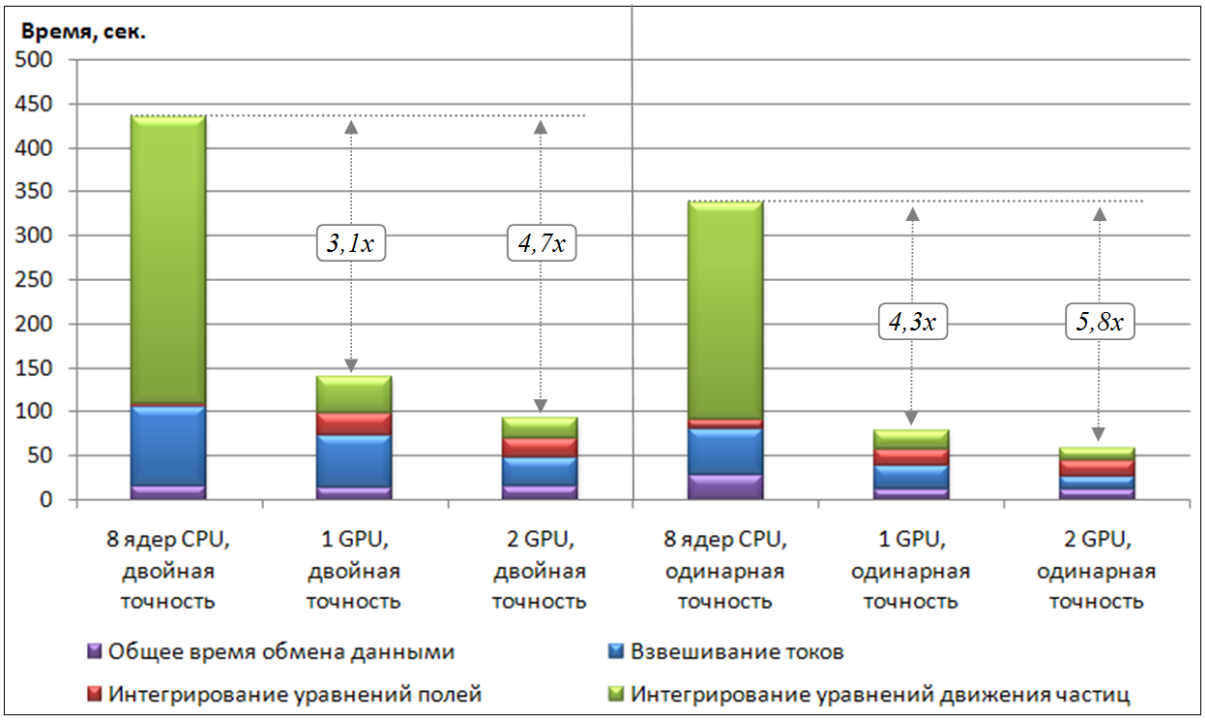


Рис. 3. Сравнение производительности CPU- и GPU-версии на кластере ННГУ

В другой серии экспериментов производилось измерение слабой масштабируемости – вычислительная трудоемкость задачи масштабировалась пропорционально количеству используемых вычислительных устройств (GPU либо ядер CPU). Трудоемкость выполнения одной итерации вычислительного цикла пропорциональна сумме количества частиц и общего количества узлов пространственной сетки. При увеличении количества используемых устройств в N раз, размер расчетной области увеличивался в N раз вдоль одной из осей и оставался неизменным вдоль других, пространственная плотность частиц оставалась неизменной (таким образом, общее количество частиц также увеличивалось в N раз). В качестве базовой выступала задача моделирования Ленгмюровских колебаний плазмы, двойная точность. Результаты с использованием только CPU, полученные на системе Akka, приведены на рис. 4а. Результаты с использованием только GPU, полученные на системе «Ломоносов», приведены на рис. 4б.

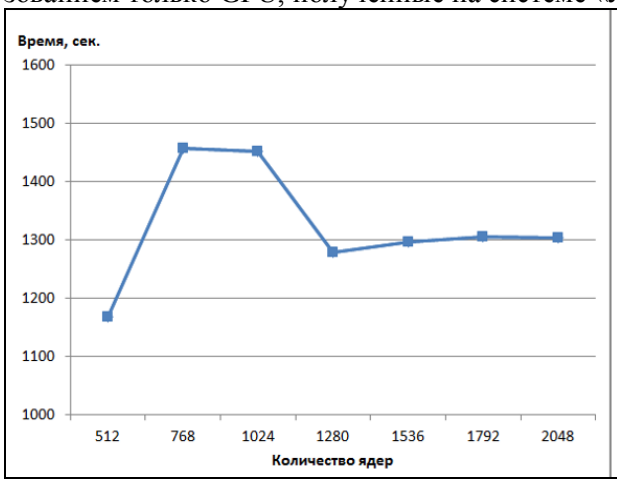


Рис. 4а. Слабая масштабируемость CPU-версии на системе Akka

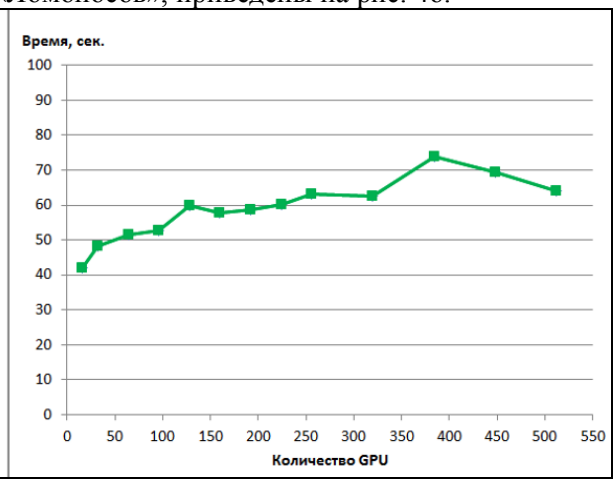


Рис. 4б. Слабая масштабируемость GPU-версии на системе «Ломоносов»

5. Заключение

В работе получены следующие результаты:

1. Разработана параллельная версия программного обеспечения для численного моделирования плазмы методом частиц в ячейках на гетерогенных кластерных системах. При решении модельных задач данная реализация показывает результаты, соответствующие теоретическим предсказаниям.
2. Проведенные вычислительные эксперименты демонстрируют масштабируемость параллельной версии для традиционных кластерных систем по крайней мере до 2048 ядер. При решении одной и той же задачи эффективность относительно запуска для 512 ядер составляет 71% и 85% на системах МВС-100К и Akka соответственно. При увеличении вычислительной трудоемкости задачи пропорционально количеству используемых ядер до 1280 время работы растет на ~11-12% и стабилизируется на этом уровне по крайней мере до 2048 ядер.
3. Разработанная версия для GPU на кластере ННГУ демонстрирует ускорение в 3-4 раза относительно 8 ядер CPU. Параллельная версия для кластера, содержащего графические процессоры, масштабируется следующим образом: при увеличении вычислительной трудоемкости задачи пропорционально количеству используемых GPU до 128 время работы растет на ~50% и стабилизируется на этом уровне по крайней мере до 320 GPU (143360 CUDA-ядер). Далее при увеличении количества используемых GPU наблюдается некоторая деградация производительности, однако при достижении 512 GPU (229376 CUDA-ядер) время работы вновь приближается к достигнутому ранее уровню (320 GPU).

Целью дальнейших исследований является увеличение производительности и масштабируемости на CPU и GPU, одновременное использование всех вычислительных устройств, а также реализация улучшенных численных схем.

Литература

1. Бэдсел Ч., Ленгдон А. Физика плазмы и численное моделирование: Пер. с англ. – М.: Энергоатомиздат, 1989. – 452 с.
2. Taflove A. Computational Electrodynamics: The Finite-Difference Time-Domain Method. – London: Artech House, 1995. – 599 P.
3. Бастраков С.И., Гоносков А.А., Донченко Р.В., Ефименко Е.С., Малышев А.С., Мееров И.Б. Исследование и поиск наиболее эффективных подходов к параллельному моделированию плазмы методом частиц в ячейках на кластерных системах // Параллельные вычислительные технологии (ПаВТ’2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2011. – С. 411-417. – URL: [http://omega.sp.susu.ac.ru/books/conference/PaVT2011/short/118.pdf].
4. Hoenig W., Schmitt F., Widera R., Burau H,, Juckeland G., Mueller M., Bussmann M. A Generic Approach for Developing Highly Scalable Particle-Mesh Codes for GPUs // SAAHPC, USA, Jul. 2010. – 15pp. – URL: [http://saahpc.ncsa.illinois.edu/10/papers/paper_10.pdf].
5. CUDA C Best Practices Guide v. 4.0. – URL: [http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf].

Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями*

В.А. Бахтин, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула

Институт прикладной математики имени М.В. Келдыша РАН

Описывается развитие алгоритмов автоматического распараллеливания Фортран-программ на кластер для использования графических ускорителей в его узлах. Результатом работы алгоритмов является текст программы на высокоуровневом языке Fortran DVMH. Основная задача алгоритмов заключается в нахождении параллельных циклов, которые могут выполняться на ускорителях, определении требуемых этим циклам данных и режимов их использования, объединении таких циклов в более крупные вычислительные регионы с целью сокращения передач данных между ускорителями и центральным процессором.

1. Введение

Программирование высокопроизводительных кластеров и других параллельных систем продолжает оставаться исключительно сложным делом, доступным узкому кругу специалистов и крайне трудоемким даже для них.

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры (GPU), еще более усложнило разработку программ. Для них требуется использовать помимо технологий MPI или SHMEM для передачи сообщений между узлами кластера еще и низкоуровневую технологию CUDA или OpenCL для программирования вычислений на ускорителе. Программировать, отлаживать, сопровождать, переносить на другие ЭВМ такие программы гораздо сложнее. Например, при переходе от GPU фирмы NVIDIA к GPU фирмы AMD придется заменить CUDA на OpenCL.

Альтернативным вариантом является использование высокоуровневых моделей и соответствующих им языков программирования [1-15] (HPF, OpenMP, DVM, DVMH, HMPP, hiCUDA, PGI APM, OpenACC, Co-Array Fortran, UPC, Titanium, Chappel, X10, Fortress). Особое место занимают модели, реализуемые посредством добавления в программы на стандартных последовательных языках спецификаций, управляющих отображением этих программ на параллельные машины. Эти спецификации, оформляемые в виде комментариев в Фортран-программах или директив компилятору (прагм) в программах на языках Си и Си++, не видны для обычных компиляторов, что значительно упрощает внедрение новых моделей параллельного программирования. Такими моделями для кластеров являются HPF и DVM, а для мультипроцессоров - OpenMP. В последние годы разработаны высокоуровневые модели и для графических процессоров (HMPP, hiCUDA, PGI APM, DVMH, OpenACC). Они имеют между собой много общего, что позволяет говорить о появлении нового подхода к программированию ускорителей типа GPU. Они позволяют программисту, также как в моделях DVM и OpenMP, полностью контролировать отображение данных и вычислений на аппаратуру.

Еще одним вариантом разработки параллельных программ, является использование высокоуровневого инструмента, который автоматически преобразовывал бы последовательную программу на стандартном языке Фортран или Си в параллельную программу для ЭВМ, в том числе с графическими процессорами или потоковыми ускорителями, как в работе [16]. Следует отметить работы по созданию систем автоматизированного распараллеливания для многоядерных кластеров [17-21] (CAPTools/Parawise, FORGE Magic/DM, BERT77, САПФОР, ДВОР), от которых можно ожидать их развития в сторону ЭВМ с графическими процессорами.

* Работа поддержана ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы» ГК № 07.514.11.4030, грантами Президента РФ МК-6772.2012.9 и НШ-4307.2012.9, программой Президиума РАН №17 и грантом РФФИ №11-01-00246.

Например, данная работа посвящена такому развитию системы САПФОР.

2. Система САПФОР

Система САПФОР состоит из следующих компонент:

1. Статический анализатор Фортран-программ, который определяет свойства последовательной программы по тексту программы;
2. База данных (БД) осуществляет передачу данных между компонентами системы;
3. Эксперты (DVM-эксперт, OpenMP-эксперт, DVM/OpenMP-эксперт, DVMH-эксперт), которые строят схемы распараллеливания, оценивают их путем прогнозирования характеристик (временных характеристик выполнения), записывают схемы и характеристики в БД;
4. Генератор по схемам из БД строит текст параллельной программы;
5. Диалоговая оболочка визуализирует содержимое БД и соотносит его с текстом программы, запускает компоненты системы, позволяет программисту (пользователю) уточнить результаты анализа.

В последнее время система развивалась в сторону поддержки современного Фортрана, создания DVMH-эксперта для кластеров с графическими ускорителями, улучшения версии диалоговой оболочки для работы с реальными приложениями.

Этап генерации текста параллельной программы получается значительно проще, если использовать в качестве промежуточного языка высокоуровневый язык, для которого уже разработаны соответствующие компиляторы. В терминах этого языка строятся схемы распараллеливания в эксперте, которые представляют собой правила преобразования текста последовательной программы.

В настоящее время известен только один высокоуровневый язык для организации вычислений в кластере с графическими ускорителями. Это язык Fortran DVMH [4], который является развитием Fortran DVM. Именно его использует DVMH-эксперт.

3. Алгоритм автоматического отображения на кластер с графическими процессорами

Работа DVMH-эксперта начинается с запуска алгоритмов DVM-эксперта [22], который осуществляет организацию параллельных вычислений (распределение данных, распределение вычислений и организацию коммуникаций) между узлами, и отбирает наилучшую схему распараллеливания. Потом DVMH-эксперт дополняет эту схему конструкциями, которые необходимы для использования графических ускорителей в узлах кластера:

- Определяет, какие DVM-циклы должны стать вычислительными регионами, что означает, что они будут выполняться на графических ускорителях (есть ограничения, например, в регионе нельзя вызывать внешние процедуры, отличные от встроенных процедур Фортрана), формирует для DVM-циклов спецификацию приватных для него переменных (этих спецификаций не было в DVM-программе);
- Объединяет регионы в более крупные, вставляет для них директивы начала и конца региона;
- Определяет нужные этим регионам данные и режим их использования;
- Вставляет во фрагменты программы, не попавшие в регионы, указания об актуализации данных и их модификации.

В результате DVMH-эксперт дополняет схему распараллеливания, полученную после работы алгоритмов DVM-эксперта, конструкциями, которые необходимы для использования графических ускорителей в узлах кластера.

Рассмотрим работу алгоритма на примере численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби (см. Рис. 1).

```
 1          PROGRAM    JAC
 2          PARAMETER    (L=2000,  ITMAX=20)
 3          REAL     A(L,L), EPS, MAXEPS, B(L,L)
 4          MAXEPS  =  0.5E - 7
 5          DO J  =  1, L
 6          DO I  =  1, L
 7              A(I,  J)  =  0.
 8              IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN
 9                  B(I,  J) = 0.
10              ELSE
11                  B(I,  J)  = ( 1. + I + J )
12              ENDIF
13          ENDDO
14          ENDDO
15
16          DO IT  =  1,  ITMAX
17              EPS  =  0.
18              DO J  =  2, L-1
19              DO I  =  2, L-1
20                  EPS = MAX ( EPS,  ABS( B( I, J) - A( I, J)))
21                  A(I, J)  =  B(I, J)
22              ENDDO
23              ENDDO
24              DO J = 2,  L-1
25              DO I = 2,  L-1
26                  B(I, J) =  (A( I-1, J ) + A( I, J-1 ) +
27       *              A( I+1, J)+ A( I, J+1 )) / 4
28              ENDDO
29              ENDDO
30              PRINT 200,  IT, EPS
31  200         FORMAT(' IT = ',I4, '   EPS = ', E14.7)
32              IF ( EPS . LT . MAXEPS ) GOTO 3
33          ENDDO
34      3   OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')
35          WRITE (3,*)   B
36          CLOSE (3)
37          END
```

Рис. 1. Программа на языке Fortran численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Алгоритмы DVM-эксперта строят схемы распараллеливания, которые отличаются вариантами распределения данных, а именно:

Вариант 0

!DVM$ DISTRIBUTE (BLOCK,BLOCK) :: a – означает распределение массива A по двум измерениям, например, при использовании 4 процессоров в конфигурации 2x2 массив A будет разрезан по каждому измерению на 2 части. Во внутреннем представлении программы все переменные обозначены малыми буквами, это объясняет их использование в директиве и отличие от переменных в тексте программы.

!DVM$ ALIGN b(i,j) WITH a(i,j) – означает точное соответствие распределения между элементами массивов A и B. Там где находится элемент A(i,j) будет расположен и B(i,j).

Вариант 1

!DVM$ DISTRIBUTE (*,BLOCK) :: a – означает распределение массива A по второму измерению, например, при использовании 4 процессоров массив A будет разрезан по столбцам на 4 части.

!DVM$ ALIGN b(i,j) WITH a(i,j)

Вариант 2

!DVM$ DISTRIBUTE (BLOCK,*) :: a

!DVM$ ALIGN b(i,j) WITH a(i,j)

Вариант 3

Пусто – означает, что не надо распределять массивы A и B, директивы распределения данных не вставлять

Затем для каждого варианта распределения данных строится вариант распределения вычислений и организация коммуникаций.

Для вариантов 0, 1 и 2

Цикл на строках с номерами 5 и 6 – "!DVM$ PARALLEL (j,i) ON a(i,j)" – означает, что виток с индексами (j,i) будет выполняться на том процессоре, на котором имеется элемент a(i,j)

Цикл на строке 16 не будет распараллелен, потому что в нем есть оператор печати (ввода-вывода)

Цикл на строках с номерами 18 и 19 – "!DVM$ PARALLEL (j,i) ON a(i,j),
DVM$ REDUCTION (MAX(eps))" – означает, что цикл будет распараллелен, а после него будет выполнена операция редукции для переменной eps, копии которой имеются на всех процессорах.

Цикл на строках с номерами 24 и 25 – "!DVM$ PARALLEL (j,i) ON b(i,j),
DVM$ SHADOW_RENEW (a(1:1,1:1))" – означает, что цикл будет распараллелен, а перед его выполнением будет произведена операция обмена между процессорами теневыми гранями массива A. На каждом процессоре будет заведена память не только под локальную часть массива, но и под теневую грань (расширение локальной части массива), в которую будут копироваться значения этого массива с соседних процессоров.

Для варианта 3

Никаких директив не будет вставлено, потому что все массивы размножены по процессорам.

Таким образом, построено 4 схемы. Эксперт оценивает их. При обмене теневыми гранями в случае одномерного распределения данных (вариант 1 и 2), например, для 100 процессоров требуется 198 (99*2) пересылок по 2000 элементов массива, а при двумерном (вариант 0) для конфигурации процессоров 10x10 – 360 пересылок по 200 элементов массива. Для оценки времени передачи данных в системе САПФОР используются параметры, которые соответствуют латентности и времени передачи каждого байта данных. Для нашего примера будем использовать, например, такие параметры Tstart = 7 мкс, Tbyte = 0.004 мкс, и будем считать, что элемент массива занимает 4 байта. Тогда для одномерного случая потребуется 11880 (198*15*4) мкс, а для двумерного 11232 (360*7.8*4) мкс, поэтому лучшей схемой по этим оценкам является схема с номером 0.

Затем DVMH-эксперт строит граф управления с крупными вершинами. Вершина соответствует нескольким операторам программы, и будем называть их блоком. Для него определяются в процессе работы алгоритма все данные, которые изменяются или читаются в блоке.

Для выбранного примера граф управления будет построен в следующем виде (см. Рис. 2). A(*,*) означает использование всего массива.

Номер блока	Строки операторов блока	Пишет данные	Читает данные	Следующий блок	Предыдущие блоки
1	4	MAXEPS		2	
2	5-14	A(*,*), B(*,*), I, J	I, J	3	1
3	16, 17	IT, EPS		4	1-6
4	18-23	A(*,*), EPS, I, J	A(*,*), B(*,*), EPS, I, J	5	1-6
5	24-29	B(*,*), I, J	A(*,*), I, J	6	1-6
6	30-33		IT, EPS, MAXEPS	3, 7	1-6
7	34-36		B(*,*)		1-6

Рис. 2. Граф управления для программы численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Определяются циклы, которые входят в состав DVM-циклов: для варианта с номером 0 – это циклы на строках 5 и 6, 18 и 19, 24 и 25. Производится проверка возможности их оформле-

ния в виде вычислительных регионов. В данном примере они удовлетворяют ограничениям для регионов. Получается три региона. Приватных для DVM-цикла переменных в них нет.

Затем производится попытка объединить регионы. Цикл на строке 18 и цикл на строке 24 вложены непосредственно в один цикл, который находится на строке 16. Поэтому производится объединение этих регионов. В итоге получается два региона (см. Рис. 3)

Номер региона	Номера блоков графа управления
1	2
2	4, 5

Рис. 3. Набор вычислительных регионов для программы численного решения краевой задачи для уравнения Лапласа в прямоугольной области итерационным методом Якоби

Для них формируется директива языка Fortran DVMH “!DVM$ region”.

Далее происходит определение используемых в регионе данных и типов их использования (входные, выходные, локальные). Рассматривается каждый регион в отдельности.

Регион 1 читает переменные I и J, но они являются итерационными переменными DVM-цикла, работа с ними обеспечивается системой поддержки выполнения программ на графическом ускорителе, поэтому для них не надо указывать тип их использования в регионе. Получается, что нет данных, которые надо пометить как входные для региона. Регион 1 пишет данные A(*,*), B(*,*), I, J. Переменные I и J являются итерационными переменными, а для массивов A и B проверяется, будут ли они использованы далее в программе. Проверка показывает, что будут использоваться, и они помечаются как выходные, иначе были бы локальными. Локальных данных в этом регионе нет.

Регион 2 читает и пишет переменные A(*,*), B(*,*), EPS, I, J. Переменные I и J являются итерационными переменными. Для переменной EPS определяется то, что перед регионом (согласно графу управления) она присваивается, а также используется после региона. То же самое и для массивов A и B, они используются на следующем витке итерационного цикла на строке 16. Поэтому переменные A, B, EPS будут помечены как входные и как выходные для региона, такой тип обозначается как inout. Локальных данных в этом регионе тоже нет.

После этого анализируются операторы программы, которые не включены в регионы, для того чтобы поставить перед ними необходимые операторы актуализации значений переменных. Дело в том, что выходные данные региона не всегда копируются из укорителя на центральный процессор после выполнения региона. Они хранятся на ускорителе с целью повторного их использования в другом регионе.

Операторы на строках 4 и 17 не читают данные, поэтому для них не надо производить подобную актуализацию. Оператор на строке 30 читает переменные IT и EPS, но переменная IT не менялась в предыдущих регионах, поэтому можно использовать ее значение с центрального процессора, копирования дополнительные не нужны. А для переменной EPS нужно копирование. Поэтому перед этим оператором вставляется директива языка Fortran DVMH “!DVM$ get_actual(eps)”. Следующий оператор на строке 32 читает переменные EPS и MAXEPS. Переменная EPS была обновлена оператором на строке 30, и между этими операторами не было регионов, которые выполнялись бы на графическом ускорителе. Поэтому дополнительное копирование для нее не нужно. А переменная MAXEPS не менялась в предыдущих регионах. Оператор на строке 35 читает массив B, который менялся в предыдущих регионах и не был скопирован на центральный процессор. Для него надо вставить директиву “!DVM$ get_actual(b)”.

В результате, получен текст программы на языке Fortran DVMH, который можно скомпилировать и выполнить на кластере с графическими процессорами. Полученный автоматически текст программы близок к тексту программы, написанному вручную, и приведенному в статье [4].

В алгоритме есть еще следующие особенности, которые на данном примере не проявились.

Во-первых, если для операторов цикла, который целиком выполняется на центральном процессоре, надо осуществлять копирования элементов с графического ускорителя, то они выполняются перед циклом, а не в нем. Это позволяет объединить операции копирования к отдельным элементам массива в операции копирования всего массива, и избежать заведомо повторных проверок, связанных с актуализацией одних и тех же данных.

Во-вторых, для массивов хранится множество элементов, которые читаются или изменяются в том или ином фрагменте программы. Несколько подряд идущих элементов объединяются в диапазоны элементов массива. А множество элементов хранится в виде списка, состоящего или из отдельных элементов массива, или из диапазонов. Задание множеств через такие списки позволяет легко находить их пересечения для разных фрагментов программы. Это требуется для определения, надо ли перед чтением элементов массивов на центральном процессоре копировать их из памяти ускорителей в память центрального процессора. Например, для случая, когда массив менялся целиком на графическом ускорителе, потом было несколько присваиваний на центральном процессоре в крайние элементы этого массива, а потом операция чтения не крайних элементов массива, надо скопировать их с графического процессора.

4. Заключение

Появление кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры, поставило вопрос об эффективном автоматическом распараллеливании последовательных программ для них.

Проведенные исследования показали, что такое отображение возможно, если при написании последовательных программ придерживаться определенной дисциплины и предоставить программисту средства для спецификации свойств его программы, недоступных для статического анализа.

Разработанные алгоритмы отображения во многом опираются на разработанные ранее алгоритмы отображения Фортран-программ на многоядерный кластер, существенно используют близость языков Fortran DVM и Fortran DVMH, и возможности компилятора Fortran DVMH.

В настоящее время ведется апробация системы САПФОР и компилятора Fortran DVMH на представительных тестовых программах (среди них тесты NAS) и других реальных приложениях.

Литература

1. High Performance Fortran Forum. High Performance Fortran Language Specification, version 2.0, January 1997. URL: http://hpff.rice.edu/versions/hpf2/index.htm (дата обращения 30.10.2011)
2. OpenMP Application Program Interface. Version 3.1. July 2011. URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf (дата обращения 30.10.2011)
3. Описание языка Fortran-DVM/OpenMP. Версия 3.0, Октябрь, 2009. URL: http://www.keldysh.ru/dvm/dvmhtm1107/rus/usr/fdvm/fdvmLDr.html (дата обращения 30.10.2011)
4. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсное будущее”, Новороссийск, сентябрь 2011 – М.: Изд-во МГУ, 2011, С. 310-315.
5. Dolbeau R., Bihan S., Bodin F. HMPP: A hybrid multicore parallel programming environment.
6. Han T., Abdelrahman T. hiCUDA: A high-level directive-based language for gpu programming. // Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units, Mar. 2009, P. 52–61.
7. The Portland Group, Inc. The PGI Fortran and C Accelerator Programming Model, March. 2010. URL: htpp://www.pgroup.com/accelerate (дата обращения 30.10.2011).
8. OpenACC. URL: http://www.openacc-standard.org (дата обращения 14.02.2012).
9. Mellor-Crummey J., Adhianto L., Jin G., Scherer W. A New Vision for Coarray Fortran. // The Third Conference on Partitioned Global Address Space Programming Models. Ashburn, VA, October 2009.

10. UPC Consortium. UPC Language Specifications, v1.2 // Lawrence Berkeley National Lab Tech Report LBNL-59208, 2005.
11. Hilfinger P., Bonachea D., Datta K., Gay D., Graham S., Liblit B., Pike G., Su J., Yelick K. Titanium Language Reference Manual. Version 2.20. // U.C. Berkeley Tech Report, UCB/EECS-2005-15.1, August, 2006.
12. Chamberlain B., Callahan D., Zima H. Parallel Programmability and the Chapel Language // International Journal of High Performance Computing Applications, August 2007, 21(3), P. 291-312. URL: http://hpc.sagepub.com/content/21/3/291.abstract (дата обращения 30.10.2011)
13. Milthorpe J., Ganesh V., Rendell A., Grove D. X10 as a Parallel Language for Scientific Computation: Practice and Experience. // IEEE International Parallel and Distributed Processing Symposium, May 2011. URL: http://cs.anu.edu.au/%7EJosh.Milthorpe/publications/Milthorpe2011_IPDPS.pdf (дата обращения 30.10.2011)
14. Cunningham D., Bordawekar R., Saraswat V. GPU Programming in a High Level Language Compiling X10 to CUDA. // ACM SIGPLAN 2011 X10 Workshop, June 2011. URL: http://x10.svn.sourceforge.net/viewvc/x10/external/X10%20Workshop%20%282011%29/papers/cuda.pdf (дата обращения 30.10.2011)
15. Allen E., Chase D., Flood C., Luchangco V., Maessen JW., Ryu S., Steele G.Jr. Project Fortress: A Multicore Language for Multicore Processors // Linux Magazine, September 2007. URL: http://labs.oracle.com/projects/plrg/Publications/linuxMagazine.pdf (дата обращения 30.10.2011)
16. Климов А.В., Окунев А.С. Автоматическое распараллеливание последовательных программ для гибридной системы с ускорителем на основе потока данных.
17. Система ParaWise: сайт.- URL: http://www.parallelsp.com (дата обращения 24.10.2011).
18. Applied Parallel Research. FORGE Magic/DM. URL: http://wotug.ukc.ac.uk/parallel/vendors/apr/ProductInfo/dpf_datasheet.txt (дата обращения 24.10.2011).
19. BERT 77: Automatic and Efficient Parallelizer for FORTRAN. сайт.- URL: http://www.basement-supercomputing.com/content/view/24/50/ (дата обращения 24.10.2011).
20. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсное будущее”, Новороссийск, сентябрь 2011 – М.: Изд-во МГУ, 2011, С. 67-70.
21. Открытая распараллеливающая система: сайт.- URL: http://www.ops.rsu.ru (дата обращения 24.10.2011).
22. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. // Вестник Нижегородского университета им. Н.И. Лобачевского, 2009, № 2, С. 128–134.

Реализация эффективных параллельных вычислений при моделировании больших задач физики плазмы методом частиц в ячейках*

Е.А. Берендеев[1], А.А. Ефимова[2]

Новосибирский государственный университет[1],
Институт вычислительной математики и математической геофизики СО РАН[2]

Рассмотрены основные варианты параллельной реализации метода частиц в ячейках для решения больших задач физики плазмы. Проведено сравнение эффективности распараллеливания на примере задачи об аномальной теплопроводности. Задача рассматривается в двумерной постановке. В работе предложен эффективный алгоритм распараллеливания, позволяющий достигнуть высокой масштабируемости вычислений.

1. Введение

Актуальность работы связана с необходимостью повышения эффективности использования суперЭВМ для решения ресурсоёмких задач физики плазмы. Для этого необходимо выполнить эффективную параллельную реализацию используемых алгоритмов. При этом для достижения масштабируемости вычислений на большое число процессоров необходимо учитывать особенности самого алгоритма. Многие простые и хорошо масштабируемые на нескольких десятках процессоров параллельные реализации могут оказаться неэффективными, если исходный последовательный алгоритм использует такие особенности архитектуры компьютера, как кэширование, SSE-инструкции, доступ к данным. В этом случае необходимо либо модифицировать исходный алгоритм, либо внести изменения в параллельную реализацию программы.

В работе рассмотрены различные варианты распараллеливания алгоритма для следующей физической задачи. В результате релаксации в плазме мощного электронного пучка в многопробочной магнитной ловушке ГОЛ-3 (ИЯФ СО РАН) наблюдается понижение электронной теплопроводности на 2-3 порядка по сравнению с классическим значением [1]. На данный момент точное теоретическое описание этого явления отсутствует, поэтому возникает необходимость в численном моделировании. Моделирование проводится на основе метода частиц в ячейках, как наиболее подходящего для решения неравновесных задач физики плазмы.

Необходимость применения суперкомпьютерных вычислений обусловлена тем, что требуется, во-первых, достаточно подробная сетка для воспроизведения резонансного взаимодействия релятивистского электронного пучка с плазмой, и, во-вторых, большое количество модельных частиц, чтобы промоделировать возникающую в дальнейшем неустойчивость.

2. Описание модели

Рассматриваемый физический процесс описывается кинетическим уравнением Власова для ионов и электронов

$$\frac{\partial f_{i,e}}{\partial t} + \vec{v}\frac{\partial f_{i,e}}{\partial \vec{r}} + \vec{F}_{i,e}\frac{\partial f_{i,e}}{\partial \vec{p}} = 0\,, \quad \vec{F}_{i,e} = q_{i,e}\left(\vec{E} + \frac{1}{c}\left[\vec{v},\vec{B}\right]\right),$$

* Работа выполнена при поддержке гранта РФФИ № 11-01-00249-а

и системой уравнений Максвелла

$$rot\vec{B} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t},$$

$$rot\vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t},$$

$$div\vec{E} = 4\pi\rho,$$

$$div\vec{B} = 0.$$

Все уравнения приведены в общепринятых обозначениях. Далее все уравнения будут приводиться в безразмерном виде. Для обезразмеривания используются следующие базовые величины:

- Характерная плотность плазмы $n_0 = 10^{14}$ см$^{-3}$;
- Характерная скорость – скорость света v_0=c = 3×10^{10} см/с;
- Характерное время $t_0 = \frac{1}{\omega_{pe}} = \sqrt{\frac{m_e}{4\pi n_e e^2}} = 1.79\times10^{-12}$ с. – величина, обратная к электронной плазменной частоте

Модель построена на основе метода частиц в ячейках. Характеристики уравнения Власова описывают движение модельных частиц. Уравнения этих характеристик имеют вид

для электронов $$\frac{d\vec{p}}{dt} = -(\vec{E} + [\vec{v}, \vec{B}]),$$

для ионов $$\frac{d\vec{p}}{dt} = \kappa(\vec{E} + [\vec{v}, \vec{B}]),$$

$$\frac{d\vec{r}}{dt} = \vec{v}.$$

$\kappa = \frac{m_e}{m_i}$ – отношение масс электрона и иона; $\vec{p} = \gamma\vec{v}$; $\gamma = (1-v^2)^{-1/2}$

Для решения уравнений движения используется схема с перешагиванием.

для электронов $$\frac{p_i^{m+1/2} - p_i^{m-1/2}}{\tau} = -(E_i^m + [\frac{v_i^{m+1/2} + v_i^{m-1/2}}{2}, B_i^m]),$$

для ионов $$\frac{p_i^{m+1/2} - p_i^{m-1/2}}{\tau} = \kappa(E_i^m + [\frac{v_i^{m+1/2} + v_i^{m-1/2}}{2}, B_i^m]),$$

$$\frac{r_i^{m+1} - r_i^m}{\tau} = v_i^{m+1/2}.$$

Здесь τ – шаг по времени; верхний индекс указывает на момент времени, в который вычисляется искомая функция; нижний индекс указывает на номер частицы, для которой производится вычисление.

Для нахождения электрических и магнитных полей используется схема, в которой поля определяются из разностных аналогов законов Фарадея и Ампера [2]. Эта схема имеет второй порядок аппроксимации по пространству и по времени. Плотности заряда и тока в узлах вычисляются при помощи точного учёта потоков плазмы через границы ячеек [3]: в этом случае разностный аналог закона Ампера выполняется автоматически. Каждая частица вносит вклад в плотность тока в ближайших к ней узлах сетки, затем проводится суммирование вкладов по

всем частицам.

3. Постановка задачи

Задача рассматривается в двумерной постановке. В области, имеющей форму прямоугольника ($0<x<L_x, 0<y<L_y$), находится плазма, состоящая из ионов и электронов. Дополнительно в области присутствуют электроны пучка (предполагается, что пучок уже вошёл в расчётную область). Условия на границах области берутся периодическими.

В начальный момент частицы распределены по области равномерно, задаётся плотность плазмы и температура электронов и ионов. Модельные частицы пучка отличаются от модельных электронов плазмы тем, что имеют меньшую массу (отношение их масс равно отношению плотности плазмы и плотности пучка), а также движутся в одном направлении (направление X) с одинаковой скоростью (определяется кинетической энергией направленного движения). Таким образом, исходными параметрами задачи являются: плотность и температура электронов плазмы, отношение плотности электронов плазмы к плотности электронов пучка.

- Плотность электронов плазмы $n_0=10^{14}$ см$^{-3}$;
- Температура электронов плазмы $T_0=500$ эВ.
- Отношение плотности электронов пучка к плотности электронов плазмы $\alpha = 2\times10^{-3}$
- Скорость электронов пучка ε=1 МэВ

4. Параллельная реализация

Возможно несколько вариантов параллельной реализации метода частиц в ячейках. Поскольку траектории модельных частиц вычисляются независимо друг от друга, проще всего распределить все частицы поровну между процессорами, независимо от их координаты. В этом случае каждый процессор будет решать систему уравнений Максвелла во всей области. Поскольку время расчёта значений электромагнитных полей существенно меньше времени расчёта траекторий частиц (в каждой ячейке сетки находится до 1000 частиц каждого сорта), такой вариант распараллеливания кажется очень удачным: даже для сетки 1000x1000 необходимо обменяться на каждом шаге только 3×10^6 значениями плотности тока, что составляет около 20 Мб. Эффективность такого распараллеливания достаточна высока.

Второй вариант: провести декомпозицию расчётной области по одному направлению (например, перпендикулярному направлению движения пучка). При этом с каждой подобластью можно связать группу процессоров и разделить частицы в подобласти между всеми процессорами группы. Каждая группа решает уравнения Максвелла только в своей подобласти. В этом случае происходит обмен граничными значениями полей между группами, также группы должны обмениваться частицами, перелетевшими в соответствующую подобласть. Внутри группы происходит обмен значениями плотности тока (как и в первом варианте распараллеливания). Такой вариант распараллеливания применяется, например, при решении трёхмерных задач (в трёхмерном случае первый вариант оказывается неэффективным, поскольку пересылать приходится, например, для сетки 1000x1000x1000 уже 20000 Мб). При этом существуют некоторые ограничения, связанные с числом групп: во-первых, оно не превосходит числа узлов по направлению, в котором ведётся декомпозиция, во-вторых, при большом числе групп число частиц в каждой подобласти может существенно различаться, делая нагрузку на процессоры неравномерной.

5. Вычислительный эксперимент

Все расчёты проводились на суперкомпьютере Новосибирского государственного университета. Использовались узлы HP BL2x220 G6, каждый из которых содержит: два 4-ядерных процессора Intel Xeon E5540 с тактовой частотой 2530 МГц (L2 cache 1Мб, L3 cache 8 МБ) и 16041 МБ ОЗУ. В расчётах задействовано до 128 процессорных ядер одновременно.

Для определения оптимального распределения процессоров по области, нами был проведён

ряд вычислительных экспериментов. Оказалось, что разбиение области на подобласти по 128x128 и менее ячеек (т.е. всего в подобласти 16384 ячейки) даёт ускорение почти в два раза по сравнению с первым вариантом параллельной реализации. Это связано с тем, что в кэш процессора может быть загружена вся сетка, и значения электромагнитных полей на каждом временном шаге для частиц в одной ячейке считывается из ОЗУ только один раз.

Сравнение параллельных реализаций проводилось при следующих параметрах:

По направлению Y выбиралась сетка в 1024 узла, в каждой ячейке 1000 частиц каждого сорта. Всего использовалось 128 процессоров. На рис.1, рис.2 приведено среднее время расчёта (в сек.) движения частиц (рис.1) за одну итерацию и среднее время расчёта одной итерации (рис.2) в зависимости от числа подобластей (1 соответствует первому варианту, 2 – подобласть делится на две группы процессоров по 64 процессора в каждой и т.д.) для различных вариантов сетки по направлению X (32,64,128,256)

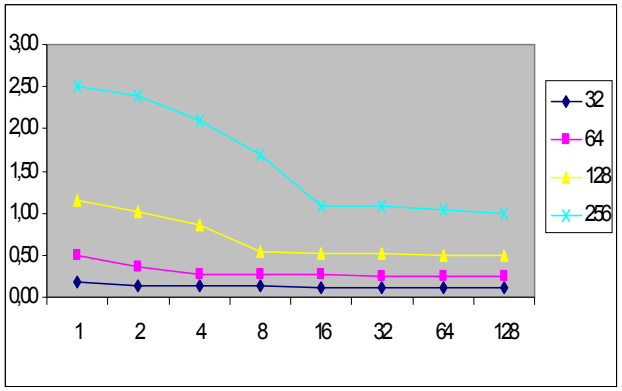


Рис. 1. Среднее время расчёта (в сек.) движения частиц за одну итерацию для различных вариантов разбиения области.

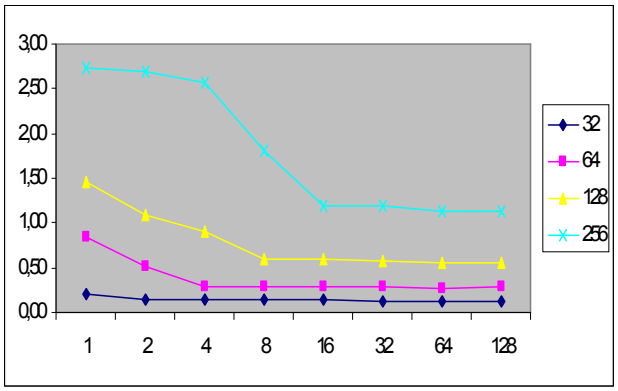


Рис. 2. Среднее время расчёта (в сек.) одной итерации для различных вариантов разбиения области.

Из рис.1 и рис.2 видно, что оптимальным является разбиение области на подобласти по 16384 ячейки, что позволяет в 2 раза ускорить вычисления.

При этом число процессоров в каждой подобласти можно увеличивать до 128, не уменьшая эффективности. На рис.3 приведено время расчёта (в сек.) одной итерации для сетки 64x256, 1000 частиц в ячейке в зависимости от числа процессоров.

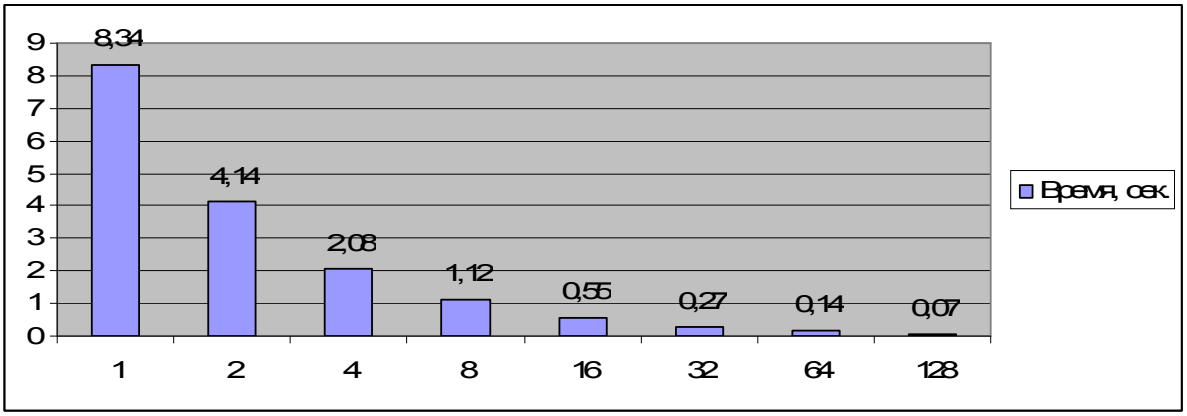


Рис. 3. Время расчёта (в сек.) одной итерации для сетки 64x256, 1000 частиц в ячейке в зависимости от числа процессоров.

Из рис.3 видно, что при равномерном распределении частиц области между процессорами, ускорение возрастает практически пропорционально числу процессоров (ускорение в 30,89, 59,57, 119,14 раза для 32, 64 и 128 процессоров соответственно). Однако такая хорошая масштабируемость получилось только потому, что мы рассмотрели лишь одну область. Если же частицы начинают перемещаться из одной подобласти в другую, нагрузка становится неравномерной.

Для определения того, насколько равномерно нагрузка распределяется между процессорами в случае, когда частицы перемещаются между процессорами хаотически, была рассмотрена следующая задача. Сетка 64x1024, 1000 частиц в ячейках, область разделена вдоль направления Y на 4 подобласти. В каждой подобласти используется 32 процессора. На рис.4 представлено среднее по процессорам время счёта одной итерации для 1000 итераций.

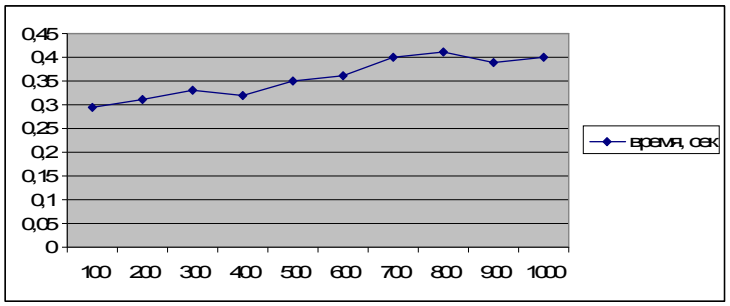


Рис. 4. Среднее по процессорам время счёта одной итерации на каждом шаге для 1000 временных шагов.

Из рис.4 видно, что время выполнения одной итерации может меняться с течением времени, однако расчёт каждого следующего шага не существенно отличается от времени расчёта текущего шага. Это связано с тем, что частицы за одну итерацию могут перемещаться только в соседние ячейки.

Таким образом, необходимо провести планирование распределения частиц как внутри подобласти, так и между подобластями.

Одним из способов планирования распределения частиц внутри подобласти является использование технологии OpenMP, в которой разделение нагрузки между потоками происходит равномерно.

Проводилось тестирование гибридной реализации MPI+OpenMP для следующей задачи.

Область 128x256, 1000 частиц в ячейках, делится на 2 подобласти. Каждая подобласть рассчитывается одним MPI-процессом, который вызывает несколько OpenMP-потоков (от 2-х до 8-ми).

Время расчёта только MPI-программы на одну итерацию – 7,95 сек, с использованием 2,4,8 OpenMP-потоков – 5,12, 4,2, 3,96 сек соответственно. При этом использование 4-х MPI-процессов без OpenMP даёт время расчёта одной итерации 3,79 сек. Таким образом,

оказывается выгоднее использовать только MPI.

Планирование по распределению частиц проводилось А.Н.Андриановым и К.Н.Ефимкиным (ИПМ им. М.В.Келдыша) [4]. Ими было предложено равномерное распределение частиц по процессорам области исходя из общего числа частиц в области на каждом шаге. Это увеличит объём пересылок частиц между процессорами, но поскольку за одну итерацию между подобластями перемещается достаточно малое число частиц, этот подход кажется перспективным.

6. Заключение

В работе нами были описаны основные параллельные реализации двумерной модели, описывающей взаимодействие релятивистского электронного пучка с плазмой. Показано, что использование декомпозиции области на группы по 128x128 ячеек даёт ускорение в 2 раза по сравнению с другими реализациями. При большом размере сетки рекомендуется использование планирования распределения частиц. Как мы показали, в этом случае внутри каждой подобласти можно эффективно использовать до 128 процессоров. Таким образом, появляются широкие возможности для масштабирования задачи на десятки тысяч процессоров.

Литература

1. Астрелин В.Т., Бурдаков А.В., Поступаев В.В. Подавление теплопроводности и генерация ионно-звуковых волн при нагреве плазмы электронным пучком // Физика плазмы, 1998, том 24, № 5, с. 45-462.
2. Вшивков В.А., Вшивков К.В., Дудникова Г.И. Алгоритмы решения задачи взаимодействия лазерного импульса с плазмой // Вычислительные технологии, том 6, № 2, с. 47-63, 2001.
3. Villasenor J., Buneman O. Rigorous charge conservation for local electromagnetic field solver // Computer Phys. Comm. 1992. Vol. 69. P. 306-316.
4. Андрианов А.Н., Ефимкин К.Н. Подход к параллельной реализации метода частиц в ячейках // Препринт ИПМ им. М.В.Келдыша №9 за 2009 г., Москва.

Эффективность построения области достижимости летательного аппарата на графических процессорных устройствах

А.В. Быстров, А.П. Карпенко

МГТУ имени Н.Э. Баумана

Рассматривается задача построения области достижимости динамической системы и подход к решению этой задачи на основе метода «мультифиниша». Приводится схема распараллеливания задачи. Исследуется ускорение, полученное при расчете на графическом процессорном устройстве в случае наличия в области достижимости системы статических и динамических препятствий. Приводятся результаты параллельного решения задачи для высокоманевренного летательного аппарата.

1. Введение

Во многих приложениях, в которых модели исследуемых объектов представляют в виде обыкновенных дифференциальных уравнений (ОДУ), возникает задача построения области достижимости соответствующей динамической системы. В данной работе эта задача рассматривается в контексте проблемы обеспечения траекторной безопасности высокоманевренного летального аппарата (ВМЛА) [1].

Важной особенностью задачи построения области достижимости ВМЛА является то, что ее приходится решать в режиме реального времени. При этом основные вычислительные затраты обусловлены затратами на интегрирование модельной системы ОДУ. Таким образом, задача быстрого построения области достижимости ВМЛА является актуальной.

Параллельные вычисления при численном построении некоторой аппроксимации области достижимости возможны на основе метода мультифиниша и нейросетевых методов. В работе использован метод мультифиниша, основная идея которого состоит в многократном численном интегрировании модельной системы ОДУ при различных допустимых управлениях [2]. Метод хорошо распараллеливается, как для MIMD, так и для SIMD многопроцессорных вычислительных систем [3, 4].

Работа продолжает исследования, начатые в работах [3, 4], и посвящена исследованию эффективности параллельного построения области достижимости на графических процессорных устройствах (ГПУ). Новым в работе является решение задачи в условиях наличия в области достижимости ЛА статических и динамических препятствий.

2. Постановка задачи

Рассмотрим динамическую систему

$$\dot{X} = F(t, X, U), \quad X(0) = X^0, \tag{1}$$

где $X = X(t) = (x_1(t), x_2(t), ..., x_n(t))^T$ – n-мерный вектор фазовых переменных системы, $F = F(t, X, U) = (f_1(t, X, U), f_2(t, X, U), ..., f_2(t, X, U))^T$ – n-мерная вектор-функция, $U = U(t) = (u_1(t), u_2(t), ...u_m(t))^T$ – m-мерный вектор управлений, $X^0 = (x_1^0, x_2^0, ..., x_n^0)^T$ – n-мерный вектор начальных условий, $t \in [0; T]$. На вектор фазовых переменных X и вектор управления U наложены ограничения

$$X \in D_X, U \in D_U \subset L_U[0; T], \tag{2}$$

где $L_U[0; T]$ – некоторое пространство m-мерных функций, определенных на интервале $[0; T]$, например, пространство функций, интегрируемых с квадратом на этом интервале.

Среди фазовых переменных $x_1, x_2, ...x_n$ выделим $\upsilon \le n$ переменных. Не ограничивая общности, положим, что эти переменные образуют ν-мерный вектор $Y = Y(t) = (x_1(t), x_2(t), ...x_\upsilon(t))^T$. Областью достижимости $D_Y = D_Y(T, X^0)$ системы (1) назовем множество всех возможных значений вектора $Y(t)$, которые достигаются на решениях системы (1) при начальных условиях X^0 и ограничениях (2).

3. Схема распараллеливания вычислений

Положим, что одну из границ множества достижимости формируют управления класса, в которых все компоненты вектора управления, кроме одной, постоянны, а компонента $u_1(t)$, например, имеет одну точку переключения:

$$u_1(t, t^S) = \begin{cases} +1, t \in [0; t^S], \\ -1, t \in (t^S; T], \end{cases} \quad .$$

$$u_2(t) = const_1, ..., u_m(t) = const_m$$

Здесь $t^S \in [0; T]$ - момент времени, когда происходит переключение управления $u_1(t)$.

Покроем интервал $[0; T]$ равномерной сеткой с шагом $\Delta t^S = \frac{T}{M}$ и узлами t_i^S, $i \in [1 : M]$. Положим, что шаг Δt^S кратен шагу Δt, так что $\Delta t^S = q\Delta t$, где $q = \frac{K}{M} \ge 1$ - целое число. Примем, что $U_i = (u_1(t, t_i^S), u_2, ..., u_m)^T$.

При равномерной декомпозиции точек переключения схема распараллеливания имеет вид, представленный на рис. 1, где P_j - *j*-й процессор используемой параллельной вычислительной системе, *N* – число процессоров; $j \in [1 : N]$.

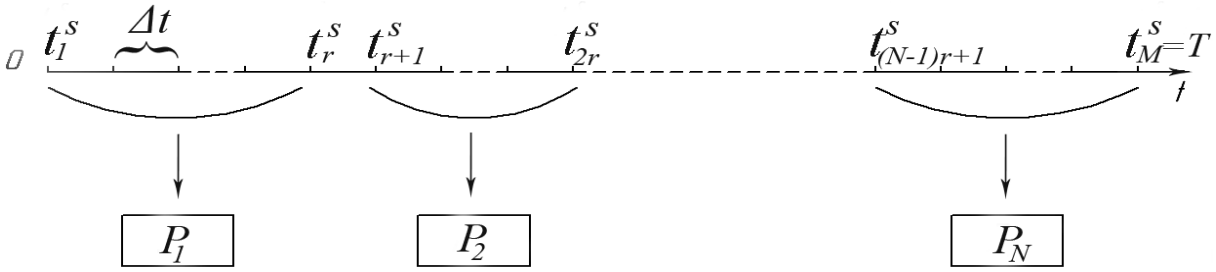


Рис. 1. Схема распараллеливания

В соответствии с этой схемой на процессоре P_1 выполняется интегрирование системы ОДУ (1) при управлениях $U_1, ..., U_r$, на процессоре P_2 - при управлениях $U_{r+1}, ..., U_{2r}$ и т.д. до процессора P_N, который выполняет интегрирование при управлениях $U_{(N-1)r+1}, ..., U_M$; *r* – ближайшее целое большее M/N.

Вычисления на процессоре $P_i, i \in [1 : N]$ организуем следующим образом.

Шаг 0 (этап «разгона»). Исходя из начальных условий X^0, выполняем интегрирование системы (1) при управлении $(1, u_2, u_3, ..., u_m)$ от начального момента до момента времени t_{ir-1}^S и сохраняем в памяти ЭВМ значения компонентов векторов $X(t_{(i-1)r+1}^S) = X_{(i-1)r+1}^0$ $X(t_{(i-1)r+2}^S) = X_{(i-1)r+2}^0, \ldots, X(t_{ir-1}^S) = X_{ir-1}^0$.

Шаг 1. Исходя из начальных условий $X^0_{(i-1)r+1}$, выполняем интегрирование той же системы при управлении $(-1,u_2,u_3,...,u_m)$ от момента $t^S_{(i-1)r+1}$ до момента времени *T*.

Шаг 2. Исходя из начальных условий $X^0_{(i-1)r+2}$, выполняем интегрирование при управлении $(-1,u_2,u_3,...,u_m)$ от момента времени $t^S_{(i-1)r+2}$ до момента времени *T*.

…

Шаг r. Исходя из начальных условий X^0_{ir-1}, выполняем интегрирование при управлении $(-1,u_2,u_3,...,u_m)$ от момента времени t^S_{ir-1} до момента времени *T*.

Варианты оптимизации данной схеме рассмотрены в работе [2].

4 Математическая модель ВМЛА

Рассмотрим ВМЛА, движение центра масс которого в нормальной земной системе координат *Oxyz* описываются системой нелинейных дифференциальных уравнений

$$\begin{cases} \dot{V} = g\cdot(n_T - \sin\Theta), \\ \dot{\Theta} = \dfrac{g}{V}(n\cdot\cos\gamma_c - \cos\Theta), \\ \dot{\Psi} = \dfrac{-g\cdot n\cdot\sin\gamma_c}{V\cdot\cos\Theta}, \\ \dot{X} = V\cdot\cos\Theta\cdot\cos\Psi, \\ \dot{Y} = V\cdot\sin\Theta, \\ \dot{Z} = -V\cdot\cos\Theta\cdot\sin\Psi \end{cases} \tag{3}$$

где V – скорость ВМЛА, Θ – угол наклона траектории, Ψ – угол поворота траектории, Y – высота ВМЛА, n_T – тангенциальная перегрузка, n – нормальная перегрузка, γ_c – скоростной угол крена; g – ускорение свободного падения [1]. Нормированный вектор управления ВМЛА имеет вид

$$u = (u_1, u_1, \gamma_c),\ |u_1| \le 1;\quad |u_2| \le 1;\quad |\gamma_c| \le \pi,$$

где $n^{\max}\cdot u_1 = n$, $n^{\max}\cdot u_2 = n_T$.

В работе [1] получена следующая структура управлений, приводящих ВМЛА на дальнюю, ближнюю и боковую границы области достижимости, соответственно:

$$u_д = \begin{bmatrix} u_1 = \pm 1 \\ u_2 = 1 \end{bmatrix}_{t\in[0,t^-]} \cup \begin{bmatrix} u_1 = 0 \\ u_2 = 1 \end{bmatrix}_{t\in[t^-,T]};$$

$$u_б = \begin{bmatrix} u_1 = \mp 1 \\ u_2 = -1 \end{bmatrix}_{t\in[0,t_1^+]} \cup \begin{bmatrix} u_1 = \pm 1 \\ u_2 = -1 \end{bmatrix}_{t\in[t_1^+,T]};$$

$$u_д = \begin{bmatrix} u_1 = \pm 1 \\ u_2 = -1 \end{bmatrix}_{t\in[0,t_2^+]} \cup \begin{bmatrix} u_1 = \pm 1 \\ u_2 = 1 \end{bmatrix}_{t\in[t_2^+,T]}.$$

Здесь t^- – момент времени перехода на особый участок управления $n_{особ} = 0$, t_1^+, t_2^+ - моменты переключения знака управления нормальной перегрузкой и переключения знака управления тангенциальной перегрузкой соответственно.

5. Отображение алгоритма мультифиниша на архитектуру GPU

Покроем интервал времени $[0;T]$ сеткой Ω_u с n_u узлами, а область допустимых значений управления γ_c - сеткой Ω_γ с m_γ узлами. Нити с идентификатором (i_u, i_γ) поставим в соответствие l соседних узлов сетки Ω_u, соответствующих узлу i_γ сетки Ω_γ. Говоря более строго, для данной границы области достижимости, например, указанная нить обрабатывает управление

$$\gamma_c = i_\gamma \frac{\gamma_c^{\max} - \gamma_c^{\min}}{m_\gamma}$$

и управления u_1 или u_2, определяемые точками переключения

$$t_i = (i_u l + i)\frac{T}{n_u l}, \quad i \in [0:(l-1)].$$

Структура *GPU*-программы имеет следующий вид:

- хост-процессор копирует в глобальную память GPU исходные данные (начальная скорость и координаты летательного аппарата);
- *GPU* реализует этап разгона, т.е. интегрирование системы (3) от начального момента времени до первой точки переключения t_k, соответствующей данной нити;
- *GPU* последовательно выполняет интегрирование той же системы при управлениях, соответствующих точкам переключения t_{k+i}, $i \in [0:(l-1)]$;
- *GPU* осуществляет копирование координат полученных граничных точек на хост-процессор;
- хост-процессор записывает все результаты в файл и осуществляет визуализацию полученного множества точек.

5. Результаты исследования

5.1 Задача без препятствий

Исследование выполнено на хост-процессоре, имеющим следующие основные параметры: процессор - *Intel Core2 Duo CPU* P8600 @ 2,40 ГГц; ОЗУ – 4 ГБ. В качестве *GPU* использована видеокарта *NVIDIA GeForce* 9650*M GT*: число мультипроцессоров – 4; число ядер в мультипроцессоре – 8; общее число вычислительных ядер – 32; версия среды выполнения - *CUDA 3.20*; *Compute Capability* - 1.1; частота - 1.38 ГГц; объем глобальной памяти – 1054 Мб.

GPU-программа реализована на языке *C* с использованием технологии *CUDA*. Для визуализации границ области достижимости использована библиотека *OpenGL*. Эффективность параллельных вычислений оценивалось ускорением *S*, равным отношению времени решения задачи на хост-процессоре к времени ее решения на *GPU*.

Полученный вид области достижимости ВМЛА иллюстрирует рисунок 2. Здесь и далее, если не оговорено противное, результаты соответствует пяти секундам полета ВМЛА со скоростью 300 м/с и начальными координатами (0, 0, 1000 м).

Выполнено широкое исследование эффективности параллельных вычислений при варьировании числа точек переключения n_u, числа m_γ узлов сетки Ω_γ, а также времени полета ВМЛА *T*. Некоторые результаты исследования иллюстрирует рисунок 3.

Рисунок 3 показывает, что, как и следовало ожидать, ускорение возрастает с ростом числа точек переключения n_u и длительности полета *T*, т.е. с ростом вычислительной сложности задачи. Максимально достигнутое ускорение в условиях исследование лишь немного превышает 10. Меньшее ускорение объясняется, в первую очередь, потерями на копирование данных из оперативной памяти хост-процессора в глобальную память *GPU* и обратно.

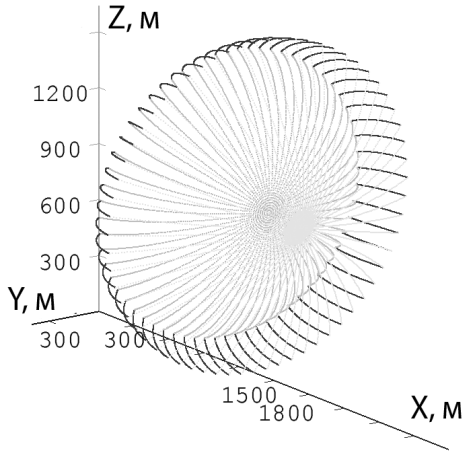


Рис. 2. Пример границ области достижимости ВМЛА

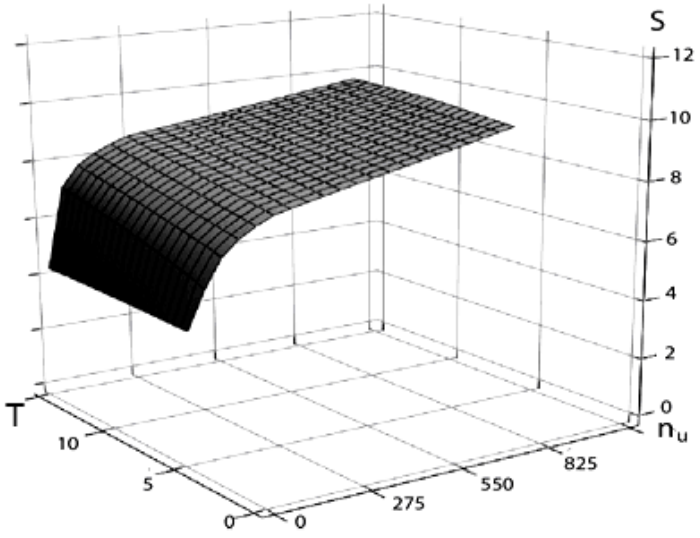


Рис. 3. Достигнутое ускорение в функции числа точек переключения n_u и длительности полета T:

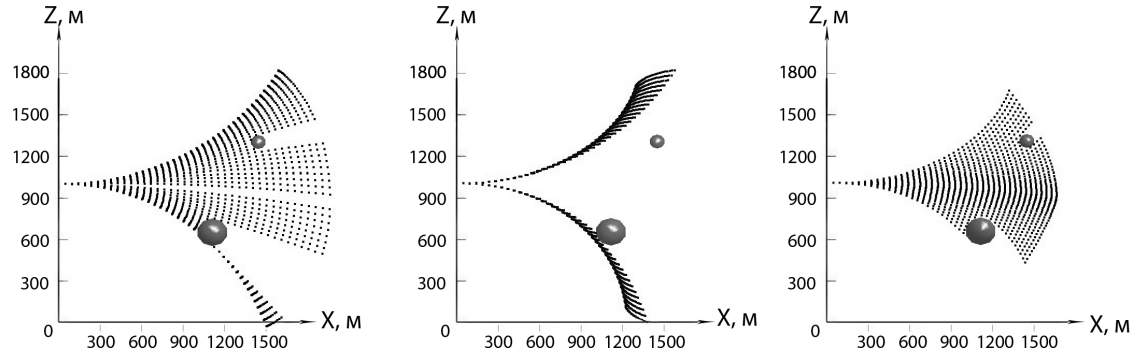
$$m_\gamma = 256$$

5.2 Стационарная задача с препятствиями

Положим, что в области достижимости ВМЛА находится одно или несколько стационарных сферических препятствий, имеющих известные радиусы и координаты центров. Наличие этих препятствий несколько меняет рассмотренный алгоритм построения области достижимости ВМЛА. На каждом шаге интегрирования в этом случае приходится проверять допустимость данной траектории и, если она пересекает то или иное препятствие, прекращать интегрирование. Характер соответствующей деформации области достижимости для двух препятствий в одной из координатных плоскостей иллюстрирует рисунок 4. Рисунок получен при следующих начальных условиях:

$$X = 0\,\text{,м};\quad Y = 0\,\text{,м};\quad Z = 1000\,\text{,м};\quad V = 330\,\text{,м/с};\quad \Theta = 0;\quad \Psi = 0\,.$$

Соответствующая трехмерная иллюстрация представлена на рисунке 5. При указанных начальных условиях получены также все остальные приведенные ниже результаты.



а) дальняя граница б) боковая граница в) ближняя граница

Рис. 4. Деформация области достижимости двумя препятствиями в плоскости

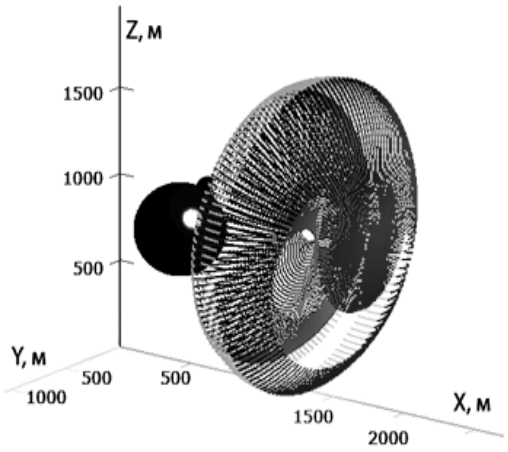
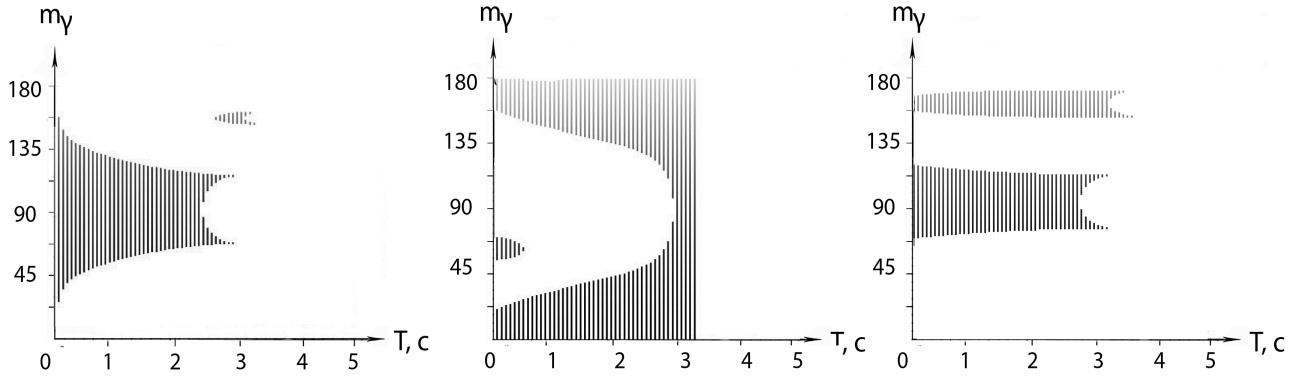


Рис. 5. Деформация области достижимости двумя препятствиями в пространстве

Одной из основных целей построения области достижимости ВМЛА при наличии препятствий является отыскание запрещенных управлений, т.е. управлений, приводящих к столкновению ЛА с препятствием. Для отыскания таких управлений выполнена еще одна модификация рассмотренной выше схемы параллельных вычислений, в соответствии с которой при обнаружении запрещенного управления его параметры заносятся в специальную таблицу. Пример визуализации такой таблицы для случая двух препятствий приведен на рисунке 6.



а) дальняя граница б) боковая граница в) ближняя граница

Рис. 6. Параметры запрещенных траекторий для случая двух препятствий

Исключение из рассмотрения запрещенных траекторий приводит к сокращению вычислительной сложности задачи и к тем большему сокращению, чем большее число траекторий яв-

ляются запрещенными. При этом время решения задачи уменьшается, но одновременно уменьшается и ускорение вычислений (рисунок 7.)

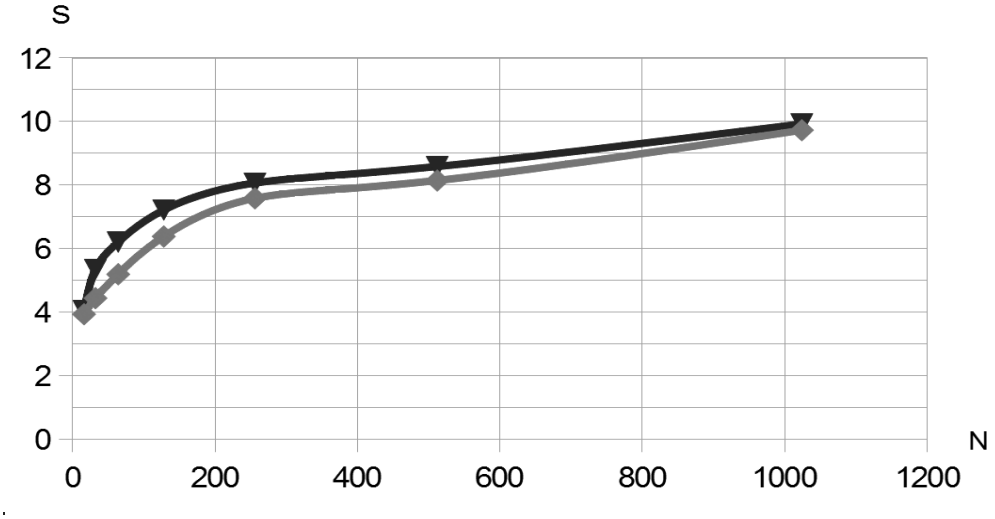


Рис. 7. Достигнутое ускорение при $m_\gamma = 256$ и одном препятствии в функции числа точек переключения n_u : o - при наличии препятствия; ∇ - при отсутствии препятствия**.**

5.3 Динамическая задача с препятствиями

Выполнено исследование эффективности параллельных вычислений также для случая одного сферического препятствия, движущегося равномерно и прямолинейно, а также для случая аналогичного тяжелого препятствия, движущегося под действием силы тяжести. Характер деформации области достижимости ВМЛА в последнем случае иллюстрирует рисунок 8.

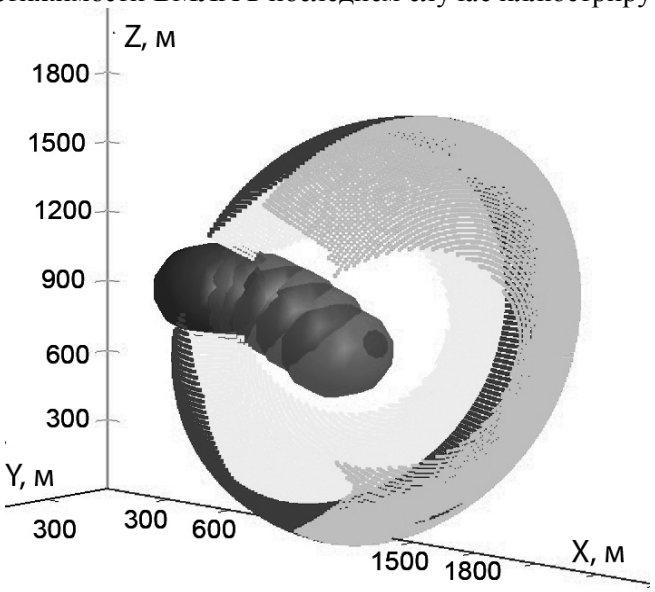


Рис. 8. Область достижимости ВМЛА для случая динамического сферического препятствия

Результаты исследования ускорения вычислений показывают результаты, близкие к представленным выше для стационарного препятствия.

6. Заключение

Результаты выполненного исследования показывают, что построения области достижимости ВМЛА на графической карте *Compute capability* 1.1 позволяет достигнуть примерно 11 кратного ускорения. Выполнено исследование причин, обуславливающих такое небольшое ускорение. Среди мер, по увеличению полученного ускорения можно назвать использование неравномерной декомпозиции точек переключения по потоковым процессорам, использование более новых графических карт (с *Compute capability* 2.0 и выше), применение нескольких графических процессорных устройств.

В целом, на основании выполненного исследования можно констатировать перспективность использования графических процессорных устройств для приближенного построения границ области достижимости ВМЛА в реальном времени, как в условия отсутствия в этой области препятствий, так и в случае наличия статических и динамических препятствий.

В развитие планируется исследование эффективности применения для решения указанной задачи нескольких графических процессорных устройств.

Литература

1. Воронов, Е.М. Алгоритм оценки границ области достижимости летательного аппарата с учетом тяги / Е.М. Воронов, А.А. Карпунин // Вестник МГТУ. Сер. Приборостроение.- 2007.- №4(69).- с. 81-99.
2. Воронов, Е.М. Численные методы построения области достижимости динамической системы / Е.М. Воронов, А.П. Карпенко, О.Г. Козлова, В.А. Федин В.А. // Вестник МГТУ им. Н.Э. Баумана, серия «Приборостроение», 2010, №2 (79), с. 3-20.
3. Воронов, Е.М. Параллельное построение множества достижимости высокоманевренного летательного аппарата методом «мультифиниша» / Е.М. Воронов, А.П. Карпенко, В.А. Федин // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2010. – с. 113-120.
4. Витюков, Ф.А. Построение области достижимости динамической системы на NVidea и AMD графических процессорах / Ф.А. Витюков, В.К. Домашнев, А.П. Карпенко, В.А. Федин В.А. // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи Труды международной суперкомпьютерной конференции (21-26 сентября 2009 г., Новороссийск). – М.: Изд-во МГУ, 2010. – с.635-641.

Квантово-химические прикладные пакеты на Российских грид-полигонах*

В.М. Волохов[1], Д.А. Варламов[1,2], А.В. Волохов[1],
А.В. Пивушков[1], Г.А. Покатович[1], Н.Ф. Сурков[1]

Институт проблем химической физики РАН [1],
Институт экспериментальной минералогии РАН [2]

В статье сделан обзор текущего состояния размещения и использования прикладных пакетов в области квантовой химии и молекулярной динамики на существующих российских грид-полигонах. Охарактеризованы основные компоненты адаптированных прикладных программных пакетов и обслуживающих их грид-сервисов. Описаны основные прикладные пакеты вычислительной химии, адаптированные в настоящее время для работы в качестве вычислительных грид-сервисов. Сформулированы перспективы применения прикладных пакетов с использованием грид-вычислений.

1. Введение

Вычислительная химия и сопряженные с ней области знаний являются одними из наиболее заинтересованных в грид-вычислениях (в том числе на входящих в состав грид-полигонов суперкомпьютерах) отраслями науки [1-3]. Исследования, проводимые в области химии и смежных наук, зачастую абсолютно неэффективны без использования сверхмощных параллельных и распределенных вычислительных ресурсов для решения задач самых разных классов. Например, некоторые задачи оптимизации крупных молекулярных структур требуют выполнения до 10^9 отдельных расчетов. Подобные расчеты требуют вычислительных ресурсов, которые не может предоставить ни один из вычислительных центров, что приводит к необходимости использования мощностей крупных распределенных грид-полигонов либо суперкомпьютеров высокой производительности.

Данная статья опирается на опыт использования прикладных программных пакетов (далее – «ППП») вычислительной химии в грид-средах, что является одним из основных направлений работы вычислительного центра Института Проблем Химической Физики в Черноголовке (ИПХФ РАН, http://www.icp.ac.ru). Адаптация прикладных пакетов проводилась авторами для различных распределенных сред, основанных на middleware gLite, Unicore, Globus Tools, в условиях основных российских грид-полигонов (Национальная Нанотехнологическая Сеть - ГридННС, СКИФ-Полигон, EGEE(EGI)-RDIG).

Институт располагает богатейшей в России библиотекой параллельных квантово-химических и молекулярно-динамических программ (авторских, «open source» и лицензионных), что позволяет проводить обширные эксперименты по адаптации подобных программ к грид-средам в интересах собственных пользователей. В течение года в институте проводится расчет от 3 до 4 тысяч вычислительных задач высокой сложности с публикацией более чем 400 печатных работ с использованием результатов проведенных расчетов. Работы с использованием грид-вычислений в интересах квантовой химии и молекулярной динамики в ИПХФ ведутся с 2004 года, и в настоящее время осуществляются под эгидой нескольких государственных программ (включая Федеральные Целевые Программы, Программы Президиума РАН, гранты РФФИ). ИПХФ является инициатором по использованию квантово-химических пакетов в ряде создаваемых в настоящее время российских грид-полигонов разного масштаба.

Данная статья может представлять интерес как для конечных пользователей-химиков, заинтересованных в интенсификации своих расчетов, так и для администраторов ресурсных грид-сайтов, заинтересованных в увеличении функциональности своих ресурсов. Отметим, что здесь не рассматриваются процедуры адаптации ППП, установки и настройки как клиентских интерфейсов пользователей грид-среды, так и собственно ресурсных сайтов грид-сред (см.[1,3]).

* Работа выполнена при поддержке гранта РФФИ № 11-07-00686-а

2. Основные компоненты адаптированных к грид-средам ППП

2.1 Собственно прикладной пакет

Использование стандартных (с точки зрения химика, да и любого конечного пользователя) ППП в грид-средах обуславливается несколькими параметрами:

- востребованность прикладных пакетов научно-техническими работниками, т.е. конечными пользователями (нет смысла проводить весьма трудоемкую адаптацию и последующую настройку грид-среды для малоиспользуемых пакетов);
- лицензионные ограничения (в том числе существующие и для свободно распространяемых пакетов); для коммерческих ППП условия лицензирования могут либо крайне ограничить возможность реализации их в грид-среде, либо впрямую запретить это;
- доступность исходных кодов ППП, что весьма желательно для некоторых модулей пакетов (например, коммуникационных или отвечающих за реализацию параллельных протоколов);
- возможность работы без интерактивного взаимодействия с пользователями и графического вывода информации (что пока маловероятно реализовать в условиях грид-среды).

Как правило, установка прикладных пакетов (из исходных текстов или бинарных дистрибутивов) на ресурсные грид-сайты почти не отличается от таковой для обычных локальных вычислительных кластеров. Некоторые особенности связаны с настройкой управляющих очередей PBS (или аналогичных систем), Как правило, основные проблемы связаны с ограничениями прав псевдопользователей («mapped users») по сравнению с локальными пользователями, ограничениями на время исполнения программ (что весьма актуально для ППП вычислительной химии, где время выполнения может достигать месяцев) и ограничениями в ресурсах. Также, при наличии выбора, рекомендуется вместо сокетных параллельных версий ППП использовать MPI или OpenMP версии, поскольку они гораздо лучше отвечают требованиям грид-сред в части детального мониторинга ресурсов и управления заданиями.

2.2 Настройка грид-шлюза для обслуживания ППП

Грид-шлюз ресурсного сайта предназначен для информирования брокера ресурсов о подчиненных ресурсах, поддерживаемых виртуальных организациях (ВО) и входящих в их состав ППП, обработки входящих грид-заданий и передачи их вычислительным элементам, мониторинга подчиненных ресурсов и выполнения грид-задач и многих других функций, описание которых не входит в задачи данной публикации. Для конкретного ППП в зависимости от используемой грид-среды вносится информация о доступности ППП различным Виртуальным Организациям (ВО), ACL листах, размещении исполняемых файлов ППП, допустимых лимитах использования ресурсов сайта.

2.3 Низкоуровневые клиентские интерфейсы ППП

Низкоуровневые клиентские интерфейсы (или интерфейсы командной строки – ИКС) включают набор shell-скриптов по формированию исходящих заданий, запуску через грид-инфраструктуру на удаленных ресурсных узлах (с возможностью выбора ресурсов), мониторингу выполнения задач, возвращению полученных результатов и «сборку» окончательных результатов на интерфейсе пользователя. Как правило, для ППП включают в свой состав модернизированные скрипты локальных запусков, а также установки необходимых переменных окружения на удаленных ресурсах.

2.4 Высокоуровневые web-интерфейсы (ПОИ, ВИГ)

Высокоуровневые web-интерфейсы (ПОИ – «Проблемно-Ориентированный Интерфейс», ВИГ – «Веб-интерфейс ГридННС») к адаптированным ППП позволяют более эффективно использовать все преимущества грид-расчетов, особенно для неподготовленного пользователя. Эта среда позволяет пользователям получить доступ к грид-ресурсам и сервисам, вызывать и настраивать их с помощью web-браузера. Web-интерфейсы являются «контейнером» для низ-

коуровневых пользовательских интерфейсов, обеспечивающих полнофункциональную работу с грид-службами. ПОИ предоставляют средства работы с файлами данных и конфигурационными файлами ППП, обеспечивая их загрузку, хранение, редактирование, удаление. Как базовые компоненты ПОИ содержит средства формирования грид-задания (с возможностью выбора доступного (для выбранной ВО) ресурса грид-полигона или использования произвольного ресурса), средства запуска сформированного задания «прозрачно» для пользователя через брокер/менеджер ресурсов грид-полигона с учетом требований безопасности и контроля передачи информации, принятым в рамках грид-полигона. Web-интерфейсы предоставляют возможность постоянного мониторинга выполнения задания (включая его останов и перезапуск) и получения конечных результатов выполнения задачи, лог-файлов и (при необходимости) прочих файлов на портал, хранение данных файлов и доставку их по запросу на компьютер пользователя.

В ряде случаев для ППП могут быть созданы дополнительные модули по интерактивному формированию пользователем сложных конфигурационных файлов и файлов данных для запуска адаптированных ППП в грид-среде. Модули на основе многоуровневых меню и системы «деревьев» выбора обеспечивают корректный выбор из большого массива альтернатив методов вычислений и различных параметров расчетов, ввод и коррекцию численных данных, тестовую проверку файлов конфигурации и данных перед запуском задания. Они могут также включать возможность работы пользователя с использованием типовых шаблонов задач вычислительной химии (с минимумом изменяемых входных параметров) для решения стандартизированных задач предметной области. В ИПХФ РАН пилотный web-интерфейс с такими возможностями реализован для пакета GAMESS, что позволяет создавать конфигурационные файлы ППП, содержащие описания нескольких сотен параметров и методов вычислений.

3. Квантово-химические и молекулярно-динамические пакеты, адаптированные для работы в составе российских грид-полигонов

Ниже приведены краткие описания функциональности прикладных пакетов вычислительной химии, адаптированных для проведения вычислений в рамках Национальной Нанотехнологической Сети (http://www.ngrid.ru), в том числе на ресурсном сайте ИПХФ РАН (http://grid.icp.ac.ru и субдомены http://nanogrid.icp.ac.ru и https://webgrid.icp.ac.ru). Все указанные ППП в разной степени (в зависимости от лицензий) доступны в рамках ВО «NanoChem», а также ВО, относящимся к конкретным ППП (Gamess, AbInit и т.п.), через портал ГридННС https://ui.ngrid.ru. Отметим, что часть ППП была также ранее реализована авторами на пространстве грид-полигонов EGEE(EGI)-RDIG (http://www.egee-rdig.ru) и СКИФ-Полигона (http://skif-grid.botik.ru). Как правило, адаптация пакетов для разных грид-сред различается особенностями реализации низкоуровневых интерфейсов между middleware и ППП. Большинство нижеперечисленных пакетов установлены на кластерах ресурсных центров, входящих в ГридННС (в том числе ИПХФ) и доступны в качестве вычислительных грид-сервисов, для части сделаны тестовые инсталляции (в основном из-за проблем с лицензиями). Для пакетов после установки и тестирования на кластерах настроены шлюзы приема входящих грид-заданий и работы с GridFTP ресурсами, установлены высокоуровневые интерфейсы (различной степени сложности) и проведено массовое тестирование (включая стресс-тесты) на различных задачах. ИПХФ выступал в качестве установщика ряда адаптированных пакетов (Gamess, Gaussian), предоставлял ресурсы своего грид-центра и производил тестирование созданных грид-сервисов через низкоуровневые и высокоуровневые web-интерфейсы грид-среды.

- **GAMESS-US** (http://www.msg.chem.iastate.edu/GAMESS, General Atomic and Molecular Electronic Structure System) – свободно распространяемый прикладной программный пакет, позволяет рассчитывать энергию, геометрию и структуры молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Основное направление – развитие методов расчета сверхбольших молекулярных систем.
- **Gaussian** (http://www.gaussian.com) предназначен для расчета структуры и свойств молекулярных систем разной размерности. Выделяется широким спектром реализованных квантово-химических методов расчетов, высокой эффективностью и удобным интерфейсом поль-

зователя (всего реализовано более 80 методов расчетов). Позволяет рассчитывать энергию, структуру молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Молекулярные модели, рассчитанные в Gaussian, могут быть применены как к стабильным соединениям, так и к тем, которые трудно или невозможно наблюдать экспериментально. Gaussian может использоваться в SMP и параллельных вариантах расчетов (с использованием пакета TCP Linda). Пакет Gaussian существенно ограничен лицензионными правами, однако, допускает получение лицензии на использование в грид-средах.

- **AbIinit** (http://www.abinit.org) используется для решения ряда задач квантово-механических расчетов с высокой вычислительной сложностью и относительной независимостью параметров от результатов расчетов предыдущих задач. Поэтому, помимо встроенной поддержки параллельных вычислений (на базе MPI), возможна параллелизация выполняемых задач по данным. ППП обеспечивает решение теоретических задач – вычисление полной энергии, плотности заряда и электронной структуры атомных систем (молекулы и периодические твердые тела). Вычисления в ППП AbInit основаны на применении теории функционала электронной плотности, псевдопотенциалов, базиса из присоединенных и расширенных плоских волн. ППП Abinit позволяет использовать теорию возмущения многих тел (GW-приближение) и теорию функционала, зависящую от времени. Он обеспечивает режим оптимизации атомной геометрии под действием межатомных DFT-сил и внешнего давления, а также позволяет проводить моделирование молекулярной динамики, используя эти силы, с определением колебательно-фононных, диэлектрических, механических и термодинамических свойств твердых тел. AbInit позволяет оптимизировать геометрию исследуемой системы, минимизируя силы или напряжения, проводить молекулярно-динамическое моделирование, вычислять распределение электронной плотности, определять динамическую матрицу, эффективный заряд и многое другое. Распространяется на основе GPL.
- **GROMACS** (http://www.gromacs.org**,** GROningen MAchine for Chemical Simulations), пакет молекулярной динамики для моделирования физико-химических процессов, в том числе - динамики крупных молекулярных систем (10^3-10^6 частиц). Представляет собой набор программ, предназначенных для расчета траекторий движения отдельных частей молекулы, аппроксимированной механической системой физических материальных точек, связанных набором сил. Пакет предназначается в основном для моделирования крупных молекул, в том числе биомолекул (белки и липиды), имеющих много связанных взаимодействий между атомами. GROMACS является пакетом прикладного программного обеспечения для расчетов как классической молекулярной динамики различных систем, так и с использованием сторонних квантово-механических пакетов таких как Gaussian, Mopac, GAMESS-UK, ORCA, гибридных расчетов (QM/MM). GROMACS обладает высоким уровнем параллелизации, рассчитанным на использование на выскопроизводительных кластерах и суперкомпьютерах с разделяемой памятью. Имеет две реализации параллельных алгоритмов: с использованием MPI-2 и с использованием POSIX threads/NPTL. Из дополнительных возможностей – поддержка выполнения части вычислений на GPU. GROMACS является программным обеспечением с открытым исходным кодом, выпущенным под лицензией GPL.
- **OpenMX** (http://www.openmx-square.org, Open source package for Material eXplorer) – квантово-механический программный пакет для моделирования наноструктур, основанный на использовании метода DFT (Density Functional Theories, теория функционала плотности), с использованием псевдопотенциалов для известных типов атомов. Данная особенность позволяет пакету хорошо масштабироваться при увеличении размерности системы примерно как O(N) и O(NlogN). Также ППП позволяет выполнять квантово-механические расчеты молекулярной динамики структур с использованием DFT. OpenMX может использовать несколько различных схем параллелизации для различных типов вычислительных систем: стандарный MPI v2.0, OpenMP и гибридное использованием MPI+OpenMP, что позволяет использовать OpenMX на широком классе параллельных вычислительных. OpenMX является мощным средством исследования наноматериалов широкого спектра, таких как биоматериалы, углеродные нанотрубки, магнитные материалы и нанопроводники. Распространение пакета и его исходных кодов соответствует лицензии GPLv2.
- **LAMMPS** (http://www.lammps.sandia.gov, Large-scale Atomic/Molecular Massively Parallel

Simulator) предназначен для моделирования физико-химических процессов с использованием уравнений классической молекулярной динамики (уравнений Ньютона). Атомы в молекулах рассматриваются как материальные точки, взаимодействующие посредством потенциальных полей. Данный пакет распространяется в виде исходного кода по лицензии GPL и создан специально для работы на высокопроизводительных параллельных системах. Он использует MPI для работы на системах с распределенной памятью, написан на языке C++. LAMMPS является многофункциональным продуктом, который позволяет моделировать физические процессы и химические реакции, происходящие в самых разных системах, таких как атомные системы, жидкости, кристаллы металлов и полупроводников, полимеры, белки, ДНК, гранулярные материалы, эллипсоидальные частицы, точечные диполи, крупнозернистые мезомасштабные модели, а также комбинации всего вышеперечисленного.

- **VASP** (http://cms.mpi.univie.ac.at/vasp) обеспечивает вычисление полной энергии, плотности заряда и электронной структуры атомных систем (молекулы и периодические твердые тела), оптимизацию атомной геометрии в статических условиях и под внешним давлением, а также позволяет проводить моделирование методом молекулярной динамики с определением важных физических свойств твердых тел (колебательно-фононных, диэлектрических, механических, термодинамических).Предназначен для моделирования процессов в объеме и на поверхности твердых тел (прежде всего катализа и ионной проводимости) в рамках неэмпирических подходов, основанных на применении функционалов плотности с использованием периодических граничных условий с базисами на плоских волнах. Подход, реализованный в программе VASP, основан на приближении локальной плотности (при конечных температурах), при этом свободная энергия считается вариационным параметром, и на каждом шаге выполняется точная оценка мгновенного электронного основного состояния. В программе VASP значительно улучшены процедуры сходимости процессов ССП и оптимизации. Имеется процедура статистического «размывания» краев запрещенной зоны и оптимизации спинового состояния для моделирования металлов и узкозонных полупроводников. Распространяется на коммерческой основе, подразумевает обязательное лицензирование, для определенных типов лицензий допускается использование в качестве грид-сервиса.
- **PWScf** (http://www.pwscf.org, http://www.quantum-espresso.org, Plane-Wave Self-Consistent Field) построен на базе теории функционала электронной плотности (DFT) и методе псевдопотенциала (PAW-метод). Представляет собой мощный инструмент для энергетических расчетов многоэлектронных систем и предназначен для моделирования на квантово-механическом уровне малых кластеров с числом атомов 10-100, определяющих существование возможных в материале фаз. Во многом является аналогом коммерческого квантово-химического ППП VASP. Описание моделируемого объекта строится на языке волновых функций и заданного гамильтониана системы. Целевыми функциями являются электронный энергетический спектр, собственные функции и плотность состояний изолированного кластера при фиксированном положении ядер, потенциальная энергия системы с учетом электронно-ядерных подсистем. С помощью ППП PWscf можно прогнозировать плотности электронных состояний произвольных кристаллических материалов и их свойств, исходя из основ квантовой теории строения вещества. Расчеты с применением ППП PWscf оптимизированы для использования сотен процессоров, поэтому большой интерес вызывает возможность использования этих пакетов на большом числе процессоров в грид-системах (на основе MPI). ППП PWscf распространяется в исходных текстах по лицензии GNU GPL, что позволяет свободно использовать его в качестве грид-сервиса.
- **NWChem** (http://www.nwchem-sw.org, New Wave Chemistry) – используется для проведения смешанного квантово-механического и молекулярно-динамического моделирования, когда определенные локальные области материала моделируются на квантово-механическом, а другие – на молекулярно-динамическом уровнях. Позволяет проводить расчеты геометрии молекулярных структур, расстояний между атомами, сил взаимодействия, свободных энергий поверхностей и т.п. Варианты применения NWChem фокусируются на предоставлении возможностей научных расчетов в области кинетики и динамики химических превращений, химических взаимодействий на границах фаз и в конденсированных (твердых и жидких) фазах. Расчеты с применением ППП NWChem оптимизированы для параллельного использования сотен и тысяч процессоров (на основе MPI). ППП NWChem

свободно распространяется под собственной лицензией в виде бинарных кодов и исходных текстов, что облегчает использование его в качестве грид-сервиса.

- **NAMD** (http://www.ks.uiuc.edu/Research/namd, NAnoscale Molecular Dynamics) – масштабируемая программа для молекулярной динамики, написанная с использованием модели параллельного программирования Charm++, обладает высокой эффективностью распараллеливания. Используется для моделирования больших и сверхбольших структур и кластеров с числом атомов 10^3-10^6 (вплоть до расчета ДНК структур) вблизи состояния равновесия, и в рамках задач, относящихся к неравновесной молекулярной динамике. Моделирование молекулярных систем может производиться в различных приближениях в зависимости от доступных вычислительных ресурсов и сложности рассчитываемой системы. К числу таких приближений относятся: моделирование молекул как твердых тел, моделирование внутримолекулярных потенциалов взаимодействия как гармонических, моделирование только локального кулоновского поля в периодических граничных условиях, или учет дальнодействующей составляющей. Может быть использована для выполнения расчета траекторий движения атомов заданной системы за счет интегрирования уравнений движения с использованием эмпирических потенциалов взаимодействия. Программа активно используется для расчетов мицелл (micelle – коллоидная частица, несущая электрический заряд и объединяющая в себе несколько крупных молекул) и подобных органических и неорганических молекулярных структур. ППП NAMD является ПО с открытым исходным кодом и свободно распространяется под собственной лицензией в виде бинарных кодов или исходных текстов, что позволяет использовать его на ресурсных грид-сайтах.
- **Firefly** (http://classic.chem.msu.su/gran/firefly, ранее известен как PC GAMESS) - одна из самых популярных и высокопроизводительных программ для теоретического исследования свойств химических систем, позволяет рассчитывать энергию, геометрию и структуры молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Основное направление – расчет больших и сверхбольших молекулярных систем. Достоинством пакета является широкомасштабный охват основных вычислительных квантово-химических алгоритмов и реализация для большого количества процессорных архитектур и параллельных сред. Основные возможности программы Firefly близки таковым для ППП GAMESS-US.

4. Заключение

Применение адаптированных к грид-средам прикладных программных пакетов в области квантовой химии и молекулярной динамики позволяет ставить и решать вычислительные задачи фундаментального и прикладного характера в области химических наук, ранее не доступные из-за ограниченности возможностей вычислительных ресурсов. Основные научные области применения – химическая физика, квантовая химия, исследование наноструктур, молекулярная динамика, биохимия, фармацевтика, разработка топливных элементов и близкие отрасли наук.

Литература

1. В.М.Волохов, Д.А.Варламов, А.В.Пивушков, Н.Ф.Сурков, Г.А.Покатович GRID и вычислительная химия // «Вычислительные методы и программирование», М.: МГУ, 2009, т.10, № 1,с.224-235
2. В.М.Волохов, Д.А.Варламов, А.В.Пивушков, А.В.Волохов «Применение GRID технологий в области вычислительной химии» // «Известия Академии наук. Серия химическая», 2011, № 7, с.1483-1490
3. В.М.Волохов, Д.А.Варламов, А.В.Волохов, А.В.Пивушков, Г.А.Покатович, Н.Ф.Сурков Грид-сервисы в вычислительной химии: достижения и перспективы // «Вестник Уфимского Государственного авиационно-технического университета. Серия «Управление, вычислительная техника и информатика», 2011, т. 15, № 5 (45), с.161-169.

Метод трехуровневого распараллеливания методики ТИМ-2D

А.А. Воропинов

ФГУП Российский Федеральный Ядерный Центр –
Всероссийский Научно-Исследовательский Институт Экспериментальной Физики

Методика ТИМ-2D предназначена для решения задач механики сплошной среды на неструктурированных многоугольных лагранжевых сетках произвольной структуры. Для методики используется метод трехуровневого распараллеливания. На первом уровне осуществляется распараллеливание счета по математическим областям. На втором уровне производится распараллеливание счета внутри математической области по параобластям. На первых двух уровнях используется модель распределенной памяти и интерфейс MPI. На третьем уровне осуществляется распараллеливание итераций счетных циклов в модели общей памяти с использованием интерфейса OpenMP. Уровни могут использоваться как по отдельности, так и в различных сочетаниях при решении одной задачи.

1. Введение

Методика ТИМ-2D [1] предназначена для решения двумерных нестационарных задач механики сплошной среды на неструктурированных многоугольных лагранжевых сетках произвольного вида. Ячейками сетки могут быть не самопересекающиеся многоугольники, в узлах может сходиться произвольное количество ребер. Для расчета задач газодинамики и упругопластичности используются явные конечноразностные схемы. Для моделирования диффузионных приближений используются неявные конечно-разностные схемы. Кинематические величины отнесены к узлам разностной сетки, термодинамические величины к ячейкам. Для решения уравнений механики сплошных сред используется цилиндрическая или плоская декартова система координат.

Для получения результатов в сжатые календарные сроки с необходимой точностью используются различные методы распараллеливания, позволяющие рассчитывать задачу одновременно на большом количестве процессорных ядер. В настоящее время наиболее распространенными прикладными стандартами для распараллеливания являются: интерфейс передачи сообщений MPI, для модели распределенной памяти, и интерфейс OpenMP, предназначенный для модели общей памяти.

При решении сложных двумерных задач начальную геометрию системы часто приходиться разбивать на матобласти. Такое разбиение иногда бывает необходимо для более точного описания взаимодействия тел на выделенной линии скольжения. Методика ТИМ-2D позволяет проводить расчеты в многообластной постановке. При этом в каждой матобласти используется независимая расчетная сетка. В начале каждого счетного шага между матобластями решается задача контактного взаимодействия, в дальнейшем положение и скорости границ выступают в качестве навязанных граничных условий при расчете матобластей. Сами матобласти рассчитываются независимо друг от друга. Это первый ресурс распараллеливания методики ТИМ-2D – по матобластям, который достаточно легко использовать. При этом более удобно использовать модель распределенной памяти, т.к. все взаимодействие между матобластями решается в одном блоке – при расчете контактного взаимодействия и при этом используется небольшой объем информации.

С другой стороны, из-за неструктурированности сетки в методике ТИМ-2D все счётные алгоритмы построены поточечно, без опоры на упорядоченность структуры сетки. Благодаря этому каждый элемент сетки можно обсчитать независимо от соседей, используя только информацию с предыдущего шага. Это второй ресурс распараллеливания – по элементам сетки внутри матобласти. В зависимости от модели памяти это приводит к двум различным подходам. В модели распределенной памяти это мелкозернистое распараллеливание с пространст-

венным разделением области на фрагменты-параобласти (декомпозиция по пространству). При этом недостающая для расчета информация может быть восполнена двумя различными способами – путем наложения между параобластями в один слой ячеек и обменом информацией в слое наложения, либо путем расчета контактного взаимодействия между параобластями по алгоритмам, аналогичным распараллеливанию по матобластям. Если использовать независимость расчета элементов сетки в модели общей памяти, то естественным будет использование распараллеливания итераций счетных циклов (которые, как правило, и построены по элементам сетки). В результате для такого распараллеливания не требуется выполнения декомпозиции по пространству, что позволяет использовать его в сочетании с другими подходами. Таким образом, в методике ТИМ-2D имеется три основных ресурса распараллеливания. Первые два основаны на пространственном разбиении задачи на фрагменты (по матобластям, и внутри матобластей по параобластям) и использовании модели распределенной памяти. И третий – распараллеливание итераций счетных циклов в модели общей памяти. Эти ресурсы и были использованы для разработки распараллеливания методики ТИМ-2D.

На первом этапе было реализовано распараллеливание итераций счетных циклов в модели общей памяти с использованием интерфейса OpenMP [2]. На втором этапе было реализовано распараллеливание по областям в модели распределенной памяти с использование стандарта MPI, а также режим счета в смешанной модели памяти с использованием двух уровней распараллеливания [3, 4]. Последними были реализованы алгоритмы мелкозернистого распараллеливания [5, 6]. Данный доклад посвящен вопросам одновременного использования уровней распараллеливания при расчете одной задачи.

2. Декомпозиция данных

Эффективное использование счетных программ методик на параллельных машинах требует выполнения декомпозиции таким образом, чтобы ядра вычислительной системы были загружены равномерно, а взаимодействия между ними были минимальны. В модели распределенной памяти задача декомпозиции состоит в распределении данных между процессами (декомпозиция данных) таким образом, чтобы количество обменов и объем передаваемых данных между ними были минимальны. В методике ТИМ-2D для распараллеливания в модели распределенной памяти используется декомпозиция по пространству.

При распараллеливании по областям максимальное количество узлов, которые можно задействовать, равно количеству областей. Разбиение задачи на матобласти выполняется на этапе подготовки расчета, исходя из геометрии задачи и начальных данных. В результирующей сетке может существенно различаться объем вычислений, приходящихся на каждую область, из-за различного количества точек, веществ, используемых приближений и счетных алгоритмов. Это может приводить к снижению эффективности распараллеливания при расчете на максимальном количестве узлов. Для того чтобы обойти эту проблему, в рамках методики ТИМ-2D реализована возможность отнесения нескольких матобластей к одному MPI процессу. То есть одним процессом может рассчитываться не одна матобласть, а несколько. Благодаря этому, как правило, удается сбалансировать вычислительную нагрузку, при этом уменьшается количество задействованных процессов. В таком виде задача формирования декомпозиции для распараллеливания по областям сводится к задаче близкой к классической задаче «о ранце» теории графов: с каждой областью связывается некоторое число (вес). Этот вес отражает время на расчет области, либо количество счетных точек, если время на расчет области недоступно. Таким образом, получается множество из K чисел, где K – количество матобластей в задаче. Затем решается задача разбиения данного множества на N групп, где N – количество доступных процессов ($N \le K$). При выполнении разбиений минимизируется максимальная сумма чисел в группах (минимизируется вес самого тяжелого «ранца»). Распределение чисел по группам и определяет распределение областей по процессам, т.е. декомпозицию.

Принципы декомпозиции для мелкозернистого распараллеливания следующие:

- Декомпозиция осуществляется по ячейкам (ячейка – основной счетный элемент сетки в методике ТИМ-2D).
- Все ячейки области распределяются по компактам, при этом каждая ячейка принадлежит одному и только одному компакту. Компакт – набор ячеек одной области.

- Каждая математическая область разбивается на компакты независимо от других.
- На основе каждого компакта формируется параобласть – полноценная счетная область пригодная для проведения расчета (при этом в зависимости от способа мелкозернистого распараллеливания между параобластями может формироваться слой наложения, или параллельная контактная граница).

Задача декомпозиции для мелкозернистого распараллеливания сводится к решению задачи о разрезании графа на подграфы. При этом используется следующий алгоритм:

- На основе неструктурированной сетки строится граф, отображающий ее структуру. При этом ячейкам сетки соответствуют вершины графа, а соседству между ячейками – ребра графа.
- Вершинам графа присваивается вес, отражающий вычислительную нагрузку, связанную с соответствующей ячейкой. Веса ребер графа используются для придания декомпозиции дополнительных свойств – для вытягивания компактов вдоль границ для уменьшения количества обменов при расчете контактного взаимодействия.
- Непосредственно задача о разрезании графа на подграфы решается при помощи алгоритмов библиотек ParMeTiS или SCOTCH [7].
- Для задач, состоящих из нескольких математических областей, используется алгоритм двухпроходной декомпозиции. На первом проходе каждая область разбивается независимо от других на большое количество микрокомпактов. На втором проходе строится макрограф, вершинами которого являются компакты первого прохода и границы между областями. Ребра макрографа отражают соседство между микрокомпактами первого прохода и соседство с границами области.

Подробное описание алгоритмов декомпозиции для методики ТИМ-2D приведено в работе [8].

3. Уровень распараллеливания по областям

На верхнем уровне распараллеливание осуществляется по математическим областям в модели распределенной памяти. Поскольку расчет математических областей полностью не зависим, то взаимодействие между процессами осуществляется только на этапе расчета контактного взаимодействия между ними. При этом сама контактная граница между двумя областями рассчитывается на одном из двух процессов, к которым отнесены области, формирующие эту границу.

Расчет контактного взаимодействия в параллельном режиме производится по следующему алгоритму (курсивом выделены блоки, относящиеся к распараллеливанию):

1. Для каждого контура формируются «*свои*» стороны (в последовательном режиме все стороны – «свои»).
2. *Производится обмен информацией о сторонах контуров («свои» отправляются, «чужие» принимаются).*
   Все последующие операции выполняются только для тех контуров, которые рассчитываются текущим процессом.
3. Совмещение сторон контура.
4. Формирование основного и вспомогательного слоя узлов на контуре.
5. Расчет расстояний между соседними точками на контуре.
6. Посадка узлов вспомогательного слоя на основной слой.
7. *Обмен информацией о сформированных контурах между процессами.*
8. Расчет ускорений и скоростей для граничных узлов «*своих*» областей.
9. *Получение процессом, рассчитывающим контур вещественной информации о «чужой» стороне контура. (Соответственно эта сторона подготавливается и отправляется процессом, рассчитывающим соседнюю область).*
10. Расчет взаимодействия сторон контура. Вычисление новых координат и скоростей точек контура.
11. *Пересылка рассчитанного контура.*

Заметим, что обмены информацией производятся только для тех контуров, которые разделяют области, рассчитываемые на разных процессах. Если контур разделяет области, рассчиты-

ваемые одним и тем же процессом, то и контур рассчитывается этим же процессом. Аналогично внешний контур задачи (типа «свободная граница» или «жесткая стенка») рассчитываются процессом, рассчитывающим соответствующую область.

4. Уровень мелкозернистого распараллеливания

В методике ТИМ-2D деление на математические области производится исходя из начальных данных, таких как параметры геометрии и распределение веществ. При этом не преследуется цель достижения высокой эффективности распараллеливания. В разных областях может быть задано различное количество точек, используются разные вещества и приближения. Это приводит к тому, что, использовать такое распараллеливание удается для небольшого количества процессоров. Дополнительное использование распараллеливания в модели общей памяти позволяет увеличить количество задействованных ядер почти на порядок. Тем не менее, в реальных расчетах удается использовать не более нескольких десятков процессорных ядер. Все это показывает, что такого распараллеливания оказывается недостаточно и необходимо использование мелкозернистого распараллеливания.

Основной вопрос, который возникает при мелкозернистом распараллеливании, это способ расчета узлов сетки, которые окружают ячейки, отнесенные к разным компактам (параграничные узлы). Рассмотрим фрагмент сетки, представленный на рисунке 1. На рисунке ячейки с белой заливкой отнесены к компакту 1, с желтой заливкой к компакту 2.

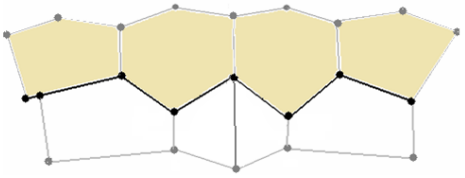


Рис. 1. Фрагмент сетки с разделением на компакты

В соответствии с разностной схемой методики [1] для расчета узла строится замкнутый контур интегрирования из центров ячеек и «центров» ребер («центр» ребра в цилиндрическом случае – точка, которая делит поверхность, образуемую вращением ребра вокруг оси симметрии, на две равные части, в плоском случае – это середина ребра). Пример контура интегрирования для узла V на рисунке 2 выделен зеленой линией. В определении контура интегрирования участвуют центры окружающих ячеек C_1, C_2, C_3, C_4 C_1, C_2, C_3, C_4 и «центры» ребер VV_1, VV_2, VV_3, VV_4. В соответствии с контуром интегрирования при расчете узла V используются величины ячеек C_1, C_2, C_3, C_4 и узлов V, V_1, V_2, V_3, V_4.

Если сохранять контур интегрирования узла неизменным, то получается первый тип мелкозернистого распараллеливания – с наложением в один слой ячеек.

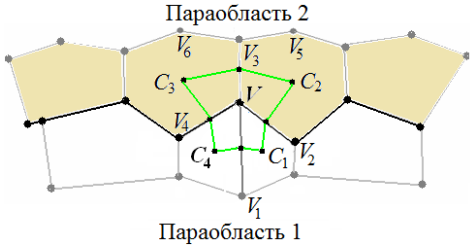


Рис. 2. Контур интегрирования узла

С другой стороны контур интегрирования можно представить как совокупность замкнутых контуров со стороны каждой из ячеек (такой способ используется и в разностной схеме для определения массы узла). Аналогично можно представить контур интегрирования как совокупность замкнутых контуров со стороны каждой параобласти. Для рассматриваемого примера такое разделение представлено на рисунке 3. Линия раздела контура интегрирования на рисунке выделена красным. При разделении контура интегрирования узлы вдоль линии между параобластями разделяются на пары (или больше по количеству параобластей сходящихся в узле),

например, узел V на V' и V''. Для каждого из узлов используется свой контур интегрирования для определения массы и ускорений, которые затем сочетаются для расчета единой скорости. Такое разделение контура интегрирования при расчете позволяет не использовать наложение между параобластями. Подобная схема используется при расчете контактной границы «без скольжения», поэтому для мелкозернистого распараллеливания был использован блок расчета контактных границ [9] с небольшими доработками.

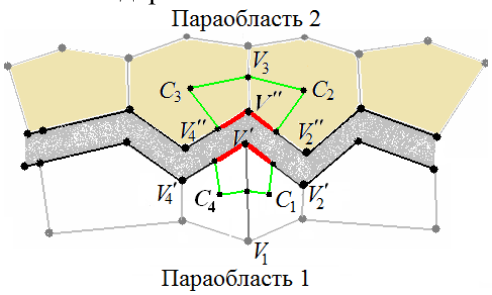


Рис. 3. Разрыв контура интегрирования для узла при мелкозернистом распараллеливании без наложения

В методике ТИМ-2D расчет газодинамических величин разделен на два основных этапа:

1. Расчет узловых величин – скорости, координаты (расчёт уравнения движения). При расчете узла используется информация из соседних узлов и окружающих ячеек с предыдущего момента времени.
2. Расчет ячеечных величин – плотность, давление, энергия (расчёт уравнения энергии). При этом используется информация о новом положении узлов рассматриваемых ячеек.

Первый метод мелкозернистого распараллеливания основан на сохранении контура интегрирования для парапограничных узлов. В результате возникает необходимость наложения между параобластями в один слой ячеек. При этом парапограничные узлы могут рассчитываться любым из процессов, рассчитывающих окружающие параобласти. Расчет узлов является относительно дешевой операцией в методике ТИМ-2D, поэтому для дополнительного контроля расчет парапограничных узлов дублируется. Второй возникающий вопрос – это обновление информации в присоединенных элементах сетки (ячейки и узлы). Это обновление производится путем асинхронных обменов информацией в парапограничных, припограничных и присоединенных элементах. Вызовы процедур выполнения обменов расположены так, чтобы необходимая информация обновлялась до ее использования.

Второй метод мелкозернистого распараллеливания основан на разбиении контура интегрирования парапограничных узлов. В результате между параобластями не требуется наложение по ячейкам. При разрыве контура интегрирования в каждой параобласти появляется свой парапограничный узел со своим контуром интегрирования. Для восстановления исходного контура интегрирования необходимо их объединение. В этом режиме мелкозернистого распараллеливания для каждого из пары узлов рассчитывается ускорение, производится обмен рассчитанными ускорениями и массами и затем расчет единой скорости. Получение единой скорости является относительно дешевой операцией, поэтому для уменьшения количества обменов получение единой скорости дублируется процессами.

5. Согласование уровней распараллеливания

Отдельной проблемой оказалось согласование алгоритмов трех уровней распараллеливания между собой. Наиболее удобно рассматривать решение этой проблемы с точки зрения мелкозернистого распараллеливания, т.к. с одной стороны это средний уровень, с другой стороны, оно было реализовано последним, и проблема согласования стояла для него наиболее остро.

5.1 Использование смешанной модели памяти

Первый вопрос заключается в использовании смешанной модели памяти и одновременном применении алгоритмов распараллеливания использующих MPI и OpenMP. При распараллели-

вании в модели смешанной памяти в рамках каждого вычислительного узла выполняется один процесс. Процесс выполняет цикл по счетным областям задачи, но области, отнесенные к другим узлам, пропускает. Областью может быть как матобласть, так и параобласть, в зависимости от режима проведения расчета. Нумеруются эти процессы так, как это принято в стандарте MPI (от 0 до $N-1$, где N – общее количество используемых вычислительных узлов для расчета задачи). Эти процессы и выступают в качестве MPI процессов, они управляют расчетом задачи в рамках вычислительного узла, производят MPI обмены. В рамках каждого вычислительного узла MPI процесс производит независимое от других узлов порождение параллельных нитей OpenMP для проведения расчетов областей, отнесенных к данному вычислительному узлу (и только для этих областей). Таким образом, OpenMP распараллеливание не зависит от распараллеливания в модели распределенной памяти.

5.2 Согласование мелкозернистого и пообластного распараллеливания

Алгоритмы пообластного и мелкозернистого распараллеливания используют модель распределенной памяти и декомпозицию по пространству. В связи с этим возникли вопросы организации описания параобластей в структурах данных и эффект «фрагментации» границ между матобластями по нескольким параобластям.

5.2.1 Организация параобластей в структуре данных

Одним из первых вопросов при разработке алгоритмов мелкозернистого распараллеливания встал вопрос об организации параобластей в структурах данных. Основным требованием при этом являлось то, что эта организация должна быть прозрачной для счетных алгоритмов. Т.е. счетный алгоритм должен иметь возможность одинаково работать как с матобластями, так и параобластями. Для этого параобласти структурно оформляются также как матобласти. Кроме того, в структуре «области» заведен ряд дополнительных величин, описывающих ее как «параобласть» и делающих привязку к «родительской» матобласти. Помимо этого организуется дополнительная структура, содержащая информацию о глобальной нумерации элементов сетки, информацию для обменов между параобластями (парагранicах) и другие данные.

Поскольку параобласть в структуре данных описывается также как матобласть, то встает вопрос о согласовании их нумераций. Параобласти в счетных структурах нумеруются в списке матобластей. При этом параобласти получают номера после матобластей. Матобласть, для которой были порождены параобласти, из счета исключается. Параобласть так же, как и матобласть, является глобальным объектом уровня задачи. Благодаря использованию такого подхода возможно использование единых алгоритмов распараллеливания по матобластям для распределения параобластей по процессам и дальнейшего управления. Рассмотрим нумерацию параобластей на примере: в задаче 3 матобласти, из первой области формируется 2 параобласти, из второй области параобласти не формируются совсем, а из третьей формируется 3 параобласти. В результате параобласти, полученные из первой матобласти, получат номера 4, 5, а параобласти из 3 матобласти номера 6, 7, 8. Общее количество областей в структуре данных устанавливается равным 8. Отметим, что исходные матобласти 1 и 3 сохраняют за собой номера, но из счета исключаются. Матобласть 2 рассчитывается под своим номером – параобласть для нее не формируется.

5.2.2 Контактные границы

При мелкозернистом распараллеливании многообластных задач может возникать эффект «фрагментации» границ, когда точки стороны границы распределены по нескольким параобластям. Одним из первых шагов в алгоритме расчета контура – является сборка сторон. В случае «фрагментации» границ этот блок требует дополнительного обмена – пересылки фрагментов границы на управляющий процесс матобласти.

Кроме того, после расчета результирующий контур рассылается всем процессам, имеющим фрагменты сторон данного контура. В остальном алгоритмы расчета фрагментированных границ остаются такими же как и при распараллеливании по областям.

5.3 Взаимодействие мелкозернистого распараллеливания с распараллеливанием итераций счетных циклов

Взаимодействие между мелкозернистым распараллеливанием и распараллеливанием итераций счетных циклов логически организуется достаточно просто, так как в этих режимах используются разные парадигмы. На общей памяти распараллеливаются итерации счетных циклов, такие как циклы по ячейкам или по узлам. В параобласти вводится локальная нумерация элементов сетки, поэтому для распараллеливания в общей памяти параобласть идентична матобласти и модификаций не требуется. Тем не менее, два обстоятельства потребовали специальной доработки алгоритмов.

5.3.1 Разделение счетных циклов

Для мелкозернистого распараллеливания с наложением необходимо выполнение обменов приграничными элементами сетки между соседними параобластями. При этом используются асинхронные обмены для того, чтобы наложить вычисления и обмены. Для того чтобы как можно раньше начать отправку данных о приграничных элементах сетки при формировании параобластей, им присваиваются номера меньше внутренних. В результате, в цикле по ячейкам или узлам они обрабатываются раньше, и можно начать их отправку соседней параобласти. Тут и возникает конфликт. Дело в том, что при распараллеливании счетных циклов нарушается последовательность обработки итераций, т.е. в цикле от 1 до 1000, 101 итерация может быть выполнена раньше 100. Нарушение последовательности итераций происходит из-за их распределения между нитями OpenMP.

Выходом из положения стало разделение счетных циклов на пары. Первый цикл выполняется по парапограничным и приграничным элементам сетки. Второй цикл по внутренним элементам. При этом оба цикла заключены в одну параллельную область OpenMP. Между циклами выполняются обмены. При этом обмены выполняются внутри параллельной области OpenMP также в параллельном режиме.

5.3.2 Обращение к MPI процедурам в параллельных областях OpenMP

Вызов процедур асинхронных обменов производится в параллельных областях OpenMP между двумя счетными циклами по приграничным и внутренним узлам или ячейкам. При этом несколькими нитями (в рамках одного процесса) одновременно могут вызваться MPI процедуры для обмена данными о параобластях с соседями. В некоторых случаях это приводило к авостам в MPI, если на вычислительной системе используется библиотека MPI, не являющаяся «безопасной для нитей» (thread safe). Чтобы избежать одновременного обращения, вызов обменов был модифицирован таким образом, чтобы вызов MPI обменов производила в один момент только одна нить. Для этого все обращения к MPI процедурам заключены в критическую область OpenMP с одинаковым именем `MPI`. Такой подход решает описанную проблему, практически не снижая скорости выполнения, при условии, что количество обменов невелико. Также его использование гарантирует, что в других блоках подобные проблемы не возникнут.

6. Различные режимы параллельного счета

Три уровня распараллеливания могут использоваться различным образом: по отдельности, в различных попарных сочетаниях, и все три одновременно. Вместе с последовательным режимом возможно 8 разных режимов счета. Выбор конкретного режима проведения расчета определяется задачей, моделируемыми приближениями и характеристиками вычислительной системы. Рассмотрим режимы счета более подробно.

Последовательный режим счета. Этот режим, как правило, используется на персональном компьютере для различных обработок расчетов, а также для методических расчетов. Этот режим часто используется разработчиками при начале разработки алгоритмов, а также для устранения проблем несвязанных с распараллеливанием.

6.1 Режимы счета с одним уровнем распараллеливания

Возможны следующие режимы счета с использованием одного уровня распараллеливания:

Распараллеливание по счетным областям в модели распределенной памяти (крупноблочное распараллеливание). Этот режим может использоваться для расчета небольших задач на малом количестве ядер (обычно менее 10). Ограничение данного режима распараллеливания обусловлено ограниченным количеством матобластей и сложностями в балансировке нагрузки, поскольку в областях используются различное количество точек, материалы, моделируемые приближения, модели и др. Этот метод распараллеливания применяется, когда использование общей памяти невозможно (например, из-за характеристик вычислительной системы).

Распараллеливание по параобластям в модели распределенной памяти (мелкозернистое распараллеливание). В отдельности такой режим может использоваться для расчета однообластных задач (в этом случае уровень управления задачей и областью фактически совмещаются). Из-за малого количества таких задач применяется такой режим редко.

Распараллеливание итераций счетных циклов в модели общей памяти с использованием интерфейса OpenMP. Использование данного режима ограничено тем, что в современных кластерных системах количество ядер, а следовательно и количество нитей, работающих на общей памяти, невелико. По этой причине распараллеливания в модели общей памяти оказывается недостаточно для проведения расчетов с большим количеством точек. Этот режим используется при выполнении некоторых сервисных операций, при проведении расчетов с небольшим количеством точек (не более 100 тысяч ячеек) и для вычислительных систем, использующих только общую память.

Как видно из описания, каждый из этих режимов имеет существенные ограничения по применимости. И каждый из них по отдельности не может быть использован как универсальный подход. Поэтому основные способы распараллеливания – смешанные режимы.

6.2 Режимы счета с использованием двух уровней распараллеливания

Попарное использование методов распараллеливания включает следующие режимы:

Смешанное распараллеливание по областям и распараллеливание итераций циклов при помощи OpenMP. В этом режиме счет задачи осуществляется так же, как в режиме распараллеливания по областям, но на одном вычислительном узле кластера запускается только один MPI процесс. Каждый MPI процесс является главной нитью OpenMP в рамках своего вычислительного узла. Такой режим активно используется при проведении расчетов для задач средних размеров (от нескольких сотен тысяч до миллиона ячеек).

Смешанное мелкозернистое и распараллеливание итераций счетных циклов. Режим работы аналогичен предыдущему, за исключением того, что распараллеливание счета в общей памяти осуществляется внутри параобласти. Этот режим может применяться для больших однообластных задач. Применение такого распараллеливания также ограничено из-за малого количества задач из одной матобласти.

Смешанное по областям и мелкозернистое распараллеливание в модели распределенной памяти. В этом режиме для каждой матобласти определяется управляющий процесс, который отвечает за ее расчет. А сама матобласть разделяется на некоторое количество параобластей. Такой режим распараллеливания может применяться для расчёта многообластных задач в случаях, когда необходимо задействовать большое количество процессов. Этот режим используется для вычислительных систем без общей памяти.

6.3 Режим счета с трехуровневым распараллеливанием

Последний режим параллельного счета – использование всех трех базовых режимов одновременно – *режим трехуровневого распараллеливания*. Этот режим наиболее сложный с точки зрения реализации и управления, но в таком режиме полностью решаются проблемы запуска на произвольном количестве ядер. Поэтому этот режим является основным для решения больших задач из нескольких миллионов точек. Работа в таком режиме аналогична работе в смешанном режиме распараллеливания по областям и мелкозернистому, но на одном вычислительном узле

запускается по одному MPI процессу, как в режиме смешанного распараллеливания по областям и распараллеливание итераций циклов.

Необходимо отметить, что выделение этих 8 режимов счета является отчасти условным. В реальных расчетах могут возникать ситуации, когда, например, для маленькой области не будет использоваться мелкозернистое распараллеливание, если размер области не превышает размера компакта. В случае же совсем маленькой области, состоящей из нескольких сотен ячеек, может отключаться и OpenMP-распараллеливание, т.к. накладные расходы на организацию параллельной области выше выигрыша от расчета этой области в параллельном режиме. Таким образом, хотя глобально на уровне задачи может быть использован режим из всех трех уровней распараллеливания, но на локальном уровне конкретной матобласти могут отключаться уровни мелкозернистого и/или OpenMP-распараллеливания.

7. Исследование эффективности распараллеливания

В качестве характеристик эффективности распараллеливания использовались следующие функции: $S_p = \frac{t_1}{t_p}$ – ускорение счета; $E_p = \frac{t_1}{p \cdot t_p} \cdot 100\%$ – эффективность распараллеливания, где t_1 – время расчёта на одном узле используемой параллельной машины, t_p – время счёта на p узлах.

Для исследования эффективности распараллеливания использовалась задача об обжатии цилиндра в многообластной постановке. Размеры геометрии 10 × 1 с разбиением на 10 матобластей размером 1×1. Между матобластями решается задача контактного взаимодействия, на боковой поверхности цилиндра задано постоянное по времени давление $P_{zp} = 5$. Остальные границы являются жесткими стенками. Вещество: Идеальный газ с параметрами: $\gamma = 3$, $\rho_0 = 1$. В каждой области использовалась сетка диаграммы Вороного из 1 миллиона ячеек со случайной расстановкой центров. Фрагмент сетки приведен на рисунке 4. При расчете на одном вычислительном узле использовался только уровень распараллеливания на общей памяти. Это значение использовалось в качестве базового для оценки эффективности распараллеливания. При расчете на 10 узлах использовалось два уровня распараллеливания – «по областям» и OpenMP. При большем количестве узлов каждая матобласть разбивалась на параобласти. При расчете на 20 узлах – каждая матобласть на 2 параоласти, на 40 – на 4, и так далее. Примеры декомпозиций для одной области для различного количества процессов приведены на рис. 5.

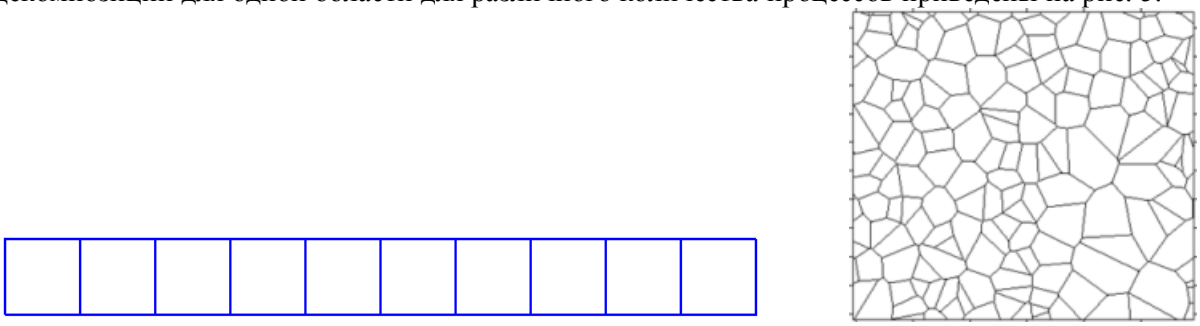


Рис. 4. Вид геометрии с разделением на области и фрагмент начальной сетки диаграммы Вороного

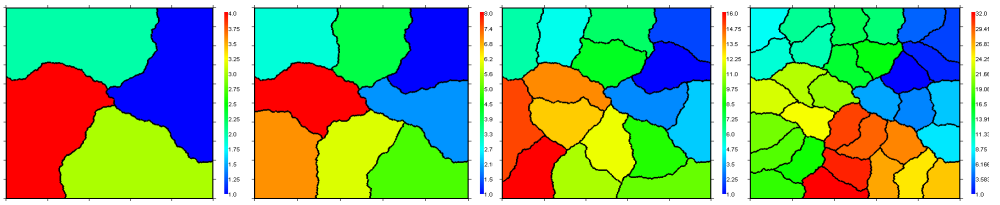


Рис. 5. Вид декомпозиции одной области для количества ядер 40, 80, 160 и 320

В таблице 1 приведены значения ускорения и эффективности распараллеливания. Расчеты проводились в модели смешанной памяти: по одному MPI процессу на каждый узел вычислительной системы, в каждом узле использовалось 8 процессорных ядер. В строке «MPI × OpenMP» первое число указывает количество MPI процессов, а второе – количество OpenMP нитей.

Таблица 1. Мелкозернистое распараллеливание с наложением

Режим	**OpenMP**	**По областям + OpenMP**	**Трехуровневое распараллеливание**			
Количество ядер	**8**	**80**	**160**	**240**	**320**	**400**
MPI × OpenMP	**1 × 8**	**10 × 8**	**20 × 8**	**30 × 8**	**40 × 8**	**50 × 8**
Ускорение, разы	1	8,92	16,08	21,61	23,18	24,5
Эффективность, %	100	89	80	72	58	49

8. Тестовый расчет

В качестве тестового расчета была выбрана газодинамическая задача о полете несферической оболочки [10]. Начальная геометрия задачи представлена на рисунке 6. Начальная форма оболочки $R(t=0,\theta)=7+0.875\cos^3\theta-0.525\cos\theta$, толщина $\Delta R=0.16$. Материал – свинец: $\rho_0=11.4$. Оболочка имеет скорость, направленную к центру: $v=5706860$. Давление на наружной поверхности оболочки $P_E=0$, давление на внутренней поверхности зависит от времени $P_i=3.495\cdot10^6\cdot(V(t))^{-5/3}$, где $V(t)$ – объём внутренней полости.

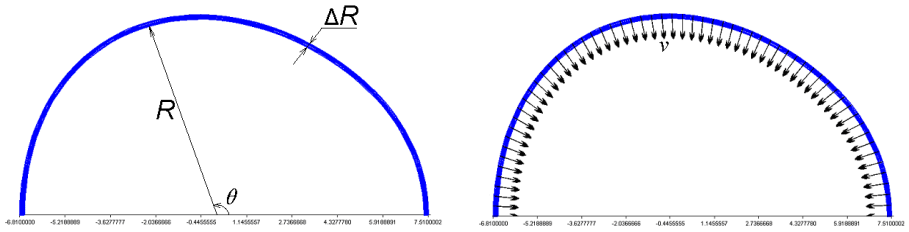


Рис. 6. Начальная форма и скорость задачи о полете несферической оболочки

Результаты расчетов приведены на рисунках 7–8. Во всех расчетах получено полное совпадение результатов между собой. Сравнение формы оболочки в последовательном и параллельном режимах показано на рисунке 7.

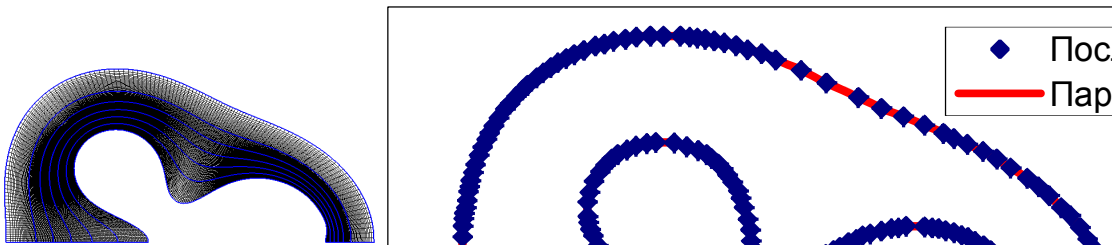


Рис. 7. Вид расчетной сетки на момент времени $t=1.1$ и сравнение формы оболочки в последовательном и параллельном расчетах

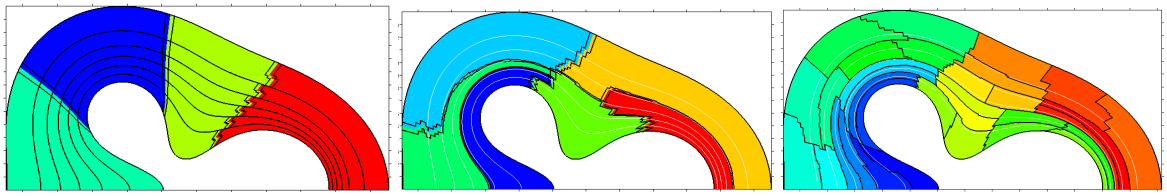


Рис. 8. Примеры результатов расчетов, в различных режимах распараллеливания

9. Заключение

В статье описывается метод трехуровневого распараллеливания, применяемый в методике ТИМ-2D. На первом уровне осуществляется распараллеливание счета по математическим областям. На втором уровне производится распараллеливание счета внутри математической области по параобластям. На первых двух уровнях используется модель распределенной памяти и интерфейс MPI. На третьем уровне осуществляется распараллеливание итераций счетных циклов в модели общей памяти с использованием интерфейса OpenMP. Уровни могут использоваться как по отдельности, так и в различных сочетаниях при решении одной задачи. При этом результаты расчетов в последовательном и параллельном режимах полностью совпадают. Использование метода трехуровневого распараллеливания позволяет гибко настроиться под конфигурацию вычислительной системы и решать задачи с использованием нескольких сотен процессорных ядер с эффективностью не менее 50%.

Литература

1. Соколов С.С., Воропинов А.А., Новиков И.Г., и др. Методика ТИМ-2D для расчета задач механики сплошной среды на нерегулярных многоугольных сетках с произвольным количеством связей в узлах // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2006. Вып. 4. С. 29-43.
2. Воропинов А.А., Новиков И.Г., Соболев И.В., Соколов С.С. Распараллеливание методики «ТИМ» в модели общей памяти с использованием интерфейса OpenMP // Вычислительные методы и программирование. 2007. Том 8. №1. С. 134-141.
3. Воропинов А.А., Соколов С.С., Новиков И.Г. Распараллеливание в модели смешанной памяти для расчета задач газодинамики в методике «ТИМ-2D» // Параллельные вычислительные технологии (ПаВТ'2008): Труды международной научной конференции (Санкт-Петербург, 28 января - 1 февраля 2008 г.). Челябинск: Изд. ЮУрГУ, 2008. С. 69 - 79.
4. Воропинов А.А., Соколов С.С., Новиков И.Г. Двухуровневое распараллеливание в модели смешанной памяти для расчета задач газодинамики в методике ТИМ-2D // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2008. Вып. 1. С. 51-59.
5. Воропинов А.А. Алгоритмы мелкозернистого распараллеливания в методике ТИМ-2D // Вычислительные методы и программирование. 2009. Том 10. С. 112-120.
6. Воропинов А.А., Новиков И.Г., Соколов С.С. Методы мелкозернистого распараллеливания в методике ТИМ-2D // Тезисы докладов XVIII Всероссийской конференции «Теоретические основы и конструирование численных алгоритмов решения задач математической физики», посвященной памяти К.И. Бабенко (Дюрсо, 13-17 сентября, 2010). М.: Институт прикладной математики им. М.В. Келдыша, 2010. С. 19.
7. Половникова Т.Н., Воропинов А.А. Опыт использования библиотек SCOTCH и MeTiS для декомпозиции неструктурированных сеток в методике ТИМ // Супервычисления и математическое моделирование. Труды XII международного семинара / под ред. Р. М. Шагалиев. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2011. С. 282 - 288.
8. Воропинов А.А. Декомпозиция данных для распараллеливания методики ТИМ-2D и критерии оценки ее качества // Вестник ЮУрГУ, серия «Математическое моделирование и программирование». 2009. №37(170), вып. 4. С. 40-50.
9. Воропинов А.А., Новиков И.Г., Соколов С.С. Расчет контактного взаимодействия между счетными областями в методике ТИМ-2D // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2008. Вып. 2. С. 5-20.
10. Sofronov I.D., Rasskazova V.V., Nesterenko L.V. The use of nonregular nets for solving two-dimensional nonstationary problems in gas dynamics // Numerical Methods in Fluid Dynamics / под ред. N. N. Yanenko, Ju. I. Shokin. Moscow: Mir Publishers, 1984. С. 82 - 121.

Организация СЭД на базе суперкомпьютера

Р.К. Газизов, А.Р. Мухтаров, Р.Р. Рахматуллин, И.У. Ямалов

ФГБОУ ВПО «Уфимский государственный авиационный технический университет»

Иллюстрируется возможность и преимущества построения системы межведомственного электронного документооборота органов государственной власти на базе суперкомпьютеров университетов. В качестве примера рассматриваются СЭД Республики Башкортостан на базе программного обеспечения JBOSS-Референт фирмы АйТи и суперкомпьютера ФГБОУ ВПО «УГАТУ».

1. Введение

В настоящее время система электронного документооборота (СЭД) становится одним из обязательных и существенных элементов ИТ-инфраструктуры современной организации, с помощью которой промышленные предприятия, коммерческие компании и другие учреждения повышают эффективность своей деятельности. В зависимости от конкретных условий задачи внедрения СЭД возможны различные варианты решений: готовое комплексное решение, решение на базе программной платформы с возможностью самостоятельной доработки или индивидуальная разработка. В большинстве случаев российские разработчики предлагают готовые решения, а иностранные предлагают платформенные решения [1]. Если для большинства учреждений и предприятий порядок и методы использования СЭД регламентируются корпоративными стандартами, то для государственных учреждений дело состоит иначе. Распоряжением Правительства Российской Федерации от 25 апреля 2011 г. №729-р утвержден перечень услуг, предоставляемых государственными и муниципальными учреждениями, а также другими организациями, которые должны включаться в реестр государственных (муниципальных) услуг, предоставляемых в электронной форме. Тем самым внедрение СЭД в органы государственной власти, как для внутреннего управления, так и для межведомственного взаимодействия и взаимодействия с населением становится задачей государственного уровня. Это, в свою очередь, требует от органов власти и управления различного уровня построения соответствующей ИТ-инфраструктуры, направленной на одновременное решение большего числа различных задач обработки или хранения информации. Очевидно, что для обработки данных необходимы многозадачные информационные системы, и многопроцессорные/многоядерные системы могут широко использоваться для этих целей. Кроме того, внедрение СЭД требует создание инфраструктуры, направленной на бесперебойность и надежность работы, катастрофа и отказоустойчивость, организацию защищённых каналов связи и т.д.

Во многом этим условиям удовлетворяют суперкомпьютерные центры, созданные в последние годы во многих ведущих университетах страны. Зачастую выделение для целей СЭД 50-100 процессоров не является критичным для основных работ, выполняемых на суперкомпьютере. Однако это позволяет, во-первых, сэкономить государственные ресурсы, во-вторых – приблизить реальные задачи к университету.

В данной работе предлагается решение по использованию классической схемы суперкомпьютера для организации СЭД республики, а так же анализируется преимущества и недостатки такого решения.

2. Особенности внедрения СЭД в ИТ-структуру учреждения

СЭД можно строить по различным технологиям, что предъявляет различные требования к программно-аппаратной архитектуре. Но при всем разнообразии систем можно выделить общие компоненты.

База данных (БД) – хранилище документов, которые на различных этапах обработки могут иметь различные формы, содержание, метки, метаданные и т.п.

Сервер приложений – программное обеспечение промежуточного уровня, обеспечивающее взаимодействие между клиентом и базой данных с документами. Сервер приложений должен обеспечивать функционирование ряда сервисов, среди которых можно выделить [2]:

- сервис управления потоками работ – обеспечивает маршрутизацию работ, заданий или документов любого типа в рамках бизнес процесса предприятия;
- сервис защиты и управления доступом к информации – обеспечивает решение задач контроля доступа к ресурсам СЭД, защиты информационного обмена между узлами СЭД, аудита деятельности пользователей, администраторов, обеспечения целостности данных, криптографической защиты информации;
- сервис управления контентом – обеспечивают процесс создания, доступа, контроля и доставки информации вплоть до уровня разделов документов и объектов.
- сервис предоставления информации – обеспечивает агрегирование, управление, доставку и унифицированный доступ к информации пользователей посредством коммуникационной структуры системы.

В основе функционирования СЭД лежит взаимодействие между БД и пользователем. И если технология построения и работы СУБД скрыта в недрах системы, то интерфейсная часть на стороне пользователя может обеспечиваться различными способами. Самым простым клиентом может выступать стандартный браузер, а в некоторых случаях целесообразным оказывается использование специализированной оболочки-клиента. Но независимо от способа реализации пользовательская компонента системы должна обеспечивать унифицированный и персонализированный доступ и работу всех категорий сотрудников организации ко всей информации.

Задачу внедрения СЭД необходимо увязывать со сложившейся в организации ИТ-инфраструктурой. Конечно, можно пытаться использовать готовое комплексное решение или платформу, пытаясь перестроить все бизнес-процессы под ее требования, но такой поход не всегда целесообразен и приемлем. В идеале хочется получить решение, которое влечет лишь минимальные изменения и адаптацию существующих информационных ресурсов. Немалое значение при этом имеет готовность пользователей к работе в СЭД в части их квалификации, а так же готовность технического персонала к обслуживанию и поддержке системы.

Относительно построения архитектуры системы в целом следует отметить несколько ключевых моментов. Во-первых, реализация СЭД должна обеспечить необходимую производительность, основным параметром оценки которой может выступать время реакции на типовые действия пользователей: авторизацию, открытие документов, создание новых документов, переход между документами, поиск и группирование документов в системе. Требуемые значения параметров производительности должны достигаться в конкретных масштабных ограничениях – количестве одновременно работающих пользователей, количестве и объеме хранящихся в системе документов. Параметры производительности должны оставаться удовлетворительными в течение среднего времени хранения документов в системе с учетом регулярного выполнения процедур обслуживания архива, включая вытеснение редко-используемых документов на медленные носители в системе хранения.

Вторым ключевым моментом при внедрении СЭД является возможность *масштабируемости* при дальнейшем развитии организации. Масштабируемость должна обеспечивать как просто подключение новых пользователей, так и добавление функционала в систему без остановки эксплуатации и существенного падения производительности. Наращивание же аппаратных ресурсов (увеличение емкости хранилища документов, серверных мощностей) должно происходить и вовсе незаметно для конечных пользователей.

Таким образом, внедрение СЭД подразумевает под собой целый комплекс организационных и технических задач, качество решения которых определяет стабильность и надежность функционирования системы. Качество же решения зависит, в свою очередь, от финансовых ограничений и от квалификации техперсонала. И если при построение СЭД республиканского уровня приобретение программно-аппаратной части требует финансовых вложений, на которые можно влиять административными силами, то внедрение и дальнейшая эксплуатация системы требует достаточно серьезных интеллектуальных ресурсов в ИТ сфере, которые не всегда имеются в наличии в органах власти. Наиболее перспективным в такой ситуации выглядит привлечение услуг компании-интегратора, имеющей опыт аутсорсинга подобных решений. Но не всякая компания-интегратор имеет в своем распоряжении полноценный центр обработки данных,

либо аренда его ресурсов может быть дорогостоящей. В качестве альтернативы такому решению может выступать аренда аппаратных мощностей у вуза, имеющего, с одной стороны, опыт работы с различными программными продуктами, и, с другой стороны – развитую ИТ инфраструктуру. При таком подходе актуальной становится оценка готовности ЦОД вуза к развертыванию СЭД, при этом на первый план выходит возможность использования имеющихся вычислительных ресурсов.

ЦОД вуза строится в первую очередь для решения собственных задач учебного заведения. В случае УГАТУ, ЦОД создавался для обеспечения функционирования суперкомпьютера, который используется, прежде всего, для решения больших вычислительных задач. На это ориентирована его архитектура, определяющая межузловую связность, и набор базовых приложений. Целью проведенного исследования стало изучение возможности использования ресурсов вычислительного суперкомпьютера для решения задачи развертывания системы электронного документооборота республиканского уровня.

3. Особенности использование Infiniband в качестве транспорта СЭД

С момента появления в 1999 году и до наших дней технология Infiniband [3] активно развивается, что привело к ее повсеместному использованию при построении высокопроизводительных вычислительных систем. Одним из факторов, оказавших большое влияния на этот процесс, стало распространение универсального программного обеспечения OFED (Open Fabrics Enterprise Distribution) [4]. Причиной распространения OFED явилось открытость его исходного кода, ориентация на высокопроизводительные сетевые приложения, требующие низкую латентность, высокая масштабируемость, поддержка протоколов удалённого прямого доступа к памяти (Remote Direct Memory Access, RDMA) и множества протоколов верхнего уровня, среди которых следует отметить SRP (SCSI RDMA Protocol) и iSER (iSCSI Extensions for RDMA). Распространение OFED позволило рассмотреть вопрос об использовании Infiniband в качестве транспорта для СЭД.

Следует отметить, что традиционная архитектура ЦОД предполагает использование FC как для системы хранения данных, так и для связи с вычислительными серверами (рис. 1а). При этом использованное в качестве базового при построении суперкомпьютера УГАТУ решение IBM System Cluster 1350 [5] предполагает использование Infiniband для связи между вычислительными серверами, тогда как FC используется только для связи с системой хранения данных. Для приведения архитектуры суперкомпьютера УГАТУ к традиционной потребовалось бы оснащение вычислительных серверов адаптерами FC, что влечет дополнительные финансовые затраты. Поэтому было принято решение об исследовании возможности построения СЭД на базе имеющейся производительной и высокодоступной телекоммуникационной инфраструктуры с поддержкой унифицированной матрицы коммутации, консолидирующей локальные сети (LAN), сети хранения (SAN) и системы хранения данных, а так же сети высокопроизводительных вычислений (InfiniBand) (рис.1б).

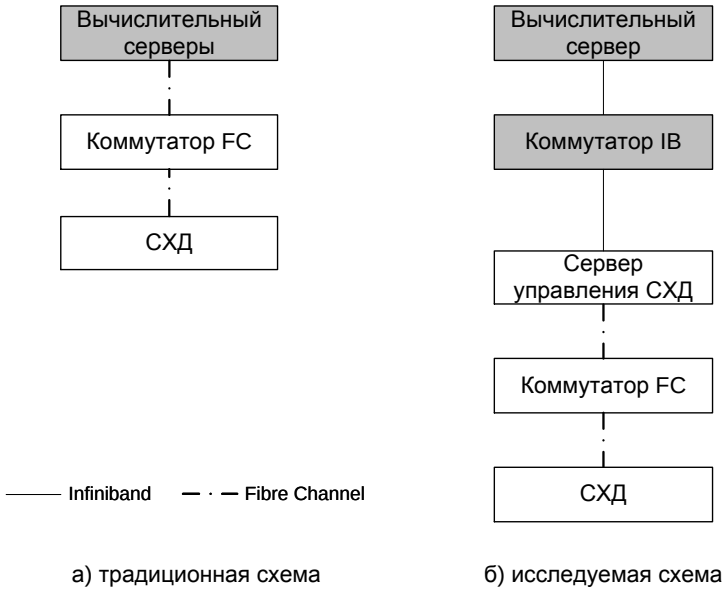


Рис. 1. Варианты инфраструктур для построения СЭД

Для выявления слабых и сильных сторон данного решения были собраны *тестовые стенды* и проведено нагрузочное тестирование, по результатам которого можно сделать выводы по возможности использования суперкомпьютеров для оказания услуг аутсорсинга вычислительных мощностей.

Нагрузочное тестирование проводилось на стенде следующей конфигурации:

- система хранения: HP MSA2312FC, массив на дисках SATA 750Gb, 16 дисков RAID6 + 4 hot spare – емкостью 10.4Tb;
- сервер СХД: HP DL360R05 E5430 2G EU Svr (Rack1U XeonQC 2.66Ghz (2x6Mb)/ P400i (256Mb/RAID5/1/0)/ iLO2std/ 2xGigEth), HP 8GB FBD PC2-5300 2x4GB LP Kit – 2 шт., HP 72.8GB Pluggable SAS 15,000 rpm Universal – 2 шт., HP FC2242SR PCI-e DC HBA, HP 4GB SW SinglePack SFP Transceiver– 2шт., HP NC364T PCI Express Quad Port Gigabit Ser;
- сервер приложений: IBM HS21EM / 2xXeon Quad E5345 / RAM 8Gb / HDD 73,4 Gb / IB HCA 4x;
- сервер базы данных: IBM HS21EM / 2xXeon Quad E5345 / RAM 8Gb / HDD 73,4 Gb / IB HCA 4x;
- сервер мониторинга: IBM HS21EM / 2xXeon Quad E5345 / RAM 8Gb / HDD 73,4 Gb / IB HCA 4x.

Тестирование проводилось для решения следующих задач: выбора наиболее подходящей технологии доставки данных при организации системы хранения и проверка возможности ее использования в системе электронного документооборота.

3.1. Тестирование программно-аппаратных архитектур

Основой построения сетей хранения данных SAN (Storage Area Network) является протокол высокоскоростной передачи данных Fibre Channel Protocol (FCP), обеспечивающий поддержку протокола SCSI в сетях Fibre Channel (FC). Изначально FC использовался в качестве среды передачи данных для суперкомпьютеров уступив в последующем технологии Infiniband. В настоящее время FC практически полностью перешел в сферу сетей хранения данных, где он используется как стандартный способ подключения к системам хранения данных уровня предприятия. В связи с тем, что архитектура современных суперкомпьютеров оптимизирована для решения больших вычислительных задач, вопросы обмена и хранения данных отходят на второй план. Вместе с тем, потенциал высокоскоростных межузловых связей и потенциал сетей хранения данных позволяют рассмотреть вопрос об использовании свободных ресурсов суперкомпьютера, построенного по стандартной архитектуре, для решения задач системы межведомственного

электронного документооборота. Решение этого вопроса может способствовать повышению эффективности использования ресурсов суперкомпьютера.

Для выбора технологии доставки было произведено сравнительное тестирование двух программно-аппаратных архитектур систем хранения данных (FC+iSER+IB и FC+SRP+IB) основанных на возможностях программного обеспечения OFED.

Протоколы iSER и SRP являются транспортом для протоколов iSCSI и SCSI соответственно. Данные протоколы транслируют команды через RDMA [6].

Тестирование перечисленных решений проводилось с использованием тестов bonnie++ [7] для следующих вариантов реализации дисковой системы:

- сервер с локальными дисками (physical.local);
- сервер с дисковой системой SRP (physical.srp);
- сервер с дисковой системой iSER (physical.iser).

Результаты измерения скорости обмена данными и латентности для различных вариантов реализации дисковой системы приведены в таблице 1.

Таблица 1. Результаты тестирования

Вариант	**Параметр**	**Последовательная запись**			**Последовательное чтение**		**Случайный доступ**
		посимвольная	блочная	перезапись	посимвольное	блочное	
physical.local	скорость, КБ/с	531	48265	24129	2287	6345	381
	задержка, мс	15	614	708	9	121	132
physical.srp	скорость, КБ/с	257	130492	71209	740	214532	5163
	задержка, мс	32	427	5401	19	58	278
physical.iser	скорость, КБ/с	526	154260	123285	2296	399503	1506
	задержка, мс	15	2309	507	14	191	1047

Проанализируем полученные результаты.

Последовательная запись

a. посимвольный вывод: локальный диск и iSER показали примерно одинаковые результаты, скорость SRP в два раза ниже, при этом в два раза выше задержки;
b. блочная запись: SRP быстрее локального диска в 2.7 раза, iSER быстрее локального диска в 3.2 раза;
c. перезапись: SRP быстрее локального диска почти в 3 раза, iSER быстрее локального диска в 5.1 раза.

Последовательное чтение

a. посимвольный ввод: локальный диск и iSER показали примерно одинаковые результаты, результаты SRP в два раза хуже, задержки iSER и SRP выше локального диска;
b. блочное чтение: SRP быстрее локального диска в 3.4 раза, iSER быстрее локального диска в 6.3 раза.

Случайный доступ

a. SRP быстрее локального диска в 13.5 раз, iSER быстрее локального диска почти в 4 раза.

Задержки в целом либо сопоставимы, либо в разы выше локального диска, что объясняется наличием промежуточных протоколов между дисковой подсистемой и серверами.

Построим график сравнения скорости обмена данными в случаях с минимальной производительностью (рис.3).

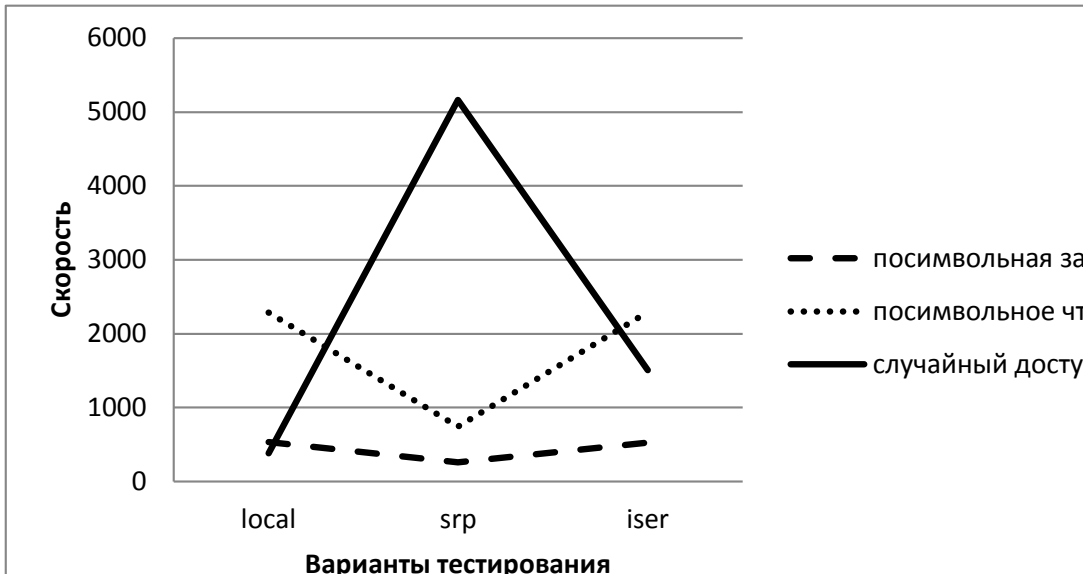


Рис. 2. Скорость обмена данными в различных реализациях дисковой подсистемы

Результаты тестирования показывают, что для построения СЭД (обращение к базам данных) решение FC+SRP+IB является более подходящим для поддержки системы хранения.

3.2. Нагрузочное тестирование работоспособности

3.2.1. Цель, условия и методика тестирования

Целью нагрузочного тестирования являлось проверка соответствия оборудования для обеспечения работоспособности СЭД при имитации реальной эксплуатации. Тестирование проводилось специалистами компании АйТИ по смешанной функционально-автоматизированной схеме. Часть нагрузки формировали активно работающие пользователи, которые создавали, регистрировали, подписывали, согласовывали, рассматривали, исполняли документы. Другая часть нагрузки обеспечивалась виртуальными пользователями, которые имитировали следующую работу: авторизация, переход по вкладкам, просмотр документов.

Нагрузка на систему подавалась в три этапа в течение 30 минут с последовательным увеличением. Такой способ дал возможность измерить и проанализировать как время отклика системы, так и тенденции изменения загрузки оборудования при изменении нагрузки.

Количество сессий, открытых на каждом этапе нагрузочного тестирования, приведено в таблице 2.

Таблица 2. Количество сессий на различных этапах

Вид нагрузки/ Этапы	открытие вкладок	открытие документов	создание документов	пользователи помощники	всего
Этап 1	150	118	120	12	400
Этап 2	380	70	120	12	582
Этап 3	550	150	120	12	832

При проведении тестирования использовались два типовых сценария:

- *Сценарий перехода по вкладкам.* Представленный сценарий эмулирует действия пользователей, которые авторизуются в системе и переключаются между вкладками.

- *Сценарий открытия документов*. Представленный сценарий эмулирует действия пользователей, которые авторизуются в системе, переключаются между вкладками и открывают документы.

Условия проведения тестирования накладывали некоторые ограничения, в частности не была использована виртуализация и отсутствовала балансировка нагрузки, но для анализа возможности функционирования системы в целом погрешностями, вносимыми этими ограничениями, было решено пренебречь.

3.2.2. Анализ времени отклика системы

Результаты тестирования и замеров среднего времени отклика системы при изменении нагрузки на различных этапах, приведены в таблице 3.

Таблица 3. Среднее время отклика при увеличении нагрузки

Показатель	1 этап (сек)	2 этап (сек)	3 этап (сек)
Авторизация	0,5	0,6	0,8
Открытие документа	5,1	6,2	6,6
Создание новой карточки документа	4,2	5,3	5,6
Переход между вкладками	9,8	10,2	17,6
Действия помощников, переход по вкладкам и открытие документов	7,45	14,1	14,6
Переход документа в следующий статус	4,75	4,86	5,24

Для наглядного представления результатов построим по полученным данным график зависимости среднего времени отклика (в секундах) от количества активных пользователей. Пользователи переключались по вкладкам, просматривали документы и создавали новые.

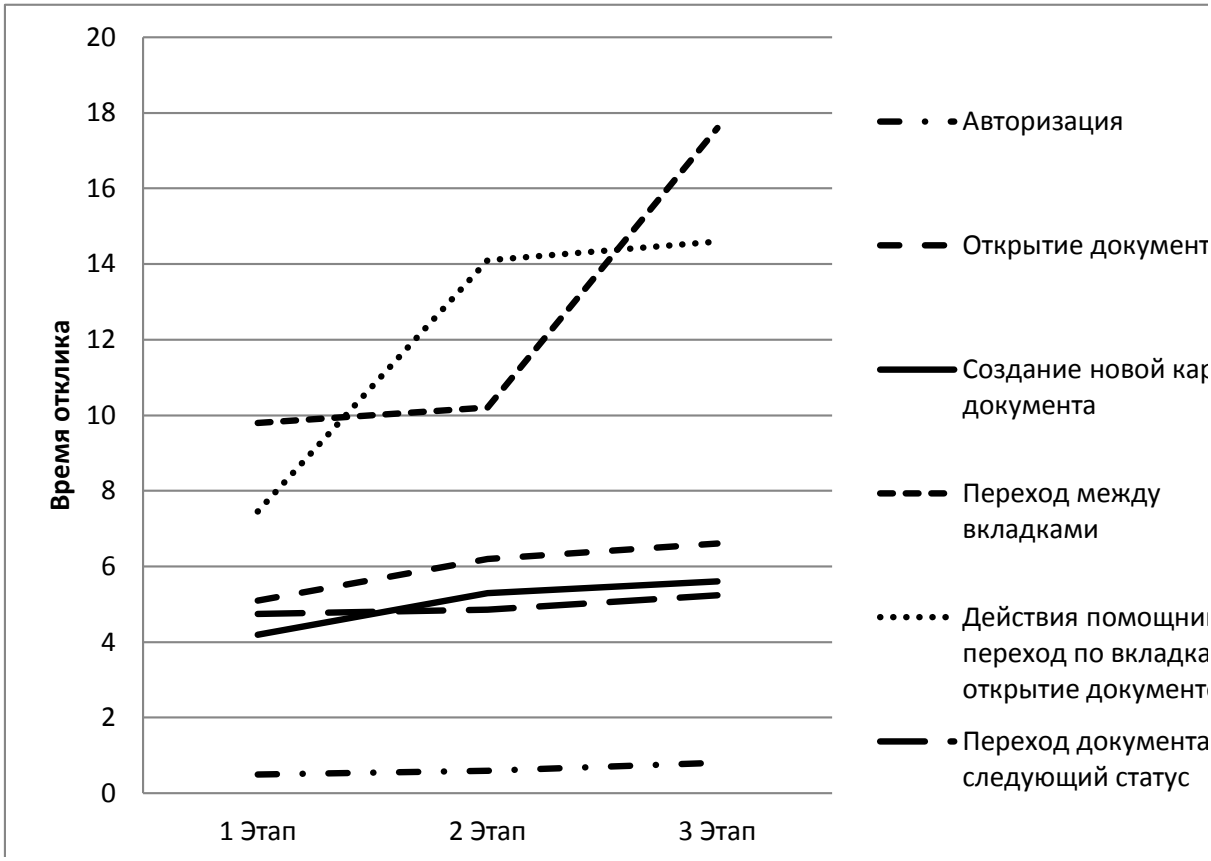


Рис. 3. График отклика системы

Рисунок 3 показывает то, что при максимальной нагрузке более 800 пользователей время отклика системы в допустимых пределах.

Уменьшение времени отклика на втором этапе тестирования на открытие новой карточки документа и на переход документа в следующий статус обуславливается уменьшением в 2 раза количества пользователей создающих документы по сравнению с первым этапом. На третьем этапе было увеличено число как активных, так и пассивных пользователей. Таким образом, можно сделать вывод о том, что уменьшение активных пользователей незначительно влияет на общую загрузку системы и на среднее время отклика.

3.3. Анализ загрузки оборудования

Для анализа загрузки оборудования на всех этапах нагрузочного тестирования проводились измерения количественных характеристик различных параметров, таких как загрузка процессора, использование ОЗУ, нагрузка на дисковую систему. Измерения проводились как для сервера баз данных, так и для сервера приложений. Значения максимальной нагрузки на различных этапах для сервера БД и сервера приложений Jboss приведены в таблицах 4 и 5.

Таблица 4. Максимальная загрузка сервера приложений Jboss

	Загрузка процессоров, %	Загрузка ОЗУ, %	Загрузка дисковой подсистемы, %
Этап 1	4.6	43.7	5.96
Этап 2	1.5	50.5	6.14
Этап 3	2.5	32.2	6.15

Таблица 5. Максимальная загрузка сервера БД

	Загрузка процессоров, %	Загрузка ОЗУ, %	Загрузка дисковой подсистемы, %
Этап 1	21.5	83.0	48.27
Этап 2	21.5	98.9	48.27
Этап 3	11.5	98.9	48.80

Для дальнейшего анализа полученных результатов построим по полученным данным сравнительные графики зависимости загрузки серверов от тестовой нагрузки на различных этапах (рисунки 4-6).

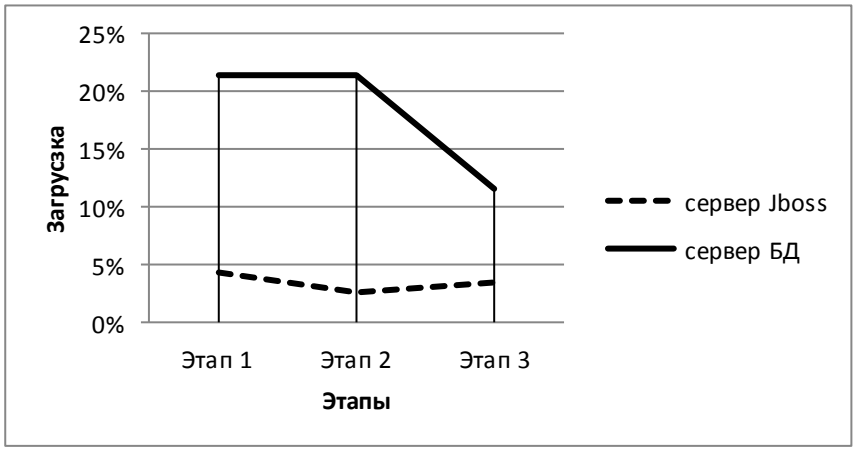


Рис. 4. Загрузка процессоров

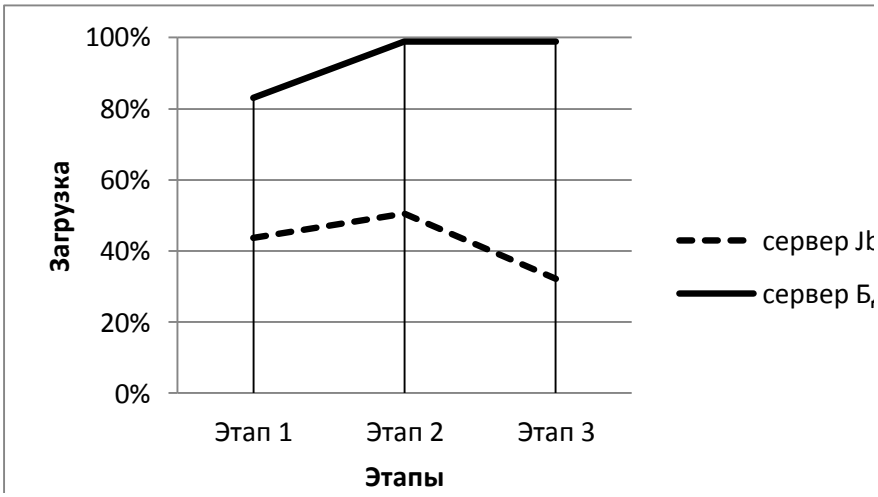


Рис. 5. Загрузка ОЗУ

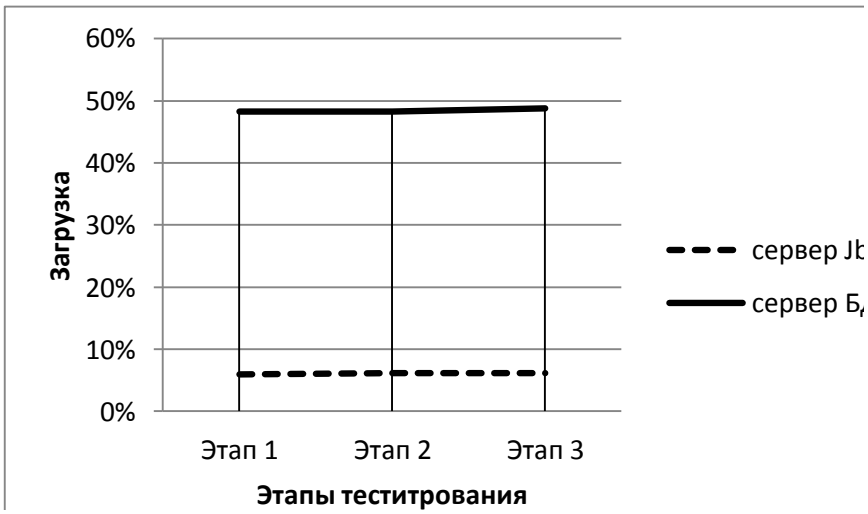


Рис. 6. Загрузка дисковой подсистемы

Как видно из графиков, основная нагрузка, создаваемая пользователями, приходится на сервер с базой данных. При этом наиболее значительную нагрузку испытывает оперативная память, тогда как на такие ресурсы как процессоры и дисковые подсистемы нагрузка незначительна.

Учитывая режим работы системы как слабо нагруженный или средне- нагруженный, то приведенная схема тестирования допускает приемлемую работу 600 пользователей с возможностью нарастания пиковой нагрузки до 800 пользователей.

Выводы

По результатам проведенного тестирования можно сделать следующие выводы.

Построение межведомственного электронного документооборота органов государственной власти на базе стандартной производительной и высокодоступной телекоммуникационной инфраструктуры с поддержкой сети высокопроизводительных вычислений (InfiniBand) становится возможным при минимальной адаптации и настройке программного обеспечения путем установки OFED.

Таким образом, можно утверждать, что использование высокопроизводительных вычислительных установок, построенных на основе технологии Infiniband, в качестве аппаратной платформы для развертывания СЭД, позволяют повысить эффективность их использования. Вместе с тем, использование такого решения позволяет сократить расходы государственных органов власти на развитие и поддержку IT инфраструктуры для обеспечения выполнения распоряжения Правительства РФ о предоставлении услуг в электронной форме.

Литература

1. *Рейнгольд Л.* Обзор систем электронного документооборота. http://www.ixbt.com/soft/sed.shtml.
2. *Антимонов Д.* СЭД осваивает регионы. http://www.cnews.ru/reviews/free/dms2007/articles/practice_1.shtml
3. *Дайерлинг К.* InfiniBand: архитектура коммутации для серверов, запоминающих устройств и коммуникационных систем // Мир компьютерной автоматизации. – 2002. – №3
4. OpenFabrics Alliance (OFA). http://www.openfabrics.org.
5. В УГАТУ запущен самый мощный в России суперкомпьютер IBM. http://www.cnews.ru/news/line/index.shtml?2007/12/19/280310
6. High-Performance Storage Solutions using RDMA http://www.voltaire.com/download/whitepapers/High%20Performance%20Storage0605.pdf
7. Тест bonnie++. http://www.coker.com.au/bonnie++

Система управления суперкомпьютером SCMS

А.Л. Головинский[1], А.Л. Маленко[1], Л.Ф. Белоус[2], С.В. Баранник[3]

Институт кибернетики им. В.М. Глушкова НАН Украины[1],
Физико-технический институт им. Б.И. Веркина НАН Украины[2],
Институт сцинтилляционных материалов НАН Украины[3]

В работе представлена система управления суперкомпьютером, соединяющая в себе простой интерфейс и широкие возможности. Система предназначена как для обычных пользователей суперкомпьютера, так и для администраторов. Поддерживается большое количество аппаратных решений.

1. Введение

Проблема понятного и доступного пользователю программного интерфейса является актуальной и важной для всех видов программного обеспечения. Особенно это касается сферы высокопроизводительных вычислений, где традиционные консольные интерфейсы для доступа пользователей очень специфичны и требуют дополнительных технический знаний. Кроме особенностей своей научной области, для работы с суперкомпьютером пользователь должен понимать работу операционной системы кластера, процесс запуска задач, работу с компиляторами и т.д.

Развитие грид-технологий не улучшило текущее состояние вещей. Ведь работа в гриде – это еще один дополнительный уровень сложности, который требует знания грид-инструментов командной строки, нового синтаксиса запуска задач, межкластерной совместимости программных окружений и т.д.

С другой стороны, задача администрирования кластерных систем остается сложной и трудоемкой, требует высокого уровня квалификации администратора, при этом отнимает много времени.

Система управления суперкомпьютером SCMS 4.2 – это попытка предложить комплексное решение как для доступа пользователей, так и для администрирования кластеров. Она представляет собой веб-приложение, на котором пользователи могут легко запускать и контролировать свои задачи без необходимости изучать многочисленные детали работы суперкомпьютера и операционного окружения грида.

Для администрирования кластера в системе предусмотрены удобные средства мониторинга состояния оборудования суперкомпьютера, управления ресурсами кластера, очередями задач и пользователями.

Проект SCMS является продолжением работ, посвященных системам управления, интерфейсам суперкомпьютера и работе в гриде [1–3].

Система управления суперкомпьютером SCMS 4.2 является готовым продуктом, который успешно используется на грид-кластерах Института кибернетики им. В.М.Глушкова НАН Украины, г. Киев, Института сцинтилляционных материалов НАНУ, в Физико-техническом институте низких температур им. Б.И. Веркина НАН Украины, г. Харьков и Институте механики сплошных сред РАН, г. Пермь.

2. Интерфейс пользователя

В рамках проекта решена задача построения эффективного интерфейса как специализированного инструмента, имеющего, с одной стороны, гибкие возможности, а с другой стороны, достаточного простого и интуитивно понятного для современного специалиста.

Исследуя типовые способы работы пользователей на украинских суперкомпьютерах, мы отметили три главных направления: работа ученого, грид-пользователя и программи-

ста. Для каждого из этих типов работы предложены соответствующие средства, обеспечивающие удобную и эффективную работу.

Большинство ученых, использующих кластер, работают с пакетами готового программного обеспечения. Для них разработана простая и удобная среда для редактирования исходных файлов, запуска параллельных программ и он-лайн просмотра результатов задач.

Прикладные программисты, в свою очередь, используют кластер в качестве инструмента разработки и тестирования параллельных программ. Для этого создано окружение для компиляции с поддержкой популярных компиляторов и библиотек, а также редактор исходного кода с подсветкой синтаксиса.

Ключевым элементом интерфейса пользователя является форма запуска вычислительных задач (рис. 1), позволяющая запускать как программы собственной разработки, так и большие программные пакеты, установленные на кластере. Форма оборудована системой профилей, облегчающих повторные запуски задач и предустановленных пакетов ПО.

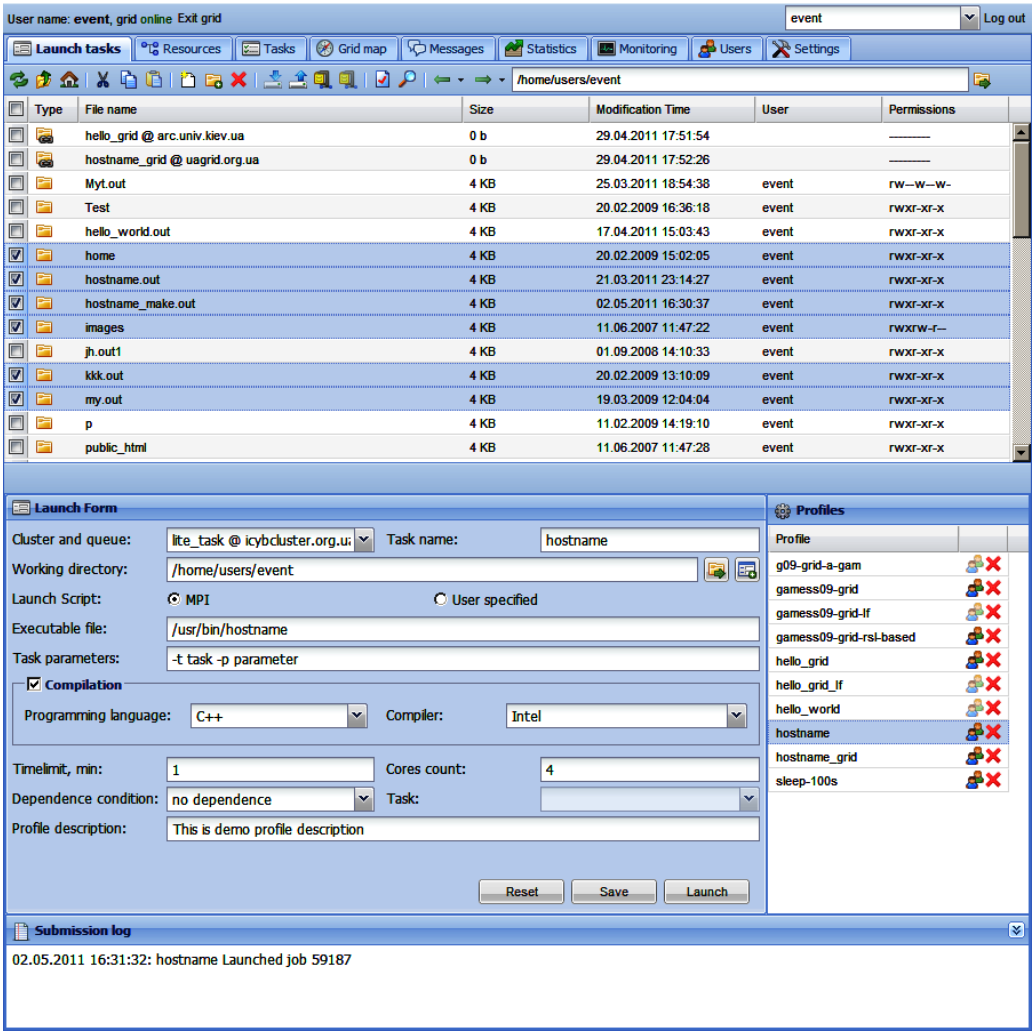


Рис. 1. Интерфейс пользователя SCMS

Для исходных файлов предусмотрена интеллектуальная система компиляции. Она определяет язык программирования и выбирает соответствующий сценарий компиляции. Поддерживаются сценарии языков программирования C/C++, Fortran 77, 90, 95 для компиляторов Intel и GNU. Для гибридного кластера на GPU-ускорителях доступно автоматическое построение сценария компиляции кода программы на языках CUDA и OpenCL.

Работа в гриде, в том числе запуск грид-задач, и на локальном кластере максимально

унифицированы. С точки зрения пользователя практически нет разницы, где запускается его задача: на локальном кластере или в гриде. Для полной поддержки грид-технологий пользователю достаточно иметь действительный грид-сертификат и грид-пароль.

3. Система контроля оборудования

Основной задачей суперкомпьютера является предоставление безотказного, круглосуточного, надежного и безопасного сервиса высокопроизводительных вычислений.

Основным способом решения этой задачи является создание единой интеллектуальной системы управления оборудованием вычислительного комплекса, которая может обеспечить непрерывный контроль всего аппаратного обеспечения (вычислительных узлов, хранилища, системы охлаждения, сетевого оборудования, каналов связи, блоков бесперебойного питания и т.д.) и базового программного обеспечения.

Было проведено исследование основных видов неисправностей и поломок больших вычислительных систем, приводящих к серьезным нарушениям их работы. Неисправности классифицированы по степени критичности и вероятности возникновения. Например:

- выход из строя системы кондиционирования, который приводит к перегреву всего кластера;
- выход из строя дисков системы хранения;
- выход из строя узлов;
- выход из строя различных программных сервисов.

Существующие системы мониторинга и диагностики, например, Ganglia и Nagios, успешно решают задачу получения подробной информации о текущем состоянии кластера по требованию администратора. Эти системы можно самостоятельно дополнить модулями извещения администратора по E-mail/SMS о критических событиях, например, перегреве узлов.

Однако, такие меры являются недостаточными, так как они предполагают наличие дежурного специалиста, который постоянно следит за состоянием кластера и в любой момент сможет решить возникшую проблему. На практике, такая методика имеет множество недостатков. Логично поручить заботу о ”здоровье” суперкомпьютера самому суперкомпьютеру.

Авторами проанализирован опыт администраторов ряда суперкомпьютеров и выделены типовые сценарии поведения администратора во время выявления проблем аппаратного обеспечения и базового ПО кластера. Большинство из них можно и нужно автоматизировать.

Разработанная система SCMS позволяет оперативно решать проблемы кластера в автоматическом и полуавтоматическом режиме. Ее интеллектуальность заключается в возможности самостоятельного выявления опасной ситуации, ее оценки и принятия решения про выполнение необходимых действий.

Опишем для примера модуль автоматического отключения оборудования кластера при перегреве.

При отсутствии средств автоматического выключения выход из строя одного кондиционера системы охлаждения провоцирует эффект ”домино”: остальные кондиционеры работают на грани своих возможностей и один за другим выходят из строя, в течение короткого промежутка времени (10-60 минут) наступает полный коллапс системы охлаждения. Температура в кластерном зале возрастает до 60–70°C и выше, вследствие чего начинают начинают выходить из строя узлы.

Такого сценария можно избежать, если кластер контролирует ситуацию. После отключения части охладителей термодинамический баланс оказывается разбалансированным, возникают зоны перегрева, при этом другая часть кластера может работать в штатном режиме.

Модуль защиты находит такие зоны и отключает узлы, оказавшиеся под угрозой, уменьшая тепловыделение. Последовательное выключение узлов кластера продолжается до наступления нового термодинамического равновесия. Такой подход рациональнее полного выключения кластера.

Эффективность системы продемонстрировала ее успешная эксплуатация на ряде украинских кластеров с 2009 года. Благодаря этой системе процент аварий сведен практически к нулю, что позволило повысить надежность работы суперкомпьютеров, уменьшить количество простоев, сэкономить на устранении аварий.

4. Обзор возможностей администратора

Интерфейс рабочего места администратора обеспечивает удобное выполнение основных действий по управлению вычислительным комплексом:

- управление узлами и разделами кластера;
- управление потоком задач;
- контроль оборудования кластера;
- управление учетными записями пользователей;
- выполнение диагностических задач;
- анализ статистики использования ресурсов;
- контроль качества предоставляемых сервисов.

Администратор создает разделы кластера из его узлов, указывает образы операционных систем узлов, включает, выключает и перезагружает узлы.

Управление менеджером ресурсов позволяет выбрать оптимальный алгоритм планировщика и приоритеты пользователей, а также просматривать очереди задач и отменять их.

Интерфейс администратора позволяет визуально контролировать основные характеристики работы кластера. В состав системы входит модуль оповещения о критических событиях с помощью электронной почты или SMS.

Администратор имеет полный комплекс средств для управления учетными записями пользователей: прием запроса регистрации пользователя, редактирование данных учетной записи, удаление пользователей.

Модуль учета системы SCMS постоянно собирает информацию об использовании ресурсов кластера и состоянии оборудования, позволяет создавать отчеты по пользователям, организациям за любой период времени. Статистика отображается в системе в табличном и графическом виде. Ее можно экспортировать в CSV или Excel формат.

Авторами разработан метод оценки качества предоставления сервиса [4]. На основании собранной статистики система автоматически вычисляет индекс ожидания ресурсов. Визуальное отображение индекса по неделям или месяцам дает администраторам четкое понимание того, насколько быстро обслуживаются запросы пользователей, позволяет оценить зависимость качества предоставленных ресурсов от нагрузки и структуры потребления.

На основе индекса ожидания администратор принимает решение об изменении параметров счетного поля и алгоритма планировщика задач, а также выставляет приоритеты и квоты пользователям.

5. Выводы

Описанная в данной статье система удовлетворяет основные потребности пользователей и администраторов суперкомпьютера. Разработанные средства позволяют работать на кластере неспециалистам, сократить время обучения пользователей работе на суперкомпьютере.

Таким образом, система способствует более широкому применению отечественных многопроцессорных вычислительных систем в науке и промышленности, поскольку упрощает их использование учеными и программистами.

Система SCMS позволяет разгрузить администраторов кластеров от рутинной работы, переключить их внимание на более важные глобальные задачи.

Разработанная система доступна в сети Интернет для публичного ознакомления с ее функционалом [6]. Параметры тестовой учетной записи: логин – *demo*, пароль – *cluster*.

Литература

1. Якуба А.А., Головинский А.Л., Бандура А.Ю., Горенко С.А., Ефременюк Д.А. Портал кластерных вычислений для управления вычислительными процессами на суперкомпьютерном комплексе // Кибернетика и системный анализ. 2009. 6. 97–105.

2. Головинский А.Л., Белоус Л.Ф., Маленко А.Л. Веб-портал системы управления суперкомпьютером // Вычислительные методы и программирование. 2010. 11/7.

3. Головинський А.Л., Маленко А.Л., Черепинець В.В. Система керування суперкомп'ютером з підтримкою роботи у гріді SCMS 4.0 // Матеріали міжнародного наукового конгресу з розвитку інформаційно-комунікаційних технологій та розбудови інформаційного суспільства в Україні. — 2011. — С. 28–29.

4. Головинський А.Л., Маленко А.Л. Аналіз ефективності планувальників черги задач для суперкомп'ютера з кластерною архітектурою // Матеріали конференції Високопродуктивні обчислення HPC-UA 2011. — 2011. — С. 64–69. Перевод статьи на русский язык: http://melkon.com.ua/files/article-scheduling-rus.pdf

5. Документация системы SCMS: http://melkon.com.ua/

6. http://scms.melkon.com.ua/

Разработка расчетно-экспериментального комплекса на базе супер-ЭВМ для анализа процессов в энергетических установках*

Н.В. Горбушина, П.В. Писарев, В.Я. Модорский

ФГБОУ ВПО «Пермский национальный исследовательский политехнический университет»

Разработан расчетно-экспериментальный комплекс для численного и физического моделирования междисциплинарных задач механики сплошных сред, в частности изучения и верификации динамических процессов в системе «поток газа – деформируемая конструкция»

Действие на динамические системы различных возмущений приводит к тому, что их выходные характеристики отклоняются от расчетных. При определенных условиях возможно возникновение колебаний с увеличивающейся во времени амплитудой, появляются неустойчивые режимы работы, что, как правило, недопустимо.

Неустойчивость существенно снижает надежность конструкции, ухудшает ее рабочие характеристики и может привести к разрушению. Поэтому, выявлению причин неустойчивости рабочих процессов, ликвидации колебаний или снижению их амплитуды до допустимого уровня уделяется большое внимание.

Различные неустойчивые рабочие процессы реализуются при наличии возмущений, образующих волны давления. Источником неустойчивости может являться резонансное взаимодействие системы «поток газа – конструкция» [1]. Резонанс возникает, когда частота колебаний давления газа близка к частоте колебаний конструкции. Частота и форма наблюдающихся при этом волн зависят от геометрических и физико-механических характеристик конструкции и от параметров нагрузки.

Для поиска опасных резонансных режимов в газонаполненной конструкции необходимо провести комплексное экспериментально-теоретическое исследование параметров динамического поведения системы «поток газа – конструкция». Данный подход предполагает проведение взаимодополняющих физического и вычислительного экспериментов. Вычислительный эксперимент по исследованию процессов в динамической системе «газ-конструкция» предполагает использование высокопроизводительного вычислительного комплекса ПНИПУ, производительностью 4 ТФлопса.

Подготовку реализации расчетно-экспериментального подхода можно провести в два этапа. На первом этапе - реализация вычислительных экспериментов, моделирующих работу экспериментальной установки и уточнение ее проектных параметров. На втором этапе – проектирование, изготовление установки и проведение экспериментальных работ по исследованию динамических процессов и верификации математических моделей.

Для реализации первого этапа предполагается использование системы инженерного анализа ANSYS CFX и ANSYS Mecanical. Рассматривается консольный тонкостенный конечномерный цилиндр, нагруженный изменяющимся внутренним давлением (**Рис.1**).

Для проведения численного эксперимента данной работы были задействованы 2 восьмиядерных вычислительных узла кластера Центра высокопроизводительных вычислительных систем ПНИПУ. Расчеты проводились по схеме, показанной на **Рис. 2**. Математическая модель изложена в [2]. В качестве приоритетной исполнительной среды была выбрана Windows Compute Cluster Server 2003. Вместе с тем, на кластере ПНИПУ реализована возможность одновременной работы разноплатформенных операционных систем Linux и Windows. Использование Windows Compute Cluster Server 2003 позволило пользователям

* Работы выполнены при поддержке гранта РФФИ №11-07-96003.

работать в более привычной среде, так как работа под Linux зачастую требует специальных знаний, это в свою очередь затрудняет работу рядовых инженеров-пользователей.

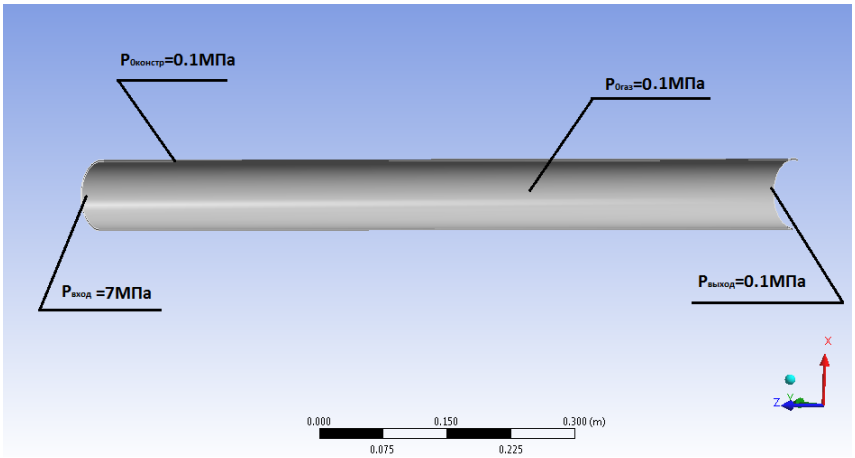


Рис. 1. Граничные условия

В ходе вычислительный экспериментов имеется возможность расчета временных зависимостей для перемещений в оболочке и давлений в газодинамической полости. Следовательно, имеется возможность оценки частот как в газе, так и в конструкции. Их совпадение свидетельствует о резонансе в системе.

Рис. 2. Схема проведения вычислительного эксперимента на кластере

На **Рис. 3** и **Рис. 4** приводятся изображения расчетной сетки для конструкции (300000 ячеек) и газодинамической полости (800000 ячеек), соответственно.

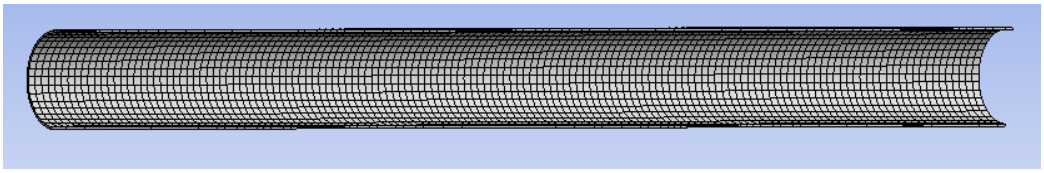


Рис. 3. Общий вид расчетной сетки для упругой трубы

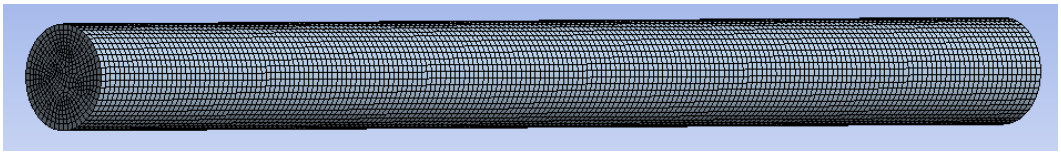


Рис. 4. Общий вид расчетной сетки для проточной части

Проводится анализ переходных процессов в газодинамическом потоке и конструкции. Рассчитываются поля давлений и скоростей в потоке газа (**Рис.5,6**).

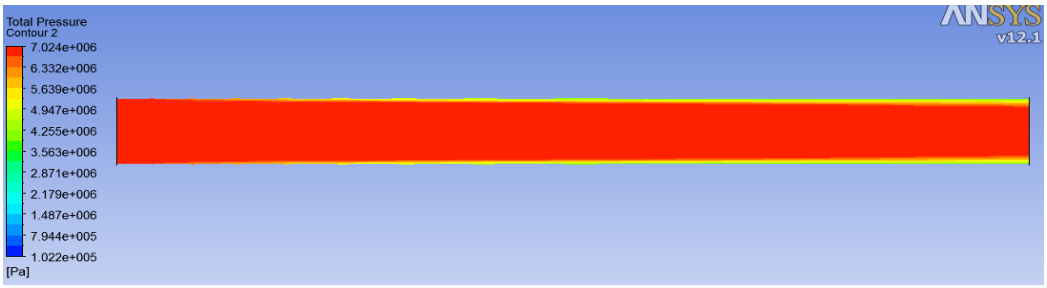


Рис. 5. Распределение полного давления по продольному сечению трубы

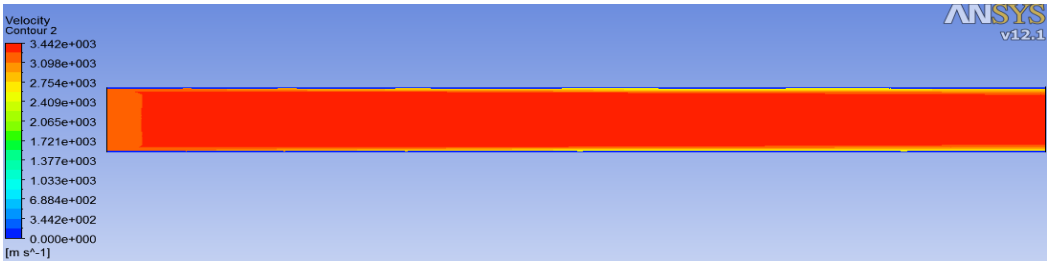


Рис. 6. Распределение модуля скорости по продольному сечению трубы

Производится оценка напряженно-деформированного состояния цилиндрической конструкции (**Рис.7,8**).

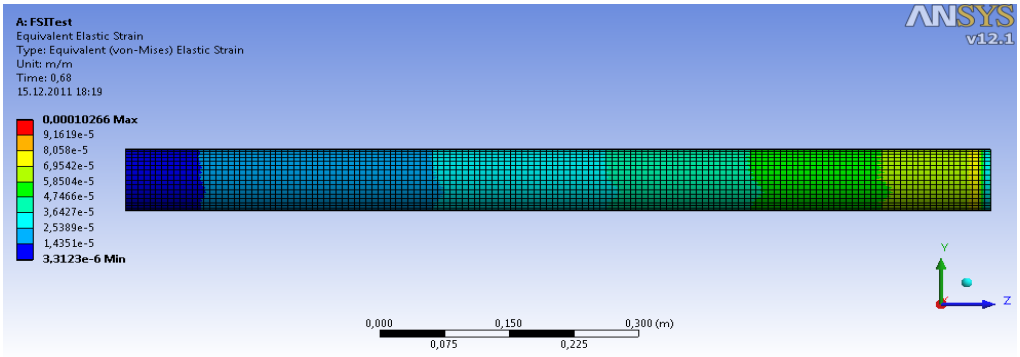


Рис. 7. Деформации по Мизесу

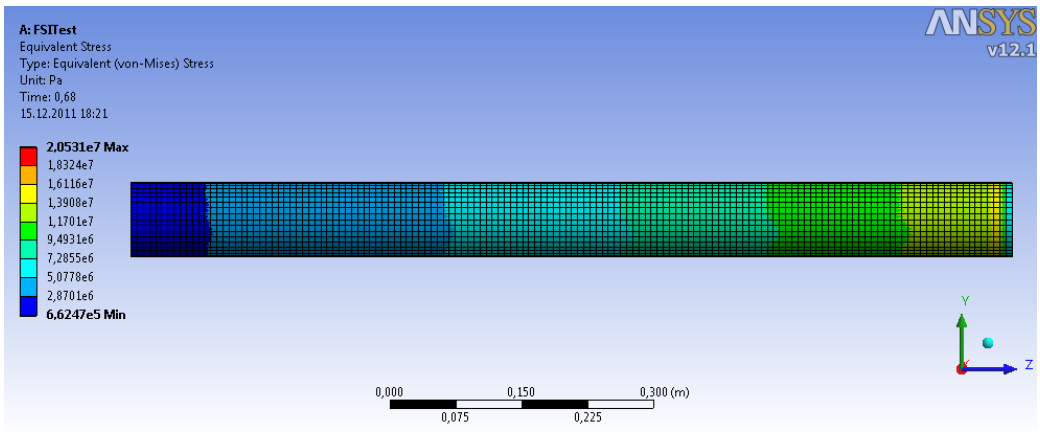


Рис. 8. Напряжения по Мизесу.

Производится гармонический анализ колебательных процессов в газе и конструкции.

На **Рис. 9** представлен график масштабируемости - зависимость ускорения (по вертикали) от количества расчетных ядер (по горизонтали).

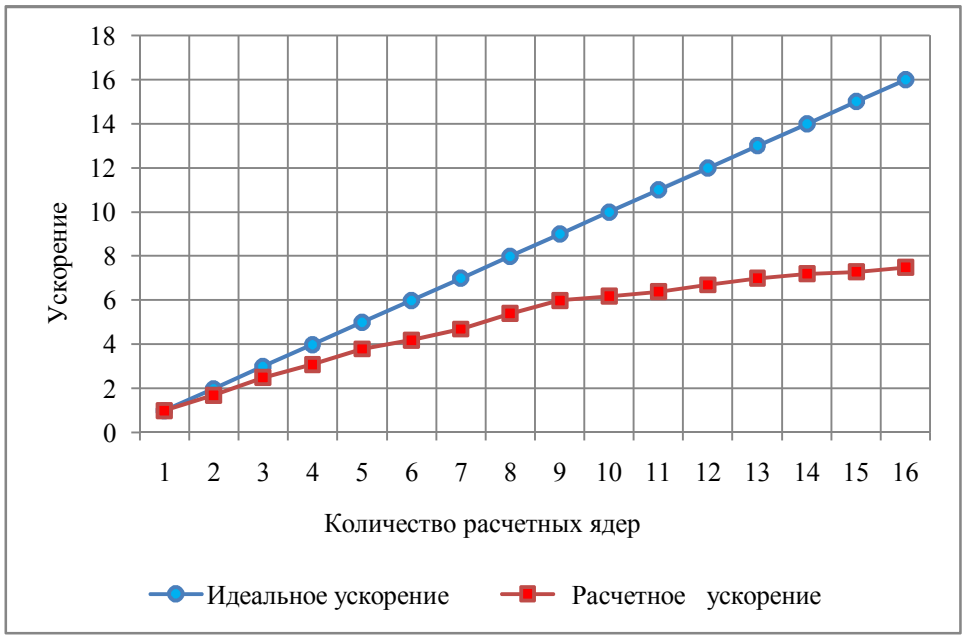


Рис. 9. График масштабируемости

Из графика видно, что эффективное ускорение для данного класса задач, количества расчетных узлов и используемого ПО наблюдается при использовании от 2 до 8 ядер, при последующем увеличении количества ядер до 16 ускорение резко снижается.

Для верификации численных расчетов и проведения физических экспериментов разработана специальная модельная установка, конструкция которой обеспечивает выполнение условий резонансного взаимодействия потока газа с деформируемой конструкцией.

В рамках этого этапа сформулированы требования к рабочим параметрам и конструкции и в соответствии с ними разработана экспериментальная установка.

Экспериментальная установка состоит из следующих основных частей: стапель, модельная камера, система подачи и отвода рабочего тела, измерительный комплекс.

Сформулированы следующие требования, предъявляемые к экспериментальной установке:

− стапель должен обладать необходимым запасом прочности и жесткости, обеспечивать необходимую ориентацию и надежное крепление средств эксперимента;

− конструкция модельной камеры должна обеспечивать настройку резонансного взаимодействия в системе «поток газа – конструкция»; возможность оснащения корпуса модельной камеры необходимыми средствами измерения, с помощью которых можно получить информацию о параметрах потока рабочего тела и уровне деформаций конструкции в контрольных точках; возможность установки сменных корпусов, отличающихся материалом и геометрическими характеристиками; модельная камера не должна является сосудом высокого давления;

− система подачи рабочего тела должна иметь предохранительный клапан, редуктор, автоматические пуск-выключение и обеспечивать постоянное давление на входе (до 1 МПа) с возможностью контроля по манометру; система отвода рабочего тела должна обеспечивать безопасную эксплуатацию установки;

− измерительный комплекс должен обеспечивать высокую надежность и точность измерений; помехоустойчивость измерительной аппаратуры при работе ее в условиях стенда; многоканальность системы измерения; регистрацию и обработку временных зависимостей на персональном компьютере.

− для повышения безопасности при работах на установке необходимо использовать бронекабину.

Стапель конструктивно выполняется в виде силовой рамы с ложементом, на которую вертикально устанавливается модельная камера. Ложемент ограничивает возможность углового смещения модельной камеры под действием волн давления.

Для моделирования потока газа предусмотрена система подачи и отвода рабочего тела (газа). На первом этапе в качестве рабочего тела взят воздух.

Исходя из предъявляемых требований, разработана конструктивная схема модельной камеры (**Рис.10**).

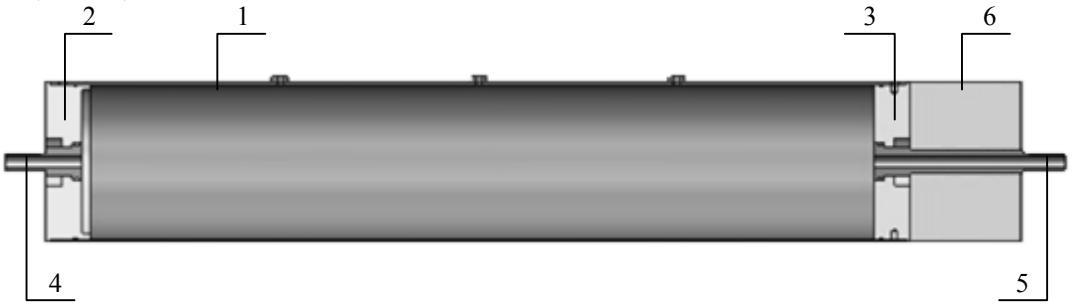


Рис. 10. Конструктивная схема модельной камеры: 1– корпус, 2– передняя крышка, 3– задняя крышка, 4,5– входной и выходной штуцеры, 6– груз

Чтобы получить качественную и количественную оценку рабочих процессов, протекающих в экспериментальной установке, используется комплекс измерительных средств и устройств. Измеряемые параметры: давление внутри модельной камеры определяется датчиками давления общего назначения и деформации корпуса определяются пьезоэлектрическими датчиками динамических деформаций.

Для измерения, регистрации и анализа результатов экспериментов предлагается модульная платформа PXI компании National Instruments. Архитектура PXI включает в себя шасси, в которое устанавливаются модульные приборы, контроллеры или интерфейсы для удаленного управления платформой.

Объектом эксперимента является модельная камера, корпус которой конструктивно представляет собой трубу.

При проектировании экспериментальной установки вопрос обеспечения необходимой резонансной частоты колебаний в системе «поток газа – конструкция» является очень важным.

Известно, что резонанс наблюдается при близости частот продольных колебаний в газе и в конструкции.

$$f_г = f_к \tag{1}$$

Из этого следует, что геометрические и физико-механические характеристики модельной камеры должны быть подобраны так, чтобы выполнялось условие (1).

Подберем геометрические характеристики модельной камеры.

В качестве математической модели продольных колебаний корпуса модельной камеры рассмотрим упругий консольный стержень длиной l, на свободном конце которого находится сосредоточенная масса. В данной задаче инерционностью стержня можно пренебречь и учитывать только его упругость.

Тогда частоты продольных колебаний, возникающие в корпусе камеры, можно вычислить по формуле [3]:

$$f_к = \frac{1}{2\pi}\sqrt{\frac{c}{M}}, \tag{2}$$

где $c = \dfrac{2\pi ER\delta}{l}$ – жесткость конструкции, E – модуль Юнга материала, R – радиус, δ – толщина стенки, l – длина; M – масса груза.

Частота продольных колебаний газа в замкнутой трубе:

$$f_г = \frac{a}{l} \tag{3}$$

где $a = \sqrt{k\frac{P}{\rho}}$ – скорость распространения возмущений в газе, k – показатель адиабаты, P – давление, ρ – плотность.

Согласно (1) можно приравнять правые части выражений (2) и (3) и записать:

$$\frac{a}{l} = \frac{1}{2\pi}\sqrt{\frac{c}{M}} \tag{4}$$

Пусть выбранная модель имеет следующие характеристики: l = 1м, R = 0,0825м, δ = 0,001м, $E = 2\cdot10^{11}$ Па, k =1,49, $P = 1{,}013\cdot10^5$ Па, ρ = 1,29 кг/м^3.

Тогда, согласно (3), частота продольных колебаний в газе: $f_г = 330 Гц.$

Из условия резонансного взаимодействия (1) следует $f_к = 330 Гц.$

Тогда, согласно (2), масса присоединенного груза $M = 24.12 кг.$

Следовательно, для обеспечения необходимой резонансной частоты колебаний к торцу модели модельной камеры должен быть присоединен груз массой 24,12 кг.

В ряде случаев, для системы в целом, значения $f_г$ и $f_к$ могут отличаться от парциальных [4]. Необходимые значения частот мы можем получать при варьировании (подборе) характеристик конструкции и параметров нагрузки. Таким образом, потребуется дополнительная настройка модельной камеры на резонансный режим.

Литература

1. Модорский В.Я. Нелинейное деформирование стержневой конструкции при наддуве. – Авиационная техника, 2003, №3, с.63 – 64.
2. Методическое руководство по ANSYS CFX 12.1 2010.
3. Модорский В.Я., Соколкин Ю.В. Газоупругие процессы в энергетических установках//Под. ред. Соколкина Ю.В. М.: Наука. Физматлит, 2007.-176 с.
4. Численное исследование актуальных проблем машиностроения и механики сплошных и сыпучих сред методом крупных частиц: монография / Ю.М. Давыдов [и др.]. – М.: Национальная академия прикладных наук, Международная ассоциация разработчиков и пользователей метода крупных частиц, 1995. – 1595с.

Объединение вычислительных кластеров для крупномасштабного численного моделирования в проекте NumGRID*

М.А. Городничев

Институт вычислительной математики и математической геофизики СО РАН

В работе представлен программный комплекс NumGRID для организации вычислений на объединении высокопроизводительных вычислительных кластеров в целях крупномасштабного численного моделирования. Дан анализ проблем организации распределенных вычислений на кластерах и обзор родственных проектов. Продемонстрированы результаты экспериментального исследования системы NumGRID.

1. Введение

Программный комплекс NumGRID[1] предназначен для объединения разнородных вычислительных кластеров в единый вычислительный ресурс на основе частичной реализации стандарта MPI-2.2 [2]. NumGRID обеспечивает возможность запускать MPI-приложение так, чтобы процессы приложения были распределены по рабочим узлам нескольких кластеров. При этом процессы составляют один коммуникатор MPI и имеют возможность обмениваться между собой средствами MPI.

NumGRID позволяет

- решать задачи, для которых недостаточно ресурсов отдельных кластеров,
- продлять жизнь устаревающего оборудования за счет объединения с новым,
- распределять части комплексных задач между специализированными кластерами в соответствии с индивидуальными требованиями частей к оборудованию и программному обеспечению,
- повысить гибкость при планировании распределения задач между кластерами в грид,
- планировать постепенное наращивание мощностей распределенной системы.

Схема устройства объединенного вычислительного ресурса NumGRID представлена на рис. 1.

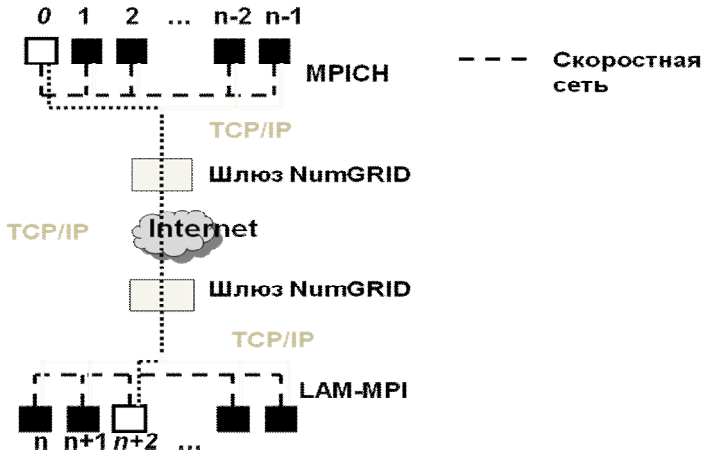


Рис. 1. Устройство NumGRID

В примере объединяются два кластера через Internet. Каждый кластер имеет рабочие узлы, объединенные высокоскоростной сетью (Myrinet, Infiniband, др.). Также узлы связаны с голов-

* Проект поддержан грантом РФФИ №10-07-00454а

ным узлом сетью с меньшей пропускной способностью, поддерживающей протокол TCP. Процессы приложения MPI, находящиеся на рабочих узлах одного кластера обмениваются между собой через высокоскоростную сеть средствами специфичной для кластера библиотеки MPI, позволяющей использовать возможности высокоскоростной сети. В примере это библиотеки MPICH и LAM-MPI. Для процессов, распределенных по рабочим узлам двух кластеров, поддерживается глобальная адресация в рамках одного коммуникатора MPI. Сообщения между процессами, расположенными на разных кластерах, проходят путь через головные узлы кластеров, где для этих целей перед стартом приложения запускаются шлюзы.

Пользователь вначале запускает шлюзы на каждом кластере. Шлюзы устанавливают между собой связь. Пользователь загружает на головные узлы кластеров исходный код своего приложения и исходные данные, собирает приложение на каждом кластере с библиотекой NumGRID-MPI, запускает необходимое число процессов на каждом кластере как локальные задачи MPI. Локальная задача получает в параметрах информацию о месте данной группы процессов в общей распределенной задаче, что позволяет обеспечить глобальную адресацию процессов. В конце работы пользователь забирает с каждого кластера результаты работы программы.

В разделе 2 ставится задача объединения вычислительных кластеров, формулируются требования к программному комплексу NumGRID, дается анализ проблем организации распределенных вычислений и обзор родственных проектов. В разделе 3 излагаются принципы организации программного комплекса NumGRID. В разделе 4 представлены результаты экспериментов по выполнению вычислений на объединении вычислительных кластеров.

2. Задача объединения вычислительных кластеров

2.1 Объединение вычислительных систем сегодня

В целях обеспечения ученых вычислительными ресурсами разрабатываются сверхмощные суперкомпьютеры [3], развиваются инфраструктурные проекты по организации доступа ученых к распределенным вычислительным ресурсам (грид) [4-6], к вычислительным центрам коллективного пользования [7,8], создаются распределенные вычислительные системы на основе объединения персональных компьютеров добровольцев [9,10] через Интернет. Широкое распространение получили относительно малые вычислительные системы (порядка десятков TFlops) кластерного типа (кластеры), которые приобретают университеты, институты и прочие организации. Такие малые системы являются наиболее доступными для ученых средствами решения вычислительных задач. Обычно, системы используются в многопользовательском режиме, и пользователи ставят свои задачи в очередь. Такого рода системы также объединяют в грид с целью распределения пользовательских задач между вычислительными ресурсами дружественных организаций. Однако, как правило, такое объединение предполагает, что задача пользователя назначается целиком в один из кластеров, а распределение процессов параллельной задачи по вычислительным узлам нескольких кластерам невозможно: штатные средства современных кластеров спроектированы так, чтобы обеспечивать высокоскоростные коммуникации между процессами, расположенными на узлах кластеров, и не предусматривают общения процессов между кластерами. Такое положение дел имеет исторические причины: во-первых, кластерные системы появились раньше, чем желание их объединить, и, во-вторых, существенно худшая производительность сетей связи между кластерами по сравнению с производительностью внутренних сетей кластеров делала нецелесообразным запуск параллельных программ с распределением процессов между кластерами. Считалось, что для распределенного счета подходят только задачи, где отсутствует необходимость в коммуникациях между процессами, либо коммуникации крайне малы. Инструменты для решения таких задач были разработаны [9, 11].

2.2 Зачем распределять процессы параллельных задач между кластерами?

Какие выгоды дала бы возможность запускать параллельные задачи с распределением процессов между кластерами? Во-первых, большее количество процессоров и памяти для одной задачи позволили бы увеличить масштаб решаемых задач. Во-вторых, возможность произвольного распределения процессов повысила бы гибкость при планировании распределения задач между кластерами в грид. В-третьих, появилась бы возможность вовлекать в расчеты устаревающие вычислительные комплексы, собственная производительность которых недостаточна для актуальных задач. В-четвертых, появилась бы возможность учитывать наличие специализированного оборудования или программного обеспечения, в т.ч. дорогостоящего, на тех или иных кластерах при распределении процессов параллельной программы: например, часть процессов программы в таком случае бы производила расчеты с использованием графических ускорителей на одном кластере, другая часть процессов получила бы возможность использовать библиотеки численных методов, имеющиеся на втором кластере. В-пятых, наличие средств для подключения дополнительных компьютеров позволило бы планировать постепенное наращивание мощностей распределенной системы.

2.3 Три технологических прорыва

Интерес к совместному использованию вычислительных ресурсов, неоднородных как в смысле процессоров, так и в смысле связей между процессорами, приводит к 1) развитию вычислительных алгоритмов, допускающих гибкость в организации коммуникаций между процессами и даже частичную потерю сообщений [12,13], 2) развитию методов организации вычислений, позволяющих выполнять коммуникации на фоне счета и динамически перераспределять вычисления с целью оптимизации загрузки вычислительных узлов и сетей связи [14-16]. В то же время, 3) характеристики (пропускная способность, латентность) каналов, связывающих кластеры, становится сопоставимыми с характеристиками сетей связи между узлами кластеров.

Эти три технологических прорыва открывают перспективу для применения распределенных вычислительных систем к решению крупных задач с более интенсивными коммуникациями.

В связи с этим актуальной является проблема разработки средств, которые позволили бы распределять процессы параллельных задач между вычислительными узлами нескольких кластеров и обеспечивать коммуникации между распределенными процессами.

2.4 Цель проекта NumGRID и требования к объединению вычислительных кластеров в проекте NumGRID

Цель проекта NumGRID заключается в разработке программного комплекса для поддержки исполнения параллельных программ на объединении вычислительных кластеров с распределением процессов параллельных программ между узлами вычислительных кластеров. В отличие от таких систем, как Globus Toolkit [17], ставящих задачу объединения вычислительных ресурсов в общем, проект NumGRID сконцентрирован на разработке простого инструмента пользовательского уровня для распределения процессов параллельлной программы между кластерами, к которым пользователь имеет непосредственный доступ посредством личных учетных записей. Принципиальным является обеспечение единой коммуникационной среды для распределенных процессов на основе стандартов MPI. Выбор стандарта MPI в качестве коммуникационного протокола обосновывается доминирующим положением MPI среди средств разработки параллельных программ численного моделирования.

К разработке программного комплекса в проекте NumGRID предъявляются требования, обоснованные сложившейся практикой организации вычислений и целью проекта:

1. Общая коммуникационная среда для процессов распределенных по нескольким кластерам должны быть рализована на основе стандартов MPI.
2. О структуре кластеров и сети связи между ними нужно предполагать следующее: каждый кластер состоит по крайней мере из головного/управляющего узла и вычислительных узлов; при этом вычислительные узлы предназначены для запуска на них вычислительных процессов и связаны между собой высокоскоростной сетью, а с головным узлом – сетью, как правило, меньшей пропускной способности, поддерживающей протокол TCP; головной

узел имеет еще по крайней мере один сетевой интерфейс, поддерживающий протокол TCP, через который узел может общаться с головными узлами других кластеров; головной узел используется для компиляции задач, управления локальными очередями задач и мониторинга задач.
3. Процессы параллельной программы должны размещаться на вычислительных узлах кластеров, узлы не имеют прямого сообщения друг с другом.
4. Должен быть предоставлен удобный интерфейс для задания конфигурации объединения кластеров, управления ресурсами и задачами.
5. Должны быть созданы условия для обеспечения динамических свойств прикладных программ (динамическая настраиваемость на доступные ресурсы, динамическая балансировка нагрузки, системы мониторинга и т.д.).
6. Должна обеспечиваться безопасность вычислений.
7. Каждому пользователю для запуска распределенных зирадач на нескольких кластерах должно быть достаточно иметь учетные записи на этих кластерах и пакет NumGRID. Организация NumGRID не должна требовать изменений в практике и политиках администрати-вания кластеров.
8. Кластеры могут быть разнородными в аппаратном, системном программном обеспечениии, линии связи могут быть разной пропускной способности, кластеры могут иметь различную административную подчиненность.

2.5 Обзор родственных проектов

Известно несколько проектов [18-22], которые ставят своей задачей объединение вычислительных систем на основе стандартов MPI. Системы, разрабатываемые в рамках этих проектов, не удовлетворяют полностью списку требований к NumGRID, однако многие проблемы стоящие перед NumGRID также решались в этих проектах. Далее рассмотрены наиболее значимые проблемы и подходы, которые применяются для их решения.

2.5.1 Обеспечение эффективности коммуникаций внутри кластеров

Как правило, в современных кластерах используется несколько типов сетей для объединения узлов. Сети с относительно малой пропускной способностью и относительно большой задержкой (латентностью) используются для задач управления узлами кластера и организации распределенных файловых систем. Для обмена данными между процессами параллельных вычислительных приложений используются скоростные сети с высокой пропускной способностью и малыми латентностями. При объединении кластеров важно, чтобы коммуникации внутри кластеров оставались эффективными. В то время как между кластерами без существенного изменения системного ПО возможно устанавливать соединение только по протоколам TCP/IP, то в зависимости от технологии скоростной сети внутри кластеров могут применяться различные реализации MPI, способные использовать оборудование более эффективно, чем на высоком уровне TCP/IP. Обычно производитель вычислительных систем поставляет такие реализации вместе с аппаратурой, если применяются специальные сетевые аппаратные решения, либо администраторы и пользователи систем могут выбрать свободно или коммерчески доступные библиотеки (MPICH-GM, OpenMPI, Intel MPI, Voltaire MPI, HP-MPI) для распространенных коммуникационных технологий (Myrinet, Infiniband).

Необходимость использования специализированных реализаций MPI для внутрикластерных коммуникаций обусловила отказ от коммуникационной библиотеки Nexus [23] в проекте MPICH-G [24] и разработку новой версии системы MPICH-G2 [19].

2.5.2 Обеспечение коммуникаций между внутренними узлами различных кластеров

Ранние подходы к объединению вычислительных систем на основе MPI [24, 19], предполагали, что процессы, располагающиеся на одной вычислительной системе, могут установить

прямое соединение в смысле TCP с процессами, расположенными на другой системе. Для современных кластерных систем установление таких прямых соединений невозможно (см. требования в разделе 2.4, п.2), поэтому необходимо организовывать трансляцию сообщений с внутренних узлов кластера через узел, имеющий сетевые интерфейсы как внутрь кластера, так и в сеть связи между кластерами. Далее такой узел будем называть узлом трансляции. Трансляция может быть осуществлена неявно с помощью технологий NAT [25] или VPN. Эти технологии обеспечивают возможность установления соединения TCP между процессами расположенными на компьютерах в различных частных сетях. Таким образом, для процессов создается иллюзия нахождения в одной сети и могут быть применены средства, например MPICH-G2 [19] для организации межкластерных взаимодействий. Однако, такая организация вычислений не будет учитывать фактическую топологию сети связи, соответственно организация межкластерных коммуникаций будет неэффективной, особенно для операций коллективного взаимодействия процессов (см. раздел 2.5.4 о коллективных коммуникациях). В проекте PACX-MPI [20] предполагается, что узел трансляции имеет доступ как в высокоскоростную сеть внутри кластера, так и в сеть между кластерами. Для организации передачи сообщений между кластерами на узле трансляции запускается дополнительный процесс MPI, помимо процессов, необходимых для выполнения прикладной логики параллельной программы. Через этот процесс все остальные процессы внутри данного кластера общаются с внешним миром. Преимущества такого подхода – PACX-MPI осведомлен о топологии сети и на этой основе может оптимизировать межкластерные взаимодействия, эффективно используется высокоскоростная сеть. Недостатки – 1) часто в современных кластерах узлы, имеющие доступ во внешние сети, не имеют подключения к скоростной сети; в таких системах требуется реализация трансляции сообщений через внутреннюю медленную TCP/IP сеть; 2) для трансляции требуются дополнительные процессы MPI, не соответствующие логике приложения.

Поскольку идеология грид предполагает сохранение локальных политик администрирования отдельных вычислительных систем при включении их в метакомпьютер, то все порты TCP, кроме, обычно, порта для доступа по протоколу SSH закрыты с помощью сетевого экрана (файервола). Поэтому, для передачи сообщений между кластерами применяются технологии проксирования и туннелирования через доступные порты [26].

В NumGRID предполагается использование ограниченного числа доверенных кластеров с выделенными линиями связи между ними, поэтому необходимые порты для связи между кластерами могут быть открыты. В случае если NumGRID нужно использовать в общественных сетях, возможно применения туннелирования соединений через SSH.

2.5.3 Учет особенностей сетевой организации для реализации эффективных межкластерных коммуникаций

Учет топологии сети рассматривается на уровне реализации взаимодействия между двумя процессами (p2p-взаимодействия) и на уровне реализации коллективных операций (см. раздел 2.5.2). Передача p2p может быть оптимизирована, когда существует более одного физического пути между коммуницирующими узлами. В таком случае применяется разбиение сообщения на части и параллельная доставка частей по разным путям [27].

В случае, когда для доставки сообщения требуется использование узлов трансляции, необходимо контролировать использование памяти на узлах трансляции, и кроме того, нельзя допускать блокирование узла трансляции в процессе доставки сообщения большого размера: нужно обеспечивать возможность прохождения сообщений малого размера на фоне доставки большого. Контроль памяти возможен только посредством разбиения сообщения на части. Избежать блокирования можно, если поддерживать многопоточную обработку сообщений на узле трансляции, либо посредством разбиения сообщений на части. Таким образом, для решения рассмотренных задач целесообразно применение пакетирования сообщений [28].

Применение классификации сообщений по приоритетам [29] целесообразно для обеспечения приоритетной доставки сообщений малого размера, имеющих смысл для управления ходом вычислений.

Учет топологии сети в реализации коллективных операций рассмотрен в следующем параграфе.

2.5.4 Эффективная реализация коллективных коммуникаций в неоднородной коммуникационной среде

Большинство упомянутых проектов занималось вопросом оптимизации коллективных коммуникаций в неоднородных сетях.

В работе [30] показано, что если время завершения посылки сообщения близко ко времени завершения приема сообщения, то оптимальной схемой рассылки сообщений для реализации широковещательной рассылки будет бинарное дерево. Если же время завершения посылки существенно больше времени завершения отправки, то оптимальная схема рассылки: корневой узел самостоятельно посылает получателям сообщение непосредственно. Таким образом, в случае объединения кластеров, оптимальной схемой коллективных коммуникаций будет бинарное дерево внутри кластеров, и посылка каждому кластеру по отдельности между кластерами [31]. Дополнительная оптимизация возможна, если кластеры не связаны физически в полный граф. В таком случае между кластерами не имеющими непосредственной связи сообщения могут двигаться через промежуточные кластеры. Поэтому, если требуется широковещательная рассылка, то корневой процесс должен послать лишь одно сообщение в по каждому физическому соединению, а промежуточные кластеры должны распространить это сообщение далее самостоятельно. Такой образом удается избежать многократной передачи одних и тех же данных по одним и тем же отрезкам сети.

2.5.5 Оптимизация приложений с учетом неоднородностей распределенной вычислительной среды

Несмотря на любые оптимизации алгоритмов межкластерных коммуникаций, неустранима проблема относительно низкой пропускной способности и высоких задержек в сети между кластерами. В таких условиях приложения, которые по своей природе требуют частых коммуникаций и в большом объеме между всем процессами приложения, не могут эффективно выполняться в распределенной среде. Для того, чтобы приложение могло выполняться эффективно, необходимо, чтобы имелась возможность равномерно распределить вычислительную загрузку между всеми узлами вычислительной системы и передавать сообщения на фоне счета: то есть сообщения должны идти по сети тогда, когда процессоры в это время заняты другой работой. Для минимизации простоев процессоров может потребоваться реализация динамического перераспределения работы между процессорами.

В проектах [32-34] реализовано автоматическое перераспределение загрузки на уровне процессов MPI: между узлами вычислительной системы с передаются именно процессы MPI. Такой подход упрощает разработку приложений, однако не учитывает структуру взаимодействий процессов прикладной программы.

2.5.6 Авторизация и шифрование сообщений

В работе [35] даются основные требования к обеспечению безопасности распределенных вычислений: 1. должно быть установлено взаимное доверие между тем, кто порождает процесс и ресурсом, где он порождается; при этом порожденные процессы должны наследовать способность к аутентификации от породивших процессов; 2. необходимо обеспечить шифрование данных на линиях передачи, где возможен перехват сообщений. Эти требования для распределенных приложений MPI были обеспечены реализацией MPICH-G [24] на основе библиотеки Nexus [23].

В отличие от MPICH-G, где предусматривался запуск процессов MPI на произвольных ресурсах, обнаруженных с помощью инструментария Globus Toolkit [16], пользователь NumGRID самостоятельно выбирает кластеры, которые он хочет объединить. Для объединения кластеров пользователь должен обладать учетными записями на каждом из них. Такая постановка облегчает проблему авторизации при порождении процессов. Авторизация может потребоваться для выполнения команд инструментария, обеспечивающих запуск процессов, например, при использовании очередей задач на выбранном вычислительном кластере. Использование выделенных сетевых линий между вычислительными кластерами позволяет на текущем этапе развития

системы NumGRID отказаться от реализации шифрования передаваемых данных. В случае необходимости использования публичных линий связи могут использоваться внешние средства для организации шифрованных туннелей [36, 37].

2.5.7 Устойчивость к сбоям

В проекте LA-MPI [27] надежность достигалась за счет применения проверки контрольных сумм и повторной пересылки данных. Признано, что использование протокола TCP для обеспечения надежности доставки внутри вычислительных систем нецелесообразно из-за больших накладных расходов. Проект [38] предоставляет возможности восстановления прерванных MPI-процессов даже после сбоя N-1 из N процессов. FT-MPI предоставляет прикладной программе сведения о состоянии процесса: завершён аварийно, прерван, восстановлен, перезапущен сначала. Сведения извлекаются с помощью обработчиков ошибок. Такой подход требует использования техник восстановления после сбоев внутри самой прикладной программы. Поскольку NumGRID в настоящее время ориентирован на объединение сравнительно небольших кластеров, полагается, что стабильность вычислительных процессов обеспечивается уровнем защищенности аппаратных компонентов вычислительной системы, на которой эти процессы запущены.

3. Программный комплекс NumGRID

Программный комплекс NumGRID состоит из трех компонентов:

- Шлюз
- Библиотека NumGRID-MPI
- Графическая система управления NumGRID

С помощью графической системы управления NumGRID пользователь задает набор кластеров, которые он хочет объединиеть для решения задачи, задает параметры авторизации на каждом из кластеров (пользователь должен иметь личные учетные записи на кластерах), параметры подзадач: сколько процессов на каждом кластере должно быть запущено и как распределены, путь на своей рабочей машине до архива с файлами своей программы и входными данными.

По команде запуска задачи, система управления осуществляет авторизацию на кластерах, отправляет файлы пользователя и системные файлы NumGRID на кластеры, выполняет сборку программы пользователя, а также шлюза и библиотеки NumGRID-MPI. Для обеспечения универсальности системы управления, сборка программы пользователя осуществляется на основе предоставленных пользователем инструкций (в виде Makefile). На головных узлах кластеров запускаются шлюзы и устанавливают между собой каналы связи в соответствии с топологией, заданной пользователем. Затем, на каждом кластере, в соответствии с параметрами подзадач, запускается или ставится в очередь программа пользователя. Сборка программы пользователя с библиотекой NumGRID-MPI вместо обычных библиотек MPI наделяет программу пользователя способностью ориентироваться в объединении кластеров. Поэтому запущенная программа пользователя автоматически в начале своей работы (в функции MPI_Init) осуществляет подключение к шлюзу, работающему на головном узле данного кластера. Шлюзы распространяют информацию о подключении подзадач между собой. Когда все подзадачи подключились к своим шлюзам, шлюзы информируют подзадачи об общей конфигурации системы и таким образом подзадачи осуществляют возврат из функции MPI_Init() с полной информацией о распределенной системе и проинициализированным коммуникатором MPI_COMM_WORLD, объединяющим все распределенные процессы.

Пользователь имеет возможность наблюдать за состоянием исполняющихся задач посредством графической системы управления NumGRID, завершать задачи, забирать результаты вычислений.

Все обращения к функциям MPI анализируются библиотекой NumGRID-MPI. Если требуемые коммуникации включают лишь процессы, расположенные в рамках одного кластера, то вызывается предустановленная на кластере библиотека MPI, способная эффективно использо-

вать высокоскоростную сеть кластера. Если требуются коммуникации между процессами, расположенными на различных кластерах, то сообщения транслируются через шлюзы.

Применяется метод коммутации пакетов при трансляции сообщений через шлюзы. Это позволяет контролировать расход памяти на головных узлах, параллельно отсылать несколько сообщений, использовать несколько путей доставки сообщения для повышения производительности сети. Реализовано расширение функций p2p станадрта MPI функциями с заданием приоритета сообщений (см. 2.5.3). Учитывается топология сети при реализации коллективных операций (см. 2.5.4).

4. Экспериментальное исследование NumGRID

Эксперименты по запуску приложений проводились на объединении кластеров Сибирского суперкомпьютерного центра (ССКЦ) [8] и Новосибирского государственного университета (НГУ). Кластеры построены на основе двухпроцессорных узлов с процессорами Intel Xeon E5540 (4 ядра) и внутренней коммуникационной сетью InfiniBand. Пропускная способность канала между головными узлами кластеров – 10 Гб/с.

4.1 Решение волнового уравнения

Ниже представлены графики, демонстрирующие характеристики исполнения программы решения волнового уравнения с помощью двухслойной явной схемы. На всех диаграммах запись «P (NxS)» означает «всего P ядер, из них N на кластере НГУ и S – на кластере ССКЦ.

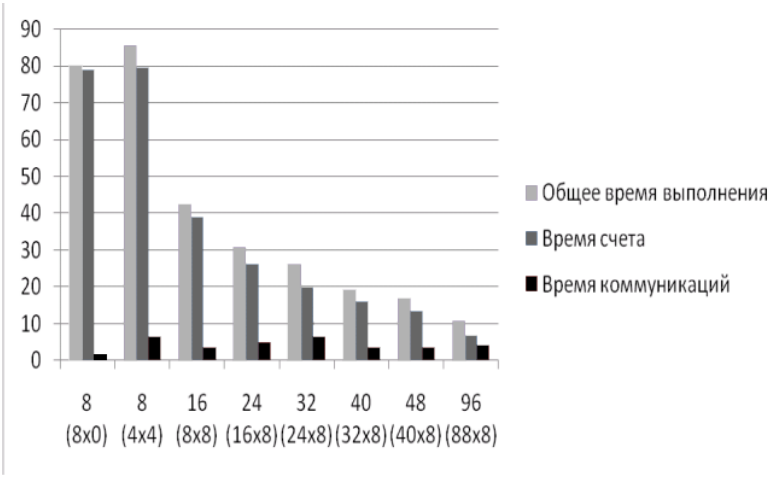


Рис. 2. Время решения волнового уравнения явным методом на NumGRID, сек.

На Рис.2 видно, что при переходе к распределенным вычислениям (с 8x0 на 4x4) увеличивается общее время работы программы за счет увеличения расходов на коммуникации. Расходы на коммуникации в данном примере увеличиваются в 5 раз, что приводит к росту времени работы программы на ~6%.

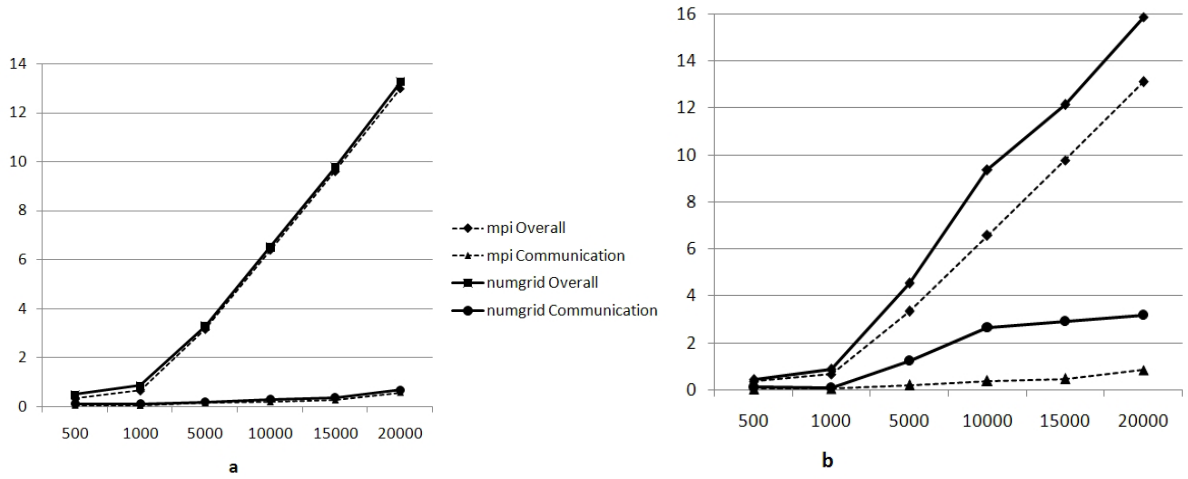


Рис. 3. Зависимость времени исполнения программы от увеличения объема коммуникаций на примере решения волнового уравнения для разных размеров задачи и способов распределения вычислений, время в сек.

На Рис. 3 показано, как изменяется время решения двумерного волнового уравнения на двух процессах в зависимости от размера задачи и способа распределения вычислений. Обозначения: mpiOverall – время работы программы, процессы которой расположены в рамках одного кластера; mpiCommunication – время, потраченное программой на коммуникации; соответствующие графики numgrid показывают время для процессов, расположенных на разных кластерах.

Распределение вычислений выполняется методом декомпозиции области. Таким образом, каждый процесс обрабатывает половину области. На части a Диаграммы показана ситуация, в которой область моделирования увеличивается (горизонтальная ось) по разрезанной размерности. При этом размер границы разреза, очевидно, не изменяется и объем коммуникаций между процессами остается постоянным. На части b Диаграммы показано, что происходит, когда изменяется размер области по другой размерности. В этом случае, с изменением размера области соответственно увеличивается длина разреза и объем коммуникаций. По сравнению с ситуацией a, видно, что в ситуации b время коммуникаций между процессами, расположенными на разных кластерах, существенно увеличивается в отношении времени коммуникаций между процессами внутри кластера.

4.2. Генерация случайных чисел и расчет статистик

Ниже представлены результаты тестирования программы, которая параллельно генерирует случайные величины и вычисляет статистики. Программа используется для моделирования физических процессов методами Монте-Карло. Программа осуществляет редкие коллективные коммуникации для сбора статистик, в остальное время процессоры выполняют существенные по объему вычисления независимо. Это позволяет ожидать, что относительно плохая пропускная способность сети между кластерами незначительно скажется на общей производительности программы.

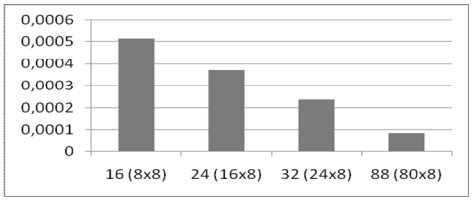


Рис. 4. Время генерации 2000 значений случайной величины и расчета статистик на NumGRID, сек. При этом время генерации 2000 значений случайной величины и расчета статистик на одном ядре кластера 1: 0,0109 сек.

Время исполнения программы при переходе от решения на одном ядре к решению на 16 ядрах, по 8 ядер на каждом кластере, уменьшается в 21 раз. Объяснить подобное сверхлинейное ускорение можно более эффективным использованием кэшей процессоров при увеличении количества процессоров и низкими коммуникционными расходами, свойственными данной задаче. При дальнейшем увеличении количества вовлеченных ресурсов так же отмечается сверхлинейное ускорение. Рис. 5 дает более ясное представление об ускорении времени решения при увеличении количества процессоров.

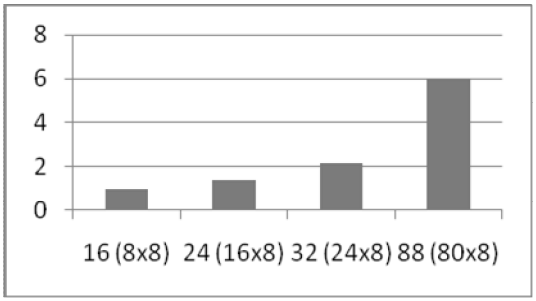


Рис. 5. Ускорение генерации 2000 значений случайной величины и расчета статистик на NumGRID относительно времени работы программы на двух узлах: один узел кластера НГУ (8 ядер) и один узел кластера ССКЦ (8 ядер).

4.3 Анализ экспериментов

В ходе экспериментов показано, что в зависимости от способа распределения вычислений между кластерами меняется эффективность работы приложений. Показано, что использование дополнительных процессоров из другого кластера позволяет уменьшать время выполнения приложения при определенных конфигурациях запуска. В частности, на задаче решения волнового уравнения достигается эффективность 90% на 40 процессорных ядрах относительно производительности программы на 8 ядрах, и 65% на 96 ядрах. На задаче генерации случайных чисел достигается эффективность 92% на 24 ядрах и 109% на 88 ядрах.

Несмотря на существенную разницу в производительности систем коммуникации внутри кластеров и между кластерами, время решения задач увеличивается умеренно для решения волнового уравнения на том же количестве процессоров и уменьшается для решения задачи поиска статистик. Можно заключить, что существует класс задач, которые могут быть решены на объединении кластеров за приемлемое для пользователей время. Вместе с тем, объединение кластеров может позволить решать задачи, большего объема, чем был бы способен решить один кластер за то же время.

5. Заключение

Растущий интерес к объединению разнородных вычислительных ресурсов для решения крупномасштабных задач и повышения эффективности использования вычислительных систем стимулирует развитие численных алгоритмов, методов планирования распределенной обработки данных, инструментов взаимодействия программных систем и средств взаимодействия пользователя с объединенными вычислительными системами.

В рамках проекта NumGRID проведен анализ проблем объединения вычислительных кластеров, существующих подходов к решению этой задачи, сформулированы требования к программному комплексу для организации распределенных вычислений на объединении вычислительных кластеров, программный комплекс реализован. Проведены испытания по объединению кластеров Сибирского суперкомпьютерного центра и Новосибирского государственного университета. Эксперименты демонстрируют, что существуют классы задач численного моделирования, которые целесообразно решать на объедниении вычислительных кластеров.

Дальнейшая работа заключается в расширении поддержки стандарта MPI-2.2, адаптации существующих приложений для работы в среде NumGRID, создании методов и средств разработки программ численного моделирования для неоднородных вычислительных сред.

Литература

1. D.Fougere, M.Gorodnichev, N.Malyshkin, V.Malyshkin, A. Merkulov, B.Roux. NumGrid Middleware: MPI Support for Computational Grids // Parallel Computing Technologies: 8th International Conference, PaCT 2005, Krasnoyarsk, Russia, September 5-9, 2005. Proceedings, Springer, 2005. - LNCS Vol. 3606, pp. 313-320.
2. MPI Documents. – URL: http://www.mpi-forum.org/docs/docs.html. Дата обращения: 15.12.2011.
3. Top 500 Supercomputer Sites. – URL: http://www.top500.org. Дата обращения: 15.12.2011.
4. XSEDE – Extreme Science and Engineering Discovery Environment. – URL: https://www.xsede.org. Дата обращения: 15.12.2011.
5. Worldwide LHC Computing Grid. – URL: http://lcg.web.cern.ch/lcg. Дата обращения: 15.12.2011.
6. Grid5000:Home. – URL: https://www.grid5000.fr. Дата обращения: 15.12.2011.
7. Межведомственный Суперкомпьютерный Центр РАН. – URL: http://www.jscc.ru. Дата обращения: 15.11.2011.
8. Сибирский Суперкомпьютерный Центр. – URL: http://www2.sscc.ru. Дата обращения: 15.11.2011.
9. BOINC. Open-source software for volunteer computing and grid computing. – URL: http://boinc.berkeley.edu/index.php. Дата обращения: 15.12.2011.
10. World Community Grid. Technology solving problems. – URL: http://www.worldcommunitygrid.org. Дата обращения: 15.12.2011.
11. Воеводин Вл.В., Жолудев Ю.А., Соболев С.И., Стефанов К.С. Эволюция системы метакомпьютинга X-Com // Вестник Нижегородского государственного университета им. Н.И. Лобачевского. №4. 2009. С 157-164.
12. F. Oboril, M. B. Tahoori, V. Heuveline, D. Lukarski, J.-Ph. Weiss. Fault Tolerance Technique for Iterative Solvers / Karlsruhe Institute of Technology – URL: http://www.emcl.kit.edu/preprints/emcl-preprint-2011-10.pdf. Дата обращения: 15.12.2011.
13. Peng Du, Piotr Luszczek, Jack Dongarra: High Performance Dense Linear System Solver with Soft Error Resilience // In. proc. of International Conference on Cluster Computing (CLUSTER), Austin, TX, USA, September 26-30, pp. 272-280.
14. Malyshkin V.E., Perepelkin V.A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem - In: Proceedings of the 11th Conference on Parallel Computing Technologis, LNCS 6873 - pp. 53-61, Springer, 2011
15. Система параллельного программирования OpenTS. Sergey Abramov, Alexei Adamovich, Alexander Inyukhin, Alexander Moskovsky, Vladimir Roganov, Elena Shevchuk, Yuri Shevchuk, and Alexander Vodomerov. 2005. OpenTS: An Outline of Dynamic Parallelization Approach // In proc of. PaCT 2005 conference, Krasnoyarsk, Russia, September 5-9, 2005, Springer, 2005. - LNCS Vol. 3606, pp. 303-312.
16. Globus Toolkit Homepage. – URL: http://www.globus.org/toolkit. Дата обращения: 15.11.2011.
17. C. Grelck and F. Penczek, Implementation Architecture and Multithreaded Runtime System of S-Net, in Implementation and Application of Functional Languages, 20th International Symposium, IFL'08, Hatfield, United Kingdom, Revised Selected Papers, 2010.
18. William L. George, John G. Hagedorn, and Judith E. Devaney. IMPI: Making MPI Interoperable //Journal of Research of the National Institute of Standards and Technology, vol. 105, 2000, pp. 343—428.
19. Nicholas T. Karonis, Brian Toonen, and Ian Foster. MPICH-G2: a Grid-enabled implementation of the Message Passing Interface // Journal of Parallel and Distributed Computing - Special issue on computational grids, Volume 63 Issue 5, May 2003, Academic Press, Inc. Orlando, FL, USA.
20. Edgar Gabriel, Michael Resch, and Roland Ruehle. Implementing MPI with Optimized Algorithms for Metacomputing // Proc. Message Passing Interface Developer's and User's Conference (MPIDC'99), pp. 31-41, Atlanta, GA, March 10-12, 1999.
21. Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. Pro-

ceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming ACM New York, NY, USA,1999.
22. IMPI Relay Trunking for Improving the Communication Performance on Private IP Clusters. R.Takano, M.Matsuda, T.Kudoh, Y.Kodama, F.Okazaki, Y.Ishikawa, and Y.Yoshizawa. In CCGrid2008, 2008.
23. I. Foster, C. Kesselman, and S. Tuecke. The Nexus Approach to Integrating Multithreading and Communication //Journal of Parallel and Distributed Computing, vol. 37, 1996, pp. 70-82.
24. I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, S. Tuecke. A wide-area implementation of the Message Passing Interface // Paral Comp (1998) Volume: 24, Issue: 12, Publisher: Elsevier, pp. 1735-1749.
25. P. Srisuresh, K. Egevang. Traditional IP Network Address Translator (Traditional NAT). Network Working Group, Request for Comments: 3022. – URL: http://tools.ietf.org/html/rfc3022. Дата обращения: 15.12.2011.
26. Yoshio Tanaka , Mitsuhisa Sato, Motonori Hirano, Hidemoto Nakada, and Satoshi Sekiguchi. Performance Evaluation of a Firewall-Compliant Globus-Based Wide-Area Cluster System // Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing IEEE Computer Society Washington, DC, USA, 2000.
27. Rob T. Aulwes, David J. Daniel, Nehal N. Desai, Richard L. Graham, L. Dean Risinger, Mark A. Taylor, Timothy S. Woodall, Mitchel W. Sukalski, "Architecture of LA-MPI, A Network-Fault-Tolerant MPI" // 18th International Parallel and Distributed Processing Symposium (IPDPS'04) – Papers, vol. 1, p.15, 2004.
28. Paul Baran, "On Distributed Communications Networks," IEEE Transactions on Communication Systems, Vol CS-12 (1), pp. 1-9, Mar 1964.
29. Real-Time Message Passing Specification based on MPI. – URL: http://www.mpirt.org. Дата обращения: 15.12.2011.
30. M. Bernaschi and G. Iannello. Collective Communication Operations: Experimental Results vs. Theory // Concurrency: Practice and Experience, 10(5), April 1998, pp. 359-386.
31. Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. Proceedings of the seventh ACM SIGPLAN symposium on Principles and practice of parallel programming ACM New York, NY, USA,1999.
32. Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth, and Gengbin Zheng. Programming Petascale Applications with Charm++ and AMPI. Petascale Computing: Algorithms and Applications. Chapman & Hall / CRC Press, USA, 2008, pp. 421-441.
33. Xiaohui Wei, Hongliang Li, Dexiong Li MPICH-G-DM: An Enhanced MPICH-G with Supporting Dynamic Job Migration. Proceedings of the 2009 Fourth ChinaGrid Annual Conference. IEEE Computer Society Washington, DC, USA 2009.
34. A. Barak, O. La'adan and A. Shiloh, Scalable Cluster Computing with MOSIX for Linux . Proc. Linux Expo '99, pp. 95-100, Raleigh, N.C., May 1999.
35. I. Foster, N. T. Karonis, C. Kesselman, G. Koenig and S. Tuecke. A secure communications infrastructure for high-performance distributed computing. HPDC '97 Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing IEEE Computer Society Washington, DC, USA,1997.
36. D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina. Generic Routing Encapsulation (GRE). // Network Working Group, Request for Comments: 2784, March 2000. – URL: http://tools.ietf.org/html/rfc2784. Дата обращения: 15.12.2011.
37. W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, B. Palter. Layer Two Tunneling Protocol "L2TP". // Network Working Group, Request for Comments: 2661, August 1999. – URL: http://tools.ietf.org/html/rfc2661. Дата обращения: 15.12.2011.
38. Graham E. Fagg and Jack J. Dongarra. FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World // RECENT ADVANCES IN PARALLEL VIRTUAL MACHINE AND MESSAGE PASSING INTERFACE, LNCS, 2000, Volume 1908/2000, pp. 346-353.

Подход к проблеме удаленной визуализации сложных 3D-моделей на базе noVNC-решений*

Д.А. Диков

Южно-уральский государственный университет

Проблема удаленной визуализации сложных 3D-моделей предполагает предоставление результатов инженерных вычислений посредством облачного сервиса в качестве одного из возможных решений. Задача подобного сервиса - обеспечить взаимодействие конечного пользователя с системой визуализации на стороне сервиса, позволяя управлять процессом и вносить корректировки в условия визуализации. Распространенный подход на основе технологии VNC накладывает технические ограничения на клиентскую часть. Альтернативой является использование noVNC-решений, использующих преимущества технологии HTML5 для организации тонкого клиента, что существенно расширяет круг устройств, способных полноценно работать с сервисом.

1. Введение

Текущее развитие суперкомпьютерных технологий и высокопроизводительных вычислений порождает проблему накопления больших объемов данных на вычислительных фермах без организации удаленного доступа для просмотра результатов и внесения изменений в процесс построения решения задачи инженерного моделирования.

Существует целый ряд широко используемых CAE-систем, таких как ANSYS, DEFORM, FlowVision, решающих задачи инженерного моделирования. Для расчета инженерных задач в рамках системы CAEBeans [6, 7] конечному пользователю предоставляется возможность работы с высокопроизводительными ресурсами распределенной грид-среды посредством простого проблемно-ориентированного пользовательского интерфейса. В основе технологии CAEBeans лежит обеспечение сервис-ориентированного предоставления программных ресурсов базовых компонентов CAE-систем и формирование иерархий проблемно-ориентированных оболочек, инкапсулирующих процедуру постановки и решения определенного класса задач. Технология CAEBeans регламентирует процесс декомпозиции задачи в иерархию подзадач [8]:

1. поиск вычислительных ресурсов;
2. сопоставление задачам соответствующих базовых компонент CAE-систем;
3. мониторинг хода решения задач;
4. передача результатов решения задач пользователю.

Большой объем данных, полученных в результате экспериментов, приводит к сложностям в процессе их обработки на пользовательском ПК. Возникает необходимость быстрой визуализации результатов моделирования без необходимости полного копирования всего объема полученных результатов на компьютер пользователя.

Для организации удаленного доступа необходимо переносимое, независимое от аппаратной платформы решение, позволяющее установить соединение с сервером, имеющим прямой доступ к ресурсам высокопроизводительной фермы [1, 4]. Ранее предложенное решение [5] позволяло получать лишь отдельные изображения результатов визуализации и обновлять полученное изображение в соответствии с запросами на перемещения камеры. Подход на базе VNC [2, 3] требует установки программы для подключения VNC на клиентской стороне, что накладывает существенные ограничения на круг устройств, которые могут использовать систему. Также необходимо знать реальный IP-адрес сервера, что снижает защищенность системы. Решение [11] использует Java-апплет для подключения к серверу, что исключает возможность использования системы на устройствах без виртуальной машины Java.

* Проект выполнен при поддержке Совета по грантам Президента Российской Федерации (номер проекта МК-1987.2011.9) и Российского фонда фундаментальных исследований (проекты № 11-07-00478 и 10-07-96007-р_урал_а) .

Организация тонкого клиента, взаимодействующего с сервером по протоколу VNC возможна с использованием существующих на сегодняшний день no-VNC решений, основывающихся на возможности создания полнодуплексных клиент-серверных соединений в актуальных версиях веб-браузеров. Такими решениями являются [9, 10].

Технология [9] требует для развертывания контейнер сервлетов Apache Tomcat на серверной стороне. В описываемом в данной статье решении используется технология [10], не накладывающая дополнительных требований на процесс развертывания, а также предоставляющая эмуляцию технологии WebSockets на базе технологии Flash браузерам, не поддерживающим стандарт HTML5.

Такие решения базируются на HTML5 и технологии WebSockets и позволяют организовать доступ к облачной CAE-системе посредством сети Интернет. Это позволяет использовать виртуальный стенд при помощи ПК или мобильной системы без дополнительного ПО и получать доступ к облачной CAE-системе посредством сети Интернет.

2. Концепция сервиса удаленной визуализации

В данной статье предлагается подход к построению системы виртуальных стендов на основе сервиса удаленной визуализации сложных 3D-моделей. Система использует бизнес-модель SaaS (Software as a Service), предоставляя пользователю удаленный доступ к системе визуализации через веб-портал CAEBeans Portal. Доступ осуществляется посредством тонкого клиента, роль которого выполняет веб-браузер.

Предлагаемая концепция сервиса удаленной визуализации включает следующие компоненты:

1. Веб-служба;
2. Система визуализации;
3. CAEBeans Server (в качестве первичного источника данных).

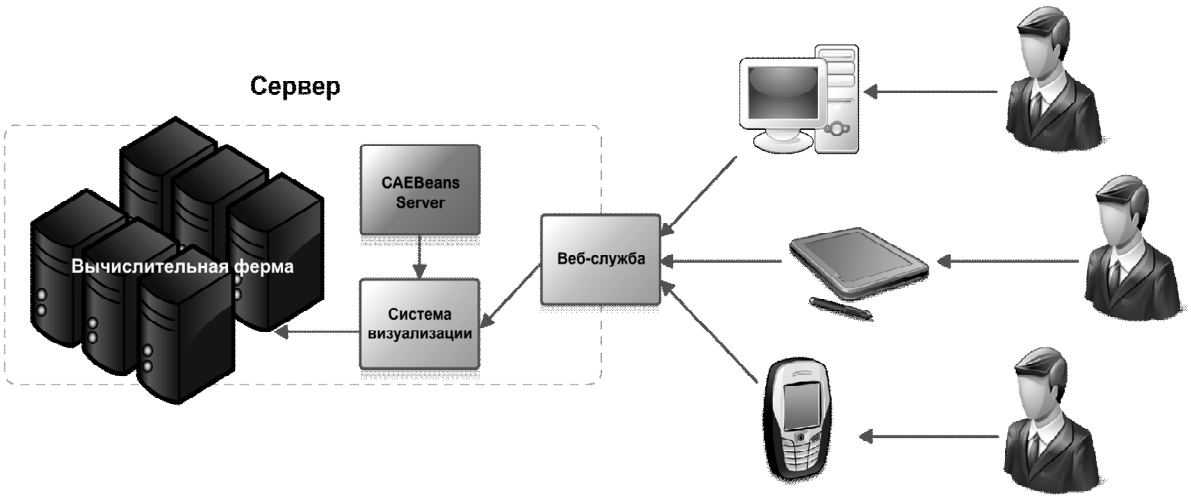


Рис. 1. Концепция сервиса удаленной визуализации

Веб-служба маршрутизирует запросы пользователя системе визуализации и обеспечивает взаимодействие с виртуальным стендом. Виртуальный стенд включает в себя три компонента:

1. Система визуализации;
2. CAEBeans Server;
3. Вычислительная ферма.

Работа пользователя с виртуальным стендом происходит посредством клиентского веб-приложения, взаимодействующего с веб-службой, расположенной на сервере. Клиентское веб-приложение отображает запущенное на сервере приложение, которое организует интерактивное взаимодействие с результатами визуализации. Принцип работы стенда отражен на **Рис. 2**.

После аутентификации пользователя в сервисе веб-служба выполняет на сервере команду запуска сервера VNC на определенном порту. Система визуализации получает адрес размещения данных решения инженерной задачи пользователя от CAEBeans Server и осуществляет ко-

пирование данных на сервер визуализации. После этого система визуализации может выполнять визуализацию решения.

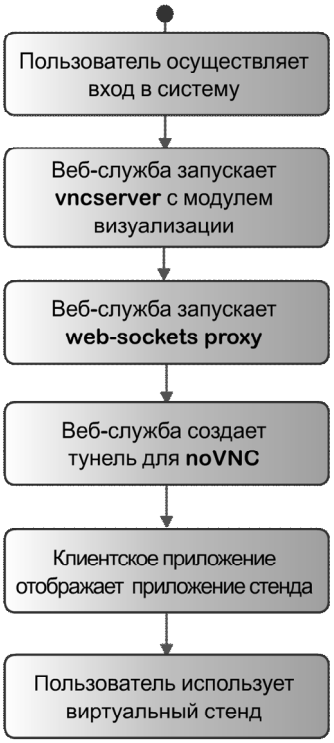


Рис. 2. Запуск и работа стенда

Запущенный сервер VNC отображает приложение стенда. Если сервер запущен успешно, то веб-служба запускает веб-сокет прокси. Если прокси создан успешно, то веб-служба создает маршрут для обеспечения доступа к порту, на котором запущен прокси. После этого запущенное приложение стенда отображается клиентом, и пользователь может приступить к работе со стендом.

2.1 Архитектура сервера

Рис. 3. Архитектура системы визуализации

Система визуализации включает в себя приложение стенда и модуль визуализации (Рис. 3). Приложение стенда представляет собой программу, предоставляющую пользовательский ин-

терфейс для просмотра результатов визуализации и позволяющий пользователю изменять параметры визуализации – поворот визуализируемой модели и приближение камеры. Модуль визуализации получает запросы на визуализацию от приложения стенда и строит изображение на основе данных решения инженерной задачи и данных поворота модели и расположения камеры.

Решение инженерной задачи пользователя копируется модулем визуализации на этапе инициализации, на основе идентификатора решения, передаваемого веб-службой. Модуль визуализации запрашивает адрес решения у CAEBeans Server.

Результат визуализации отображается приложением стенда.

3. Реализация сервиса удаленной визуализации

Веб-служба реализована на базе фреймворка Ruby On Rails. Пользователь системы CAEBeans аутентифицируется на веб-портале CAEPortal и может перейти к просмотру готового решения инженерной задачи. В этом случае браузер осуществит подключение к системе виртуальных стендов, интегрированной в веб-портал. При подключении клиента к сервису удаленной визуализации веб-служба осуществляет запуск сервера VNC на определенном порту. Веб-служба передает клиенту URI для подключения noVNC. При попытке подключения noVNC с использованием данного URI, веб-служба создает туннель, который перенаправляет подключение с выданного клиенту URI на конкретный порт, на котором запущен VNC сервер пользователя. Это позволяет защитить сервис от несанкционированных подключений, не позволяя подключаться к портам сервера напрямую.

Веб-служба, принимая запрос на подключение к серверу VNC от клиентского приложения, запускает модуль визуализации для визуализации решения пользовательской задачи, которое хранит CAEBeans Server.

Узел вычислительной фермы, к которому подключается клиент, работает под управлением ОС Linux. Клиент отображает только рабочую область окна приложения стенда.

Система использует в качестве клиентского веб-приложения HTML-страницу веб-портала CAEBeans Portal, выполняющую асинхронные запросы серверу, используя технологию AJAX. Веб-служба использует API на основе протокола REST для запросов аутентификации и установления подключения.

4. Заключение

В представленной работе был предложен подход к организации системы виртуальных стендов с тонким клиентом на базе no-VNC решений. Освещены механизмы подключения клиента и организации туннеля для передачи данных. Описана архитектура и детали реализации решения, интегрированного в систему CAEBeans и, непосредственно, в веб-портал CAEBeans Portal.

Литература

1. Baratto R., Kim L., Nieh J. Thinc: a virtual display architecture for thin-client computing // SOSP ’05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. New York, NY, USA: ACM, 2005, pp. 277–290.
2. Margalef F.P., Vall M. R., Garcia A.I., et al Usage of thin clients on STB for secure interactive applications. – URL: http://nem-summit.eu/wp-content/plugins/alcyonis-event-agenda//files/Usage_of_thin_clients_on_STB_for_secure_interactive_applications.pdf. Дата обращения: 14.11.11
3. Shizuki B., Nakasu M., Tanaka J. VNC-based Access to Remote Computers from Cellular Phones // CSN ’02: Proceedings of the IASTED International Conference on Communication Systems and Networks, 2002, pp. 74–79.

4. Zhong L., Wo T., Li J., Li B. A Virtualization-Based SaaS Enabling Architecture for Cloud Computing // Proceedings of the 2010 Sixth International Conference on Autonomic and Autonomous Systems (ICAS '10). IEEE Computer Society, Washington, DC, USA, 144-149.
5. Диков Д.А. Создание сервис-ориентированной системы удаленной визуализации сложных 3D-моделей для системы CAEBeans // Научный сервис в сети Интернет: экзафлопсное будущее: Труды международной научной конференции (Новороссийск, 19-24 сентября 2011 г.). М.: Изд-во МГУ, 2011. С. 599-602.
6. Радченко Г.И. Сервисно-ориентированный подход к использованию систем инженерного анализа в распределенных вычислительных средах // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XI Всероссийской конференции (Н. Новгород, 2–3 ноября 2011 г.). Нижний Новгород: Изд-во Нижегородского госуниверситета, 2011. С. 247-251.
7. Радченко Г.И., Соколинский Л.Б. Технология построения виртуальных испытательных стендов в распределенных вычислительных средах // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. № 54. 2008
8. Радченко Г.И. Технология построения проблемно-ориентированных иерархических оболочек над инженерными пакетами в грид-средах // Системы управления и информационные технологии. 2008. № 4(34).
9. Проект Guacamole. - URL: http://guac-dev.org. Дата обращения: 12.11.11
10. Проект noVNC. - URL: http://kanaka.github.com/noVNC. Дата обращения: 12.11.11
11. Проект Oxalya. - URL: http://www.oxalya.com. Дата обращения: 28.10.11

Высокопроизводительные реконфигурируемые вычислительные системы на основе плис Virtex-6 И Virtex-7[1]

А.И. Дордопуло, И.А. Каляев, И.И. Левин, Е.А. Семерников

В статье рассматриваются конструктивные особенности и характеристики реконфигурируемых вычислительных систем (РВС), построенных на основе программируемых логических интегральных схем (ПЛИС), объединенных в большие вычислительные поля с помощью высокоскоростных интерфейсов. Отличительной характеристикой таких систем является высокая удельная производительность и энергоэффективность при решении прикладных задач, а также близкий к линейному рост производительности при увеличении аппаратного ресурса. В статье приводятся примеры реализации вычислительных модулей и блоков на основе ПЛИС семейств Virtex-6 и новые разработки на основе ПЛИС Virtex-7, их технические характеристики и описывается разрабатываемый комплекс системного программного обеспечения.

1. Введение

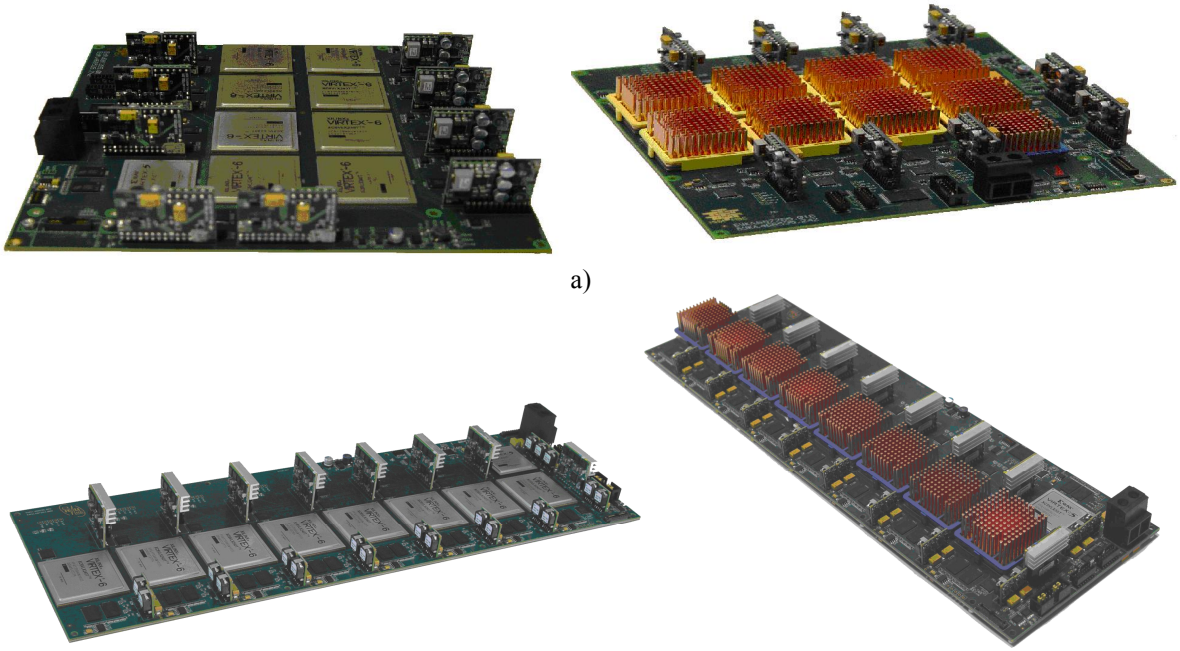
Разработчики высокопроизводительных вычислительных систем все чаще обращаются к поиску новых архитектурных решений на пути достижения рекордных характеристик производительности суперЭВМ. В последние годы в качестве вычислительных узлов помимо универсальных микропроцессоров используются многоядерные графические микропроцессоры и программируемые логические интегральные схемы (ПЛИС). В большинстве вычислительных систем, использующих ПЛИС, например таких как Jaguar - Cray XT5-HE, программируемые кристаллы используются в качестве дополнения к микропроцессорам, на которых выполняются трудно- или неэффективно реализуемые фрагменты вычислений.

В то же время известно [1,2], что ПЛИС обладают значительно большим вычислительным потенциалом, который в полной мере может быть реализован в реконфигурируемых вычислительных системах, содержащих множество кристаллов ПЛИС, используемых как основной вычислительный элемент. Методы разработки и создания таких систем успешно развиваются в НИИ многопроцессорных вычислительных систем Южного федерального университета (г. Таганрог). Концепция построения РВС позволила создать целый ряд высокопроизводительных систем различных архитектур и конфигураций, предназначенных для решения вычислительно трудоемких задач различных предметных областей, успешно эксплуатируемых организациями и ведомствами Российской Федерации. В качестве элементной базы для построения таких РВС использовались ПЛИС Xilinx семейства Virtex-5 (семейство РВС, разработанное по госконтракту №02.524.12.4002 от 20.04.2007) [3,4] и Virtex-6 большой интеграции, соединенные в единый вычислительный ресурс высокоскоростными каналами передачи данных – LVDS и Rocket GTX.

2. РВС нового поколения

Новые разработки НИИ МВС ЮФУ на основе ПЛИС семейства Virtex-6 и Virtex-7 используют для построения РВС открытую масштабируемую архитектуру [1], позволяющую преодолеть многие ограничения, накладываемые архитектурой с ортогональной коммутацией. В 2010-2011 годах в НИИ МВС ЮФУ были разработаны и серийно производятся РВС нового поколения на основе вычислительных модулей (ВМ) с ПЛИС Virtex-6, построенные на основе принципов открытой масштабируемой архитектуры, в двух перспективных конструктивных исполнениях – в модуле высотой 6U «Саиф» и в модуле высотой 1 U «Ригель», фотографии плат которых представлены на рисунке 1,а-б.

[1] Исследования выполнены при финансовой поддержке Министерства образования и науки РФ.



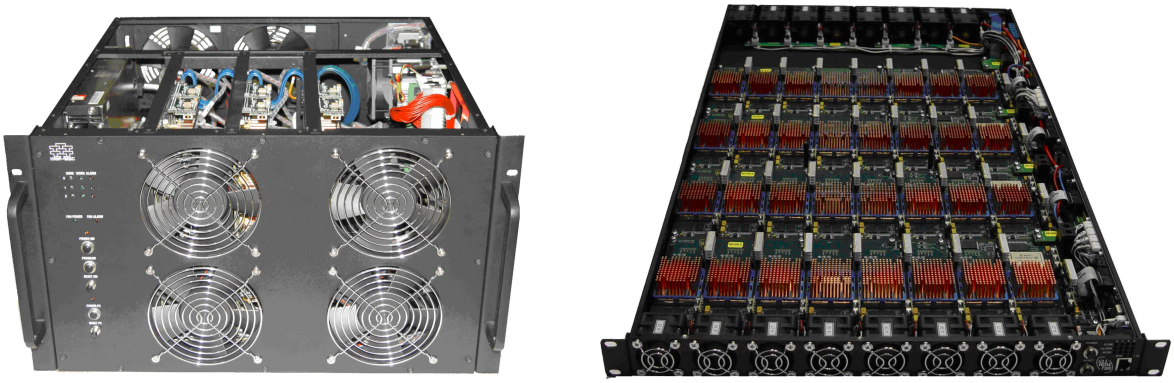
а)

б)

а) плата вычислительного модуля «Саиф»; б) плата вычислительного модуля «Ригель»

Рис. 1. Платы вычислительных модулей на ПЛИС Virtex-6

Фотографии вычислительных модулей «Саиф» и «Ригель» представлены на рисунке 2.



а) б)

а) вычислительный модуль «Саиф»; б) вычислительный модуль «Ригель»

Рис. 2. Вычислительные модули на основе Virtex-6

Одновременно с этим ведутся разработки перспективных изделий на ПЛИС Virtex-7: разрабатываемая в настоящее время по государственному контракту №14.527.12.0004 от 03.10.2011 реконфигурируемая вычислительная система РВС-7 будет содержать вычислительное поле на основе ПЛИС семейства Virtex-7 с пиковой производительностью до 10^{15} операций с фиксированной запятой в секунду в одностоечном конструктиве 47U.

На рис. 3 показана структурная схема платы вычислительного модуля (ПВМ) 6V7-180, являющейся основой для построения РВС-7. Вычислительное поле ПВМ 6V7-180 выполнено на микросхемах XC7V585T-FFG1761, содержащих около 58 миллионов эквивалентных вентилей.

В состав ПВМ 6V7-180 входят:

- контроллер ПВМ, выполненный на ПЛИС XC6V130T-1FFG1156C производства Xilinx;
- вычислительное поле, состоящее из 6-ти ПЛИС XC7V585T-1FFG1761 семейства Virtex-7 производства Xilinx. Между собой ПЛИС вычислительного поля соединены последовательно.

Связь между соседними ПЛИС обеспечивается по 144 дифференциальными линиями LVDS интерфейса на частоте 800 МГц;

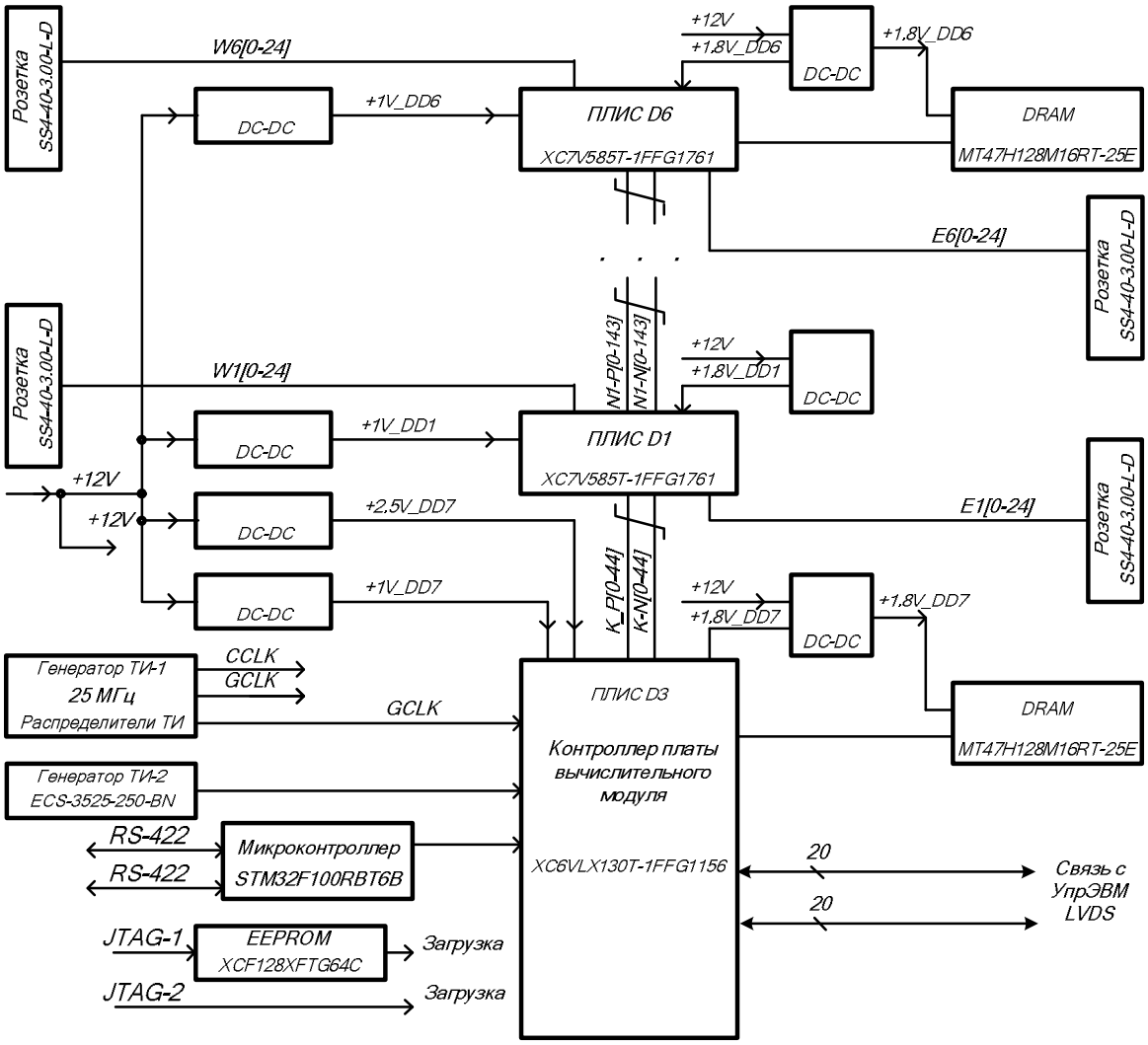


Рис. 3. Структурная схема 6V7-180 на ПЛИС Virtex-7

- 12 каналов интерфейса LVDS на частоте 800 МГц по 25 дифференциальных пар каждый (разъёмы типа SS4) для связи с другими вычислительными модулями;

- узлы основной и резервной загрузки ПЛИС по интерфейсам JTAG-1 и JTAG-2;

- подсистема синхронизации (генераторы ECS-2033-250-BN и распределители тактовых импульсов IDT5T9316NLI;

- распределённая память в составе 12-ти микросхем динамической памяти (MT47H128M16HR-25E с организацией 128 M*16 и частотой записи/чтения до 400 МГц). К ПЛИС вычислительного поля, а также к ПЛИС контроллера базового модуля подключено по две микросхемы памяти DDR2. Объем оперативной памяти на ПВМ 3 Гбайта;

- 2 канала интерфейса LVDS по 20 дифференциальных пар для связи с персональным компьютером и внешней аппаратурой;

- подсистема загрузки ПЛИС;

- подсистема питания, в состав которой входят DC-DC преобразователи напряжения, вырабатывающие напряжения питания: +1 В – питание ядер ПЛИС; +2,5 В – питание узла тактирования; +1,8 В – питание микросхем памяти DDR2, +3,3 В – буферных каскадов ПЛИС.

Технические характеристики ПВМ 6V7-180 представлены в таблице 1.

Таблица 1. Технические характеристики ПВМ 6V7-180

Технический параметр		Значение
ПЛИС XC7V585T-FFG1761 (вычислительная ПЛИС) (58 млн. экв. вент.), шт.		6
ПЛИС XC6V130T-FFG1156 (контроллер ПВМ) (13 млн. экв. вент.), шт.		1
М/с памяти DDR2 MT47H128M16HR-25E (128 М * 16 = 2048 Мбит), шт.		12
Объем памяти, Гбайт		3
Частота обработки информации ПЛИС, МГц		до 400
Тактовая частота каналов между соседними ПЛИС, МГц (не менее)		800
Производительность вычислений, приведённых операций в секунду		$5 \cdot 10^{13}$ оп/с
Интерфейсы	Каналы LVDS для связи с УпрЭВМ, дифф. пар	20
	Разъемы SS4, шт.	12
	Каналы LVDS для обмена со смежными ПЛИС, дифф. пар	144
Потребляемая мощность, не более, Вт		300
Габариты ПВМ, мм		140 x 325

В таблице 2 приведены сравнительные характеристики плат вычислительных модулей «Саиф», «Ригель» и 6V7-180.

Таблица 2. Технические характеристики плат вычислительных модулей

Плата вычислительного модуля	Число ПЛИС	Тип и наименование ПЛИС	Количество эквивалентных вентилей в 1 ПЛИС, млн. шт.	Интерфейс и скорость межмодульного обмена, Гбит/сек	Потребляемая мощность, ВА
«Саиф»	8	Virtex-6	24	Gigabit Ethernet, 1	300
«Ригель»	8	Virtex-6	24	Gigabit Ethernet, 1	300
6V7-180	6	Virtex-7	58	LVDS, 120	300

На рисунке 4 представлена компоновка вычислительного модуля 24V7-750 на основе четырех ПВМ 6V7-180 с управляющим модулем УМ-7. В состав 24V7-750 входят: четыре платы 6V7-180; управляющий модуль УМ-7; подсистема питания; подсистема охлаждения и другие подсистемы. Характеристики ВМ 24V7-750 приведены ниже в таблицах 3 – 5.

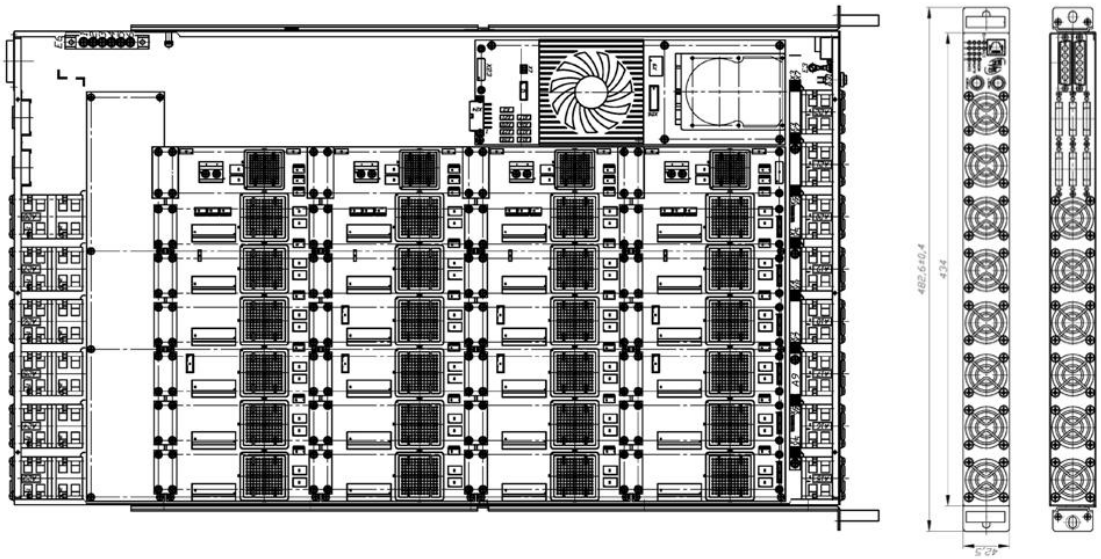


Рис. 4 Компоновка ВМ 24V7-750

Вычислительные модули конструктивных исполнений «Саиф», «Ригель» и 24V7-750 имеют высоту 6U и 1U и предназначены для установки в стандартную 19" вычислительную стойку, которая является базовым компонентом для создания сверхвысокопроизводительных комплексов на основе ПЛИС.

Применение ПЛИС семейства Virtex-6 в качестве элементной базы для построения вычислительных модулей «Саиф» и «Ригель» позволяет при сохранении стоимости поставки вычислительного модуля увеличить производительность в 1,5-2 раза по сравнению с аналогичным решением на основе ПЛИС семейства Virtex-5, а использование Virtex-7 улучшает характеристики примерно в 1,7 раза по сравнению с изделиями на основе Virtex-6. Этот факт позволяет рассматривать созданные вычислительные модули нового поколения как наиболее перспективные варианты для построения РВС различных архитектур и конфигураций и обеспечивает им существенное конкурентное преимущество по большинству технико-экономических параметров: удельной производительности, энергоэффективности и др.

Производительность рассматриваемых вычислительных модулей и вычислительных стоек на их основе представлена в таблице 3. Производительность соответствует обработке данных с одинарной (Pi_{32}) и двойной (Pi_{64}) точностью в соответствии со стандартом IEEE-754 для вычислительных модулей и стоек описанных изделий. Технические характеристики вычислительных модулей представлены в таблице 3.

Таблица 3. Производительность вычислительных модулей и стоек

Наименование вычислительного модуля	Производительность вычислительного модуля Pi_{32}/Pi_{64} (Гфлопс)	Число вычислительных модулей в 19 " стойке	Производительность стойки Pi_{32}/Pi_{64} (Тфлопс)
«Саиф»	1600/500	6	9/3
«Ригель»	1600/500	24-36	34,5 – 51,8/12,0 – 18,0
24V7-750	2600/820	24-36	62,4 – 95,0/19,4 – 29,4

В таблице 4 приведены производительности вычислительных модулей на задачах символьной обработки данных, использующих битовые преобразования, и задачах математической физики на основе арифметики с плавающей запятой одинарной точности.

Таблица 4. Производительность вычислительных модулей

Вычислительный модуль	Символьная обработка данных (Топ/с)	Математическая физика, арифметика с плавающей запятой (Тфлопс)
«Саиф»	199,6	1,6/0,5
«Ригель»	199,6	1,6/0,5
24V7-750	320,2	2,2/0,8

В таблице 5 приведены суммарные скорости передачи данных между кристаллами ПЛИС и блоками распределенной памяти, между ПЛИС в пределах одного вычислительного модуля и других вычислительных модулей.

Таблица 5. Скорость передачи данных

Вычислительный модуль	С блоками распределенной памяти (Гбит/с)	Между ПЛИС вычислительного поля (Тбит/с)	С другими вычислительными модулями (Тбит/с)
«Саиф»	12,8	1,0	0,001
«Ригель»	12,8	1,0	0,001
24V7-750	16,4	2,0	0,5

3. Многоуровневое программирование РВС

Традиционно программирование РВС содержит два этапа. На первом этапе создается вычислительная структура для решения прикладной задачи. На втором этапе программируется организация вычислительного процесса в созданной вычислительной структуре с целью полу-

чения результата. При этом наибольшие трудности у пользователей вызывает именно первый этап, поскольку большинство пользователей привыкли программировать организацию вычислительного процесса, опираясь на неизменную аппаратную платформу компьютера. С целью преодоления возникающих затруднений разработчики предлагают многоуровневый подход к программированию РВС, позволяющий в едином процессе создавать как вычислительные структуры в поле ПЛИС, так и организацию в них вычислительного процесса.

Можно выделить четыре основных уровня программирования реконфигурируемых вычислительных систем:

- уровень использования функционально законченных фрагментов в прикладной масштабируемой программе;
- уровень программирования унифицированного макрообъекта для прикладной масштабируемой программы;
- уровень синтеза и программирования макрообъектов в прикладной масштабируемой программе;
- уровень трансляции прикладной масштабируемой программы с языка высокого уровня в логические ячейки ПЛИС и связи между ними.

Уровни программирования РВС и применяемые на них основные средства разработки представлены на рисунке 5.

Рис. 5. Уровни программирования РВС, доступные пользователю

Программирование РВС на первом уровне выполняется с помощью вызова внешних библиотечных функций из программ на традиционных высокоуровневых языках программирования. Вызываемая функция осуществляет запуск потоков данных, следующих через вычислительную систему, при этом структурная, потоковая и процедурная составляющие прикладной масштабируемой программы не меняются и должны быть уже загружены в РВС. Достоинствами такого подхода являются скорость и простота разработки программ для традиционных процессоров, а также наибольшая гибкость при вызове структурно-реализованного фрагмента, недостатками – сложность изменения аппаратной реализации, созданной специалистом-схемотехником, и снижение общей производительности задачи в целом из-за более низких скоростей обмена между управляющим модулем и вычислительным полем ПЛИС.

Программирование на уровне унифицированных макрообъектов (представляющих собой совокупность вычислительных устройств, выполняющих определенную группу команд и соединенных между собой коммутационной системой) позволяет программисту задавать коммутацию как внутри макрообъектов, так и макрообъектов между собой, что, в свою очередь, позволяет перестраивать структуру РВС в процессе решения задачи и обеспечивает пользователя более гибкими средствами разработки прикладных программ. Естественным требованием унификации и универсальности макрообъекта является наличие единого интерфейса команд для программирования макрообъектов, что налагает ряд ограничений как на схемотехническую реализацию макрообъектов, так и на используемую систему команд для программирования

макрообъектов. Это приводит к тому, что для программирования макрообъектов используется низкоуровневый язык ассемблера, содержащий команды настройки коммутационной системы и выполняемых макрообъектом функций и команды чтения/записи потоков данных, обрабатываемых вычислительной структурой, составленной из макрообъектов. Преимуществом этого подхода является возможность перестройки архитектуры вычислительной системы при решении задачи, недостатком – существенное снижение реальной и удельной производительности получаемого решения.

На уровне синтеза и программирования макрообъектов создание прикладных масштабируемых программ возможно двумя способами. Первый способ предполагает создание технического решения с участием специалиста-схемотехника, при котором специалист-схемотехник по заданию алгоритмиста фактически создает проблемно-ориентированную вычислительную структуру для решения фрагмента задачи определенного класса, например, линейной алгебры, цифровой обработки сигналов, математической физики и т.п. В состав такого макрообъекта при этом включаются необходимые вычислительные блоки, интерфейсы, блоки внутренней памяти, функциональные преобразователи, объединенные пространственной коммутационной системой макрообъекта, а также устройство управления макрообъектом. Затем созданный макрообъект каскадируется и распараллеливается в необходимом количестве с целью создания вычислительной структуры для решения всей задачи. Достоинством этого подхода является простота использования макрообъектов, созданных заранее для различных проблемных областей, недостатком – сложность разработки макрообъекта, необходимость привлечения специалиста-схемотехника и необходимость отображения вычислительного графа на макрообъектную архитектуру реконфигурируемой вычислительной системы.

Второй способ предполагает использование только языковых средств. В этом случае используются библиотечные элементы, созданные схемотехниками на стадии создания библиотек элементов, для ряда проблемных областей. Библиотечные элементы, включенные в библиотеки, описываются как функции языка высокого уровня. Затем, используя языковые конструкции, эти функции вызываются для исполнения. В результате создается макрообъект с определенными свойствами исключительно средствами языка высокого уровня. При трансляции такой программы в аппаратуре РВС создаются программные макрообъекты, управление которыми осуществляется процедурной, потоковой и управляющей составляющими параллельной программы. Преимуществами такого метода является исключение из процесса создания прикладной программы для РВС специалиста-схемотехника, а также более рациональное использование аппаратного ресурса РВС. К недостаткам можно отнести необходимость наличия проблемно ориентированных библиотек для различных предметных областей.

Общим для первого и второго способов является программирование макрообъектов, которое включает в себя загрузку во все используемые в РВС макрообъекты управляющего пакета с целью их настройки на параметры решаемой задачи. Управляющий пакет включает в себя настройку всех компонентов макрообъекта на выполнение необходимых функций, способы адресации данных в макрообъектной памяти и необходимые функции коммутации пространственного коммутатора макрообъекта. Для загрузки управляющих пакетов используется низкоуровневый язык ассемблера, содержащий команды настройки всех компонентов. Ассемблер используется также для программирования потоков данных, обрабатываемых вычислительной структурой, составленной из макрообъектов.

Программирование на уровне трансляции с языка высокого уровня в логические ячейки ПЛИС и связи между ними осуществляется на языке высокого уровня, при этом осуществляется преобразование программы на языке высокого уровня в структурную, процедурную, потоковую и управляющую составляющие. Достоинством этого подхода являются простота модификации программы и существенное повышение скорости разработки прикладных программ, поскольку исключается участие специалиста-схемотехника при разработке прикладной программы, а также сравнимая с первым уровнем высокая реальная производительность получаемого решения. Недостатком этого уровня является ограничение на смену вычислительной структуры, при котором любое изменение в структуре решаемой задачи требует ее перетрансляции и перегрузки конфигурации РВС.

Поддержка третьего уровня программирования для рассматриваемых систем нового поколения с открытой масштабируемой архитектурой на основе Virtex-6 и Virtex-7, характеризую-

щихся отличной от представителей семейства РВС топологией микросхем ПЛИС и связей между ними, обеспечит при тех же принципах программирования возможность простой адаптации программных компонентов средств разработки для РВС при переходе на новые топологии ПВМ без внесения существенных изменений в код программных компонентов комплекса.

Для поддержки третьего уровня программирования необходима разработка комплекса программного обеспечения РВС-7, включающего новые программы-синтезаторы параллельно-конвейерных вычислительных структур из макрообъектов и обеспечивающего поддержку вводимых расширений всеми средствами разработки прикладных программ на всех необходимых для этого уровнях.

Структура разрабатываемого комплекса программного обеспечения РВС-7 представлена рис.6.

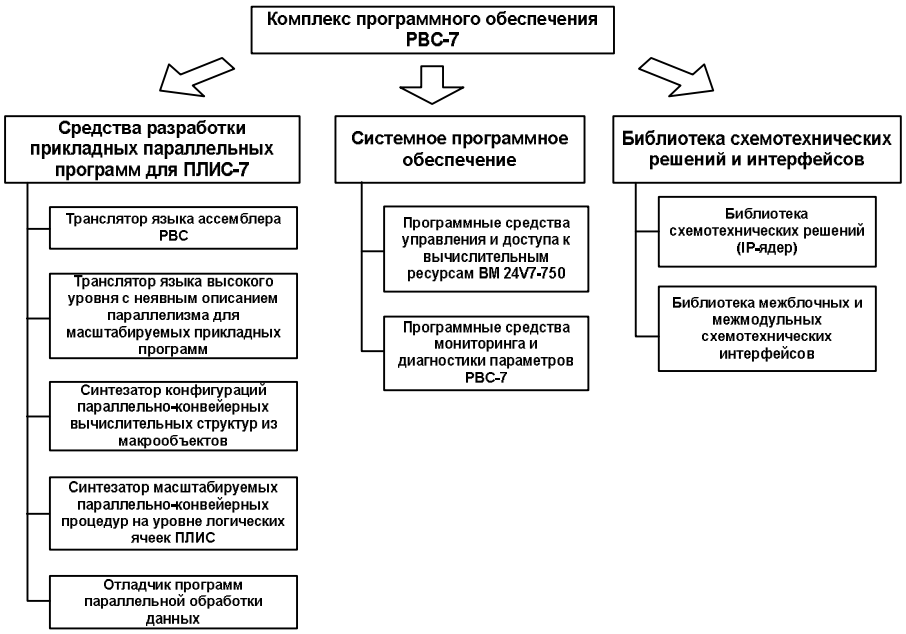


Рис. 6 Структура КПО РВС-7

Основными принципами при проектировании КПО РВС-7 и программных компонентов, входящих в его состав, являются:

- модульная структура КПО РВС-7;
- обеспечение программирования всех составляющих масштабируемых прикладных программ на языке высокого уровня;
- обеспечение реконфигурации прикладных масштабируемых программ без участия высококвалифицированного специалиста-схемотехника;
- обеспечение совместимости и переносимости проектов между РВС разных архитектур;
- возможность масштабирования прикладной задачи при увеличении ресурса.

КПО РВС-7 должен обеспечивать:

- рациональную реализацию прикладных задача различных областей на произвольном количестве взаимосвязанных кристаллов ПЛИС в составе вычислительных блоков ВБР-7 и вычислительных модулей (ВМ) 24V7-750 для любых допустимых конфигураций РВС-7;
- разработку прикладных масштабируемых программ на языке высокого уровня с вызовом библиотечных функций, которые будут настраивать архитектуру системы и реализовывать необходимые вычислительные структуры на множестве ПЛИС;
- тестирование и контроль эксплуатационных параметров составных частей РВС-7;
- управление и администрирование оборудования, в том числе удаленное, включение, выключение, остановку и запуск как отдельных ВМ, так и стоек РВС-7.

Языковые средства программирования прикладных задач должны содержать:

- транслятор языка ассемблера для программирования на уровнях унифицированного макрообъекта и структурно- и процедурно-программируемого макрообъекта;

- транслятор языка программирования РВС высокого уровня с неявным описанием параллелизма для трансляции в логические ячейки ПЛИС и связи между ними;

- среду разработки прикладных программ, поддерживающую языки ассемблера и высокого уровня для РВС;

- среду синтеза масштабируемых параллельно-конвейерных процедур для трансляции структурной составляющей с языка высокого уровня в конфигурацию ПЛИС;

- библиотеку функционально-законченных структурно-реализованных аппаратных устройств (IP-ядер) для различных предметных областей и интерфейсов для согласования скорости обработки информации и связи в единую вычислительную структуру;

- библиотеку программных функций доступа к аппаратным ресурсам базовых модулей РВС для программирования на уровне использования функционально законченных фрагментов задачи.

Разрабатываемый комплекс программного обеспечения позволит создавать эффективные прикладные программы для РВС при решении задач различных предметных областей, обеспечивая удобство программирования и сокращая время разработки прикладного решения, что, с учетом высоких показателей удельной производительности, до 10 раз превосходящей кластерные суперЭВМ на широком классе задач, позволяет считать РВС перспективным направлением развития высокопроизводительной вычислительной техники и средством создания суперЭВМ нового поколения.

4. Заключение

Вычислительные модули нового поколения «Саиф», «Ригель» и 24V7-750 на основе ПЛИС семейства Virtex 6 и Virtex 7 открывают перспективы для построения вычислительных систем более высокой производительности при сохранении стоимости системы по сравнению с РВС на основе ПЛИС семейства Virtex 5. В то же время вычислительные модули обладают достаточной автономностью и могут легко комплексироваться с персональным компьютером типа IBM PC в качестве ускорителей и использоваться при решении различных задач.

Для вычислительных модулей нового поколения «Саиф» и «Ригель» сохраняется преемственность принципов программирования: программирование всех рассмотренных вычислительных модулей осуществляется с помощью единого комплекса системного программного обеспечения, поддерживающего структурно-процедурные методы организации вычислений и определяющие не только организацию параллельных процессов и потоков данных, но и структуру вычислительной системы в поле логических ячеек ПЛИС.

Литература

1. Левин И.И. Реконфигурируемые вычислительные системы с открытой масштабируемой архитектурой // Труды Пятой Международной конференции «Параллельные вычисления и задачи управления» PACO'2010. - М.: Учреждение Российской академии наук Институт проблем управления им. В.А. Трапезникова РАН, 2010. - С.83-95.
2. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. - М.: Янус-К, 2003. – 380 с.
3. Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И. Реконфигурируемые мультиконвейерные вычислительные структуры /Изд. 2-е, перераб. и доп. / Под общ. ред. И.А. Каляева. - Ростов-на-Дону: Изд-во ЮНЦ РАН, 2009. – 344 с.
4. Каляев И.А., Левин И.И. Семейство реконфигурируемых вычислительных системы с высокой реальной производительностью // Труды международной научной конференции «Параллельные вычислительные технологии» (ПАВТ'2009). – Нижний Новгород: электронное издание НГУ имени Н.И. Лобачевского, 2009. – С.186-196.

5. Дмитренко Н.Н., Каляев И.А., Левин И.И., Семерников Е.А. Развитие аппаратной платформы реконфигурируемых вычислительных систем // Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи. – М.: Изд-во МГУ, 2010. – С. 315-320.
6. Дордопуло А.И., Каляев И.А., Левин И.И., Семерников Е.А. Высокопроизводительные реконфигурируемые вычислительные системы нового поколения // Труды Международной суперкомпьютерной конференции с элементами научной школы для молодежи «Научный сервис в сети Интернет: экзафлопсное будущее». – М.: Изд-во , 2011. – С. 42-49.
7. Каляев И.А., Левин И.И., Семерников Е.А., Дордопуло А.И. Реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex-6 // Сборник трудов Международной научной конференции «Параллельные вычислительные технологии 2011» (ПАВТ 2011). – Челябинск-М.: Издательский центр ЮУрГУ [Электронный ресурс], 2011. - С. 203–210.

Реализация параллельного алгоритма обучения в методе градиентного бустинга деревьев решений для систем с распределенной памятью*

П.Н. Дружков, А.Н. Половинкин

Нижегородский государственный университет им. Н.И. Лобачевского

В работе описывается реализация одного из наиболее перспективных алгоритмов обучения с учителем - градиентного бустинга деревьев решений (Gradient Boosting Trees). Предлагается схема параллельной реализации алгоритма обучения для систем с распределенной памятью. Приводятся результаты вычислительных экспериментов и анализ эффективности предложенного подхода к распараллеливанию.

1. Введение

Машинное обучение является подразделом весьма обширной области науки, изучающей искусственный интеллект. Алгоритмы, относящиеся к данному направлению, используются при решении задач, для которых зачастую сложно или невозможно придумать явный алгоритм решения: предсказание погоды, прогнозирование экономических и социальных процессов, медицинская диагностика, детектирование объектов на фото или видео, распознавание текста, речи, создание антивирусных программ и алгоритмов фильтрации рекламы и спама.

В настоящее время известно достаточно много алгоритмов обучения с учителем, предназначенных для решения задачи восстановления регрессии или классификации: машина опорных векторов [15], метод *K* ближайших соседей [10], нейронные сети [10], AdaBoost [10], деревья решений [2] и их различные ансамбли (случайные деревья [1], полностью случайные деревья [9], градиентный бустинг деревьев решений [7, 8]). В рамках данной работы рассматривается программная реализация одного из наиболее перспективных алгоритмов обучения с учителем – алгоритма градиентного бустинга деревьев решений (GBT – gradient boosting trees), которая является первой полнофункциональной C/C++ реализацией данного метода с открытым кодом. Результаты вычислительного эксперимента, проведенного с использованием широко распространенных наборов реальных данных, взятых из репозитория UCI [14], свидетельствуют о конкурентоспособности предлагаемой реализации по сравнению с реализациями других алгоритмов. Разработанный код интегрирован в одну из наиболее известных свободно распространяемых библиотек компьютерного зрения OpenCV [4, 11].

Необходимо отметить, что многие из решаемых в настоящее время практических задач машинного обучения и компьютерного зрения требуют обработки значительного объема входных данных. Это связано с тем, что каждый исследуемый объект может быть описан вектором признаков, содержащим сотни или даже тысячи переменных, а обучающая и тестовая выборки могут содержать десятки тысяч описаний объектов. В связи с этим, наряду с качеством предсказания на одно из первых мест встает вопрос производительности используемого алгоритма. В работе дается обзор различных аспектов параллельной реализации алгоритма обучения модели и предсказания на новых данных, а также предлагается и анализируется подход к распараллеливанию алгоритма обучения модели для систем с распределенной памятью.

* Авторы благодарят И.Б. Меерова и Н.Ю. Золотых (ННГУ) за ценные замечания и полезные обсуждения. Работа выполнена в рамках программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы», государственный контракт № 11.519.11.4015.

2. Градиентный бустинг деревьев решений

2.1 Постановка задачи

Одной из задач, изучаемой в машинном обучении, является *задача обучения с учителем*. В рамках этой задачи дано некоторое множество *объектов* $X = X_1 \times X_2 \times \ldots \times X_p$ *(пространство признаков)*. Каждому объекту $x = (\xi_1, \xi_2, \ldots, \xi_p) \in X$ поставлена в соответствие величина y, называемая *выходом*, или *ответом*, и принадлежащая множеству допустимых ответов Y. Упорядоченная пара «объект-ответ» (x,y), где $x \in X$, $y \in Y$ называется *прецедентом*. Требуется восстановить зависимость между входом и выходом, основываясь на данных о конечном наборе прецедентов, называемом *обучающей выборкой*: $\{(x_i, y_i) \mid x_i \in X, y_i \in Y, i = 1, ..., n\}$. Другими словами, задача состоит в построении функции $f: X \to Y$ из некоторого множества K, которая, получив на вход x, предсказала бы значение ответа y как можно точнее. В случае конечного Y, говорят о *задаче классификации*, если $Y = \mathbf{R}$ – *задаче восстановления регрессии* [10]. Процесс нахождения f называется *обучением* (*тренировкой*, *настройкой)* модели, процесс определения выхода по некоторому входу с помощью уже построенной модели – *предсказанием*.

2.2 Метод решения

Один из общих подходов решения задач обучения заключается в комбинировании моделей. Две основные конкурирующие идеи данного подхода – бэггинг (*bagging* от *Bootstrap Aggregating*) [3] и бустинг (*boosting*) [6]. Первая из них состоит в построении множества независимых между собой моделей с дальнейшим принятием решения путем голосования в случае задачи классификации и усреднения в случае регрессии. Данный подход реализован в алгоритме случайных деревьев (*random trees* или *random forest*). Бустинг, в противоположность бэггингу, обучает каждую следующую модель с использованием данных об ошибках предыдущих моделей. Распространенным выбором базовой модели в упомянутых выше алгоритмах являются деревья решений [2], что обусловлено их универсальностью и наличием эффективных алгоритмов обучения. Напомним, что дерево решений рекурсивно разбивает пространство признаков на непересекающиеся области с помощью правил вида $\xi_j \le t$, если $\xi_j \in \mathbf{R}$, и вида $\xi_j \in A \subset X_j$, где X_j – конечное множество возможных значений переменной ξ_j. Каждой из полученных областей присваивается некоторое значение результирующей переменной y. Таким образом, дерево решений определяет кусочно-постоянную функцию.

Алгоритм градиентного бустинга деревьев решений является развитием бустинг-идеи. Он позволяет строить аддитивную функцию в виде суммы деревьев решений итерационно по аналогии с методом градиентного спуска. Данный подход позволяет расширить круг решаемых этим алгоритмом задач, а также зачастую получить выигрыш в точности предсказания.

Далее приводится краткое описание алгоритма обучения градиентного бустинга деревьев решений для задачи восстановления регрессии в случае использования квадратичной функции потерь. Пусть обучающая выборка содержит n прецедентов, y_i – значение ответа для i-го прецедента, $T_j(x_i)$ – значение, предсказанное j-м деревом в ансамбле для i-го объекта, ν – коэффициент масшабирования. Тогда псевдоостатком для i-го объекта на m-м шаге алгоритма обучения называется значение $\tilde{y}_i = y_i - T_0(x_i) - \nu \cdot \sum_{s=1}^{m-1} T_s(x_i)$, что соответствует разности между истинным значением ответа и предсказанием ансамбля деревьев решений, построенным на (m – 1)-м шаге алгоритма обучения. Пусть M – общее число деревьев в ансамбле, тогда общая схема тренировки модели, изображенная на рис.1, может быть сформулирована следующим образом:

1. Обучить дерево T_0 на исходном наборе данных $(x_i, y_i), i = 1,...,n$
2. Для каждого $m = 1,2,...,M$
   - для всех объектов в обучающей выборке вычислить псевдоостатки $\tilde{y}_i, i = 1,...,n$
   - добавить в ансамбль новое дерево, обученное на наборе данных $(x_i, \tilde{y}_i), i = 1,...,n$

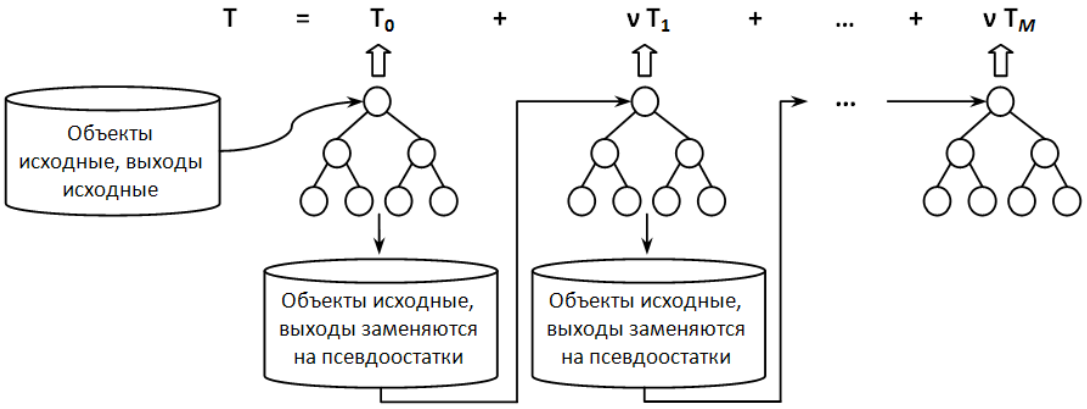


Рис. 1. Алгоритм обучения модели градиентного бустинга деревьев решений

Детальное описание алгоритма обучения, а также подробности, связанные с тренировкой отдельных деревьев решений, и особенности реализации алгоритма для задач восстановления регресии с другими функциями потерь, а также классификации с двумя и более классами можно найти в [7]. Здесь же отметим лишь, что построение одиночного дерева решений), если считать количество возможных значений каждой категориальной переменной константным, имеет трудоемкость $O(pn\log(n)+pnd)$, где p – размерность пространства признаков, n – объем обучающей выборки, d –глубина дерева. Алгоритм построения модели градиентного бустинга деревьев решений имеет трудоемкость $O(MK(pn\log(n)+pnd))$, где M – общее число деревьев в одном ансамбле, K – количество категорий в задаче классификации, для регрессионных задач K=1. При этом исходные данные (обучающая выборка) занимают порядка $O(pn)$ памяти.

Таким образом, на выходе алгоритма обучения мы получаем набор из M деревьев решений, и для осуществления предсказания, т. е. определения выхода y для нового объекта x, следует вычислить сумму $y = T_0(x) + \nu \cdot \sum_{m=1}^{M} T_m(x)$.

Таблица 1. Результаты экспериментального сравнения алгоритмов обучения с учителем: ошибки, полученные методом перекрестного контроля

Название набора данных	*Градиентный бустинг* (GBT)	*Дерево решений* (CvDTree)	*Случайные деревья* (CvRTrees)	*Случайные деревья* (CvERTrees)	*Машина опорных векторов* (CvSVM)
auto-mpg	2	2.238	*1.879*	2.147	2.981
Computer hardware	12.62	15.62	11.62	*9.631*	37
Concrete slump	2.257	2.923	2.6	2.359	*1.767*
Forestfires	18.74	17.26	17.79	16.64	*12.9*
Boston housing	*2.033*	2.602	2.135	2.196	4.049
imports-85	1306	1649	*1290*	1487	1787
Servo	*0.238*	0.258	0.247	0.42	0.655
Abalone	*1.47*	1.604	1.492	1.498	2.091

2.3 Реализация алгоритма градиентного бустинга деревьев решений

Авторами данной работы была выполнена программная реализация алгоритма градиентного бустинга деревьев решений [16], включающая как алгоритм тренировки модели, так и ее дальнейшее использование для предсказания. В данном разделе приведены некоторые экспериментальные результаты, показывающие достоинства и недостатки метода градиентного бустинга. Наряду с описываемым подходом были рассмотрены конкурирующие алгоритмы: одиночные деревья решений (алгоритм CART) [2], случайные деревья (случайные леса) [1, 8], ма-

шина опорных векторов [15]. Программной основой проведенных экспериментов является открытая библиотека компьютерного зрения OpenCV: все результаты, относящиеся к конкурирующим алгоритмам, были получены непосредственно с помощью ее компонентов: CvDTree, CvRTrees, CvERTrees и CvSVM. Эксперименты проводились на наборах реальных данных, взятых из репозитория UCI, при этом мерой качества модели считалась средняя абсолютная ошибка 10-кратного перекрестного контроля, см. табл. 1. Приведенные результаты говорят о том, что алгоритм градиентного бустинга деревьев решений для различных задач дает лучшие, или сопоставимые по качеству результаты в сравнении с другими алгоритмами.

3. Параллельная реализация алгоритма градиентного бустинга деревьев решений для систем с распределенной памятью

3.1 О подходах к распараллеливанию вычислений в алгоритме градиентного бустинга

Значительное количество задач из области машинного обучения характеризуется большим объемом входных данных, обусловленным как количеством объектов в обучающей и тестовой выборках, так и размерностью пространства признаков. В зависимости от характера использования алгоритма в прикладной задаче наиболее требовательной к быстродействию может оказаться как стадия обучения модели, так и стадия предсказания результата на новых данных.

Несмотря на то, что предсказание в алгоритме градиентного бустинга деревьев решений является менее трудоемким по сравнению с построением модели, время работы этого алгоритма зачастую также является критичным. На практике часто приходится выполнять не единичные предсказания, а целые серии: например, в некоторых алгоритмах, решающих задачу детектирования объектов на изображении методом «скользящего окна», может потребоваться выполнение десятков тысяч предсказаний на одно изображение [5]; кроме того, в некоторых прикладных задачах из данной области требуется работа в режиме реального времени.

В работе [17] авторами были предложены и реализованы два различных подхода к распараллеливанию алгоритма предсказания для систем с общей памятью с использованием обученной модели градиентного бустинга деревьев решений. Напомним, что для получения выхода y по некоторому входу x необходимо вычислить сумму предсказаний всех деревьев из имеющегося ансамбля. Так как значение каждого слагаемого может быть получено независимо от остальных, можно рассмотреть первую параллельную схему предсказания, в которой для одного объекта x значения предсказаний отдельных деревьев $T_m(x)$ в сумме $y = T_0(x) + \nu \cdot \sum_{m=1}^{M} T_m(x)$ вычисляются параллельно несколькими потоками. Другой рассмотренный подход: распараллеливание по данным – основан на необходимости одновременного предсказания для большого количества новых объектов. В данном случае выполняется параллельное вычисление значений $y_i = T_0(x_i) + \nu \cdot \sum_{m=1}^{M} T_m(x_i)$ для нескольких объектов x_i. Очевидно, что предложенные схемы распараллеливания алгоритма предсказания применимы и к системам с распределенной памятью.

Возможные подходы к распараллеливанию алгоритма обучения не являются столь тривиальными. К сожалению, бустинг-алгоритмы в целом являются плохо распараллеливаемыми: подход, связанный с одновременным построением нескольких классификаторов в ансамбле, здесь невозможен в силу того, что тренировка каждого следующего дерева решений требует результатов предсказания всех предыдущих. В связи с этим, основные усилия предпринимаются для создания параллельных алгоритмов на уровне базовых моделей, в частности деревьев решений [2]. Большинство таких подходов основано на параллелизме по данным и упрощенных алгоритмах построения деревьев решений [12, 13]. Таким образом, можно говорить, об ориентированности большинства методов на большие объемы исходных данных. В данной работе предлагается схема параллельной реализации алгоритма обучения для отдельного широко распространенного класса задач: классификация объектов с большим числом категорий (к дан-

ному классу относятся, например, задача классификации изображений, автоматизированное построение электронных каталогов научных текстов). Предлагаемый подход не затрагивает алгоритм построения отдельных деревьев решений, что позволяет достичь ускорения процесса обучения модели без потери ее качества. Также, данный метод может быть скомбинирован с параллельными алгоритмами построения базовых моделей, что может, в свою очередь, привести к улучшению масштабируемости параллельного алгоритма градиентного бустинга в целом. Также следует отметить, что, несмотря на наличие других подходов к распараллеливанию рассматриваемого алгоритма, функциональность существующих открытых реализаций весь ограничена, что не позволяет произвести непосредственное сравнение с ними.

3.2 Параллельная реализация алгоритма градиентного бустинга деревьев решений для задач классификации

В отличие от приведенного выше алгоритма обучения модели для задач восстановления регрессии, где строится одна последовательность деревьев, модель для решения задачи классификации [7] состоит из K последовательностей деревьев решений заданной длины M, где K – число категорий, к которым может относиться выходная переменная. Каждая из таких последовательностей $\{T_{0k}, T_{1k}, ..., T_{Mk}\}, k = 1, ..., K$ оценивает вероятность принадлежности объекта к k-му классу. Таким образом, в этом случае общее число обучаемых деревьев увеличивается в K раз, что оказывает серьезное негативное влияние на производительность процедуры обучения. Общая схема построения модели градиентного бустинга деревьев решений при использовании в качестве функции потерь кросс-энтропии, применяемой в текущей реализации для решения задач классификации, выглядит следующим образом:

1. Для всех $m = 1, ..., M$
   a. Для всех $k = 1, ..., K$
      a. Вычислить антиградиент функции потерь $\tilde{y}_{ik}, i = 1, ..., n$ [7]
      b. Обучить дерево решений T_{mk} на данных $(x_i, \tilde{y}_{ik}), i = 1, ..., n$
   b. Обновить модель, добавив деревья $T_{m1}, T_{m2}, ..., T_{mK}$ в соответствующие последовательности

Необходимо отметить, что значение функции антиградиента для определенного k зависит от значения функций предсказания, соответствующих всем уже построенным последовательностям, поэтому ансамбль деревьев решений для одного класса не может быть получен независимо от остальных последовательностей. При этом построение K деревьев решений на m-й итерации алгоритма может быть выполнено параллельно. Так как при обучении одиночного дерева решений производится интенсивная работа с памятью и реализация процесса построения дерева в библиотеке OpenCV является параллельной на общей памяти, параллельное построение нескольких деревьев решений (особенно при большом числе категорий K) имеет смысл производить на распределенной памяти.

Основываясь на этой идее, была выполнена реализация алгоритма обучения модели градиентного бустинга деревьев решений с использованием технологии MPI. Построение модели осуществляется K процессами, каждый из которых строит деревья одной последовательности. После построения очередного дерева решений работа всех процессов синхронизируется, и каждый процесс рассылает всем остальным данные о текущих предсказаниях соответствующего ему ансамбля на объектах обучающей выборки, которые необходимы для выполнения следующей итерации алгоритма. Данные сообщения представляют собой вектора значений типа float из n компонент, где n – объем обучающей выборки. По окончанию построения деревьев производится пересылка всех полученных последовательностей на нулевой процесс, где происходит сбор модели. Общая схема предлагаемого параллельного подхода представлена на рис. 2.

С использованием описанной реализации были проведены эксперименты, оценивающие целесообразность использования данного подхода. Вычислительный эксперимент проводился с использованием кластера ННГУ, в состав которого входят 64 двухпроцессорных двухъядерных сервера Intel Xeon 3.2 GHz, 4 Gb RAM.

С помощью алгоритма градиентного бустинга деревьев решений были решены две задачи: распознавание рукописных букв латинского алфавита (задача взята из репозитория UCI [14]) и рубрикация статей базы журнала «Вестник ННГУ». Описание использованных наборов данных приведено в табл. 2. Следует отметить, что коллекции данных, являющиеся стандартом при оценке качества работы алгоритмов обучения с учителем, такие как UCI, содержат малое количество многоклассовых задач, на решение которых и ориентирована предлагаемая параллельная реализация. В связи с этим, выбор «стандартных» задач был существенно ограничен.

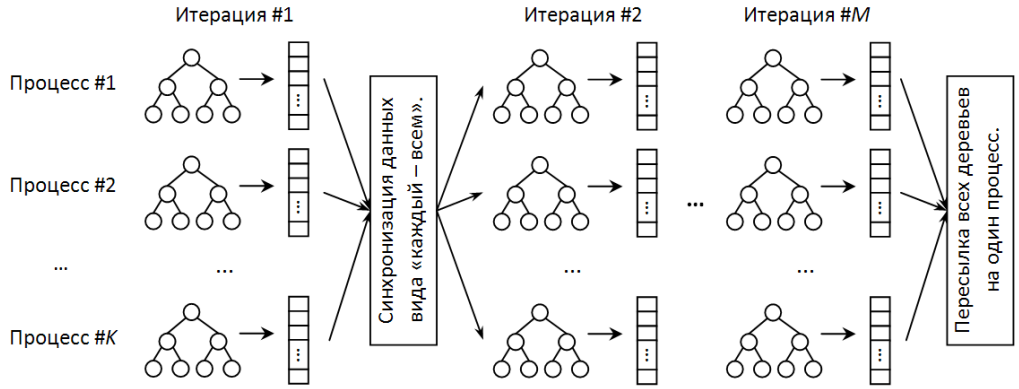


Рис. 2. Схема параллельной версии алгоритма обучения модели градиентного бустинга деревьев решений для решения задачи классификации

Таблица 2. Описание наборов данных, использованных в вычислительном эксперименте

Название набора данных	*Объем обучающей выборки*	*Размерность пространства признаков*	*Количество классов*
Letter Recognition	16000	16	26
База журнала «Вестник ННГУ»	1855	181822	18

Таблица 3. Результаты вычислительного эксперимента

Название набора данных	*Время работы последовательной версии, сек.*	*Кол-во процессов для параллельной версии*	*Время работы параллельной версии, сек.*	*Ускорение*
Letter Recognition	2743	26	170	~16.1
База журнала «Вестник ННГУ»	149114	18	9888	~15

Параллельным алгоритмом обучения для каждого набора данных использовалось число процессов, равное числу классов K в решаемой задаче. Каждый из создаваемых процессов строил последовательность из 1000 деревьев. Время работы сравнивалось с версией программы, работающей на одном узле и строящей все деревья последовательно. Результаты вычислительного эксперимента указаны в табл. 3.

Как можно видеть из приведенных результатов, эффективность параллельной версии зависит от характеристик конкретной задачи. Во-первых, малая размерность пространства признаков для задачи распознавания букв при относительно небольшом объеме обучающей выборки приводит к тому, что построение отдельного дерева происходит достаточно быстро, и, следовательно, время, затрачиваемое на пересылки данных, дает больший вклад в общее время работы алгоритма обучения. Во-вторых, больший объем выборки и количество классов соответствуют большему объему пересылок на каждой итерации алгоритма, что также негативно сказывается на эффективности параллельного подхода.

Таким образом, можно сделать вывод о том, что задачи (наборы данных) с относительно небольшим объемом обучающей выборки и большим числом предикативных переменных являются наиболее подходящими для текущей реализации кластерной версии GBT.

4. Заключение

В статье рассмотрены аспекты параллельной реализации одного из наиболее перспективных алгоритмов обучения с учителем – градиентного бустинга деревьев решений. Предложена и реализована параллельная схема алгоритма обучения модели для задачи классификации объектов с большим числом классов для систем с распределенной памятью. Эффективность предложенного подхода подтверждена результатами вычислительных экспериментов.

Литература

1. Breiman L. Random Forests. // Machine Learning. 2001. V. 45, №. 1, P. 5-32.
2. Breiman L., Friedman J., Olshen R., Stone C. Classification and Regression Trees. Wadsworth, 1983.
3. Breiman L. Bagging predictors // Machine Learning. 1996. V. 26, № 2, P. 123-140.
4. Druzhkov P. N., Eruhimov V. L., Kozinov E. A., Kustikova V. D., Meyerov I. B., Polovinkin A. N., Zolotykh N. Yu. On some new object detection features in OpenCV Library // Pattern Recognition and Image Analysis. 2011. V. 21, № 3. P. 384–386.
5. Enzweiler M., Gavrila D. M . Monocular Pedestrian Detection: Survey and Experiments // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2009. V.31, № 12. P. 2179–2195.
6. Freund Y., Schapire R. Experiments with a New Boosting Algorithm // Machine Learning: Proceedings of the Thirteenth International Conference. 1996.
7. Friedman J. H. Greedy Function Approximation: a Gradient Boosting Machine. Technical report. Dept. of Statistics, Stanford University, 1999.
8. Friedman J. H. Stochastic Gradient Boosting. Technical report. Dept. of Statistics, Stanford University, 1999.
9. Geurts P., Ernst D., Wehenkel L. Extremely Randomized Trees // Machine Learning. 2006. V. 36, № 1. P. 3–42.
10. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. Springer, 2008.
11. OpenCV Wiki. URL: http://opencv.willowgarage.com/wiki (дата обращения: 02.12.2011).
12. Panda B., Herbach J.S., Basu S., Bayardo R.J. PLANET: massively parallel learning of tree ensembles with MapReduce // VLDB Endow. 2009. V. 2, №. 2, P. 1426–1437.
13. Tyree S., Weinberger K.Q., Agrawal K., Paykin J. Parallel boosted regression trees for web search ranking // WWW. 2011. P. 387-396.
14. UCI Machine Learning Repository. URL: http://archive.ics.uci.edu/ml (дата обращения: 02.12.2011).
15. Вапник В. Н. Восстановление зависимостей по эмпирическим данным. М.: Наука, 1979. 448 с.
16. Дружков П.Н., Золотых Н.Ю., Половинкин А.Н. Программная реализация алгоритма градиентного бустинга деревьев решений // Вестник Нижегородского государственного университета им. Н. И. Лобачевского. 2011. № 1, С. 193–200.
17. Дружков П.Н., Золотых Н.Ю., Половинкин А.Н. Реализация параллельного алгоритма предсказания в методе градиентного бустинга деревьев решений // Вестник ЮУрГУ. Сер. «Математическое моделирование и программирование». 2011. Вып. 10, №37(254), С. 82 – 89.

О численном исследовании ламинарно-турбулентного перехода с использованием различных параллельных архитектур

Н.М. Евстигнеев, О.И. Рябков

Институт Системного Анализа Российской Академии Наук

В работе построен численный метод решения уравнений Навье-Стокса для несжимаемой среды высокого порядка аппроксимации, позволяющий разрешать нестационарные аттракторы, возникающие в системе при изменении числа Рейнольдса. Были реализованы два варианта метода: MPI и CUDA. Использование GPU Tesla C1060 дает четырехкратное ускорение по сравнению с использованием восьми ядер Intel Xenon. Также было реализовано обобщение метода для решения задач несжимаемой МГД среды, но только для MPI-совместимых архитектур. При решении последней задачи среди прочих был использован вычислительный комплекс IBM Blue Gene/P факультета ВМК МГУ.

1. Введение

Вопрос о том, что представляет собой турбулентное течение жидкости, и как происходит ламинарно-турбулентный переход, был поставлен еще до появления мощных ЭВМ. Различные варианты (сценарии) были предложены физиками еще в середине XX века. Например, сценарий Ландау-Хопфа [4]. Насколько нам известно, прямое численное моделирование для разрешения данной задачи не применялось до работы [9]. Поводом для проведения подобного численного эксперимента послужила серия работ [5], выявившая некий универсальный сценарий перехода к хаосу в различных малоразмерных динамических системах, и предположение о том, что в задаче ламинарно-турбулентного перехода может реализовываться аналогичный сценарий. Данная работа является прямым продолжением указанной серии работ.

2. Постановка задачи

В работе исследовалась начально-краевая задача (НКЗ) для уравнений Навье-Стокса для несжимаемой жидкости [2]:

$$\frac{\partial V}{\partial t} + (V \cdot \nabla)V + \nabla P = \frac{1}{R}\nabla^2 V + \frac{1}{Fr} \text{ в } Q = \Omega \times (0,t), \tag{1}$$

$$\nabla \cdot V = 0 \text{ в } Q, \tag{2}$$

$$V = f(\vec{x}), \text{ на } \partial\Omega_0 \times (0,t),\ V = 0, \text{ на } \partial\Omega_1 \times (0,t), \frac{\partial V}{\partial \vec{n}} = 0, \text{ на } \partial\Omega_2 \times (0,t), \tag{3}$$

$$V(\vec{x},0) = V_0(x,y,z), \text{ на } \Omega \text{ при } \nabla \cdot V_0(x,y,z) = 0. \tag{4}$$

Область расчета Ω представляет собой канал прямоугольного сечения разбитый на две части. Область стыка двух частей канала формирует ступеньку по одной из выбранных осей. Торцевые части канала задаются как входные и выходные граничные условия.

Целью работы было определение начальных стадий ламинарно-турбулентного перехода (ЛТП) [4,5,9] для данной НКЗ в терминах бифуркаций решений данной системы уравнений.

Также исследовалась НКЗ МГД течения в каверне [3]:

$$\frac{\partial V}{\partial t} + (V \cdot \nabla)V - \frac{N}{R_m}(B \cdot \nabla)B + \nabla P = \frac{1}{R}\nabla^2 V \text{ в } Q = \Omega \times (0,t), \tag{5}$$

$$\frac{\partial B}{\partial t} - (B \cdot \nabla)V + (V \cdot \nabla)B = \frac{1}{R_m}\nabla^2 B \text{ в } Q, \tag{6}$$

$$\nabla \cdot V = 0 \text{ в } Q, \nabla \cdot B = 0 \text{ в } Q, \tag{7}$$

$$V = f(\vec{x}), \text{ на } \partial\Omega_0 \times (0,t), \; V = 0, \text{ на } \partial\Omega_1 \times (0,t), \tag{8}$$

$$V(\vec{x},0) = V_0(x,y,z), \text{ на } \Omega \text{ при } \nabla \cdot V_0(x,y,z) = 0. \tag{9}$$

$$B = f(\vec{x}), \text{ на } \partial\Omega_0 \times (0,t), \tag{10}$$

$$B(\vec{x},0) = B_0(x,y,z), \text{ на } \Omega \text{ при } \nabla \cdot B_0(x,y,z) = 0. \tag{11}$$

Область расчета Ω данной НКЗ представляет собой двумерный прямоугольник с равными сторонами, причем на одной из них задано условие Дирихле на скорость («движущаяся крышка»), а на остальных условие «прилипания», соответствующие твердой стенке. Вдоль всех границ задано условие Дирихле на магнитное поле, соответствующее приложенному внешнему магнитному полю (величины внешнего поля и скорости течения вдоль крышки равны 1).

3. Численный метод и реализация

Для решения НКЗ (1)-(4) использовались два метода: PISO и проекционный метод.

Полный цикл расчета для всех проекционных методов основан на следующей полудискретной схеме метода расщепления по физическим процессам:

$$\left.\begin{array}{l} 1.\ V' - V^n = -\delta t (V^n \cdot \nabla) V^n \\ 2.\ \widetilde{V} - V' = \delta t \cdot R^{-1} \nabla^2 \widetilde{V}^{\sigma} \\ while\ \nabla \cdot V^{n+1} \neq 0; \left\{\begin{array}{l} 3.\ \nabla^2 P = -\delta \cdot \widetilde{V}^{\beta} / \Delta t \\ 4.\ V^{\beta+1} = \widetilde{V} - \delta t \nabla P \end{array}\right. \\ 5.\ V^{n+1} = V^{\beta+1} \end{array}\right\}, \tag{12}$$

В использованной в работе схеме применялся неявный метод Рунге-Кутта 4-ого порядка аппроксимации. Диффузионный член аппроксимировался методом конечных разностей шестого порядка, конвективный член – методом WENO5 [11]. Решение уравнения Пуассона, а также аппроксимация градиента давления были выполнены методом конечных элементов. Схема PISO была реализована в качестве контрольного метода для сравнения результатов, получаемых при разных методах аппроксимации уравнений.

Решение НКЗ (5)-(11) производилось аналогичным проекционным методом (см. [7,8]). Конвективная часть для переменных V и B аппроксимировалась совместно, путем перехода к характеристическим переменным. Также в схеме был реализован метод коррекции дивергенции магнитного поля, необходимый для устранения численного магнитного монополя. Данная процедура была реализована аналогично процедуре коррекции дивергенции давления, но для аппроксимации градиента давления и дивергенции поля B использовались центральные разности, а в качестве дискретного оператора Лапласа бралась композиция указанных дискретных операторов дивергенции и градиента. Для дискретизации по времени использовалась полностью явная схема Рунге-Кутта 3-его порядка аппроксимации.

Описанная выше схема для НКЗ (1)-(4) была реализована в двух вариантах: для MPI-совместимых архитектур и с использованием архитектуры CUDA (в обоих случаях использовался язык программирования C++). Результаты тестирования показали, что использование GPU Tesla C1060 дает четырехкратное ускорение по сравнению с использованием восьми ядер Intel Xenon.

НКЗ (5)-(11) была реализована только для MPI-совместимых архитектур. Решение дискретного уравнения Пуассона проводилось двумя методами: итерационным с применением

multilevel предобуславливателя и методом быстрого преобразования Фурье (использовалась библиотека FFTW). Расчет проводился на кластере с 8 процессорами Intel Xeon и на вычислительном комплексе IBM BlueGene/P факультета ВМК МГУ. Размер расчетной сетки (192x192 и 256x256) не позволил применять метод Фурье на системе BlueGene (в силу очень большого минимального количества запускаемых MPI-процессов в данной архитектуре). В результате скорость расчета в системе BlueGene/P была всего в два раза больше, чем при использовании указанного кластера. Данный факт объясняется тем, что описанная задача (в силу своей двухмерности) является слишком маленькой для такой системы как BlueGene/P. В дальнейшем нами предполагается использовать эту систему для решения более трудоемких (например, трехмерных) задач.

4. Анализ результатов моделирования

4.1 Гидродинамическое течение с уступа

Для получения данных по времени и проведения анализа требовалась запись данных из точек внутри расчетной области в течении длительного времени для функций V_x, V_y, V_z. На рисунках 1-3 эти переменных обозначены как U, V, W соответственно. Поскольку хранение данных для всей расчетной области за промежуток времени достаточный для статистического анализа невозможен ввиду отсутствия такого количества дискового пространства (для одной серии расчета, т.е. для фиксированного числа Рейнольдса требуется около 6,43 терабайта), в области расчета были выбраны несколько точек, и на основе этих точек проводился дальнейший анализ. Для этого было выбрано пять точек внутри Ω с декартовыми координатами: p1={0.1 0.5 0.5}; p2={0.2 0.5 0.5}; p3={0.5 0.5 0.5}; p4={0.7 0.5 0.5}; p5={0.8 0.1 0.1} в относительных величинах к полным длинам по каждому направлению. По результатам полученных данных строились трехмерные подпространства бесконечномерного (в дальнейшем бесконечномерным считается конечномерное фазовое пространство, порожденное численным решением, размерность которого равна или больше размерности возникающего в системе (1)-(4) аттрактора при данном числе R) фазового пространства в координатах составляющих вектор-функции скорости Θ={Vx, Vy, Vz}. Бифуркационным параметром системы уравнений Навье-Стокса (1)-(4) выступает число Рейнольдса R.

При увеличении числа Рейнольдса со 100 до 736 течение остается ламинарным с формированием стационарных рециркуляционных зон. При этом стационарному решению соответствует устойчивая точка в бесконечномерном фазовом пространстве и, соответственно, во всех ее подпространствах, включая трехмерное подпространство вектор-функций скоростей. Начиная с $R = 737$ в системе наблюдается периодическое решение с одной частотой. В Θ из каждой устойчивой неподвижной точки рождается свой устойчивый предельный цикл, что соответствует рождению цикла из неподвижной точки в бесконечномерном фазовом пространстве задачи н.к.з. для уравнений Навье-Стокса. Установлено что при увеличении числа до $R = 850$ цикл теряет устойчивость, порождая двумерный инвариантный тор как $T^2 = C_1 \otimes C_2$ в результате бифуркации Андронова-Хопфа. На рисунке 1 показаны проекции фазового пространства в точке p4 при $R = 849$ (цикл) и $R = 850$ (тор). При этом важно отметить, что этот тор находится во всем бесконечномерном пространстве, т.е. тор обнаруживается во всех пробных точках P_i. Это связано, скорее всего, с условием несжимаемости, поскольку все возмущения передаются одновременно во все стороны через градиент давления. Так точное значение параметра бифуркации R можно установить из анализа фазовых портретов в разных точках на уровне порогового числа.

Дальнейшее увеличение бифуркационного параметра приводит к значительному усложнению формы инвариантного двухмерного тора, наблюдаемого в системе.

Начиная с $R = 883$ во всех точках инвариантный двухмерный тор теряет устойчивость и через бифуркацию Андронова-Хопфа переходит в тор большей размерности – в трехмерный инвариантный тор представляющей собой топологическое произведение трех циклов некрат-

ных частот $T^3 = C_1 \otimes C_2 \otimes C_3$. Проекцию трехмерного тора в трехмерные фазовые пространства видно на рис.2-3 для $R \le 883$.

Для анализа получаемого трехмерного тора необходимо провести еще одно сечение плоскостью, например, $W = 0.0$. Выяснение дальнейшего усложнения трехмерного тора при увеличении числа Рейнольдса потребовало значительного количества данных. Уже при $R = 883.8$ трехмерный инвариантный тор теряет устойчивость и через бифуркацию удвоения периода переходит в трехмерный тор удвоенного периода. На рисунках 4-5 изображены: первое сечение (слева) и вспомогательное сечение (справа). Сечение на рисунке 4 соответствует простому трехмерному тору, а на рисунке 5 – удвоенному. Для получения достоверных данных потребовалось два года расчетов и 373Гб данных на одну точку.

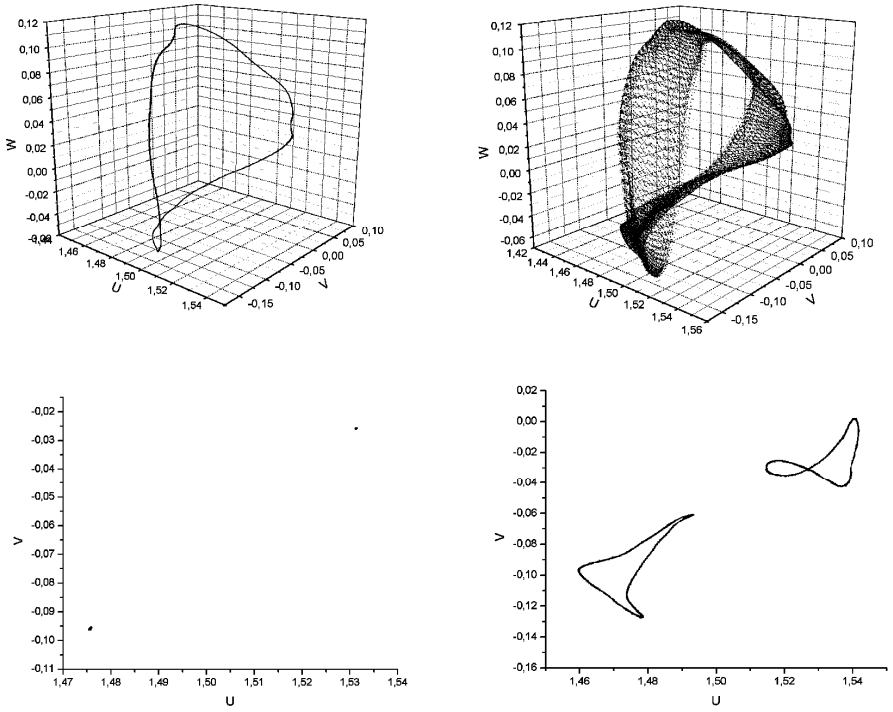


Рис. 1. $R = 849$ слева: сверху фазовое подпространство $\Theta(P_4)$, снизу сечение плоскостью $\{V_x; V_y; 1/2(W_{\max} + W_{\min})\}$; $R = 850$ справа: сверху фазовое подпространство $\Theta(P_4)$, снизу сечение плоскостью $\{V_x; V_y; 1/2(W_{\max} + W_{\min})\}$.

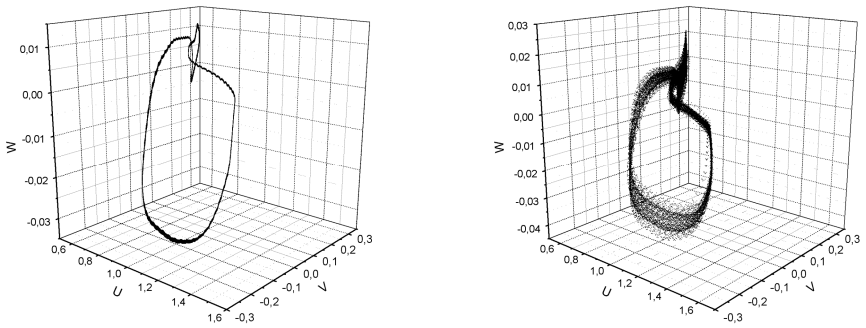


Рис. 2. $R = 850$ слева: фазовое подпространство $\Theta(P_1)$; $R = 851$ справа: фазовое подпространство $\Theta(P_1)$.

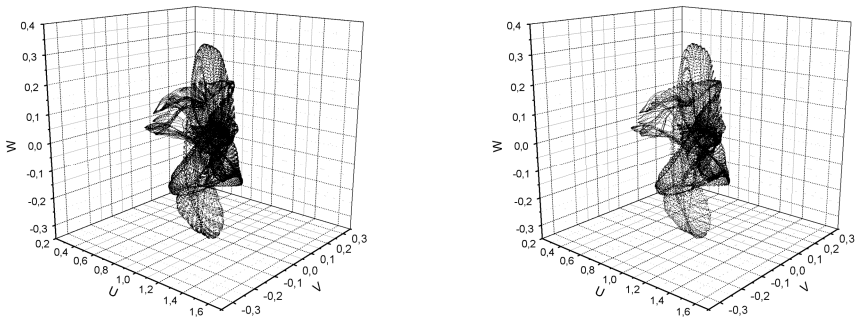


Рис. 3. $R = 882.5$ слева: фазовое подпространство $\Theta(P_1)$; $R = 883$ справа: фазовое подпространство $\Theta(P_1)$.

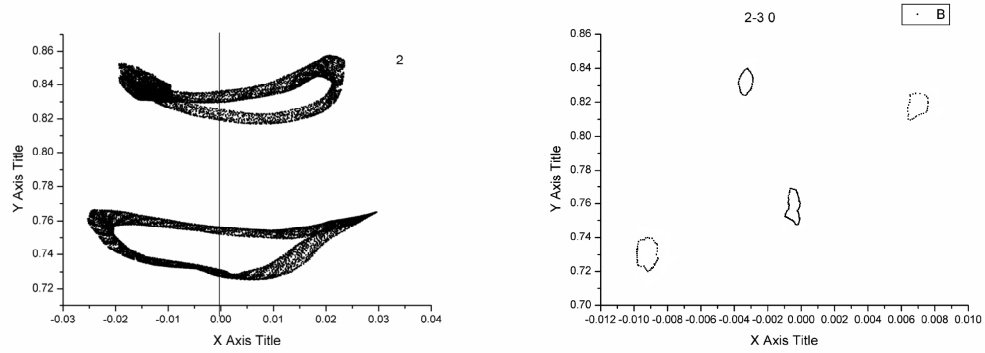


Рис.4. Точка Р3. $R = 883$, слева сечение трехмерного подпространства Θ плоскостью $\{1/2(U_{\max} + U_{\min}); V_y; V_z\}$; слева – сечение в дополнительной плоскости, $\{W = 0.0\}$.

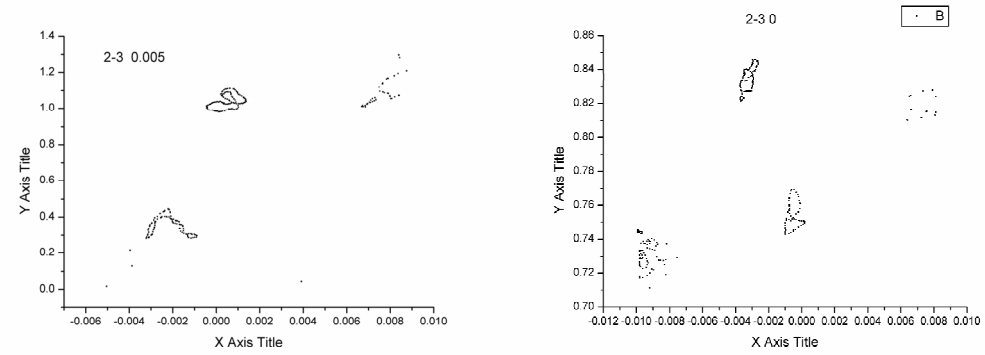


Рис.5. Точка Р3. $R = 883.8$, сечения в дополнительной плоскости, $\{W = 0.0005; 0.0\}$.

Таким образом, можно заключить, что при анализе ЛТП в задаче течения с уступа для граничных условий G_1 показано, что усложнение происходит по сценарию: $C^1 \to T^2 \to T^3 \to T^{3\otimes 2} \to ?$. Т.е. как по сценарию Ландау [4] (каскад бифуркаций увеличения размерности торов) так и по сценарию ФШМ [5]. Необходимо отметить, что для данной задачи сценарий Рюэля-Таккенса по образованию и мгновенному разрушению трехмерного тора не обнаружен.

4.2 МГД течение в каверне

Для данной задачи было выбрано три точки внутри прямоугольной области Ω с относительными координатами: p1={0.5 0.5}; p2={0.25 0.25}; p3={0.125 0.125}. В каждой точке рассматривались величины V_x, V_y (на рисунках 6-7 эти переменных обозначены как U, V). Поскольку физическая размерность задачи меньше трех, для графического представления проекции фазового пространства были использованы величины из разных контрольных точек: p2 и p3 (на рисунках 6-7 номер точки соответствует индексу после переменных U, V). Значения параметров $R_m = 1000, N = 0{,}2$ в данном случае фиксировались, а параметр R варьировался. Обнаружен следующий сценарий: рождение цикла (рис.6, слева); рождение тора (рис.6, справа); рождение цикла топологического периода 2 на месте резонансного тора с соотношением частот 1:2 (рис.7, слева), при этом не удалось точно установить, что именно происходит с тором (потеря устойчивости или разрушение); каскад удвоений получившегося цикла в соответствии с первой стадией сценария ФШМ (рис.7, справа).

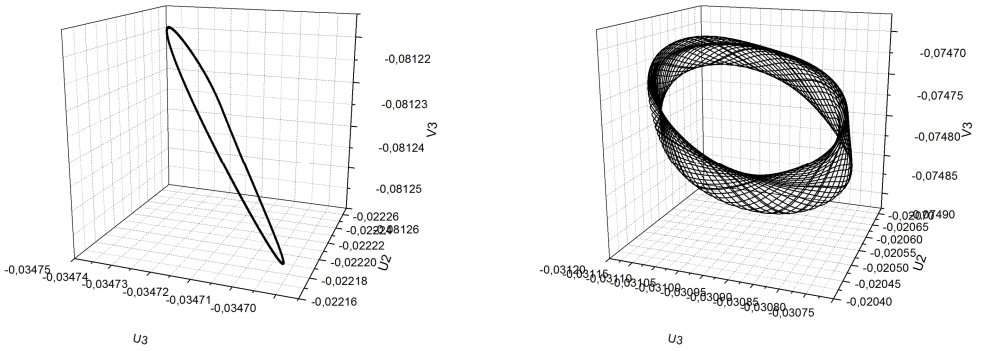


Рис. 6. $R = 2000$ слева: фазовое подпространство $\Theta(P_2, P_3)$; $R = 2500$ справа: фазовое подпространство $\Theta(P_2, P_3)$.

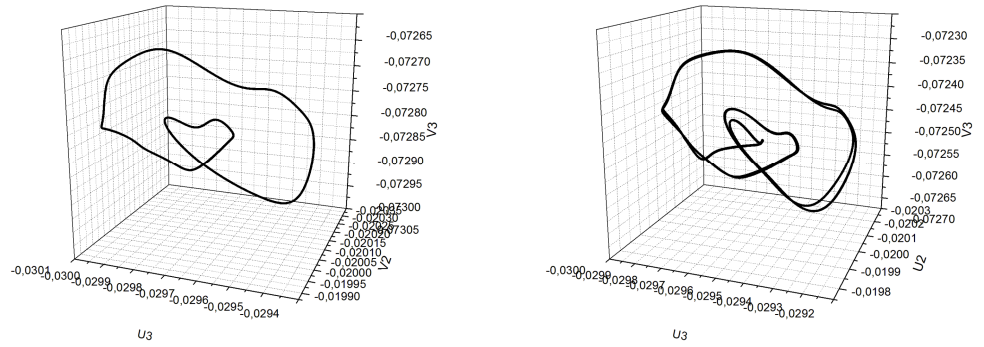


Рис. 7. $R = 2685$ слева: фазовое подпространство $\Theta(P_2, P_3)$; $R = 2720$ справа: фазовое подпространство $\Theta(P_2, P_3)$.

5. Заключение

В работе построены методы прямого численного моделирования для задач несжимаемой гидродинамики и МГД высокого порядка точности, способные разрешать нестационарные аттракторы (циклы и торы) соответствующих краевых задач. Гидродинамический метод реализован в двух вариантах: для MPI-совместимых архитектур и для архитектуры CUDA. Исследовано ускорение, которое дает использование GPU по сравнению с CPU для данного типа методов. Методы для МГД реализованы только для MPI-совместимых архитектур. На их примере произведено сравнение производительности небольших кластеров и вычислительного комплекса

IBM BlueGene/P. Рассмотрены две краевые задачи: гидродинамическое течение с уступа (трехмерная постановка) и МГД-течение в каверне (двухмерная постановка). Для этих задач были обнаружены начальные этапы сценария перехода к турбулентности. Для обеих задач начальные стадии представляют собой сочетание сценария Ландау-Хопфа и сценария ФШМ перехода к хаосу. Сценарий Рюэля-Таккенса подтвержден не был.

Литература

1. Куликовский А.Г., Погорелов Н.В., Семёнов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. М.: ФИЗМАТЛИТ. 2001.
2. Ландау Л.Д., Лифшиц Е.М. Теоретическая физика. Том VI. Гидродинамика. М.: Наука. 1982.
3. Ландау Л.Д., Лифшиц Е.М. Теоретическая физика. Том VIII. Электродинамика сплошных сред. М.: Наука. 1982.
4. Ландау Л.Д. К проблеме турбулентности // Доклады Академии Наук СССР. 1944. Т.44. С.339-342.
5. Магницкий Н.А., Сидоров С.В. Новые методы хаотической динамики. М.: Едиториал УРСС. 2004.
6. A. Kurganov, E. Tadmor Solution of two-dimensional Riemann problems for gas dynamics without Riemann problem solvers // Num. Meth. Part. Diff. Eq. 2002. Vol. 18.
7. Mistrangelo C. Three-dimensional MHD flow in sudden expansions // Wissenschaftliche Berichte FZKA. 2006. Vol. 7201.
8. Cheng-Chin Wu A high order WENO finite difference scheme for incompressible fluids and magnetohydrodynamics // Geophysical & Astrophysical Fluid Dynamics. 2007. Vol. 101. No. 1 pp. 37–61.
9. N.M. Evstigneev, N.A. Magnitskii, S.V. Sidorov On the nature of turbulence in a problem on the motion of a fluid behind a ledge // Differential Equations. 2009. Vol. 45. No. 1.
10. U. Muller, L. Buhler Magnetofluiddynamics in Channels and Containers // Springer – 2001.
11. Xu-Dong Liu, Stanley Osher, Tony Chan Weighted Essentially Non-Oscillatory Schemes // Journal of Computational Physics. Volume 115 Issue 1, Nov. 1994.

Исследование устойчивости параллельного алгоритма решения задачи сильной отделимости на базе фейеровских отображений*

А.В. Ершова, И.М. Соколинская

Южно-Уральский национальный исследовательский университет

В работе рассматривается задача сильной отделимости, заключающаяся в разделении двух выпуклых непересекающихся многогранников слоем наибольшей толщины. Описывается параллельный алгоритм решения задачи сильной отделимости на базе фейеровских отображений, допускающий эффективную реализацию на многопроцессорных системах с массовым параллелизмом. Вводится понятие устойчиво фейеровского отображения. Доказывается теорема, определяющая условия, при которых фейеровское отображение будет устойчиво фейеровским.

1. Введение

Задача разделения двух выпуклых непересекающихся многогранников имеет большое значение теоретического и прикладного характера в распознавании образов, включающем задачи дискриминации, классификации и др. Задача сильной отделимости может быть решена в ходе итерационного процесса, использующего операцию проектирования. Однако на практике применение этого метода существенно ограничивается тем, что далеко не всегда удается построить конструктивную формулу для вычисления проекции точки на выпуклое множество. Поэтому целесообразно заменить операцию проектирования последовательностью фейеровских отображений [1], строящих некоторую *псевдопроекцию*. Фейеровские методы относятся к итерационным методам проекционного типа, применяемым для решения систем линейных неравенств и задач линейного программирования. Их конструкция, то есть конструкция соответствующих фейеровских операторов, базируется на той или иной суперпозиции элементарных проектирований, а именно – проектирований на полупространства.

Среди современных методов классификации и распознавания образов одно из ведущих мест занимает метод опорных векторов, известный также как метод обобщенного портрета или машины опорных векторов (Support Vector Machines, SVM) [2]. Системы, разработанные на его основе, успешно решают задачи в таких областях, как биоинформатика, машинное зрение, категоризация текстов, распознавание рукописных символов и др. Однако метод опорных векторов оказывается неэффективным в случае изменяющихся исходных данных. Алгоритмы разделения многогранников на основе фейеровских отображений обладают тем преимуществом по сравнению с этим и другими известными методами, что они применимы к нестационарным задачам [3], т.е. к задачам, в которых исходные данные могут меняться в процессе решения задачи.

Для итерационных алгоритмов, решающих нестационарные задачи особенно важным является вопрос их устойчивости, так как динамическое изменение входных данных и погрешности вычислений могут привести к тому, что алгоритм выдаст неверный результат, либо итерационный процесс вообще перестанет сходиться.

В данной работе исследуется вопрос устойчивости параллельного алгоритма решения задачи сильной отделимости на базе фейеровских отображений, предложенного в работе [4]. Далее описывается математическая модель сильной отделимости с использованием фейеровских отображений. Приводится итерационный алгоритм **F** решения задачи сильной отделимости, основанный на построении псевдопроекций. Рассматривается масштабируемый алгоритм **S** построения псевдопроекции, позволяющий реализовать параллельную версию

* Работа выполнена при поддержке РФФИ, грант № 12-01-00452-а.

алгоритма **F**. Вводится понятие устойчиво фейеровского отображения. Доказывается теорема, определяющая условия, при которых фейеровское отображение будет устойчиво фейеровским.

2. Математическая модель

Пусть даны два выпуклых непересекающихся многогранника $M \subset \mathbf{R}^n$ и $N \subset \mathbf{R}^n$, заданные системами линейных неравенств:

$$\begin{gathered} M = \{x \mid Ax \le b\} \ne \varnothing; \\ N = \{x \mid Bx \le d\} \ne \varnothing; \\ M \cap N = \varnothing. \end{gathered} \tag{1}$$

Задача сильной отделимости – это задача нахождения слоя наибольшей толщины (π-слоя), разделяющего M и N. Сильная отделимость, по существу, эквивалентна задаче отыскания расстояния между M и N в смысле метрики

$$\rho(M,N) = \min\{\|x-y\| \mid x \in M,\ y \in N\}. \tag{2}$$

Если $\overline{x} \in M$ и $\overline{y} \in N$ являются arg-точками задачи (2), то есть $\rho(M,N) = \|\overline{x} - \overline{y}\|$, то слоем наибольшей толщины, разделяющим множества *M* и *N*, является $P := \{x \mid x \in P_1 \cap P_2\}$, где P_1 и P_2 – полупространства, задаваемые линейными неравенствами

$$(x - \overline{x}, \overline{x} - \overline{y}) \le 0 \ \text{ и } \ (y - \overline{y}, \overline{x} - \overline{y}) \ge 0.$$

Задача сильной отделимости может быть решена с помощью известного метода *последовательного проектирования* [1]. Если множества *M* и *N* достаточно просты в смысле простоты реализации операции проектирования точек на них, то алгоритм на базе последовательного проектирования может быть использован на практике. Но если *M* и *N* – произвольные многогранники, то такой алгоритм не может быть признан эффективным, так как не известен универсальный конструктивный метод построения проекции точки на многогранник. Ситуацию можно исправить, если вместо операции проектирования использовать фейеровские отображения.

Дадим определение фейеровского отображения. Пусть $\varphi \in \{\mathbf{R}^n \to \mathbf{R}^n\}$. Отображение φ называется *M-фейеровским*, если выполняются следующие два условия:

$$\varphi(y) = y,\ \forall y \in M;$$
$$\|\varphi(x) - y\| < \|x - y\|,\ \forall y \in M,\ \forall x \notin M.$$

Сконструируем *M*-фейеровское отображение, следуя работе [5]. Представим систему линейных неравенств, задающих многогранник *M*, в следующем виде:

$$Ax \le b : l_j(x) = (a_j, x) - b_j \le 0,\ j = 1,\ldots,m, \tag{3}$$

где $a_j \ne 0$ для любого j. Определим $l_j^+(x)$ следующим образом:

$$l_j^+(x) = \max\{l_j(x), 0\},\ j = 1,\ldots,m. \tag{4}$$

Тогда отображение вида

$$\varphi(x) = x - \sum_{j=1}^{m} \alpha_j \lambda_j \frac{l_j^+(x)}{\|a_j\|^2} a_j \tag{5}$$

будет *M*-фейеровским для любой системы положительных коэффициентов $\{\alpha_j > 0\}$, $j = 1,\ldots,m$,

таких, что $\sum_{j=1}^{m}\alpha_j = 1$ и коэффициентов релаксации $0 < \lambda_j < 2$. Аналогичным образом сконструируем *N*-фейеровское отображение ψ. Используя отображения φ и ψ, мы можем построить следующий алгоритм **F**, решающий задачу сильной отделимости с использованием фейеровских отображений.

Алгоритм F. Пусть задано произвольное начальное приближение $z_0 \in \mathbf{R}^n$. Зафиксируем положительное вещественное число ε. Алгоритм состоит из следующих шагов:

Шаг 0. $k := 0$.

Шаг 1. $x_{k+1} := \lim_{u\to\infty}\varphi^u(z_k)$.

Шаг 2. $y_{k+1} := \lim_{u\to\infty}\psi^u(z_k)$.

Шаг 3. $z_{k+1} := \frac{x_{k+1} + y_{k+1}}{2}$.

Шаг 4. $k := k + 1$.

Шаг 5. Если $\min\left\{\left\|x_{k+1} - x_k\right\|, \left\|y_{k+1} - y_k\right\|\right\} \geq \varepsilon$, перейти на шаг 1.

Шаг 6. Стоп.

Алгоритм **F** был исследован в работе [6]. Проведенные вычислительные эксперименты на модельных и случайных задачах подтвердили его эффективность. Однако для больших размерностей работа алгоритма **F** требовала значительного времени. Например, при размерности задачи $n = 512$ время счета на одном процессорном ядре составило 16 часов, а при размерности задачи $n = 1024$ – 608 часов (более 25 дней). В связи с этим возникла необходимость разработки параллельной версии этого алгоритма для многопроцессорных систем с массовым параллелизмом. Очевидно, что в алгоритме **F** ресурсоемкими являются шаги 1 и 2. На каждом из этих шагов реализуется *последовательный* фейеровский процесс, в результате которого мы получаем *псевдопроекцию* точки на многогранник. Исследования показали, что подобные фейеровские процессы не допускают эффективного распараллеливания на большом количестве процессорных узлов (предел масштабируемости в экспериментах не превышал 8-16 узлов). В следующем разделе описывается масштабируемый алгоритм построения псевдопроекции точки на многогранник с использованием фейеровских отображений, предложенный в работе [4].

3. Масштабируемый алгоритм S построения псевдопроекции

В основе масштабируемого алгоритма построения псевдопроекции на многогранник лежит метод разбиения пространства на подпространства. Для каждого подпространства организуется независимый фейеровский процесс. Через каждые *s* шагов результаты, полученные на подпространствах, соединяются в один вектор, который и является очередным приближением. Если расстояние между соседними приближениями меньше заданного положительного числа ε, то полученный вектор принимается в качестве псевдопроекции. В противном случае вычисления продолжаются.

Дадим формальное описание алгоритма построения псевдопроекции на выпуклый многогранник, допускающего эффективное распараллеливание на большом количестве процессорных узлов. Введем следующие обозначения. Для произвольного линейного подпространства $\mathrm{P} \subset \mathrm{R}^n$ через $\pi_{\mathrm{P}}(x)$ будем обозначать ортогональную проекцию $x \in \mathbb{R}^n$ на линейное подпространство P. Везде далее линейное подпространство будем называть просто подпространством. Через $\rho(\mathrm{P}, x) := \min_{p\in\mathrm{P}}\|p - x\|$ будем обозначать расстояние от точки x до подпространства P. Пусть линейное многообразие L получается из P сдвигом на некоторый вектор z: $\mathrm{L} = \mathrm{P} + z$. Через $\pi_{\mathrm{L}}(x)$ обозначим ортогональную проекцию $x \in \mathbb{R}^n$ на линейное многообразие L:

$$\pi_{\mathrm{L}}(x) = \pi_{\mathrm{P}}(x) + z. \qquad (6)$$

Алгоритм S. Пусть задано однозначное непрерывное M-фейеровское отображение $\varphi \in \{\mathrm{R}^n \to \mathrm{R}^n\}$, M – выпукло и замкнуто. Зададим разбиение пространства R^n в прямую сумму ортогональных подпространств: $\mathbf{R}^n = \mathrm{P}_1 \oplus \ldots \oplus \mathrm{P}_r$, $\mathrm{P}_i \perp \mathrm{P}_j$ при $i \neq j$. Для каждого подпространства P_i $(i = 1,\ldots,r)$ построим линейное многообразие L_i следующим образом. Пусть $\overline{x}^i \in \operatorname{Arg}\min_{x \in M} \rho(\mathrm{P}_i, x)$. Положим $\overline{z}^i = \pi_{\mathrm{P}_i^{\perp}}(\overline{x}^i) \in \mathrm{P}_i^{\perp}$. Здесь $\mathrm{P}_i^{\perp}$ обозначает ортогональное дополнение к подпространству P_i. Построим линейное многообразие L_i путем сдвига подпространства P_i на вектор $\overline{z}^i$:

$$\mathrm{L}_i = \mathrm{P}_i + \overline{z}^i. \tag{7}$$

Для каждого $i \in \{1,\ldots,r\}$ определим отображение $\varphi_i \in \{\mathrm{R}^n \to \mathrm{L}_i\}$:

$$\varphi_i(x) = \pi_{\mathrm{L}_i}\left(\varphi\left(\pi_{\mathrm{L}_i}(x)\right)\right). \tag{8}$$

Зафиксируем некоторое натуральное число s и положительное вещественное число ε. Положим $x_0 = \mathbf{0} \in \mathbb{R}^n$. Алгоритм состоит из следующих шагов:

Шаг 0. $k := 0$.

Шаг 1. $x_{k+1} := \sum_{i=1}^{r}\left(\varphi_i^s\left(\pi_{\mathrm{L}_i}(x_k)\right) - \overline{z}^i\right)$.

Шаг 2. $k := k + 1$.

Шаг 3. Если $\|x_{k+1} - x_k\| \geq \varepsilon$ & $d_M(x_{k+1}) \geq \varepsilon$, перейти на шаг 1.

Шаг 4. Стоп.

Для выхода из итерационного процесса в алгоритме **S** на шаге 3 используется критерий завершения, включающий в себя функцию невязки d_M, определяющую степень близости точки x к многограннику M. В качестве такой функции в нашей реализации используется следующая функция

$$d_M(x) = \sum_{j=1}^{m} \max\left\{\left\langle a_j, x\right\rangle - b_j, 0\right\}.$$

На основе описанного подхода на языке программирования C++ была разработана параллельная программа, решающая задачу сильной отделимости многогранников для произвольных входных данных. Для организации обменов данными между процессами была использована система параллельного программирования MPI. Исходные тексты программ доступны в Интернет по адресу http://life.susu.ru/discr/. Проведенные вычислительные эксперименты на высокопроизводительном кластере подтвердили эффективность предложенного подхода [4]. В следующем разделе доказывается теорема, из которой следует устойчивость описанного алгоритма по отношению к динамически меняющимся исходным данным задачи.

4. Теорема об устойчиво фейеровском отображении

Пусть задана система линейных неравенств в пространстве R^n:

$$Ax \leq b. \tag{9}$$

Пусть $y = [A, b]$ – *информационный вектор*, задающий все параметры системы (9), $y \in \mathrm{R}^{nm+m}$. Обозначим через M_y многогранник решений системы (9), определяемой информационным вектором y. Имеем $M_y \subset \mathrm{R}^n$.

Лемма. Пусть $\overline{y} \in \mathrm{R}^{nm+m}$ – информационный вектор, задающий устойчиво совместную систему [7] неравенств

$$\overline{A}x \leq \overline{b}. \tag{10}$$

Тогда существует некоторая окрестность V точки $\overline{y}$, такая, что любая точка $\tilde{y} \in V$ также определяет устойчиво совместную систему неравенств

$$\tilde{A}x \le \tilde{b}, \qquad (11)$$

где $\tilde{y} = \left[\tilde{A}, \tilde{b}\right]$.

Доказательство. Устойчивая совместность системы (10) означает, что существует $\bar{x} \in \mathrm{R}^n$ такой, что $\bar{A}\bar{x} < \bar{b}$, или, что то же самое, $\bar{A}\bar{x} - \bar{b} < 0$. Поскольку вектор-функция $A\bar{x} - b$ непрерывна по $y = [A, b]$, то для всех точек $\tilde{y} = \left[\tilde{A}, \tilde{b}\right]$ некоторой окрестности V точки $\bar{y}$ также будет выполняться неравенство $\tilde{A}\bar{x} - \tilde{b} < 0$, что равносильно $\tilde{A}\bar{x} < \tilde{b}$. А это, в свою очередь, означает, что для всех точек этой окрестности система (11) будет устойчиво совместной, что и требовалось доказать. Лемма доказана.

Определение. Пусть задано отображение $\varphi : \mathrm{R}^{nm+m} \times \mathrm{R}^n \to \mathrm{R}^n$ двух аргументов $y \in \mathrm{R}^{nm+m}$ и $x \in \mathrm{R}^n$. Обозначим через φ_y отображение из R^n в R^n, которое получается из φ, путем фиксации аргумента y. Отображение φ является *устойчиво фейеровским* относительно точки $\bar{y} \in \mathrm{R}^{nm+m}$, если $\varphi_{\bar{y}} \in \mathbf{F}_{M_{\bar{y}}}$ и существует окрестность $V \subset \mathrm{R}^{nm+m}$ точки $\bar{y}$ такая, что для любого $\tilde{y} \in V$ имеем $\varphi_{\tilde{y}} \in \mathbf{F}_{M_{\tilde{y}}}$.

Теорема. Пусть отображение $\varphi : \mathrm{R}^{nm+m} \times \mathrm{R}^n \to \mathrm{R}^n$ двух аргументов $y \in \mathrm{R}^{nm+m}$ и $x \in \mathrm{R}^n$ является непрерывным по x и y[1]. Пусть система (10), определяемая информационным вектором $\bar{y}$, является устойчиво совместной и $\varphi_{\bar{y}} \in \mathbf{F}_{M_{\bar{y}}}$. Тогда отображение φ является *устойчиво фейеровским* относительно точки $\bar{y} \in \mathrm{R}^{nm+m}$.

Доказательство. Доказательство проведем от противного. Предположим, что для любой окрестности V точки $\bar{y}$ существует

$$\tilde{y} \in V \qquad (12)$$

такой, что $\varphi_{\tilde{y}} \notin \mathbf{F}_{M_{\tilde{y}}}$. В соответствии с определением фейеровского отображения это означает, что не выполняется одно из следующих условий:

1) $\varphi_{\tilde{y}}(z) = z, \qquad \forall z \in M_{\tilde{y}}$;

2) $\left\|\varphi_{\tilde{y}}(x) - z\right\| < \left\|x - z\right\|, \qquad \forall z \in M_{\tilde{y}}, \qquad \forall x \notin M_{\tilde{y}}$.

В силу свойства 39.4 [5] при доказательстве мы можем ограничиться $\operatorname{int} M_{\tilde{y}}$, представляющим подмножество внутренних точек множества $M_{\tilde{y}}$. В силу леммы 1 мы можем выбрать окрестность V таким образом, чтобы для всех $\tilde{y} \in V$ получалась устойчиво совместная система, то есть $\operatorname{int} M_{\tilde{y}} \neq \varnothing$.

Предположим сначала, что *не выполняется условие 1)*. Это означает, что существует точка $\tilde{z} \in \operatorname{int} M_{\tilde{y}}$ такая, что

$$\varphi_{\tilde{y}}(\tilde{z}) \neq \tilde{z}. \qquad (13)$$

Так как $\tilde{z}$ является внутренней точкой многогранника $M_{\tilde{y}}$, то $\tilde{A}\tilde{z} - \tilde{b} < 0$. Поскольку вектор-функция $A\tilde{z} - b$ непрерывна относительно $y = [A, b]$, то существует окрестность $\tilde{V}$ точки $\tilde{y}$ такая, что для любой точки $\tilde{y}' \in \tilde{V}$ имеем $\tilde{A}'\tilde{z} - \tilde{b}' < 0$. Выберем окрестность V таким образом, чтобы выполнялось вложение $V \subset \tilde{V}$. Тогда получается, что $\tilde{z}$ также является внутренней точкой многогранника $M_{\bar{y}}$. Поскольку по условию теоремы $\varphi_{\bar{y}} \in \mathbf{F}_{M_{\bar{y}}}$, отсюда следует, что

$$\varphi_{\bar{y}}(\tilde{z}) = \tilde{z}. \qquad (14)$$

---

[1] Отображение $\varphi(y, x)$ *непрерывно* по x в точке $\bar{x}$, если при фиксированном y для любого $\varepsilon > 0$ существует $\delta > 0$ такое, что отображение φ определено в δ-окрестности точки $\bar{x}$ и $\left\|x - \bar{x}\right\| < \delta \Rightarrow \left\|\varphi(y, x) - \varphi(y, \bar{x})\right\| < \varepsilon$. Непрерывность по y определяется аналогичным образом.

Так как отображение φ является непрерывным по y, то из (13) следует, что существует окрестность $\tilde{V}'$ точки $\tilde{y}$ такая, что для любой точки $\tilde{y}' \in \tilde{V}'$ имеем $\varphi_{\tilde{y}'}(\tilde{z}) \neq \tilde{z}$. Выберем окрестность V таким образом, чтобы выполнялось вложение $V \subset \tilde{V}'$. Тогда получается, что $\bar{y} \in \tilde{V}'$, откуда следует $\varphi_{\bar{y}}(\tilde{z}) \neq \tilde{z}$. Получили противоречие с (14). Таким образом, мы доказали, что условие 1) выполняется.

Теперь предположим, что *не выполняется условие 2)*. Это означает, что существует точка $\tilde{z} \in \operatorname{int} M_{\tilde{y}}$ и точка $\tilde{x} \notin M_{\tilde{y}}$ такие, что

$$\left\| \varphi_{\tilde{y}}(\tilde{x}) - \tilde{z} \right\| \geq \left\| \tilde{x} - \tilde{z} \right\|. \tag{15}$$

Так как $\tilde{z}$ является внутренней точкой многогранника $M_{\tilde{y}}$, то $\tilde{A}\tilde{z} - \tilde{b} < 0$. Поскольку вектор-функция $Az - b$ непрерывна относительно $y = [A, b]$, то существует окрестность $\tilde{V}$ точки $\tilde{y}$ такая, что для любой точки $\tilde{y}' \in \tilde{V}$ имеем $\tilde{A}'\tilde{z} - \tilde{b}' < 0$. Выберем окрестность V таким образом, чтобы выполнялось вложение $V \subset \tilde{V}$. Тогда получается, что $\tilde{z}$ является внутренней точкой многогранника $M_{\bar{y}}$. Поскольку по условию теоремы $\varphi_{\bar{y}} \in \mathbf{F}_{M_{\bar{y}}}$, отсюда следует, что

$$\left\| \varphi_{\bar{y}}(\tilde{x}) - \tilde{z} \right\| < \left\| \tilde{x} - \tilde{z} \right\|. \tag{16}$$

Так как отображение φ является непрерывным по y, то из (16) следует, что существует окрестность $\bar{V}$ точки $\bar{y}$ такая, что для любой точки $\bar{y}' \in \bar{V}$ имеем $\left\| \varphi_{\bar{y}'}(\tilde{x}) - \tilde{z} \right\| < \left\| \tilde{x} - \tilde{z} \right\|$. Выберем окрестность V таким образом, чтобы выполнялось вложение $V \subset \bar{V}$. Тогда в силу (12) имеем $\tilde{y} \in \bar{V}$, откуда следует $\left\| \varphi_{\tilde{y}}(\tilde{x}) - \tilde{z} \right\| < \left\| \tilde{x} - \tilde{z} \right\|$. Получили противоречие с (15). Таким образом, мы доказали, что условие 2) выполняется. Т е о р е м а д о к а з а н а .

В заключение осталось заметить, что фейеровское отображение (5), в соответствии с доказанной теоремой, является устойчиво фейеровским, что обеспечивает устойчивость предложенного алгоритма.

Литература

1. *Еремин И.И.* Фейеровские методы сильной отделимости выпуклых полиэдральных множеств // Известия вузов. Сер. математика, 2006. № 12. С. 33-43.
2. *Boser B., Guyon I., Vapnik V.* A training algorithm for optimal margin classifiers // Proc. of the 5th Annual ACM Workshop on Computational Learning Theory. Pittsburgh: ACM Press, 1992. 144–152.
3. *Еремин И.И., Мазуров В.Д.* Нестационарные процессы математического программирования. М.: Наука, 1979.
4. *Ершова А.В., Соколинская И.М.* Параллельный алгоритм решения задачи сильной отделимости на основе фейеровских отображений // Вычислительные методы и программирование. 2011. Т. 12, № 2. С. 178-189.
5. *Еремин И.И.* Теория линейной оптимизации. Екатеринбург: Изд-во “Екатеринбург”, 1999.
6. *Ершова А.В.* Алгоритм разделения двух выпуклых непересекающихся многогранников с использованием фейеровских отображений // Системы управления и информационные технологии, 2009. №1(35). С. 53-56.
7. *Черников С.Н.* Линейные неравенства. М.: Изд-во "Наука", 1968. 488 с.

Разработка распределенных алгоритмов и высокопроизводительной программной системы для облачного хранения, потоковой обработки и сбора в реальном времени сверхбольших наборов научных данных*

М.Н. Жижин, А.Н. Поляков, А.А. Пойда, Д.П. Медведев

НИЦ «Курчатовский институт»

В статье описываются предварительные результаты, полученные в ходе первого этапа выполнения научно-исследовательской работы по разработке программных систем, объединяющих высокопроизводительные технологии и параллельные алгоритмы управления сверхбольшими наборами научных данных, адаптируемых для решения широкого круга многодисциплинарных научных задач в различных областях приложения, включая: физику высоких энергий и астрофизику, науки о Земле и глобальные изменения климата, дистанционное зондирование и обработку многомасштабных изображений, биоинформатику, нанотехнологии и т.д.

1. Введение

Целью исследований является разработка программных систем, объединяющих высокопроизводительные технологии и параллельные алгоритмы управления сверхбольшими наборами научных данных, адаптируемых для решения широкого круга многодисциплинарных научных задач в различных областях приложения, включая: физику высоких энергий и астрофизику, науки о Земле и глобальные изменения климата, дистанционное зондирование и обработку многомасштабных изображений, биоинформатику, нанотехнологии и т.д.

Необходимость автоматизации управления научными данными вызвана прежде всего лавинообразным нарастанием объемов собираемых экспериментальных и данных и увеличением их сложности, связанным с ростом числа и разрешающей способности научных сенсоров, а также с экспоненциальным ростом вычислительных возможностей и объемов результатов вычислений, требующих хранения для повторного анализа.

Еще одним важным направлением исследований является адаптация существующий сервисов доступа и управления данными, работающими преимущественно с локальными файловыми системами и отчасти с удаленными наборами данных, к работе с облачным хранилищем данных. За счет высокого параллелизма и множественности точек доступа, в системах облачных вычислений есть возможность превысить производительность, показываемую при работе с локальными данными.

Специализированные системы хранения, такие как RasDaMan [1] и SciDB [2, 3], представляют большой интерес для научных приложений, так как они используют модель данных, специально адаптированную под научные приложения, при этом к их недостаткам можно отнести относительную сложность настройки и использования, в частности сложности при масштабировании на большое число машин.

Распределенные системы хранения общего назначения, такие как Cassandra [4] и Swift [5], удобны тем, что они позволяют пользователю максимально абстрагироваться от аппаратного обеспечения и обеспечивают хорошие показатели отказоустойчивости и масштабируемости. При этом используемая модель данных слишком проста для многих научных задач.

Наиболее перспективным представляется направление исследований, сочетающее особенности специализированных систем хранения научных данных с гибкостью и надежностью современных распределенных систем хранения общего назначения NoSQL, подобных Cassandra и Swift. Возможным вариантом реализации такой гибридной системы может стать разработка промежуточного программного обеспечения, реализующего

* Государственные контракты № 07.514.11.4045 от 26.09.2011 г. и № 07.514.11.4022 от 23.09.2011

специализированную модель данных поверх хранилища более общего назначения, подобно тому как RasDaMan опирается на традиционную реляционную СУБД для хранения данных.

Для проведения научных исследований чистых данных мало, необходимо иметь технологию для их анализа и обработки. В работе с большими наборами данных, хранящимися на нескольких удаленных серверах, возникает задача о том, чтобы системы анализа и обработки работали распределенно на удаленных серверах, ближе к месту хранения данных. Но распределенная обработка решает задачу обработки и анализа сверхбольшого объема данных. Какими бы характеристиками не обладал сервер, размер данных может быть настолько велик, что не поместится в оперативную память целиком. В этом случае придется делать свопинг на жесткий диск, что приведет к потере скорости. При этом в конвейере распределенных обработчиков возникнет простой: следующий обработчик не сможет начать работу пока не закончит предыдущий.

Одним из решений, устраняющих вышеперечисленные недостатки, является использование потоковой обработка данных, для которой объектом обработки становится не набор данных или его законченный фрагмент, а непрерывный поток данных. Обработчики могут использовать лишь небольшое окно в этом потоке для вычисления производных величин.

Проведя анализ существующих программных систем для потоковой обработки, исполнители проекта остановились на двух: OGSA-DAI [7,8] и Twitter Storm [9]. Сравнивая OGSA-DAI и Twitter Storm, надо отметить следующие особенности:

1. Twitter Storm изначально создавался для обработки потоков, тогда как система OGSA-DAI позволяет как передачу целых наборов данных, так и поблочную передачу с последовательной передачей.

2. В обеих системах пользователю потребуется реализовывать свои элементы обработки на языках программирования. В системе Twitter Storm поддерживается большее число языков программирования, чем для OGSA-DAI. В системе Twitter Storm достаточно написать jar-файл с кодом, который система автоматически подключит, в то время как в системе OGSA-DAI требуется вручную изменять конфигурационные файлы и перезагружать систему.

3. Storm автоматически распараллеливает работу между узлами, в то время как при организации потока в OGSA-DAI требуется явно прописывать хосты обработчиков.

4. Обработчики в OGSA-DAI могут накапливать (кэшировать) данные, в то время как в системе Twitter Storm нет встроенных функций для организации хранилища. Это может затруднить организацию блочно-потоковых алгоритмов (т.е. алгоритмов, в которых используется не только текущее значение потока, но и некоторое число предшествующих элементов).

Перечисленные особенности, кроме пункта 4, склоняют выбор технологии потоковой обработки в пользу системы Twitter Storm, но невозможность организовать поблочную схему обработки может стать существенным препятствием к реализации важных алгоритмов.

2. Постановка задачи

Мировая практика работы со сверхбольшими наборами данных и потоками данных с сенсорных сетей в реальном времени накопила положительный опыт использования данных технологий в различных областях человеческой деятельности. Существуют частные решения как проприетарные, так и открытые для внедрения параллельных и распределенных облачных технологий хранения и обработки данных в различных предметных областях. В России имеются собственные технологии хранения и доступа к большим массивам данных, например, созданный авторами настоящего отчета Интерактивный ресурс данных по солнечно-земной физики SPIDR. Но на сегодня пока не существует достаточно универсальной междисциплинарной платформы для интеграции распределенных массивов, потоков и сервисов данных.

Первая задача состоит в создании эффективного способа распределенного хранения и параллельного чтения-записи данных с учетом слабо структурированного характера научных данных (многие коллекции данных представляют собой просто многомерные числовые массивы). При этом метаданные должны обеспечивать эффективный поиск источников (сервисов) данных, а также содержать всю необходимую информацию о содержании,

происхождении (data provenance [6]) и различных характеристиках того или иного набора данных. Также необходимо предусмотреть возможность помечать и комментировать отдельные элементы данных или более крупные фрагменты наборов данных, например, выделение отдельных событий во временных рядах.

Вторая задача состоит в создании гибкого и достаточно простого в использовании языка запросов, учитывающего гетерогенный характер данных. При этом необходимо исходить из ограниченного набора моделей и форматов данных, общепринятых в конкретных предметных областях.

Третья задача состоит в создании индексов для быстрого поиска и обработки распределенных данных для создания сервисов и языков управления рабочим потоком распределенных вычислений (locality), которые оптимизированы по использованию памяти и ресурсов локального диска и отказоустойчивы при работе с большими объемами данных и позволяют кэшировать промежуточные результаты.

3. Реализация

Решение перечисленных задач может лежать в следующем:

1. Для эффективного извлечения новых знаний из сверхбольших объемов данных, ученые все чаще обращаются к распределению параллельных вычислений в облаке (data cloud). Вычислительное облако представляет собой абстракцию для удаленных, неограниченно масштабируемых вычислений и объемов хранения. На практике оно базируется в больших ЦОД, содержащих тысячи серверов и дисковых носителей. Для того чтобы можно было надежно хранить в прямом доступе сверхбольшие наборы научных данных, они должны быть распределены и тиражированы на тысячах серверов.

2. Необходимо разработать методы и алгоритмы, позволяющие создавать высокопроизводительные распределенные облачные хранилища сверхбольших массивов научных данных на основе общей модели данных и метаданных с возможностью отслеживания происхождения и цикла изменения данных и реализующие общий язык запросов для распределенного выполнения и управления рабочим потоком параллельных вычислительных задач по их обработке, анализу и визуализации.

3. Разработанные методы и алгоритмы должны позволить интеграцию и совместный анализ в следующих предметных областях:

- в экспериментах, обсерваторских наблюдениях и вычислительных моделях физики высоких энергий и астрофизики;
- в вычислительных моделях и сенсорных сетях в области метеорологии, экологии, глобального изменения климата;
- для сбора в реальном времени, распределенного хранения и интерактивного многомасштабного анализа данных дистанционного зондирования Земли из космоса;
- в вычислительных моделях и сетях сенсоров и обсерваторий для фундаментальной и прикладной геофизики.

4. Для реализации сервисов распределенной потоковой обработки данных необходимо:

- обеспечить конечному пользователю возможность создания, отслеживания, управления рабочим потоком на множестве реализованных обработчиков в соответствии с требованиями базовой системы (OGSA-DAI или Twitter Storm);
- обеспечить взаимодействие с сервисами распределенной потоковой обработки данных по технологии REST;
- реализовать библиотеку потоковых алгоритмов для научного и ниженерного анализа данных; в первую очередь будут реализованы алгоритмы: детектирования сейсмических P-волн [10], быстрого преобразования Фурье, алгоритмы модального анализа колебаний по методам «peak-picking» и «frequency domain decomposition» [11].

5. Для проверки эффективности разработанных методов и алгоритмов необходимо создать экспериментальный образец программного комплекса, осуществляющий решение основных

прикладных задач, возникающих при создании центров данных и облачных сервисов для работы со сверхбольшими наборами научных данных, источниками которых являются сети сенсоров и вычислительные модели в предметных областях, указанных в разделе.

6. Должны быть разработаны эталонные наборы данных и тестовые примеры и методики, обеспечивающие измерение и сравнение эффективности различных способов хранения и типов обработки и анализа сверхбольших массивов данных, источником которых являются вычислительные модели и сенсорные сети, включая оценки надежности, скорость доступа и масштабируемости хранилища, оценки функционала и производительности алгоритмов и программ обработки и анализа данных; критерии эффективности созданных систем хранения/передачи, управления и визуализации масштабными потоками и многомерными массивами научных данных и т.п. Список создаваемых эталонных наборов данных должен включать:

1) в области астрофизики и физики высоких энергий временные ряды наблюдений Мировой системы центров данных и обсерваторий космических лучей, геомагнитных обсерваторий INTERMAGNET и солнечных радиотелескопов Radio Solar Telescope Network (RSTN);

2) в области метеорологии и изменений климата архивы наблюдений Всемирной метеорологической организации и реанализ климата NCEP/NCAR Reanalysis, а также загрузку потоков данных в реальном времени для глобального оперативного прогноза NWS HOAA США;

3) в области дистанционного зондирования архив и обработка в реальном времени потока данных с американских метеорологических спутников DMSP (Defence Meteorological Satellite Program);

4) в области фундаментальной и прикладной геофизики архивы наблюдений и обработка в реальном времени потоков данных с сети сейсмических станций и GPS обсерваторий ДВО РАН.

Разработчики видят следующие направления исследований в области процесса обработки данных:

- Разработка RESTful сервиса создания и управления рабочим потоком, поддерживающая возможность распределенной параллельной обработки, а также REST-языка запросов к нему. Система управления рабочим потоком должна позволять использовать как существующие сервисы обработки (в том числе локальные программы и удаленные REST сервисы), так и позволять создавать собственные.
- Разработка отдельных RESTful сервисов обработки данных, которые можно использовать в рабочем потоке.

В качестве реализации первого направления может быть выбрана одна из существующих систем управления создания и управления рабочим потоком.

Из рассмотренных ранее систем управления рабочим потоком в системах LONI, TAVERNA и KEPLER плохо организована функциональность организации распределенных вычислений на нескольких машинах. Однако из этих трех систем TAVERNA в наибольшей степени подходит для использования в распределенной среде, так как имеет серверное ядро и веб-API с поддержкой REST-интерфейса.

Система OGSA-DAI имеет наиболее мощный функционал для организации вычислений в распределенной среде, но не имеет REST-интерфейса. Также OGSA-DAI не имеет графического интерфейса, что может существенно осложнить понимание синтаксиса запросов и в конечном итоге привести к снижению его использования со стороны конечного пользователя.

Поэтому разработчики в данном направлении иследования видят два варианта: использовать TAVERNA и дооснащать ее функциональность либо использовать OGSA-DAI и писать для него REST-оболочку.

Набор сервисов во втором направлении исследования должен быть определен в ходе дальнейшей работы над проектом, однако среди обязательных сервисов (разработка каждого из них – отдельное направление исследования) должны быть:

- сервис поиска данных на основе нечеткой логики;

- сервис подготовки пирамиды сверхбольших изображений по технологии DeepZoom [12] и позволяющий визуализацию подготовленных изображений;
- сервис обработки данных средствами пакета прикладных математических программ Scilab;
- сервис анализа и визуализации геологических данных на основе системы CoreWall [13, 14];
- сервисы конвертации форматов данных, поддерживающие стандартные промежуточные форматы и сетевые протоколы представления данных и метаданных: FITS, GRIB, NetCDF/HDF, OPeNDAP, CDF.

Методы, по которым будут проводиться исследования по представленным выше направлениям, включают следующую последовательность действий:

1. Разработка архитектуры основополагающих механизмов сервиса создания и управления рабочим потоком.
2. Разработка архитектуры и спецификации базовых сервисов обработки данных (см. выше) в соответствии с требованиями, накладываемыми архитектурой, разработанной в пункте 1. Параллельно - уточнение архитектуры сервиса управления рабочим потоком.
3. Разработка спецификации расширенного набора сервисов обработки данных (исходя из конкретных предметных задач) в соответствии с требованиями, накладываемыми архитектурой, разработанной в пункте 1 и доработанной в пункте 2. Параллельно – доработка архитектуры сервиса управления рабочим потоком.
4. Создание прототипа системы управления рабочим потоком. При необходимости – доработка спецификаций и архитектур, разработанных в пунктах 1-3.
5. Реализация прототипов базовых сервисов обработки данных, их соединение через систему управления рабочим потоком в экспериментальный образец.
6. Проведение профилирования и анализ полученных результатов. Оптимизация и доработка экспериментального образца.
7. Реализация прототипов расширенного набора сервисов из пункта 3, их внедрение в экспериментальный образец.
8. Анализ и доработка системы, выводы.

4. Заключение

Создаваемые методы и программы могут быть использованы как в фундаментальных приложениях, таких как изучение влияния солнечной активности на климат, так и в чисто практических целях, например для экологического мониторинга, в частности прогноза распространения вулканического пепла или радиоактивных загрязнений. Методы, применяемые в настоящее время для этой цели, в значительной степени определяются конкретными задачами и объектами, для которых они разрабатывались, и неэффективны для сравнения и анализа сврехбольших наборов данных, возникающих на стыке естественнонаучных дисциплин. Областью приложения алгоритмов автоматизированного управления научными данными явятся задачи, возникающие в области сбора и оперативного анализа потоков данных с большого числа сенсоров или детального численного моделирования динамики астро-, гео- и биофизических явлений . Кроме того, мы предполагаем, что часть создаваемых методов найдет применение в анализе телеметрии и изображений для медицинскими приложений.

Накопленные в результате работ по проекту сверхбольшие наборы данных и созданные информационные технологии должны упрощать междисциплинарную интеграцию и позволять масштабируемое и надежное хранение научных данных с общими моделью данных и языком запросов, ускорять запись и чтение данных, позволять распределенный поиск метаданных и отслеживание происхождения данных (data provenance), облегчать интерактивный анализ и визуальный поиск закономерностей, а также повторное использование данных и гибкое управление доступом.

Коммерциализация полученных в рамках проекта научно-технических результатов возможна в виде заключения договоров с лабораториями, занимающимися прикладными и

фундаментальными исследованиями, связанными со сбором и анализом сверхбольших наборов данных, по оказанию услуг в области доступа к уже имеющимся «эталонным» сверхбольшим базам данных, хостинга наборов данных для заказчика и организации сервисов по коммерческому доступу, аренде вычислительных ресурсов для анализа и аппаратно-программных средств визуализации этих данных.

Литература

1. Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, Norbert Widmann: *The Multidimensional Database System RasDaMan.* Proceedings ACM SIGMOD'98, Seattle, Washington, USA. June 1998. http://www.faculty.jacobs-university.de/pbaumann/iu-bremen.de_pbaumann//Papers/sigmod98.zip
2. Олег Бартунов, Павел Велихов и др., SciDB – новая СУБД для больших объемов научных данных, 2011, http://supercomputers.ru/index.php?option=com_k2&view=item&id=167:scidb
3. Сергей Кузнецов, Год эпохи перемен в технологии баз данных, 2009, http://citforum.ru/database/articles/epoch/
4. Eben Hewitt, Cassandra: The Definitive Guide, O'Reilly Media, 2010
5. Ken Pepple, Deploying OpenStack, O'Reilly Media, 2011
6. Efficient provenance storage over nested data collections. Anand, Manish Kumar and Bowers, Shawn and McPhillips, Timothy and Ludascher, Bertram (2009) . Pages: 958--969. url: http://portal.acm.org/citation.cfm?id=1516470
7. Jackson, M., Antonioletti, M., Dobrzelecki, B., Chue Hong, N. Distributed data management with OGSA-DAI. Grid and Cloud Database Management (eds. Fiore, S. and Aloisio, G.), Springer-Verlag, July 2011, pp63-86. ISBN 978-3-642-20044-1. DOI 10.1007/978-3-642-20045-8.
8. Dobrzelecki, B., Krause, A., Hume, A., Grant, A., Antonioletti, M., Alemu, T., Atkinson, M., Jackson, M. and Theocharopoulos, E. Integrating distributed data sources with OGSA–DAI DQP and Views. Phil. Trans. R. Soc. A 13 September 2010 vol. 368 no. 1926 4133-4145 DOI: 10.1098/rsta.2010.0166.
9. Распределенная вычислеительная система Twitter Storm. Сайт проекта, url: https://github.com/nathanmarz/storm/wiki
10. M.N. Zhizhin, D. Rouland, J. Bonnin, A.D. Gvishiani, A. Burtsev. Rapid Estimation of Earthquake Source Parameters from Pattern Analysis of Waveforms Recorded at a Single Three-Component Broadband Station, Port Vila, Vanuatu. Seismological Society of America, 2006. Vol. 96, no. 6.
11. Zimmerman, A. T., Shiraishi, M., Swartz, R. A. and Lynch, J. P., "Automated modal parameter estimation by parallel processing within wireless monitoring systems," *Journal of Infrastructure Systems*, 14(1), 102-113, 2008.
12. Технология построение пирамид изображений DeepZoom. Сайт проекта, url: http://msdn.microsoft.com/ru-ru/library/cc645050%28v=vs.95%29.aspx
13. Chen, Y., Hur, H., Lee, S., Leigh, J., Johnson, A., Renambot, L. Case Study - Designing An Advanced Visualization System for Geological Core Drilling Expeditions. Proceedings of the ACM Conference on Human Factors in Computing Systems 2010 (CHI 2010), Atlanta, GA, 04/10/2010 - 04/15/2010
14. Conze,R., Krysiak, F., Reed, J., Chen, Y., Wallrabe-Adams, H., Graham, C. and the New Jersey Shallow Shelf Science Team, Wennrich, V. and the Lake El'gygytgyn Science Team. New Integrated Data Analyses Software Components. Scientific Drilling Journal, ISSN: 1816-8957, 04/01/2010 - 04/01/2010

Математическое моделирование радиационной эмиссии электронов на гибридных суперкомпьютерах

М.Е. Жуковский, Р.В. Усков

ИПМ им. М.В. Келдыша РАН

Рассматриваются вопросы построения алгоритмов и параллельных программ для статистического моделирования процессов переноса гамма-излучения и электронов на суперкомпьютерах с гетерогенной (гибридной) архитектурой. Обсуждаются принципы распараллеливания вычислений с использованием графических процессоров в качестве арифметических сопроцессоров. Предложена эффективная весовая модификация метода Монте-Карло, ориентированная на проведение вычислений с применением технологии NVIDIA© CUDA. Приводятся результаты моделирования эмиссии электронов с внешних и внутренних поверхностей трубы, облучаемой потоком 100-кэвных фотонов на суперкомпьютере К-100.

1. Введение

При прохождении проникающего излучения через объект частицы (фотоны) с энергией от нескольких кэВ до нескольких МэВ (рентгеновские аппараты и естественные радиоизотопы, а также некоторые виды космического излучения) в результате взаимодействия с веществом объекта (комптоновское рассеяние, фотопоглощение, рождение электрон-позитронных пар) порождают потоки быстрых электронов, которые способны покинуть границы объекта. В результате снаружи и во внутренних полостях объекта происходит процесс радиационной электронной эмиссии. Электрический ток эмитируемых электронов является источником внутреннего и внешнего радиационного электромагнитного поля, которое может заметно влиять, например, на работу электронной аппаратуры, находящейся внутри объекта, что может привести к сбоям этой аппаратуры. Кроме того, электроны эмиссии могут образовывать ионизованный слой вокруг объекта, обладающий высокой проводимостью, что меняет свойства внешней оболочки объекта.

Учет процессов электронной эмиссии важен при интерпретации различного рода экспериментов с ионизирующим излучением, например, в задачах радиационного неразрушающего контроля материалов и конструкций. В особенности это важно при анализе процессов измерения и регистрации проникающего излучения, прошедшего сквозь опытный объект. Моделирование перспективных рентгеновских источников тормозного излучения, а также приборов детектирования такого излучения невозможно без корректного учета процессов радиационной электронной эмиссии с внешних и внутренних граничных поверхностей этих приборов и аппаратов.

Исследование указанных процессов требует решения сложных граничных задач переноса излучения в трехмерных постановках. Эффективным способом решения таких задач является метод Монте-Карло. Однако применение этого метода для моделирования переноса излучения в объектах со сложной геометрической структурой требует значительных вычислительных ресурсов и предполагает использование многопроцессорных, в том числе гибридных, суперкомпьютеров.

В настоящей работе рассмотрены алгоритмы моделирования радиационной электронной эмиссии на гетерогенной вычислительной технике с применением технологии NVIDIA© CUDA.

2. Принципы моделирования радиационной электронной эмиссии

Алгоритмы моделирования процессов эмиссии электронов с граничных поверхностей объектов, облучаемых проникающим излучением, строятся в настоящей работе главным образом на основе работы [1]. Однако есть и ряд существенных отличий. В частности,

моделирование переноса электронов проводится с использованием модели индивидуальных соударений [2], которая, обладая простой внутренней логикой, гораздо эффективнее при реализации на гибридной вычислительной технике, чем широко используемые модели, основанные на идее вложенных траекторий [1, 3, 4]. Кроме того, модификации метода Монте-Карло строятся с учетом особенностей использования графических ускорителей в качестве арифметических сопроцессоров [5, 6].

Моделирование радиационной эмиссии электронов включает в себя следующие составные части:

- математическую модель переноса фотонов в многокомпонентных объектах со сложной внутренней структурой, с учетом различных процессов упругого и неупругого взаимодействия рентгеновского и гамма излучений с веществом;
- модель переноса электронов в веществе;
- алгоритм моделирования генерации потоков быстрых электронов, появляющихся в результате взаимодействия ионизирующего излучения с веществом объекта, с учетом основной характерной особенности этого процесса – существенным (до 2 порядков) различием длин пробегов фотонов и электронов;
- модель регистрации эмитируемых электронов детектирующей системой.

Авторами настоящей статьи построены подходы к применению технологии NVIDIA© CUDA для решения задач переноса гамма излучения в многокомпонентных объектах [5]. Разработаны модификации метода Монте-Карло, ориентированные на использование в расчетах графических процессоров. Реализован подход к организации данных для эффективного использования памяти видеоадаптера. В работе [6] описаны алгоритмы статистического моделирования переноса электронов для проведения вычислительных экспериментов на гибридных суперкомпьютерах с использованием технологии NVIDIA©CUDA.

Ниже рассмотрены основные особенности моделирования эмиссии электронов с границ облучаемых объектов на гетерогенных суперкомпьютерах.

Схема моделирования процесса эмиссии электронов изображена на **Рис.1**.

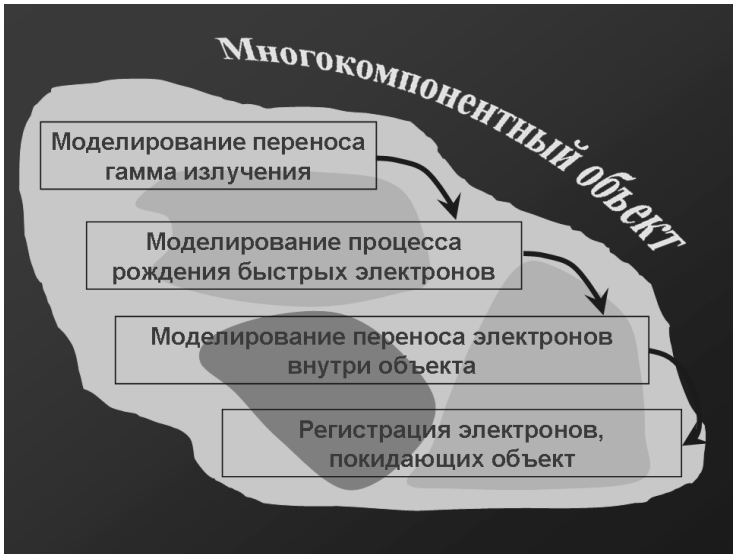


Рис. 1. Моделирование радиационной эмиссии электронов.

Алгоритмы компьютерного моделирования радиационной электронной эмиссии разработаны на основе эффективных весовых модификаций метода Монте-Карло. Основная особенность исследуемого процесса заключается в следующем. Длина пробега фотона до его поглощения или вылета за границы объекта значительно (до двух порядков) превышает длину пути электрона до его термализации. В этой ситуации прямое статистическое моделирование эмиссии неэффективно, поскольку лишь незначительная часть появившихся внутри объекта электронов способна достичь его границ. Используемые в рамках проекта алгоритмы построены на основе принципа максимальной информационной ценности фотонных

траекторий. Фактически это означает следующее. Если на данном звене своей траектории фотон с некоторой вероятностью может дать вклад в искомую величину – плотность потока эмитируемых электронов, то вместо того, чтобы разыгрывать случайное событие – реализация вклада фотона – полагаем, что фотон дал этот вклад. При этом указанная вероятность учитывается в виде весового множителя.

Область рождения электрона, способного покинуть объект, строится до вычисления точки появления этого электрона. Причем на каждом звене фотонной траектории рассматривается процесс рождения двух электронов – и комптоновского, и фотоэлектрона. Каждому из них приписывается статистический вес, равный вероятности соответствующего процесса. Таким образом, маловероятный процесс рождения электрона, способного покинуть граничные поверхности объекта, моделируется с помощью весового алгоритма, значительно снижающего дисперсию результатов.

Рассмотренные алгоритмы обладают высокой степенью однородности и, в силу независимости траекторий частиц, легко распараллеливаются, в том числе с использованием графических ускорителей.

3. Реализация алгоритма на гибридных вычислительных комплексах

Для эффективной реализации вычислительного алгоритма на гибридных вычислительных комплексах необходимо учитывать ряд особенностей:

1. Графические процессоры, в отличие от процессоров общего назначения, хорошо пригодны для вычислений, требующих большого числа арифметических операций при минимуме условных переходов. Обилие условных переходов может, наоборот, значительно снизить производительность графического процессора.

2. Объем доступной графическому процессору памяти не велик. Для современных моделей он исчисляется гигабайтами. В версии 4.0 технология CUDA позволяет GPU использовать оперативную память, однако при этом значительно снижается производительность.

3. Для эффективной загрузки вычислительных средств необходима балансировка загрузки CPU и GPU, что не всегда является тривиальной задачей из-за слишком разных их вычислительных способностей.

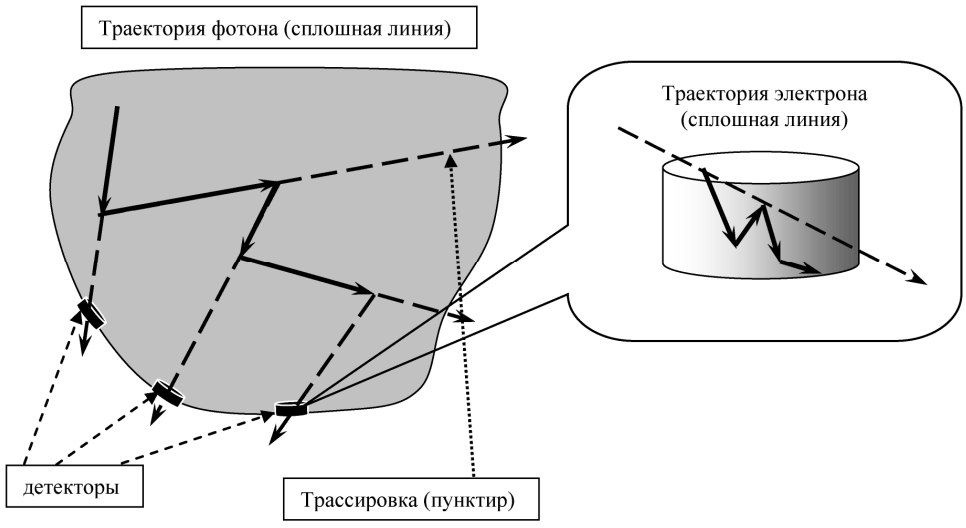


Рис. 2. Алгоритмическая схема моделирования электронной эмиссии

3.1 Вычислительная схема моделирования

На **Рис. 2** изображена общая алгоритмическая схема моделирования процесса эмиссии электронов с граничных поверхностей объектов, облучаемых фотонным излучением. Последовательность этапов моделирования следующая:

– выполняется трассировка объекта (определение точек пересечения луча вдоль очередного направления движения фотона с гомогенными оболочками объекта) из очередной точки поворота фотонной траектории;
– определение детекторов, пересекаемых указанным выше лучом (трассировка детекторов);
– моделирование процессов рождения комптоновского и фотоэлектрона в пересекаемых детекторах;
– моделирование траектории электрона до его термализации или вылета из объекта;
– регистрация электрона, если он пересек внешнее основание детектора;
– продолжение траектории фотона.

Последовательность этапов моделирования очередного звена фотонной траектории показана на **Рис. 3**. Первым шагом вычислительной схемы является трассировка объектов – определение интервалов пересечения луча вдоль текущего направления движения фотона с однородными составляющими объекта. С использованием рассчитанных данных об интервалах пересечения выполняется розыгрыш точки взаимодействия, определение типа взаимодействия и изменение характеристик фотона. Вновь рассчитанные характеристики фотона (энергия, статистический вес и направление движения) сохраняются для моделирования следующего звена фотонной траектории.

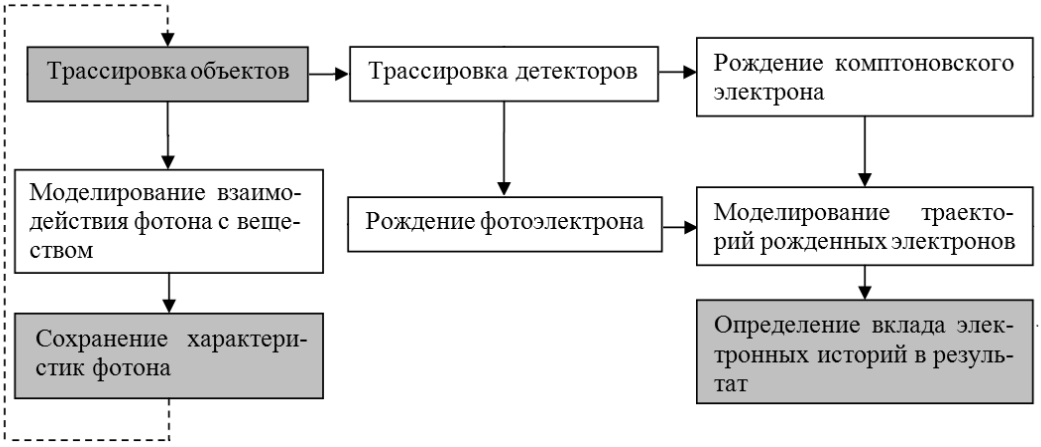


Рис. 3. Схема моделирования звена фотонной траектории

Следующим этапом является трассировка детекторов. Для каждого из пересекаемых фотонным лучом детекторов выполняются следующие шаги:

1. моделирование рождения фотоэлектрона в данном детекторе;
2. моделирование рождение комптоновского электрона в данном детекторе;
3. моделирование траектории каждого из указанных электронов;
4. регистрация электронов, покидающих объект.

Алгоритмы моделирования переноса излучения в веществе, основанные на модификациях метода Монте-Карло, имеют большое число независимых вычислительных ветвей. Такие алгоритмы хорошо поддаются распараллеливанию на любых архитектурах, в том числе имеющих мало возможностей для синхронизации доступа к данным (гибридные архитектуры, узлы которых включают «обычные» процессоры и графические ускорители). В то же время такие алгоритмы требуют огромного объема арифметических вычислений. Эти обстоятельства определяют высокую пригодность такого рода алгоритмов для реализации на гибридных вычислительных комплексах. Как показано в [5, 6] реализация алгоритмов моделирования переноса фотонов и электронов на графических процессорах имеет высокую эффективность. Это обусловлено относительно равномерной плотностью вычислений на всех этапах указанных алгоритмов (геометрическая часть, физическая часть, статистическая часть), а также небольшим объемом исходных данных и малой размерностью результатов.

Моделирование радиационной электронной эмиссии существенно отличается от задач, рассмотренных в упомянутых работах, что затрудняет прямое использование обсуждаемых в [5, 6] подходов к распараллеливанию вычислений.

Решением задачи в актуальных постановках является информация о характеристиках потоков электронов в очень большом числе точек на граничных поверхностях объектов. Это приводит к высокой размерности массивов результатов (миллионы ячеек), что вместе с необходимостью описания объектов большим числом треугольников (десятки и сотни тысяч) определяет высокий уровень требований к памяти гибридных узлов вычислительной системы. Чрезвычайно большой объем информации, требуемый для описания объекта и детекторов, делает невозможным полный перенос вычислений на графический процессор из-за ограниченности памяти GPU.

Алгоритм трассировки объекта основан на переводе вершин треугольников в локальные системы координат, связанные с направлением движения фотона. Это преобразование необходимо производить до вычисления координат точек пересечения фотонного луча и треугольника. В противном случае, операцию перевода вершины в локальную систему координат пришлось бы производить несколько раз – для каждого из треугольников, содержащего данную вершину. Такое увеличение количества операций резко снизило бы эффективность всего алгоритма.

Использование графического процессора ориентировано на модель массового параллелизма. Для обеспечения массовости (а, значит, и эффективности использования GPU), необходимо одновременное выполнение десятков тысяч вычислительных потоков. В рассматриваемом случае это означало бы трассировку объектов для всех фотонов одновременно. С учетом вышесказанного, это означает, что для каждой моделируемой частицы необходимо иметь в памяти собственную копию данных о вершинах треугольников.

В практических задачах для корректного описания объекта, имеющего сложную гетерогенную структуру, требуется около порядка 10^5-10^6 вершин треугольников, то есть, при использовании одинарной точности, на хранение информации о вершинах требуется порядка 12 Мб памяти. Получается, что, с учетом объемов памяти современных графических процессоров, максимальное число одновременно моделируемых (отдельными вычислительными потоками) фотонов не может превышать нескольких сотен. В то же время, для обеспечения массового параллелизма на последующих этапах вычислительной схемы необходимо одновременно строить десятки тысяч звеньев фотонных траекторий, что оказывается невозможным в задачах со сложной геометрией.

Помимо этого, одновременно необходимо моделировать электронные траектории и хранить статистическую выборку высокой размерности (до десятков миллионов ячеек), что ещё больше ограничивает объем доступной памяти и, соответственно, снижает степень параллелизма в расчетном алгоритме.

Учитывая вышесказанное, можно сделать вывод том, что использование GPU в качестве вычислителя на всех этапах моделирования оказывается неэффективным.

Следовательно, возникает необходимость реализации одних частей общего алгоритма на центральном процессоре, а других – на графическом. А именно, части алгоритма с высокой плотностью вычислений выполняются на GPU, а с относительно низкой – на CPU. Такое разделение является нетривиальной задачей и требует тщательного анализа плотности вычислений на всех этапах алгоритма решения задачи.

Проведенные исследования вычислительной загрузки и плотности вычислений на каждом из этапов рассматриваемого метода моделирования радиационной эмиссии электронов, позволил оценить относительную плотность вычислений различных частей общего алгоритма. Анализ разработанного метода показал, что одновременное моделирование траекторий фотонов и порожденных ими электронов проводит к сильно неравномерной плотности вычислений на разных этапах расчетного алгоритма. Части алгоритма с низкой плотностью вычислений – моделирование взаимодействия фотона с веществом, рождение электронов, статистическая часть - чередуются с частями, имеющими высокую вычислительную плотность – трассировка объектов, трассировка детекторов и моделирование траекторий рожденных электронов.

Сделанные выводы позволили разделить вычислительную загрузку между центральными и графическими процессорами, что позволило значительно повысить эффективность общего алгоритма.

Рассмотрим подробнее различные этапы предложенной вычислительной схемы.

3.2 Трассировка объекта и детекторов

Трассировка многокомпонентного объекта, то есть вычисление точек пересечения луча вдоль очередного направления движения фотона с гомогенными компонентами и определение оптических толщин этих компонент вдоль указанного направления, является одной из центральных задач моделирования переноса излучения.

Эта задача решается в два этапа:

- определение координат точек пересечения фотонного луча с оболочками, ограничивающими гомогенные компоненты объекта;
- формирование оптических толщин пересекаемых компонент вдоль этого луча.

Для описания кусочно-однородных структур, соответствующих сложным многокомпонентным объектам, используется триангуляционная модель [7]. При этом в практически важных случаях оболочки однородных компонент задаются сотнями тысяч треугольников. Поэтому задача определения координат точек пересечения требует огромного числа логически простых вычислений. С учетом указанных выше особенностей использования гибридных систем, этот этап трассировки выполняется на GPU.

Рассмотрим этот этап подробнее.

Известно, что вычислительное задание для GPU или «ядро», представляет собой совокупность вычислительных потоков, объединенных в блоки. Блоки в свою очередь объединяются в решетку (GRID). В общем случае блок представляет собой двумерный массив потоков, а решетка – трехмерный массив блоков.

Для выполнения первого этапа трассировки объекта используются два ядра.

Первое ядро предназначено для перевода всех вершин треугольников в локальную систему координат, связанную с направлением движения данного фотона. Конфигурация ядра следующая: блок представляет одномерный массив из 512 потоков (максимально возможное количеству потоков в блоке); решетка представляет собой двумерный массив блоков размера NxM, где N – количество фотонов, M*512 – количество вершин. Таким образом, каждому вычислительному потоку ставится в соответствие пара фотон-вершина (**Рис. 4**).

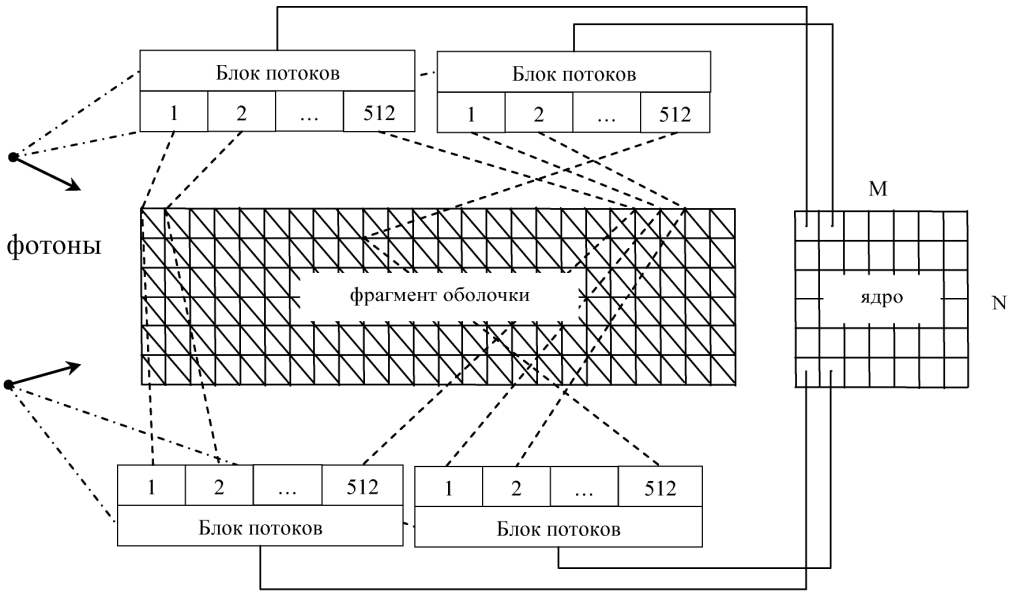


Рис. 4. Конфигурация ядра, выполняющего перевод вершин в локальные системы координат

Второе ядро определяет координаты точек пересечения лучей вдоль текущих направлений фотонов со всеми пересекаемыми треугольниками. Конфигурация ядра следующая (**Рис. 5**): блок представляет собой одномерный массив из 512 потоков, решетка – одномерный массив размера N. То есть, с каждым фотоном ассоциируется отдельный вычислительный блок, вычислительные потоки которого, путем перебора всех треугольников определяют координаты точек пересечения (**Рис. 5**). Результат возвращается в виде N массивов, содержащих координаты пересечения. Для увеличения скорости работы эти массивы изначально

формируются в разделяемой памяти блоков, а для синхронизации доступа к элементам массивов используются атомарные операции, исключающие проблемы одновременного доступа к памяти.

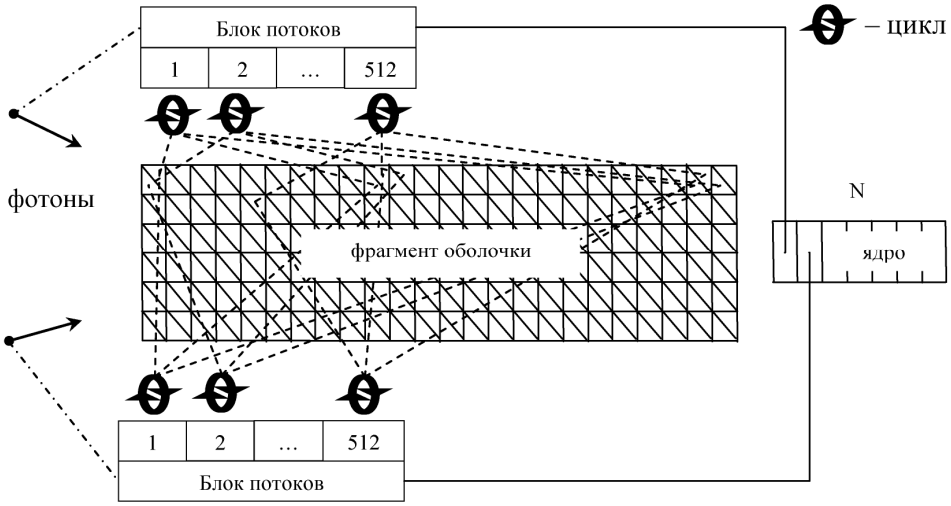


Рис. 5. Конфигурация ядра, выполняющего вычисление координат точек пересечения

Второй этап трассировки объекта - формирование оптических толщин однородных компонент, пересекаемых лучом вдоль направления движения фотона, - требует выполнения, главным образом, логических операций сравнения и перегруппировки. Поэтому этот этап выполняется на центральном процессоре.

Для регистрации электронов, эмитируемых с граничных поверхностей объекта, строятся виртуальные детекторы [1], которые имеют форму цилиндров. Высота этих цилиндров определяется величиной тормозного пути рожденного в данном детекторе электрона, а радиус основания выбирается из априорных геометрических соображений. При этом диаметр основания цилиндра много больше его высоты.

Задачей трассировки детекторов является определение интервалов пересечения указанных цилиндров лучом вдоль направления движения фотона. Трассировка детекторов выполняется в два этапа:

- грубый отсев части детекторов, заведомо не пересекаемых фотонным лучом;
- определение отрезка пересечения детектора фотонным лучом.

Грубый отсев проводится по следующему правилу: отбрасываются детекторы, для которых расстояние от центра их основания до текущего фотонного луча больше диаметра основания. Основная операция при выполнении этого этапа - вычисление расстояния от точки (центра основания детектора) до прямой (фотонного луча). В практически важных задачах количество детекторов исчисляется десятками тысяч. Поэтому этап грубого отсева выполняется на GPU. Обычно этот этап позволяет отбросить более 99% детекторов. Это позволяет значительно снизить вычислительную нагрузку в дальнейшем.

Для каждого из оставшихся детекторов разыгрываются энергии фото- и комптоновского электронов, вычисляются соответствующие величины тормозного пути. Исходя из этого, для каждого из двух указанных электронов вычисляется высота детектора и проводится построение интервалов пересечения детектора фотонным лучом. На построенном интервале в соответствии с распределением условной вероятности разыгрывается точка рождения электрона и направление его первоначального движения в детекторе. Эта часть алгоритма выполняется на CPU.

Затем моделируется траектория каждого из рожденных электронов до его термализации или до вылета за границы объекта. Расчет электронной траектории проводится на GPU с использованием модели индивидуальных соударений (МИС) [2]. Выбор этой модели обусловлен тем обстоятельством, что в разномасштабных объектах, содержащих микроструктурные элементы, применение логически сложных моделей, основанных на широко используемом методе группировки столкновений, может оказаться не эффективным. Кроме

того, МИС обладая простой внутренней логикой, но требуя большого числа арифметических операций, отлично подходит для реализации на графических процессорах.

Моделирование электронных историй требует переконфигурирования памяти графического процессора (выгрузка данных о геометрии объекта, загрузка табличных данных для электронов), поэтому проводить его на каждом звене фотонной траектории нецелесообразно. Вместо этого характеристики рожденных электронов сохраняются в электронном пуле и обрабатываются разом после заполнения данного пула. Причем моделирование электронных траекторий происходит, как отмечено выше, на графическом процессоре, а определение их вклада в статистическую выборку – на CPU. Это связанно с большой размерностью выборки, для хранения которой GPU имеет слишком мало доступной памяти.

После завершения траектории электрона и его регистрации (в случае, если он вылетел через границу объекта) моделируется взаимодействие фотона с веществом той части объекта, в которой находится очередная точка поворота его траектории. Вычисляются новая энергия и новое направление движения. Алгоритмы решения этих задач обладают достаточно сложной логической структурой и, поэтому, выполняются на центральном процессоре.

3.3 Схема проведения расчетов

Проведенный в п. 3.2 анализ плотности вычислений на различных этапах моделирования позволил распределить части алгоритма с различной плотностью между графическими и центральными процессорами узла гибридной вычислительной системы.

На **Рис. 6** показано распределение различных частей вычислительной схемы между различными типами процессоров.

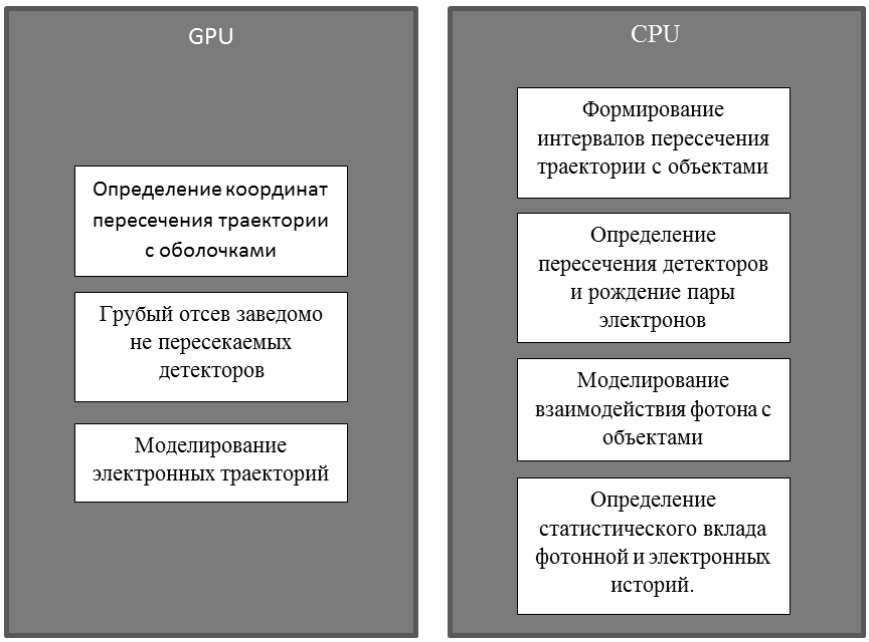


Рис. 6. Распределение частей расчетной схемы между вычислителями

Каждая итерация вычислительного алгоритма (моделирование очередного звена траектории фотона) представляет собой последовательность этапов, на каждом из которых вычисления проходят либо на CPU, либо на GPU. Независимость моделирования траекторий фотонов позволяет проводить серию расчетов одновременно. При этом в каждый момент времени разные расчеты серии находятся на разных этапах, таких образом вычисления одновременно проходят как на CPU так и на GPU. Это позволяет еще больше увеличить эффективность использования гибридной вычислительной техники.

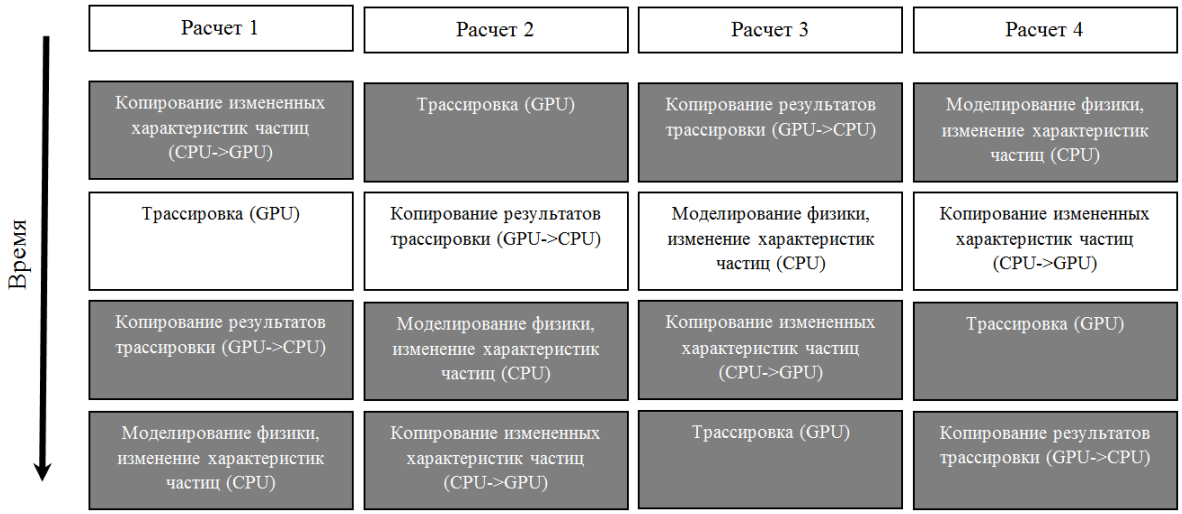


Рис. 7. Проведение серии расчетов для одновременной загрузки CPU и GPU

На **Рис. 7** показан пример схемы проведения серии расчетов.

- первый расчет проводит трассировку на GPU на основе новых координат и направлений движения частиц, скопированных на GPU на предыдущем шаге;
- второй расчет копирует результаты трассировки, выполненной на предыдущем шаге на CPU;
- третий расчет серии, на основе данных трассировки, полученных с GPU шагом ранее, производит моделирование физики;
- четвертый расчет копирует изменённые характеристики частиц на GPU для трассировки на следующем шаге.

4. Пример моделирования электронной эмиссии

Разработанные алгоритмы моделирования реализованы в виде параллельного программного кода с рабочим названием HCEE (Hybrid Calculation of Electron Emission).

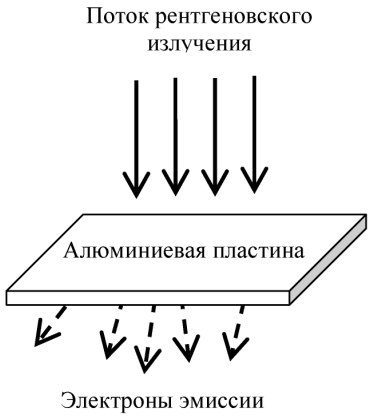


Рис. 8. Схема вычислительного эксперимента с пластиной

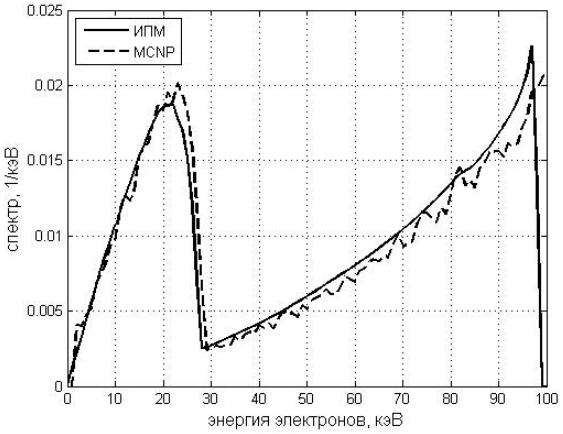


Рис. 9. Результаты сравнительного расчета спектра эмитируемых электронов

На этапе тестирования HCEE проведено сравнение результатов вычислительного эксперимента, изображенного на **Рис. 8** с аналогичными результатами, полученными с помощью пакета MCNP [8]. В этом эксперименте алюминиевая пластина толщиной 1 мм облучается потоком рентгеновских фотонов с энергией 100 кэВ. Регистрируется электронный поток за пластиной (**Рис. 8**).

На **Рис. 9** представлены графики спектра электронов эмиссии. Получено удовлетворительное согласие результатов.

Отметим, что пик в области 20 кэВ соответствует потоку комптоновских электронов, а правый пик в области более 95 кэВ – потоку фотоэлектронов, потерявших энергию при переносе до нижней границы пластины.

Время расчетов с помощью MCNP превысило время, затраченное при использовании НСЕЕ, в сотни раз. Однако сравнение быстродействия в рассматриваемом случае не является корректным, поскольку расчеты с использованием MCNP проведено на кластере с обычной архитектурой, в то время как моделирование с помощью НСЕЕ выполнено на вычислительном кластере К-100 (http://www.kiam.ru/MVS/resourses/k100.html) с гибридной архитектурой.

По заказу Федерального Института контроля и исследования материалов (BAM, Берлин, Германия) проведены расчеты плотности потока эмитируемых электронов с внешних и внутренних поверхностей трубы, облучаемой потоком 100-кэвных фотонов.

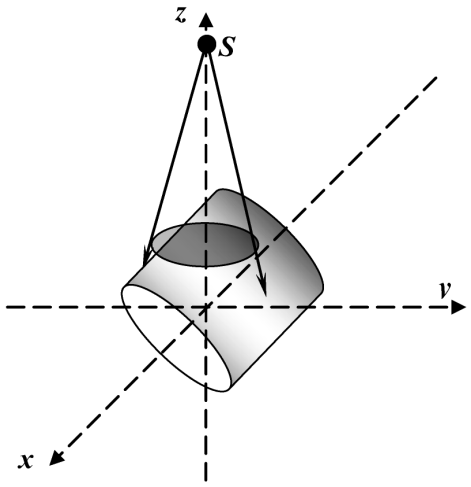


Рис. 10. Схема вычислительного эксперимента с трубой

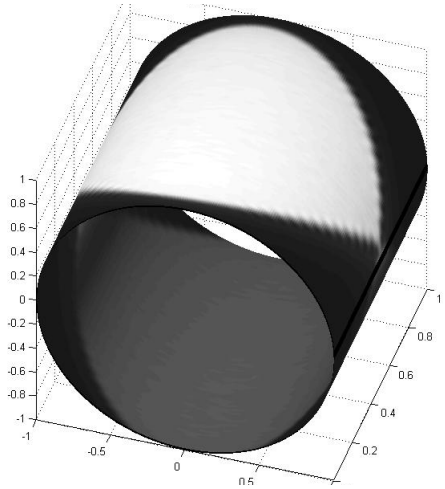


Рис. 11. Распределение плотности потока электронов с внешней облучаемой стороны трубы

Схема вычислительного эксперимента показана на **Рис. 10**. На **Рис. 11** изображены результаты расчета плотности потока электронов с внешней облучаемой стороны трубы. По координатным осям на **Рис. 11** отложены пространственные координаты в относительных единицах. Величина плотности потока изображена в градациях серого цвета: светлым участкам соответствует бо́льшая плотность, темным – меньшая плотность электронного потока.

5. Заключение

Разработаны эффективные статистические алгоритмы математического моделирования процесса радиационной электронной эмиссии с использованием гибридной вычислительной техники. Эти алгоритмы построены с учетом особенностей проведения вычислений на многопроцессорных суперкомпьютерах с использованием графических ускорителей в качестве арифметических сопроцессоров. Анализ вычислительной схемы алгоритма дал возможность провести разделение частей алгоритма по степени плотности вычислений, что позволило повысить эффективность метода решения задачи путем выравнивания загрузки центральных и графических процессоров.

Литература

1. Жуковский М.Е., Скачков М.В. Моделирование эмиссии электронов с поверхностей объектов, облучаемых ионизирующим излучением. Вестник МГТУ им. Н.Э.Баумана, 4, 2009.

2. Жуковский М.Е., Подоляко С.В., Усков Р.В. Модель индивидуальных соударений для описания переноса электронов в веществе. // Математическое моделирование, Том 23, № 6, стр. 147-160, 2011.
3. Stephen M. Seltzer, “An Overview of ETRAN Monte Carlo Methods,” in Monte Carlo Transport of Electrons and Photons, edited by Theodore M. Jenkins, Walter R. Nelson, and Alessandro Rindi, (Plenum Press, New York, 1988) 153.
4. Vilches M., Garcia-Pareja S., Guerrero R., Anguiano M., Lallena A.M. Monte Carlo simulation of the electron transport through thin slabs: A comparative study of PENELOPE, GEANT3, GEANT4, EGSnrc and MCNPX. // Nucl.Instrum.Meth., B254:219-230,2007.
5. Жуковский М.Е., Усков Р.В. Моделирование взаимодействия гамма-излучения с веществом на гибридных вычислительных системах. // Математическое моделирование, том 23, № 7, стр.20-32, 2011.
6. Жуковский М.Е., Подоляко С.В., Усков Р.В. Моделирование переноса электронов в веществе на гибридных вычислительных системах. // Вычислительные методы и программирование, том 12, 2011, с.152-159.
7. Жуковский М.Е., Загонов В.П., Подоляко С.В., Скачков М.В., Тиллак Г.-Р., Беллон К.. Применение поверхностно ориентированного описания объектов для моделирования трансформации рентгеновского излучения в задачах вычислительной диагностики. Математическое моделирование. 2004, т.16, №5,.с.103-116.
8. Briesmeister J.F. (ed.) MCNP – A General Monte Carlo N-Particle Transport Code. LANL Report LA-13709-M, Los Alamos, 2000.

Моделирование нейтронно-физических процессов активной зоны реактора АЭС в реальном времени с применением распределенных вычислений

О.О. Казьмин[1], И.А. Капацкая[3], С.А. Карпов[2]

НИИ Механики МГУ[1], ОАО «ВНИИАЭС»[2], ОАО «Джэт»[3]

В данной статье представлены результаты работ, направленных на адаптацию существующих алгоритмов моделирования нейтронно-физических процессов в активной зоне реактора АЭС для работы на суперкомпьютерных платформах в рамках создания ПТК «Виртуальная АЭС с ВВЭР». За основу модификации была взята модель активной зоны реактора ВВЭР-1000 полномасштабного тренажера Нововоронежской АЭС 2. Проанализирована производительность данной модели с целью выявления наиболее затратных ее частей. Исследованы отдельные компоненты модели и связи между ними для определения возможности распараллеливания. Разработана и реализована распределенная версия алгоритмов модели. Реализованные методы были апробированы, исследована их эффективность и предложены направления дальнейшей их оптимизации.

1. Введение

Задача обеспечения безопасности существующих и проектируемых АЭС становится еще более актуальной в связи с рядом чрезвычайных ситуаций, произошедших на атомных электростанциях в последние десятилетия. Принимая во внимание цену каждой ошибки при эксплуатации АЭС и отсутствие реальной альтернативы атомной энергетике, следует самое серьезное внимание уделить моделированию различных внештатных ситуаций и их вероятных последствий, в первую очередь - чрезвычайного характера.

В настоящее время во Всероссийском научно-исследовательском институте атомных электростанций (далее ВНИИАЭС) создается суперкомпьютерная полномасштабная модель атомной электростанции, именуемая "Виртуальной АЭС", с помощью которой можно серьезно продвинуться в решении таких задач, как

1) расчет штатных режимов эксплуатации, а также режимов с нарушением нормальных условий эксплуатации существующих и проектируемых АЭС;
2) качественная подготовка оперативного персонала АЭС с помощью симуляторов управления АЭС в режиме реального времени;
3) разработка оптимальных сценариев минимизации последствий внештатных ситуаций на существующих и проектируемых АЭС;
4) упреждающее моделирование в процессе принятия решений по выходу из опасных внештатных ситуаций.

Полномасштабная модель создается на основе прошедших апробацию и хорошо зарекомендовавших себя алгоритмов и их программных реализаций на ЭВМ традиционной последовательной архитектуры. Среди них немаловажную роль играет нейтронно-физический код ТРЕК, предназначенный для моделирования нестационарных процессов в активной зоне реактора - изменения пространственного распределения плотности потока нейтронов, энерговыделения, а также остаточного энерговыделения в активной зоне.

Повышение точности расчетов при условии моделирования в режиме реального времени, необходимость многовариантных расчетов а также упреждающего моделирования требуют высокопроизводительной реализации расчетных кодов. Для этого в настоящее время ведется их адаптация к современным суперкомпьютерным вычислительным установкам, особенностью которых является массовый параллелизм и распределенная по узлам оперативная память. Компактные варианты подобных систем могут в перспективе позволить оснастить высокоточными системами моделирования все заинтересованные организации, в первую очередь ситуационные и учебные центры.

Однако современным суперкомпьютерам присущи некоторые ограничения, которые делают распараллеливание данного программного комплекса достаточно сложной задачей. В настоящей статье рассматриваются следующие вопросы, связанные с переносом указанного кода на суперЭВМ:

1) производится анализ компонентов системы, занимающих наибольшее количество времени, для определения возможности их распараллеливания;
2) предлагается схема распараллеливания;
3) проводится анализ эффективности и выявляются механизмы, ограничивающие ее производительность;
4) предлагаются перспективные методы оптимизации расчетов с учетом целевой платформы, механизмов распараллеливания и особенностей модели.

2. Общее описание модели

Существуют различные методы моделирования нейтронно-физических процессов в активной зоне атомного реактора, программные реализации которых могут существенно отличаться друг от друга. При адаптации программной модели к суперкомпьютерным вычислительным установкам следует учитывать особенности как алгоритма, так и его программной реализации. По этой причине прежде, чем переходить к описанию схемы распараллеливания, необходимо представить использующиеся алгоритмы и особенности их программной реализации. В данной главе описывается методика нейтронно-физического расчета и особенности структуры ее реализации, которые оказывают существенное влияние на адаптацию расчетов к суперкомпьютерным установкам.

2.1. Методика нейтронно-физического расчета

В нейтронно-физическом расчете используется 3-х мерное, 2-х групповое диффузионное приближение с разделением переменных с 6 группами запаздывающих нейтронов. Амплитуда изменения мощности определяется в точечном приближении.

При расчете остаточного энерговыделения учитываются 11 групп осколков деления, распределенных по объему активной зоны (АЗ). Из продуктов деления рассматриваются только ксенон, самарий, йод и прометий, т.к. существенное влияние на временное поведение реактора оказывают только эти элементы, остальные продукты деления учтены в сечениях, рассчитанных для определенного момента времени. Зависимость средней времени жизни нейтрона от плотности воды и выгорания незначительна, поэтому при расчетах используется усредненное по реактору время жизни нейтрона.

Программа обеспечивает расчет объемного распределения энерговыделения в стационарных и нестационарных режимах работы реактора до 10 раз в секунду, в режиме реального времени.

Выгорание топлива не моделируется. Поведение реактора в различные моменты кампании моделируется через использование сечений, зависящих от выгорания топлива.

Исходная система уравнений, отвечающая диффузионной двухгрупповой модели, выглядит следующим образом:

$$\begin{cases} \dfrac{1}{v^{(1)}}\dfrac{\partial\varphi^{(1)}(\vec{r},t)}{\partial t} = \nabla D^{(1)}\nabla\varphi^{(1)}(\vec{r},t) + (1-\beta)\left[\upsilon_f^{(1)}\Sigma_f^{(1)}(\vec{r})\varphi^{(1)}(\vec{r},t) + \upsilon_f^{(2)}\Sigma_f^{(2)}(\vec{r})\varphi^{(2)}(\vec{r},t)\right] - \\ \qquad -\Sigma_a^{(1)}(\vec{r})\varphi^{(1)}(\vec{r},t) - \Sigma_d^{(1\to 2)}(\vec{r})\varphi^{(1)}(\vec{r},t) + \sum_{m=1}^{6}\lambda_m C_m(\vec{r},t) + S^{(1)}(\vec{r},t) \\ \dfrac{1}{v^{(2)}}\dfrac{\partial\varphi^{(2)}(\vec{r},t)}{\partial t} = \nabla D^{(2)}\nabla\varphi^{(2)}(\vec{r},t) - \Sigma_a^{(2)}(\vec{r})\varphi^{(2)}(\vec{r},t) + \Sigma_d^{(1\to 2)}(\vec{r})\varphi^{(1)}(\vec{r},t) + S^{(2)}(\vec{r},t) \\ \dfrac{\partial C_m(\vec{r},t)}{\partial t} = \beta_m\left[\upsilon_f^{(1)}\Sigma_f^{(1)}(\vec{r})\varphi^{(1)}(\vec{r},t) + \upsilon_f^{(2)}\Sigma_f^{(2)}(\vec{r})\varphi^{(2)}(\vec{r},t)\right] - \lambda_m C_m(\vec{r},t), \qquad m = 1..6 \end{cases} \tag{2.1.1}$$

где $\varphi^{(g)}(\vec{r},t)$ - функция потока нейтронов, g – номер энергетической группы

$D^{(g)}$ – коэффициент диффузии

β - доля запаздывающих нейтронов

$\Sigma_{f,a,d}^{(g)}(\vec{r})$ – макросечение деления, поглощения, увода нейтронов

λ_m, C_m - постоянная распада и концентрация ядер эмиттеров запаздывающих нейтронов

$S^{(g)}(\vec{r},t)$ – источники нейтронов

Функция потока нейтронов представляется как $\varphi^{(g)}(\vec{r},t) = T(t)\Phi^{(g)}(\vec{r},t)$, где $T(t)$ - амплитудная функция, $\Phi^{(g)}(\vec{r},t)$ – функция пространственного распределения, предполагается слабое ее изменение по отношению к $T(t)$. Для нахождения пространственной функции распределения потока нейтронов используется итерационный метод.

2.2. Структура программной модели

Нейтронно-физический расчетный код ТРЕК реализован в рамках базовой среды моделирования S3, которая допускает использование MPI-процессами всех вычислительных ядер суперкомпьютера. Запуск расчетного кода осуществляется в S3 в виде запуска с определенной частотой контрольных модулей программы, в которых уже вызываются расчетные модули. Все модули системы реализованы на языке Фортран. Описание всех модулей нейтронно-физического кода представлено в таблице 1.

Таблица 1. Структура нейтронно-физического кода ТРЕК.

Частота вызова	Контрольный модуль	Подпрограммы	Описание
10	cr4dy1	scrd01	Интерфейс с моделями других технологических систем АЭС
		scrd02 yscrstk yscrsor	Нейтронно-физические константы для ТВС[1] Полиномы для ТВС Полиномы для СУЗ[2]
		scrd04	Реактивность. Амплитудная функция
		scrd05	Остаточное тепловыделение
10	cr4dy2	scrd03	Пространственная функция
10	cr1dy1	scrd06	Матрица для пространственной функции, коэффициентов-утечек
1	cr1dy2	scrd07	Отравление

Расчетные модули комплекса представляют собой относительно самостоятельные расчетные единицы. Как следствие, в большинстве своем они связаны друг с другом довольно небольшим объемом данных. По указанной причине было принято решение проводить модификацию кода программного комплекса на уровне модулей.

2.3. Результаты профилировки расчетного кода ТРЕК

В целях выявления наиболее затратных компонентов программного комплекса была проведена профилировка работы модулей. Анализ производился при моделировании трех минут работы реактора. Результаты представлены в таблице 2.

[1] Тепловыделяющая сборка

[2] Система управления и защиты

Таблица 2. Результаты профилировки расчетного кода ТРЕК.

Контрольный модуль	Подпрограммы	Затраченное время	
		Абсолютное, сек	Относительное, %
cr4dy1	scrd01	менее 0.01	менее 0.01%
	scrd02	0.35	0.47%
	yscrstk	20.51	27.47%
	yscrsor	23.21	31.10%
	scrd04	0.05	0.07%
	scrd05	0.13	0.17%
cr4dy2	scrd03	25.78	34.54%
cr1dy1	scrd06	4.24	5.68%
cr1dy2	scrd07	0.37	0.50%

Нетрудно отметить, что три модуля занимают в общей сложности более 90% времени всей системы. Причем, модули расчета полиномов для ТВС и СУЗ (yscrstk и yscrsor) являются подпрограммами модуля расчета нейтронно-физических констант (scrd02). На основе этих данных было принято решение ограничиться модификацией только указанных компонентов системы.

В модуле scrd06 происходит расчет перетечек нейтронов между точками и поправок к этим перетечкам. Задача распараллеливания немного усложняется, поскольку перетечки между нодами зависят как от состояния самой ноды, так и от состояния соседних нод. Модуль scrd06 занимает порядка 5% времени расчета всей системы, однако он предоставляет большинство входных данных для модуля scrd03. Разработка параллельной версии модуля расчета коэффициентов перетечescrd06 позволяет существенно снизить количество пересылок, что повышает производительность всей системы.

3. Распараллеливание расчета нейтронно-физических констант

Расчет нейтронно-физических констант основывается на сеточном методе вычислений. Расчетная сетка является цилиндром высотой в 15 точек и с 163 точками в основании. Основание поделено на 7 областей, а именно - центральную в виде круга и 6 секторов. По высоте расчетная сетка поделена на 5 групп по 3 слоя в каждой. Схематичное изображение расчетной сетки представлено на рисунке 1, где цифрами обозначено количество точек расчетной сетки в области. На рисунке 2 представлена расчетная сетка в горизонтальной проекции. В каждой точке указанной сетки выполняется расчет коэффициентов для органов регулирования систем управления защиты и для тепловыделяющей сборки, а затем пересчет по ним констант в каждой точке. Особенностью данных вычислений в плане возможности распараллеливания является то, что для каждой точки значения полиномов зависят только от теплогидравлических параметров в самой этой точке и не зависят от параметров других точек. Из этого следует, что расчет в каждой области расчетной сетки может осуществляться независимо друг от друга, а, следовательно, может быть распараллелен без дополнительных обменов данными, кроме рассылки начальных параметров расчета.

Были разработаны две схемы, использующие различные методы распараллеливания - статический и квазистатический для произвольного количества процессов. Данный подход обусловлен различной спецификой этих методов. При статическом распараллеливании снижаются затраты на накладные расходы за счет того, что область расчетов для каждого процесса определена на этапе реализации алгоритма. Однако при использовании большого количества процессов КПД распараллеливания будет падать, как следствие, придется выбирать оптимум с точки зрения затраченные ресурсы / производительность. Квазистатическое распараллеливание, в отличие от статического, позволяет подобрать описанное оптимальное количество процессов. Далее представлены схемы указанных методов.

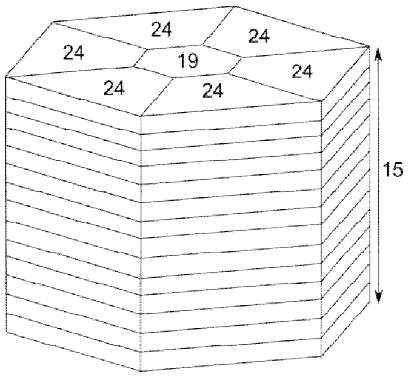


Рис. 1. Разбиение АЗ реактора.

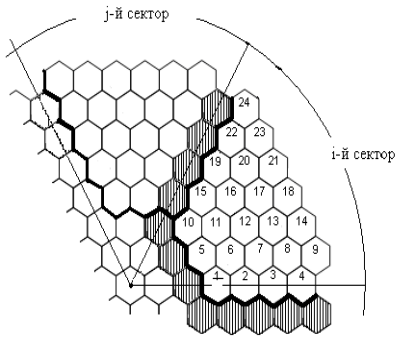


Рис. 2. Разбиение АЗ реактора в горизонтальной проекции.

Для использования произвольного количества процессов вводится одномерная параметризация областей расчетной сетки. Каждая область нумеруется числом от 1 до 105, что сводит распределение вычислительной работы между процессорными элементами к разбиению множества 105 целых чисел на определенное количество подмножеств. В настоящее время процесс с рангом вычисляет полиномы в областях с номерами от до , которые вычисляются по следующим формулам:

3

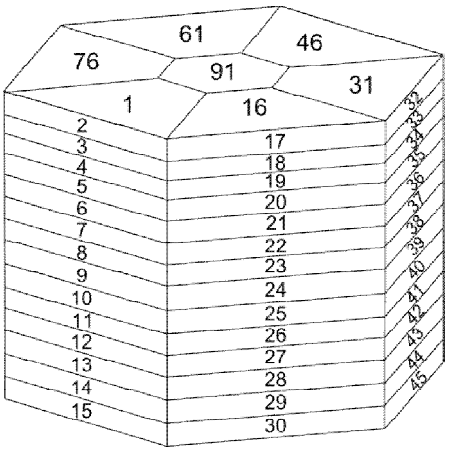


Рис. 3. Одномерная параметризация расчетной области.

Указанный метод гарантирует максимально равномерное распределение вычислительной нагрузки между произвольным количеством узлов, однако уступает по накладным расходам статическому распараллеливанию на несколько арифметических операций и одну передачу данных (параметры отображения трехмерной параметризации в одномерную). Следует также отметить тот факт, что подобный подход не зависит от способа нумерации расчетных областей. Следовательно, возможна оптимизация алгоритма с учетом архитектуры целевой вычислительной установки. В настоящее время используется одномерная параметризация, представленная на рисунке 3.

[3] Остаток от деления на

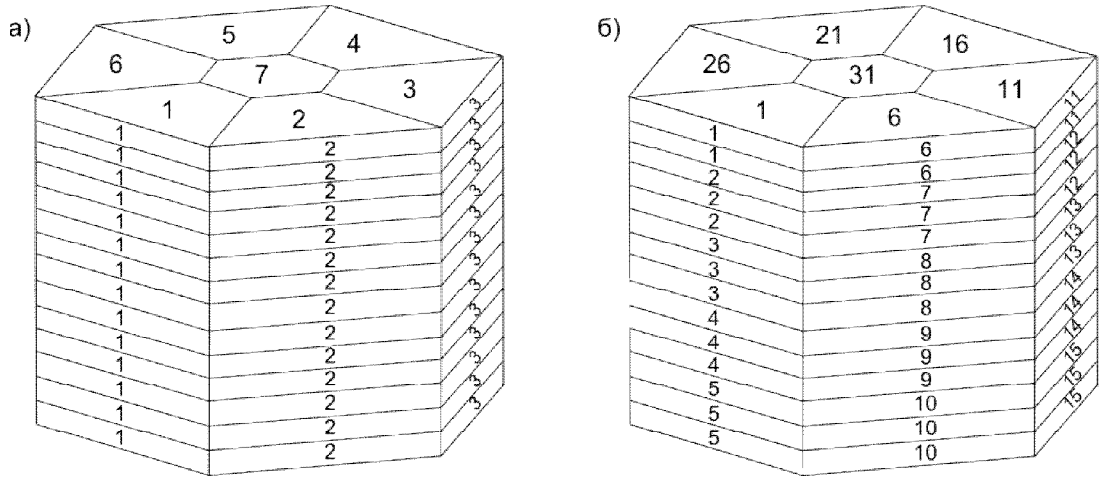


Рис. 4. Распределение расчетных областей по процессам.

Схема со статическим распараллеливанием является схемой расчета с минимальным количеством накладных расходов, которые обусловлены разнесением задачи на несколько процессоров. Данная схема была реализована для 7 и 35 процессов. При запуске на 7 процессорах каждый процесс считает те области расчетной сетки, которым соответствуют одинаковые горизонтальные проекции. При запуске на 35 процессорах указанные области дополнительно делятся на 5 областей по вертикали. Распределение расчетных областей по процессам схематично изображено на рисунке 4 (часть а - для 7 процессов, часть б - для 35), где цифрами указаны номера процессов, рассчитывающих константы в точках данной области. В целом необходимо отметить, что структура алгоритма и реализующего его программного модуля расчета нейтронно-физических констант имеет высокий внутренний параллелизм. Как следствие этот модуль распараллеливается с высоким КПД. Анализ эффективности реализации представленного метода распараллеливания можно найти в главе 6.

4. Распараллеливание расчета пространственной функции

Расчет пространственной функции производится методом итерации источников [2]. Используется та же расчетная сетка, что и в расчете нейтронно-физических констант (см. главу 3). В указанном методе поток нейтронов в точке определяется значениями потока в этой и 8 соседних точках (для гексагональной геометрии), а также источником нейтронов в этой точке. Источник нейтронов в точке определяется суммой потоков нейтронов по группам в этой точке. Потоки и источники нейтронов определяются внутри итерационного цикла. Итерации прекращаются, когда эффективный коэффициент размножения нейтронов перестает меняться с заданной точностью. На каждой итерации производится нормировка источника нейтронов на суммарное энерговыделение по зоне. Таким образом, чтобы приступить к следующей итерации, вначале нужно узнать локальные потоки и источники нейтронов по всей активной зоне.

Расчет пространственной функции на текущий момент распараллелен статически для 7 и 35 процессоров, используя разбиение расчетной области, описанное в разделе 3. В отличие от модуля расчета нейтронно-физических констант, в модуле вычисления пространственной функции присутствуют зависимости по данным между гранулами параллелизма. По этой причине количество пересылок увеличивается, и, соответственно, эффективность распараллеливания снижается. На рисунке 5 схематично приводится алгоритм параллельного вычисления пространственной функции.

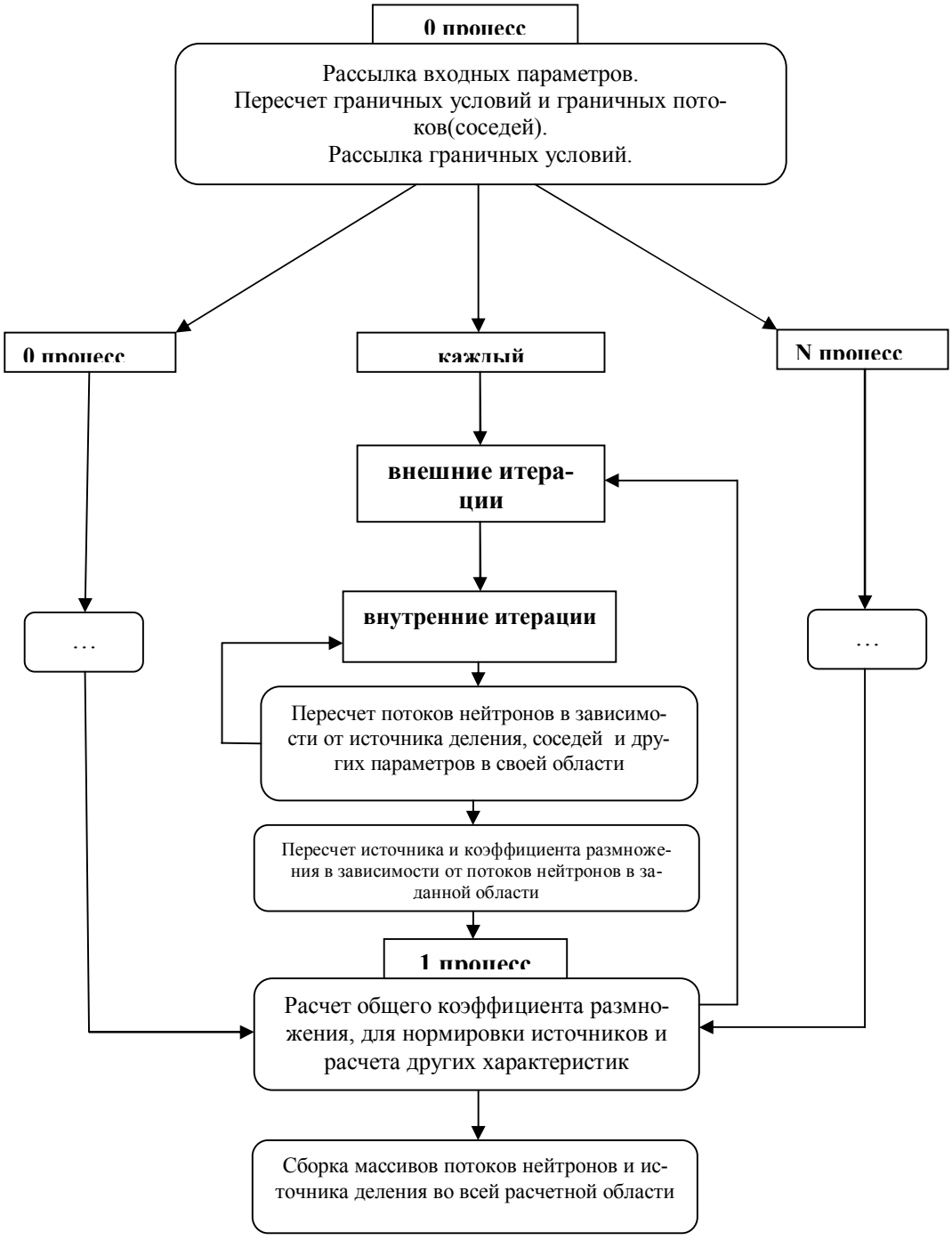


Рис. 5. Алгоритм вычисления пространственной функции.

На схеме алгоритма можно заметить, что во время выполнения внутренних итераций не используются данные, получаемые другими процессами во время выполнения итераций внутренних. Следовательно, весь цикл внутренних итераций может выполняться на удаленном вычислительном узле без дополнительных обменов данными. Однако при пересчете потоков нейтронов требуются данные о всей расчетной области, полученные на прошлой внешней итерации, например общий коэффициент размножения и поток нейтронов в соседних ячейках. По этой причине при инициализации новой внешней итерации реализован механизм коллективного обмена данными и синхронизация процессов.

Описанный выше алгоритм содержит коллективные обмены данными о потоках нейтронов в соседних ячейках. Они были реализованы по схеме "один-всем" и "все-одному". Указанный подход позволяет без существенных издержек поддерживать механизмы сохранения состояния

расчета, которые накладывают условие "все данные при выходе из модуля должны содержаться в памяти нулевого процесса".

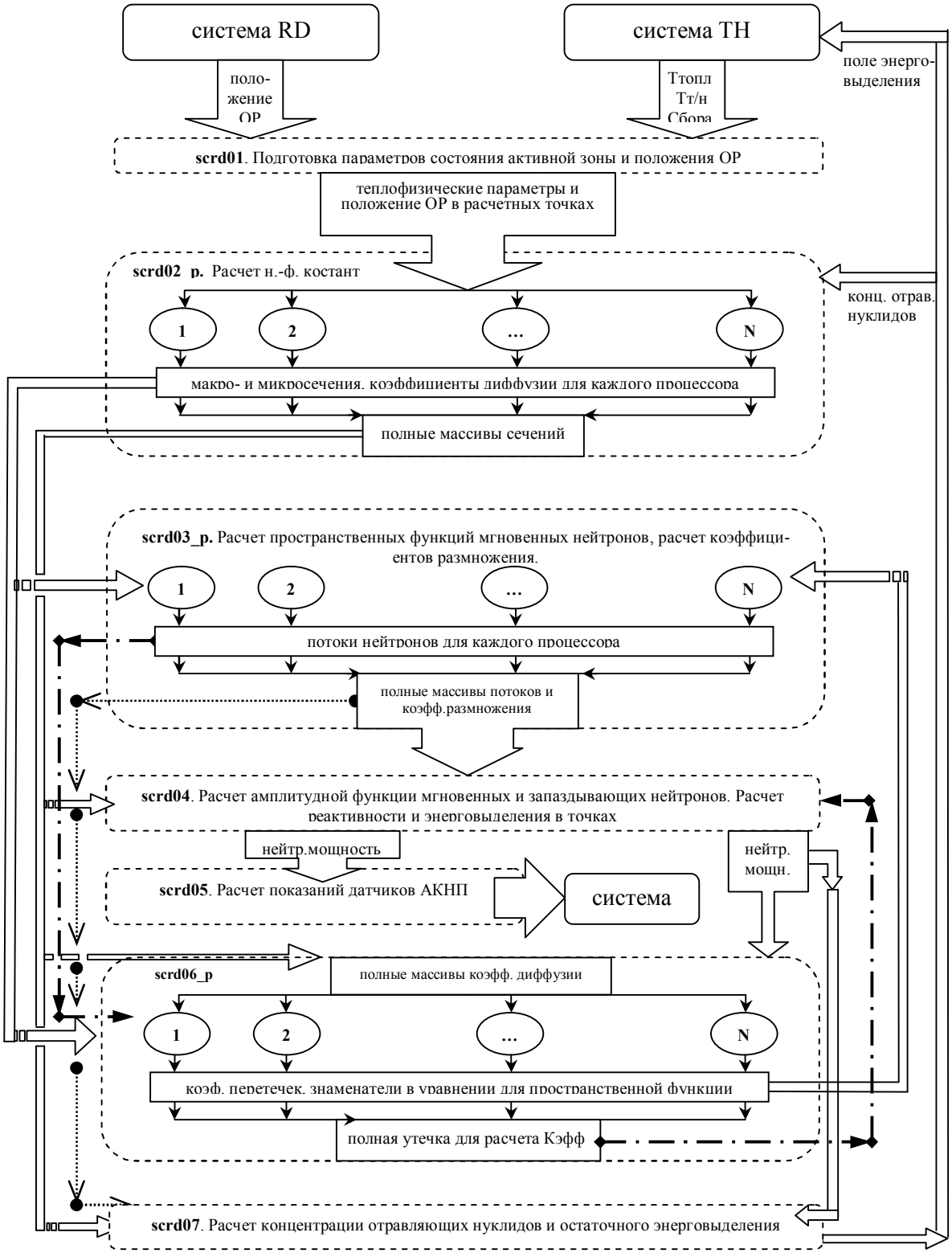


Рис. 6. Схема работы нейтронно-физического кода.

5. Интеграция параллельных модулей в расчетный код

В предыдущих разделах описана схема распараллеливания компонентов системы, которые занимают в сумме более 90% времени работы расчетного кода. Однако в нем также присутствуют модули, работающие последовательно на главном процессоре, и потому возникает проблема "стыковки" параллельных и последовательных частей расчета.

На схеме, изображенной на рисунке 6, можно видеть, что модуль расчета нейтронно-физических констант предоставляет данные для модулей, которые не подвергались модификации. В частности в scrd02 рассчитываются полные массивы сечений, которые используются при вычислении амплитудной функции нейтронов, реактивности и энерговыделения (scrd04, scrd07). Аналогичные зависимости можно отметить и у расчета пространственной функции. В силу этих обстоятельств при завершении параллельного расчета необходим сбор указанных данных в памяти главного процесса, что несколько снижает эффективность распараллеливания.

Подобных издержек можно избежать, разработав параллельные версии для всех модулей системы. В таком случае большая часть данных будет распределена между процессами и необходимость частых пересылок исчезнет. Однако этому препятствует несколько факторов. Во-первых, алгоритмы не всех компонентов системы имеют внутренний параллелизм и могут быть распараллелены. Во-вторых, нейтронно-физический расчетный код ТРЕК реализован в рамках среды моделирования S3, которая накладывает некоторые ограничения на запускаемые расчеты. В данной системе присутствует механизм сохранения и восстановления текущего состояния расчетов, что обуславливает необходимость в определенные моменты собирать все параметры расчета на главном процессоре. И в-третьих, данная задача достаточно трудоемка, что в совокупности с неизвестным по большей части результатом делает ее решение нецелесообразным в рамках данного исследования.

6. Оценка производительности расчета нейтронно-физических констант

Как было указано в главе 3, расчет нейтронно-физических констант имеет высокий внутренний параллелизм и распараллеливается с высокой эффективностью. Под эффективностью (или КПД) распараллеливания здесь и далее понимается отношение минимально возможного времени вычисления на заданном количестве процессоров к полученному времени расчета. Результаты временных раскладок это подтверждают. Была проведена промежуточная оценка времени выполнения модели с распараллеленным модулем scrd02. В качестве объекта тестирования был принят процесс моделирования минуты работы реактора. Тестирование проводилось на сервере ВНИИАЭС. Особенностью данной машины является то, что она использует общую память. Ее наличие сильно ускоряет пересылки данных. Результаты тестирования и их сравнение с идеальным распараллеливанием представлены на рисунке 7. Под идеальным распараллеливанием понимается распараллеливание без каких-либо накладных расходов, то есть ускоряющее расчет в N раз, если N - количество процессоров. Для оценки времени использовался профилировщик gprof.

Представленные на рисунке 7 данные указывают на эффективность распараллеливания при использовании небольшого количества процессоров. При работе на 7 и меньше процессорах КПД распараллеливания колеблется между 80% и 90%. Однако дальнейшие измерения на целевой универсальной компактной Супер-ЭВМ производства ФГУП «РФЯЦ-ВНИИЭФ» [3] выявили сильное падение КПД при увеличении количества процессоров.

Измерения производились при помощи встроенного механизма S3 для получения временной статистики. Он показывает время работы только контрольных модулей. Однако модуль cr4dy1, в котором вызывается расчет нейтронно-физических констант, содержит модули, которые работают последовательно. В силу этого обстоятельства замеры производились не только на главном процессоре, но и на одном второстепенном. Результаты времени счета второстепенного процесса могут быть несколько занижены в силу того, что во всех распараллеленных модулях присутствуют последовательные участки кода.

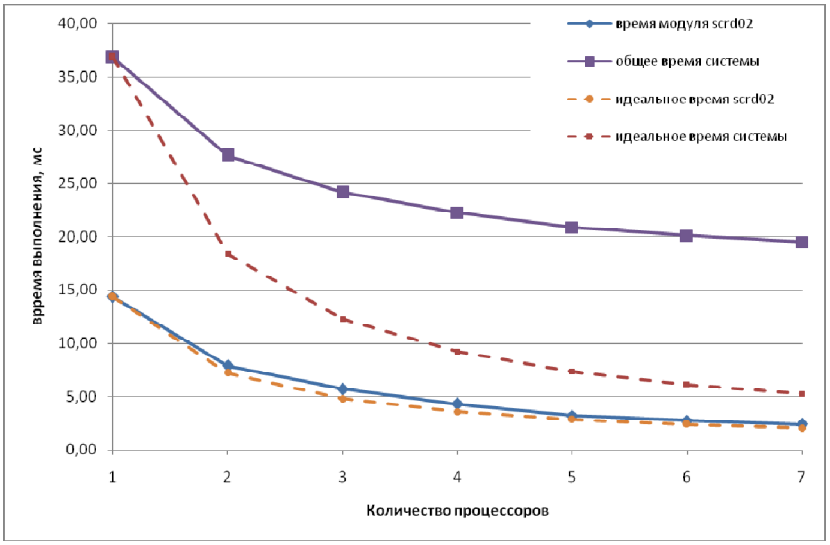


Рис. 7. Промежуточное тестирование параллельной версии расчета нейтронно-физических констант

Результаты, представленные в таблице 3, отражают среднее время выполнения модуля cr4dy1 при моделировании 3 минут работы реактора. В таблице представлены результаты работы контрольного модуля cr4dy1 при моделировании на 1, 7 и 35 процессорах. Результаты счета при параллельном моделировании представлены как на главном процессоре, так и на второстепенном.

Таблица 3. Время счета контрольного модуля cr4dy1.

время выполнения последовательного расчета, мс	время выполнения на 7 процессорах, мс		время выполнения на 35 процессорах, мс	
	главный процесс	второстепенный процесс	главный процесс	второстепенный процесс
17.92	3.43	3.27	3.45	1.39

Время работы распараллеленной модели существенно сокращается по сравнению с временем последовательного расчета, однако полученное на 7 процессорах ускорение сохраняется и на 35 процессорах. Для более подробного анализа производительности было принято решение отделить сам параллельный расчет и пересылки данных. Вследствие этого контрольный модуль cr4dy1 был разделен на 4 контрольных модуля, описание которых представлены в таблице 4.

Таблица 4. Модификация контрольного модуля cr4dy1.

Контрольный модуль	Вызываемые расчетные модули	Описание
cr4dy1_p1		рассылка начальных входных теплофизических параметров для модуля scrd02p
cr4dy1_p2	scrd01	последовательный модуль интерфейса с системами (расчет идет только на главном процессоре)
	scrd02p (scrd02p_1)	расчет нейтронно-физических констант для ТВС (без обмена данными между процессами)
cr4dy1_p3		сбор посчитанных данных на главном процессе
cr4dy1_p4	scrd04	последовательный модуль расчета реактивности и амплитудной функции потоков нейтронов
	scrd05	Последовательный модуль расчета остаточного тепловыделения

Было измерено время выполнения всех указанных частей по тому же принципу (3 минуты работы реактора). Результаты приведены в таблице 5.

Таблица 5. Детализованное время исполнения модуля cr4dy1.

название модуля	время выполнения последовательного расчета, мс	время выполнения на 7 процессорах, мс		время выполнения на 35 процессорах, мс	
		главный процесс	второстепенный процесс	главный процесс	второстепенный процесс
cr4dy1_p1	17.92	0.12	0.31	0.24	0.74
cr4dy1_p2		2.69	2.62	0.62	0.53
cr4dy1_p3		0.51	0.34	2.48	0.12
cr4dy1_p4		0.11	0.00	0.11	0.00

Исходя из данных, представленных в таблице 5, можно утверждать, что резкое снижение КПД распараллеливания обусловлено все большей долей времени, затраченной на пересылки данных. Если при расчете на 7 процессах она составляла 20% времени расчета, то при 35 уже 82%. В свою очередь скорость параллельных вычислений без передачи данных увеличивается более чем в 4,3 раза, что довольно близко к идеальному значению.

Таким образом, для увеличения эффективности вычислений следует снизить долю времени, затраченного на пересылки данных. Решение данной задачи является наиболее перспективным направлением дальнейшего исследования.

7. Производительность расчета пространственной функции

В таблице 6 приведены результаты распараллеливания модуля расчета пространственной функции (scrd03), произведенные по той же методике, как и в предыдущем разделе.

Таблица 6. Время расчета пространственной функции.

время выполнения последовательного расчета, мс	время выполнения на 7 процессорах, мс		время выполнения на 35 процессорах, мс	
	главный процесс	второстепенный процесс	главный процесс	второстепенный процесс
14.01	5.62	5.27	5.99	5.28

На 7 процессах достигнуто ускорение всего в 2,5 раза, что составляет КПД 35%. На 35 процессах ускорение немного ниже (2,33), что составляет КПД всего в 6%. Данный факт обуславливается тем, что сам алгоритм расчета пространственной функции имеет довольно тесные связи между гранулами параллелизма. Как следствие количество пересылок увеличивается, доля времени, затраченного на пересылки, растет, из-за чего падает производительность расчета.

К сожалению, из-за структуры алгоритма невозможно проанализировать долю времени, затраченного на пересылки. Однако по времени последовательного расчета и количеству пересылок в коде можно приблизительно ее оценить. Время последовательного исполнения модулей примерно одинаковое, однако при расчете нейтронно-физических констант функции коллективного обмена вызываются 5 раз (4 BCAST при рассылке входных параметров и 1 REDUCE при сборе посчитанных данных), а при расчете пространственной функции они вызываются более 40 раз, что больше на порядок. Исходя из этого можно предположить, что пересылки данных занимают почти все время расчета.

Таким образом, сокращение затрат на обмен данными становится основным способом оптимизации вычислений. Альтернативой обычному коллективному обмену данных является алгоритм передачи данных, учитывающий особенности разбиения расчетной сетки и алгоритма расчета. Основой такого подхода является тот факт, что для расчета каждому процессу нужны

данные не во всей расчетной сетке, а в ее части. Расчетная область каждого процесса граничит максимум с 8 областями других процессов (для разбиения на 35 процессов). В силу этого существует возможность применения децентрализованного обмена данными на этапе внешних итераций, который осуществляется только между процессами с соседними областями расчета.

Данный подход более затратен, как с точки зрения количества пересылок, так и с точки зрения вычисления номеров процессов с граничащими областями. Однако он позволяет проводить расчет асинхронно в большей части времени исполнения программы, что может дать существенный выигрыш по времени, который покроет излишние расходы на пересылки и вычисление соседей.

К тому же описанный выше подход может быть существенно оптимизирован под целевую аппаратную платформу. Это обуславливается тем, что пропускная способность канала между процессорами разная, следовательно, минимизировав количество общих границ расчетных областей "далеких" процессоров минимизируется среднее время одной пересылки. Указанный подход является довольно трудоемкой задачей, потому он не был реализован в рамках данного исследования. Однако его реализация и апробация является перспективным направлением для дальнейших работ по настоящей теме.

8. Заключение

Проведенные на данном этапе работы является лишь первым шагом к эффективной адаптации нейтронно-физического расчетного кода ТРЕК для использования на суперкомпьютерных платформах. В первую очередь необходима дальнейшая оптимизация механизмов обмена данными в системе.

Снижение затрат времени на передачи данных позволит использовать массовый параллелизм, присущий современным и архитектурным суперЭВМ, которые зачастую используют не только традиционные CPU, но и специализированные аппаратные ускорители.

Поддержка массового параллелизма предоставляет возможность для дальнейшего безболезненного увеличения точности расчетов за счет использования более точной расчетной сетки. Как правило, чем объемнее задача, тем лучше она решается распределенными вычислительными методами.

В случае невозможности повысить эффективность параллельной версии нейтронно-физического расчетного кода будет иметь смысл рассмотреть возможность перехода от итерационного метода поиска решения к вычислению точного решения. Данный метод не рассматривался в рамках данной работы, однако с учетом возможностей современных высокопроизводительных вычислительных установок он имеет право на жизнь.

Литература

1. MPI-2: Extensionsto the Message-PassingInterface // Message PassingInterface Forum, http://www.mpi-forum.org/docs/mpi-20.ps, 1997г.
2. Вычислительные методы в физике реакторов // Сб. статей под ред. Х. Гринспена, К. Келбера, Д. Окрента, Атомиздат, М., 1972г.
3. Компактные Супер-ЭВМ производства ФГУП «РФЯЦ-ВНИИЭФ» // Эл. ресурс: http://www.vniief.ru/directions/grazrab/catalog/infprod/razrabotki/super-evm.html
4. Бартеньев О.В. Современный Фортран // Диалог-МИФИ, М., 2005г.
5. Глесстон С, Эдлунд М., Основы теории ядерных реакторов // Инострлит., М., 1954г.
6. Демидович Б.П., Марон И.А. Основы вычислительной математики. // Наука, М., 1966г.
7. Марчук Г.И., Численные методы расчета ядерных реакторов. // Атомиздат, М., 1958г.

Высокопроизводительный алгоритм для задачи радиационно-кондуктивного переноса тепла в слое*

А.Е. Ковтанюк

Дальневосточный федеральный университет

Рассматривается задача радиационно-кондуктивного теплообмена в рассеивающем слое с отражающими границами. Для ее решения предлагается итерационный рекурсивный алгоритм, основанный на методе Монте-Карло. Проведен анализ и сравнение различных подходов параллелизации алгоритма.

1. Введение

Изучение радиационно-кондуктивного теплообмена [1, 2] является важным во многих инжинерных приложениях. Так в [3-6] изучаются термосвойства некоторых полупрозрачных и изоляционных материалов в рамках радиационно-кондуктивной модели переноса тепла. Математические подходы решения этой нелинейной системы были рассмотрены в работах [7-9]. В [10] доказаны теоремы существования и единственности решения рассматриваемой задачи в случае изотропного рассеяния и неотражающих границ. Решение нестационарной задачи радиационно-кондуктивного теплообмена на основе параллельной реализации метода Монте-Карло проведено в [11].

В данной работе предлагается итерационный рекурсивный алгоритм для решения стационарной задачи радиационно-кондуктивного теплообмена в рассеивающем слое с отражающими границами. Для решения уравнения переноса теплового излучения на каждом шаге итерационной процедуры используется рекурсивный алгоритм, реализующий метод Монте-Карло. Этот алгоритм удобен для применения технологии параллельных вычислений и способен обеспечить требуемую точность за приемлемое вычислительное время. Численные эксперименты проведены для случая изотропного рассеяния и отражающих границ слоя. Проанализированы два различных подхода в параллелизации алгоритма: в первом, параллелизация осуществляется по точкам слоя, во втором - по траекториям метода Монте-Карло. Показано, что разработанные параллельные алгоритмы обеспечивают хорошее, близкое к линейному, ускорение времени выполнения программы. Вычисления осуществлены с использованием технологии параллельных вычислений MPI.

2. Теоретическая часть

Рассмотрим задачу радиационно-кондуктивного переноса тепла в слое [7, 8]. Уравнение радиационного переноса тепла в обезразмеренной форме имеет вид:

$$\nu f_z(z,\nu) + f(z,\nu) = \frac{c}{2}\int_{-1}^{1} p(\nu,\nu')f(z,\nu')d\nu' + (1-c)u^4(z), \qquad (1)$$

где $f(z,\nu)$ обезразмеренная плотность радиационного потока в точке $z \in [\bar{z}_1, \bar{z}_2]$ и в направлении, косинус угла которого с положительным направлением оси z составляет $\nu \in [-1,1]$; $c < 1$ есть альбедо однократного рассеяния, описывающее уровень рассеяния в среде; $p(\nu,\nu')$ - фазовая функция; $u(z)$ - обезразмеренная температура. Введем следующие множества для

*Работа выполнена при финансовой поддержке Федеральной целевой программы 1.4 "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы"(гос. контракт 07.514.11.4013); Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России на 2009-2013 годы"(гос. контракты 14.740.11.0289, 14.740.11.1000, 16.740.11.0456)

задания граничных условий:

$$\Gamma^- = \Big(\{\bar{z}_1\} \times (0,1]\Big) \cup \Big(\{\bar{z}_2\} \times [-1,0)\Big),\ \Gamma^+ = \Big(\{\bar{z}_1\} \times [-1,0)\Big) \cup \Big(\{\bar{z}_2\} \times (0,1]\Big).$$

Присоединим к уравнению (1) следующие граничные условия:

$$f(\bar{z}_i,\nu) = h(\bar{z}_i) + (Bf)(\bar{z}_i,\nu), \quad i = 1,2, \quad (\bar{z}_i,\nu) \in \Gamma^-, \tag{2}$$

где

$$h(\bar{z}_i) := \varepsilon_i U_i^4, \quad (Bf)(\bar{z}_i,\nu) := \rho_i^s f(\bar{z}_i,-\nu) + 2\rho_i^d \int_0^1 f(\bar{z}_i, -sgn(\nu)\nu')\nu' d\nu'.$$

Здесь, U_1 и U_2 обезразмеренные температуры на границах слоя; ρ_i^s и ρ_i^d - коэффициенты зеркального и диффузного отражения; $\varepsilon_i = 1 - \rho_i^s - \rho_i^d$ - коэффициенты излучения для обоих границ.

Уравнение кондуктивного переноса тепла запишем в следующем виде

$$u''(z) = \frac{1}{2N_c}\left(\int_{-1}^{1} f(z,\nu)\nu d\nu\right)', \tag{3}$$

где N_c есть кондуктивно-радиационный параметр [7]. К уравнению (3) присоединим следующие граничные условия:

$$u(\bar{z}_1) = U_1, \quad u(\bar{z}_2) = U_2. \tag{4}$$

Для нахождения решения системы (1)-(4) применим метод простой итерации с параметром. В соответствии с которым, начальное приближение для температуры $u(z)$ (например, линейное приближение, которое соответствует нулевому значению правой части уравнения (3)) обозначим через $u^{\langle 0\rangle}(z)$. Затем, подставляя $u^{\langle 0\rangle}(z)$ в (1) вместо функции $u(z)$, находим решении задачи (1)-(2), и обозначим его через $f^{\langle 1\rangle}(z,\nu)$. Далее, находим решение задачи (3)-(4), соответствующее заданной функции $f^{\langle 1\rangle}(z,\nu)$, и обозначим его как $\tilde{u}^{\langle 1\rangle}(z)$. Выберем малый положительный параметр α и положим $u^{\langle 1\rangle}(z) = \alpha\tilde{u}^{\langle 1\rangle}(z) + (1-\alpha)u^{\langle 0\rangle}(z)$ в качестве следующего приближения функции $u(z)$. Затем, подставляя $u^{\langle 1\rangle}(z)$ вместо функции $u(z)$ в уравнение (1), находим следующее приближение $f^{\langle 2\rangle}(z,\nu)$, и т.д. Таким образом, на j-ом шаге мы используем функции $u^{\langle j-1\rangle}(z)$ и $\tilde{u}^{\langle j\rangle}(z)$ для нахождения следующего приближения функции $u(z)$ по формуле:

$$u^{\langle j\rangle}(z) = \alpha\tilde{u}^{\langle j\rangle}(z) + (1-\alpha)u^{\langle j-1\rangle}(z). \tag{5}$$

Основная сложность в численной реализации итерационного метода заключается в решении краевой задачи для уравнения переноса теплового излучения. Для нахождения решения задачи (1),(2) будем использовать рекурсивный алгоритм, основанный на методе Монте-Карло.

Предварительно сформулируем некоторые ограничения. Положим, что функция $u(z)$ является неотрицательной и $u(z) \in C_b(\bar{z}_1,\bar{z}_2)$, где $C_b(X)$ есть Банахово пространство функций непрерывных и ограниченных на X с нормой $\|\varphi\|_{C_b(X)} = \sup\limits_{x\in X}|\varphi(x)|$. Также, пусть $p(\nu,\nu') \in C_b(\Omega\times\Omega)$, где $\Omega = [-1,0)\cup(0,1]$, и

$$\frac{1}{2}\int_{-1}^{1} p(\nu,\nu')d\nu' = 1.$$

Заметим, что оператор $B : C_b(\Gamma^+) \to C_b(\Gamma^-)$ является линейным, ограниченным и неотрицательным, $\|B\| < 1$.

Для дальнейших рассуждений обозначим

$$X = (\bar{z}_1,\bar{z}_2) \times ([-1,0)\cup(0,1]),$$

$$\xi(\nu) = \begin{cases} \bar{z}_1, & \nu \in (0,1], \\ \bar{z}_2, & \nu \in [-1,0). \end{cases}$$

Введем также следующие интегральные выражения

$$(A\varphi)(z,\nu) = \frac{1}{\nu}\int\limits_{\xi(\nu)}^{z} \exp\left(-\frac{z-z'}{\nu}\right)\varphi(z',\nu)dz',$$

$$(S\varphi)(z,\nu) = \frac{c}{2}\int_{-1}^{1} p(\nu,\nu')\varphi(z,\nu')d\nu',$$

$$(Tf)(z,\nu) = (Bf)(\xi(\nu),\nu)\exp\left(-\frac{z-\xi(\nu)}{\nu}\right) + (ASf)(z,\nu). \tag{6}$$

Отметим, что согласно [12] при выполнении неравенств: $\|B\| < 1$ и $c < 1$, существует единственное решение задачи (1)-(2), которое может быть представлено в виде следующего ряда Неймана, сходящегося в норме $C_b(X)$:

$$f(z,\nu) = \sum_{k=0}^{\infty}(T^k f_0)(z,\nu), \tag{7}$$

$$f_0(z,\nu) = \exp\left(-\frac{z-\xi(\nu)}{\nu}\right)h(\xi(\nu)) + (1-c)(Au^4)(z,\nu).$$

3. Реализационная часть

Рассмотрим итерационный алгоритм (5). Для вычисления решения задачи (1)-(2), соответствующего заданной функции $u(z)$, предложим рекурсивный алгоритм, основанный на методе Монте-Карло. В соответствии с [12] существует единственное решение задачи (1)-(2), которое представляется в виде ряда Неймана (7). Применим алгоритм метода Монте-Карло для вычисления конечной суммы

$$f_N(z,\nu) = \sum_{n=0}^{N}(T^n f_0)(z,\nu). \tag{8}$$

Для реализации метода перепишем (8) в виде следующего рекурсивного отношения

$$f_n(z,\nu) = (Tf_{n-1})(z,\nu) + f_0(z,\nu), \quad n = 1,2,...,N.$$

Рассмотрим структуру оператора T (см. равенство (6)). Он состоит из двух слагаемых: первое описывает эффекты отражения, второе - эффекты рассеяния. Рассмотрим второе слагаемое более подробно. Используя простые преобразования, мы запишем его в следующей форме

$$I(z,\nu) = \frac{c}{2}\left(1-\exp\left(-\frac{z-\xi}{\nu}\right)\right)\times$$

$$\times\int_{\xi}^{z}\int_{-1}^{1}\frac{\exp\left(-(z-z')/\nu\right)}{\nu(1-\exp\left(-(z-\xi)/\nu\right))}p(\nu,\nu')f(z',\nu')d\nu'dz', \tag{9}$$

где $\xi = \xi(\nu)$. В соответствии с методом Монте-Карло мы апроксимируем интеграл в этом выражении как математическое ожидание от случайных последовательностей, определенных через случайные величины z' и ν', распределенные на интервалах (ξ, z) и $(-1,1)$ с плотностями

$$\frac{\exp(-(z-z')/\nu)}{\nu(1-\exp(-(z-\xi)/\nu))}, \quad \text{и} \quad \frac{1}{2}p(\nu,\nu'), \tag{10}$$

Следовательно интеграл (9) апроксимируется следующей суммой

$$\overline{I}(z,\nu) = \frac{c}{M}\left(1-\exp\left(-\frac{z-\xi(\nu)}{\nu}\right)\right)\sum_{k=1}^{M} f(z_k,\nu_k).$$

Здесь, ν_k, z_k, $k = 1, 2, ..., M$ есть независимые реализации случайных величин z' и ν', распределенных на интервалах (ξ, z) и $(-1, 1)$ с плотностями (10). Следовательно, мы можем следующим образом апроксимировать функции $f_n(z,\nu)$, $n = 1, 2, ..., N$:

$$f_n(z,\nu) \approx \overline{f}_n(z,\nu) = \frac{1}{M}\sum_{k=1}^{M} s_{n,k}(z,\nu), \quad \overline{f}_0(z,\nu) = f_0(z,\nu), \tag{11}$$

$$s_{n,k}(z,\nu) = \left(B\overline{f}_{n-1}\right)(\xi(\nu),\nu)\exp\left(-\frac{z-\xi(\nu)}{\nu}\right) +$$

$$+c\left(1-\exp\left(-\frac{z-\xi(\nu)}{\nu}\right)\right)\overline{f}_{n-1}(z_k,\nu_k) + f_0(z,\nu). \tag{12}$$

Таким образом, конечная сумма (8) может быть вычислена на основе рекуррентных соотношений (11), (12).

Для конструирования устойчивого варианта итерационной процедуры запишем аналитическое представление решение задачи (3)-(4) при заданной функции f . После интегрирования равенства (3) получим

$$u(z) = \frac{1}{2N_c}\int_0^z\int_{-1}^{1} f(\zeta,\nu)\nu d\nu d\zeta + C_1 z + C_2, \tag{13}$$

Постоянные C_1 и C_2 определяются из граничных условий (4).

Предположим, что приближение температуры $\overline{u}^{\langle j-1\rangle}(z)$ уже найдено на $(j-1)$ -ом шаге итерационной процедуры (5). На основе (5) и (13) мы получаем следующую апроксимацию для температуры на j-ом шаге:

$$\overline{u}^{\langle j\rangle}(z) = (1-\alpha)\overline{u}^{\langle j-1\rangle}(z) + \alpha\left(\frac{z}{MN_c}\sum_{k=1}^{M}\nu_k\overline{f}_N^{\langle j\rangle}(z_k,\nu_k) + C_1 z + C_2\right), \tag{14}$$

где случайные величины z_k и ν_k являются равномерно распределенными на соответствующих интервалах $(0, z)$ и $(-1, 1)$. Вычисление $\overline{f}_N^{\langle j\rangle}(z,\nu)$ осуществляем на основе формул (11) и (12), полагая, что температура равна $\overline{u}^{\langle j-1\rangle}(z)$. Постоянная C_1 также вычисляется с помощью метода Монте-Карло. Таким образом, приближение $\overline{u}^{\langle j\rangle}(z)$ функции $u^{\langle j\rangle}(z)$ вычисляется с помощью формул (11), (12) и (14).

Для вычисления второго слагаемого в правой части равенства (12) генерируются точки (z_k, ν_k), $k = 1, ..., M$, на соответствующих интервалах $(\xi(\nu), z)$ и $(-1, 1)$ с плотностями (10). Генерация точек z_k осуществляется по следующей формуле:

$$z_k = z + \nu\ln\left(1-\alpha_k(1-\exp(-(z-\xi)/\nu))\right),$$

здесь α_k является независимой реализацией равномерно распределенной на $(0, 1)$ случайной величины. Генерация угловой переменной ν_k определяется конкретным видом фазовой функции $p(\nu,\nu')$.

При вычислении первого слагаемого в правой части равенства (12) используется угловая переменная, соответствующая зеркальному отражению: $-\nu$, и также, генерируется угловая переменная, соответствующая диффузному отражению: $-sgn(\nu)\sqrt{\alpha_k}$.

Заметим, что рекуррентный алгоритм, основанный на методе Монте-Карло, удобен для использования параллельных компьютерных вычислений. Имеется два основных пути параллелизации вычислительного процесса. В первом, вычисление функции f в каждой точке слоя выполняется в отдельном процессе. Другой подход основывается на генерации каждой рекурсивной траектории метода Монте-Карло в отдельном процессе.

4. Экспериментальная часть

Параллелизация рекурсивного вычислительного алгоритма была реализована в C++ коде с использованием технологии MPI. Анализировались перспективы использования двух основных способов параллелизации алгоритма: в первом, параллелизация осуществлялась по точкам слоя, во втором - по траекториям метода Монте-Карло. Вычисление нормализованной температуры проводилось в 32 точках слоя, при этом для нахождения температуры в одной точке генерировалось 8192 (или 2^{13}) траекторий. При проведении вычислительных экспериментов были взяты следующие значения параметров задачи: $N = 8$, $N_c = 0.00001$, $c = 0.9$, $\bar{z}_2 - \bar{z}_1 = 3$, $U_1 = 1$, $\rho_1^s = 0.1$, $\rho_1^d = 0.2$, $\varepsilon_1 = 0.7$, $U_2 = 0.5$, $\rho_2^s = 0.3$, $\rho_2^d = 0.1$ и $\varepsilon_2 = 0.6$.

В таблице 1 приведено время выполнения 10 итераций алгоритма и получаемое ускорение в зависимости от количества используемых вычислительных ядер. Время выполнения программы находилось с помощью метода clock. Ускорение, соответствующее k ядрам, рассчитывалось как отношение времени выполнения программы на 1 ядре ко времени выполнения параллельного кода на k ядрах. Данные, представленные в таблице 2, соответствуют алгоритму, в котором распараллеливание осуществлялось по траекториям метода Монте-Карло. Из анализа полученных результатов следует, что оба параллельных алгоритма обладают высокой эффективностью и обеспечивают хорошее ускорение времени выполнения программы, близкое к линейному. Так, отклонение от "идеального"линейного ускорения при распараллеливании по траекториям на 64 ядрах не превосходит 6%. При этом большую эффективность демонстрирует алгоритм, в котором параллелизация осуществляется по точкам слоя. С другой стороны, алгоритм, осуществляющий параллелизацию по траекториям, достаточно перспективен для реализации на многоядерных суперкомпьютерных комплексах, поскольку содержит большое количество независимых процессов ($2^{11} \sim 2^{13}$), равное количеству генерируемых траекторий. При разработке комбинированного алгоритма, осуществляющего одновременную параллелизацию и по точкам слоя, и по траекториям, количество независимых процессов может достигать 2^{18}.

Таблица 1.Ускорение времени выполнения 10 шагов итерационного алгоритма при распараллеливании по точкам слоя.

Количество узлов	**Время выполнения (мс)**	**Ускорение**
1	11914080	
2	5954700	2,00
4	3003840	3,97
8	1513550	7,87
16	760690	15,66
32	381900	31,20

Целью следующей серии вычислительных экспериментов являлся анализ вычислительной устойчивости предложенного алгоритма. Параметры задачи были взяты как и в предыдущей серии экспериментов, за исключением количества членов ряда Неймана (N), которое в данном случае равнялось 14. Следует отметить, что используемое в экспериментах значение параметра N_c соответствует случаю высоких температур и требует использование

Таблица 2. Ускорение времени выполнения 10 шагов итерационного алгоритма при распараллеливании по траекториям метода Монте-Карло

Количество узлов	**Время выполнения (мс)**	**Ускорение**
1	11914080	
2	6009780	1,98
4	3072630	3,88
8	1536630	7,75
16	761570	15,64
32	394970	30,16
64	196370	60,67

достаточно малого значения параметра α итерационной процедуры (5). При проведении расчетов было взято значение $\alpha = 0.0001$, что в свою очередь замедлило сходимость итерационной процедуры.

На рисунках 1,2 представлены аппроксимации температуры, соответствующие 2000 траекториям метода Монте-Карло, для различного количества шагов итерационной процедуры. Рисунок 1 соответствует 600 шагам итерационного алгоритма; рисунок 2 - 1500 шагам. На рисунках продемонстрирована неустойчивость вычислительной процедуры в случае недостаточного числа траекторий метода Монте-Карло.

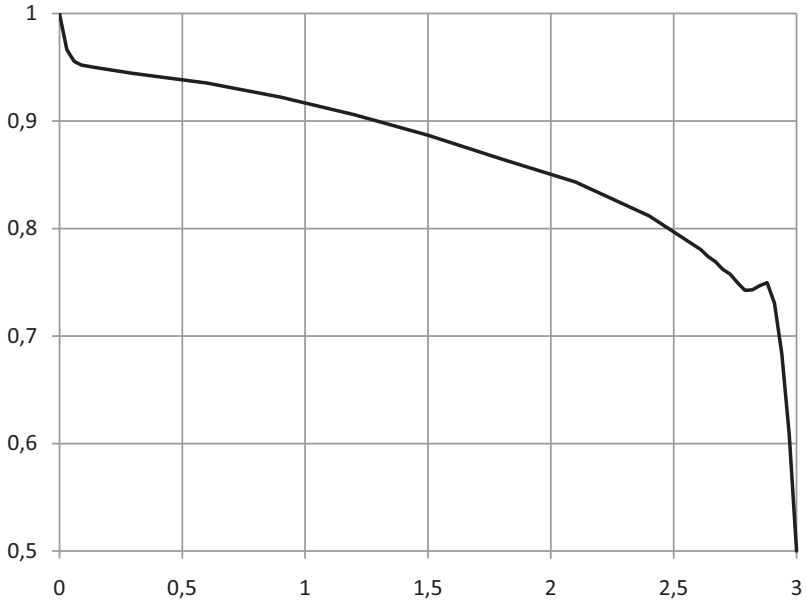


Рисунок 1. Численный эксперимент, демонстрирующий неустойчивость итерационной процедуры, основанной на методе Монте-Карло. Неустойчивость имеет место в случае недостаточного числа траекторий ($M = 2000$). График соответствует 600 шагам итерационной процедуры.

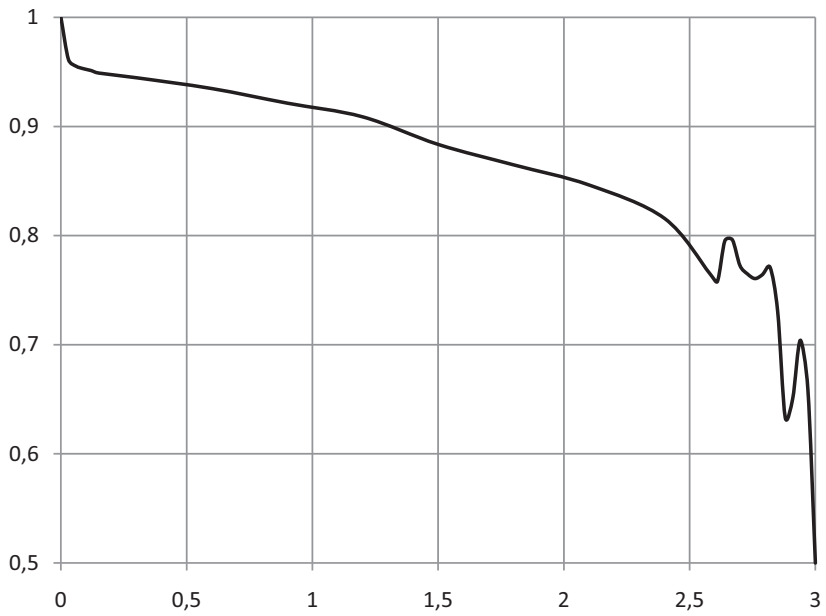


Рисунок 2. Численный эксперимент, демонстрирующий неустойчивость итерационной процедуры, основанной на методе Монте-Карло. Неустойчивость имеет место в случае недостаточного числа траекторий ($M = 2000$). График соответствует 1500 шагам итерационной процедуры.

На рисунке 3 представлены результаты вычислительного эксперимента, соответствующего 10000 траекториям метода Монте-Карло и 1200 шагам итерационной процедуры. Таким образом, увеличение числа траекторий способствует устойчивости вычислительного процесса.

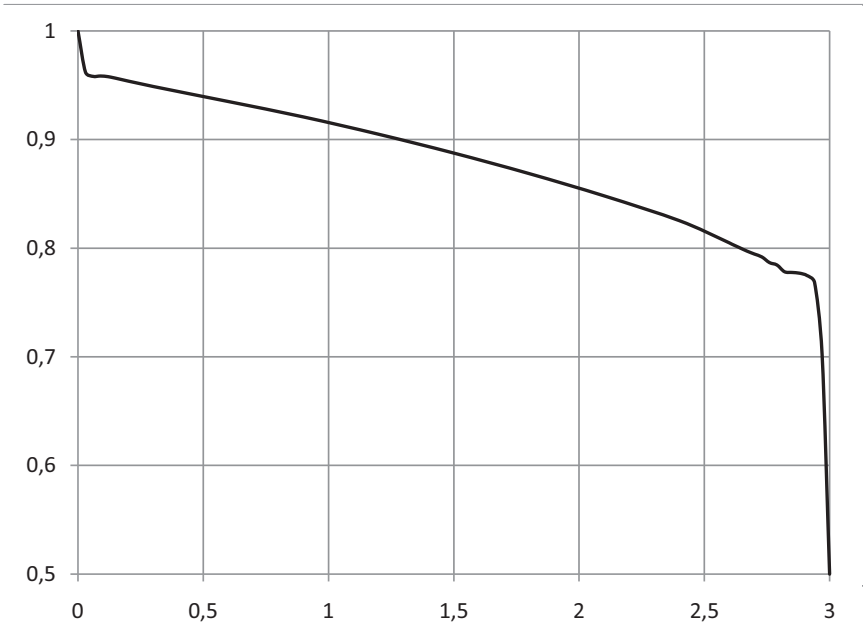


Рисунок 3. Температурный профиль, соответствующий 10000 траекториям метода Монте-Карло и 1200 шагам итерационной процедуры.

5. Заключение

Предложен итерационный рекурсивный алгоритм, основанный на методе Монте-Карло, для нахождения температурного профиля в задаче радиационно-кондуктивного теплообмена в рассеивающем слое с отражающими границами. Данный подход удобен для использования параллельных вычислений. Проанализированы перспективы распараллеливания алгоритма по точкам слоя, и по траекториям метода Монте-Карло. В обоих случаях параллельный алгоритм демонстрирует хорошее ускорение времени выполнения программы, близкое к линейному. Распараллеливание по траекториям является перспективным для выполнения на многоядерных вычислительных комплексах, поскольку позволяет выделить до 10^4 независимых процессов. Представляет интерес разработка комбинированного параллельного алгоритма, осуществляющего параллелизацию по точкам слоя и по траекториям. Число независимых процессов в его реализации может составить $3 \cdot 10^5$, что является основой для эффективной параллелизации алгоритма. Таким образом, реализация алгоритма с использованием суперкомпьютерных вычислений позволит обеспечить хорошую точность рассчета за приемлемое время.

Литература

1. Ozisik M.N. Radiative Transfer and Interaction with Conduction and Convection. John Wiley, New York, 1973.

2. Modest M.F. Radiative Heat Transfer, McGraw-Hill, New York, 1993.

3. Andre S., Degiovanni A. A theoretical study of the transient coupled conduction and radiation heat transfer in glass: phonic diffusivity measurements by the Nash technique// Int. J. Heat Mass Transfer. 1995. Vol. 38, N 18. P. 3401–3412.

4. Andre S., Degiovanni A. A new way of solving transient radiative-conductive heat transfer problems// J. Heat Transfer. 1998. Vol. 120, N. 4. P. 943–955.

5. Banoczi J.M., Kelley C.T. A fast multilevel algorithm for the solution of nonlinear systems of conductive-radiative heat transfer equations// SIAM J. Sci. Comp. 1998. Vol. 19, N. 1. P. 266–279.

6. Klar A., Siedow N. Boundary layers and domain decomposition for radiative heat transfer and diffusion equations: applications to glass manufacturing process// Eur. J. Appl. Math. 1998. Vol. 9, N. 4. P. 351–372.

7. Siewert C.E., Thomas J.R. A computational method for solving a class of coupled conductive-radiative heat-transfer problems// J. Quant. Spectrosc. Radiat. Transfer. 1991. Vol. 45, N. 5. P. 273–281.

8. Siewert C.E. An improved iterative method for solving a class of coupled conductive-radiative heat-transfer problems// J. Quant. Spectrosc. Radiat. Transfer. 1995. Vol. 54, N. 4. P. 599–605.

9. Barichello L.B., Rodrigues P., Siewert C.E. An analytical discrete-ordinates solution for dual-mode heat transfer in a cylinder// J. Quant. Spectrosc. Radiat. Transfer. 2002. Vol. 73. P. 583–602.

10. Kelley C.T. Existence and uniqueness of solutions of nonlinear systems of conductive-radiative heat transfer equations// Transport Theory Statist. Phys. 1996. Vol. 25, N. 2. P. 249–260.

11. Шаенко А.Ю. Распределенный параллельный расчет радиационно-кондуктивного теплообмена методом Монте-Карло на базе графических ускорителей// Доклады пятой международной конференции "Параллельные вычисления и задачи управления Москва. 2010. С. 281–293.

12. Prokhorov I.V., Yarovenko I.P., Krasnikova T.V. An extremum problem for the radiation transfer equation// J. Inverse Ill-Posed Problems. 2005. Vol. 13. P. 365—382.

Разработка прямого решателя для разреженных систем линейных уравнений с симметричной положительно определенной матрицей*

Е.А. Козинов, И.Г. Лебедев, С.А. Лебедев, А.В. Линев, А.Ю. Малова, И.Б. Мееров, А.В. Сысоев, Т.А. Сысоева, С.С. Филиппенко

Нижегородский государственный университет им. Н.И. Лобачевского

Рассмотрен подход к решению СЛАУ с разреженной симметричной положительно-определенной матрицей. Описана поэтапная схема решения СЛАУ с использованием методаХолецкого. Выполнена базовая последовательная программная реализация представленной схемы. Реализована модификация решения на основе супернодального подхода, выполнены ее последовательная и параллельная реализации. Приведены результаты экспериментов, дано их сравнение с результатами, полученными с помощью некоторых известных библиотек.

1. Введение

Одной из актуальных задач алгебры разреженных матриц является решение систем линейных алгебраических уравнений $Ax = b$ с разреженной симметричной положительно определенной матрицей A. Разработка алгоритмов решения таких задач и их высокопроизводительная программная реализация, ориентированная на современные вычислительные архитектуры, представляет большой практический интерес.

На сегодняшний день в мире разработано большое количество специализированного программного обеспечения для решения больших разреженных СЛАУ – так называемые «решатели» СЛАУ. В соответствии с используемыми методами эти решатели подразделяются на прямые и итерационные, некоторые из них имеют высокопроизводительные реализации. В данной работе идет речь о разработке прямого решателя разреженных СЛАУ. Среди известныхпрямых решателей – MKL PARDISO, SuperLU, MUMPS, CHOLMOD и многие другие. На сегодняшний день существуют решатели, ориентированные на различные режимы работы: последовательный, параллельный для систем с общей памятью, параллельный для систем с распределенной памятью. Некоторые из решателей поддерживают режим работы out-of-core, используя жесткий диск в качестве «продолжения» оперативной памяти.Часть решателейработает только для симметричных положительно определенных матриц, другие – для матриц общего вида.Постоянно обновляемый обзор прямых решателей от авторов SuperLU можно найти по ссылке http://crd.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf.Сравнительный анализ функциональности программного обеспечения, реализующего численные методы линейной алгебры, включая прямые и итерационные решатели разреженных СЛАУ, расположен на странице Дж. Донгарра (J. Dongarra):http://www.netlib.org/utk/people/JackDongarra/la-sw.html.

Задачей коллектива авторов является разработка собственной реализации высокопроизводительного прямого решателя разреженных СЛАУ с симметричной положительно определенной матрицей A. Мотивация для создания своего программного комплексасостоит в потенциальной возможности создания конкурентноспособногоинструмента, а также перспективах его использования в учебном процессе.В данной работе приведены текущие результаты проекта, дано их сравнение с результатами некоторых известных библиотек, а также определены пути дальнейшего развития.

* Работа выполнена в лаборатории «Информационные технологии» ВМК ННГУ впри поддержке проекта «Подготовка и переподготовка профильных специалистов на базе центров образования и разработок в сфере информационных технологий», госконтракт № 07.P20.11.0030.

2. Постановка задачи

Дана система линейных уравнений:

$$Ax = b \quad (1)$$

Здесь A – разреженная симметричная положительно определенная матрица, b– плотныйвектор, x – вектор неизвестных. Необходимо найти решение системы x.

3. Метод решения

Прямые методы решения задачи (1), как правило, основаны на применении разложения Холецкого к матрице A в виде:

$$A = U^T U \quad (2)$$

где U – верхнетреугольная матрица. В этом случае решение системы сводится к последовательному решению двух треугольных систем:

$$U^T y = b, Ux = y \quad (3)$$

Особенностью процедуры разложения Холецкого для разреженной матрицы является то, что матрица обычно претерпевает заполнение, что на практике может привести к неудовлетворительным требованиям по памяти. Степень заполненности матрицы можно уменьшить с помощью переупорядочивания ее строк и столбцов. Это соответствует нахождению матрицы перестановки Pи переходу к эквивалентной системе (4):

$$\bar{A}(Px) = Pb, \bar{A} = PAP^T \quad (4)$$

Таким образом, при численном решении разреженной системы с использованием методаХолецкого можно выделить следующие этапы: переупорядочивание – вычисление матрицы перестановки P и переход к системе (4); символическое разложение – построение портрета матрицы U, выделение памяти для хранения ненулевых элементов; численное разложение – вычисление значений матрицы U и размещение их в выделенной памяти; обратный ход – решение треугольных систем уравнений (3).

4. Программная реализация

4.1 Базовый подход

На основе описанного вышеметода решения выполнена последовательная программная реализация решателя СЛАУ и ее модификация с применением супернодального подхода. Подготовленапараллельная версия для систем с общей памятью. Программная реализация выполнена на языке C. Для хранения разреженных матриц выбран формат CRS. Вычисления проводились в двойной точности. Рассмотрим подробнее реализацию каждого этапа метода:

Этап 1. Для оптимизации заполнения фактора был использован *метод вложенных сечений* (nesteddissection)[6], основанный на разбиении графа матрицы при помощи разделителей. По сравнению с методом минимальной степени, он позволяет достичь большей степени параллелизма на стадиях факторизации. Реализация метода вложенных сечений выполнена в соответствии с описанием, приведенномв [14]. Реализована модификация метода с применением алгоритма уменьшения размера разделителя, предложенного в [2].

Этап 2. Во время *символической фазы* подготавливаются структуры данных для факторизации. Задачамиэтого этапа являются вычисление шаблонарасположения ненулевых элементов и построение вспомогательных структур данных для численной фазы.Так, для нахождения числа ненулевых элементов в факторе и выполнения параллельных вычислений строится специальная структура – *дерево исключения*. Оно отражает зависимость между строками матрицы: если вершины дерева не соединены ребром, то соответствующие им строки фактора могут вычисляться параллельно. В рассматриваемой реализациипостроение дерева исключения выполняется по алгоритму, описанному в [13]. Кроме того, к переупорядоченной на этапе 1 матрице дополнительно применяется *постперестановка*на основеалгоритма, приведенного в [3]. Этот прием повышает структурированность портрета фактора, не ухудшая его заполненности, что позволяет ускорить численную фазу факторизации. Далее выполняется *оценка общего числане-*

нулевых элементов в факторе и выделяется память для массива столбцовых индексов по алгоритму, предложенномув [7]. Непосредственно *символическая факторизация* выполняется на основе процедуры, описанной в [15].
Этапы 3-4. Во время *численной фазы* выполняется нахождение значений элементов верхнего треугольника. Это самая трудоемкая часть факторизации, состоящая в последовательном рассмотрении всех строк исходной матрицы и проведении операции Гауссова исключения для всех ненулевых элементов, лежащих левее главной диагонали. Базовая версия численной части (этап 3), а также *обратный ход*(этап 4) реализованы в соответствии с работой [15].

4.2 Супернодальный подход

Недостатком базового подхода к разложению Холецкого является низкая производительность на матрицах больших размерностей из-за возникновения существенного количества кеш-промахов. Для решения этой проблемы существует два широко распространенных подхода: супернодальный (supernodal, «суперэлементный») и мультифронтальный (multifrontal).

*Супернодальный подход*для алгоритма факторизации использует так называемые «*суперноды*» (supernode) и позволяет производить факторизацию поблочно с применением матричных операций BLAS. Повышение производительности в этом случае можно получить путем применения оптимизированных реализаций BLAS для плотных подматриц, повышения эффективности работы с памятью, создания базы для распараллеливания. Подобный подход используется в таких решателях, как MKL Pardiso, CHOLMOD и др.

Мультифронтальный подход (авторы Дюфф и Рэйд [5]) состоит в разбиении всего процесса факторизации на факторизацию небольших плотных матриц [11, 12]. При этом используются те же механизмы повышения производительности. Одним из наиболее известных решателей, реализующих данный подход, является MUMPS.

В данной работе для модификации численной части использовался супернодальный подход. Для выделения блоков применялись так называемые *ослабленные суперноды* (relaxedsupernode [1]) – группы строк, имеющих различие в структуре левее плотного треугольного блока не более чем в фиксированном числе элементов. *Выделение супернодов* выполняется *после символической части* на основе процедур, введенных в [1].При этом задается параметр огрубления, отвечающий за количество нулей, допускаемых в BCRS-блоке матрицы, что позволяет сформировать блоки большего размера и сократить время работы матричных операций. Однако сильное увеличение параметра может приводить к неудовлетворительным требованиям по памяти, его оптимальное значение может быть подобрано под конкретную задачу.

Модифицированная численная фаза при супернодальном подходе состоит из нескольких этапов, а именно: перевод исходной матрицы в формат BCRS на основе выделенных супернодов; выполнение факторизации с использованием матричных операций (BLAS 3 уровня).Для устранения необходимости перевода результата факторизации в формат CRS реализован алгоритм обратного хода метода Гаусса для формата BCRS. Реализация указанных алгоритмовоснована наработах [4], [8-10]. Использовались функцииBLASиз библиотеки IntelMKL.

4.3 Параллельная версия для систем с общей памятью

Авторами выполнена реализация параллельной версии решателя для систем с общей памятью на основе супернодального подхода. На данном этапе работы распараллелена численная фаза разложения Холецкого, как наиболее трудоемкий этап вычислений.

Схема параллельных вычислений формировалась на основе модификации дерева исключения, построенного по BCRS-портрету матрицы. В этом случае дерево исключений показывает, над какими группами строк операции могут выполняться независимо. Для минимизации накладных расходов при организации паралеллизма и равномерной загрузки потоков по дереву исключения строится так называемое дерево распараллеливания. При этомобъединеняются некоторые вершины дерева исключения, принадлежащие одному и тому же поддереву. Вычисления, соответствующие отдельному узлу, могут выполняться независимо.

Распараллеливаниевыполнено на основе потоков Windowsи шаблона параллельного программирования «мастер – рабочий». Назначение задач «потокам-рабочим» происходит согласс-

но дереву распараллеливания. Обработка вершин дерева начинается с листьев. Очередная вершина становится доступной для независимых вычислений и назначается «потоку-рабочему» после завершения вычислений для ее потомков.

5. Результаты экспериментов

Для анализа производительности программного комплекса был проведен ряд экспериментов на матрицах из коллекции [16] университета Флориды. Характеристики тестовых матриц представлены в таблице ниже (Таблица 1). Все они являются симметричными положительно определенными.

Таблица 1. Характеристики тестовых матриц

Название матрицы	Порядок	Число ненулевых элементов	Заполненность, %
pwtk	217918	5926171	0,0125
msdoor	415863	10328399	0,0060
parabolic_fem	525825	2 100225	0,0008
tmt_sym	726713	2903837	0,0005
G3_circuit	1585 478	4 623152	0,0002
ecology2	999999	2997995	0,0003
audikw_1	943695	39297771	0,0044

Параметры тестовой инфраструктуры приведены в следующей таблице (Таблица 2):

Таблица 2. Параметры тестового окружения

Процессор	2 четырехъядерных процессора Intel® Xeon E5520 (2.27 GHz)
Память	16 GB
Операционная система	Windows 7
Среда разработки	MicrosoftVisualStudio 2008
Компилятор, библиотеки	Intel® ParallelStudio XE 2011

Были проведеныэксперименты для базовой и супернодальной последовательной версии. Для сокращения времени работы супернодальной реализации использованы матричные операции BLAS библиотеки Intel® MKL, а также настройка параметра построения супернодов (параметра огрубления, описанного выше). Далее были проведены эксперименты для параллельной версии решателя с настройкой параметра алгоритма –числа поддеревьев.

Также были проведеныэксперименты на тех же тестовых матрицах с использованием библиотеки MKL PARDISO из пакета Intel® MKL, SuperLUVersion 4.1, MUMPSна той же аппаратной платформе под управлением операционной системыLinuxopensuse 11.2.

В дальнейшем для решателя авторов указано лучшее время работы из тестируемых модификаций. Для матриц parabolic_fem, ecology2, audikw_1 это базовая версия решателя, для матриц pwtk, msdoor, tmt_sym, G3_circuit – супернодальная версия. Для матрицы G3_circuit применялся модифицированный метод вложенных сечений, для остальных матриц – базовая версия алгоритма переупорядочивания.

Для пакета SuperLU приведено время работы при использовании одного из встроенных алгоритмов перестановки, дающегоменьшее время работы. Для матриц pwtk, parabolic_fem, tmt_sym, ecology2 это приближенный столбцовый метод минимальной степени (COLAMD), для матрицы msdoor – множественный метод минимальной степени (MMD). Отметим, что на матрице G3_circuit программа завершила работу из-за внутренней ошибки работы с памятью; на матрице audikw_1 – в связи с нехваткой ресурсов для хранения фактора.

Рассмотрим текущие результаты последовательной версии решателя в сравнении с последовательными версиями SuperLU, MUMPS и MKL PARDISO (Таблица 3).Как видно из таблицы ниже, последовательная версия решателя в основном показывает лучшие результаты, чем SuperLU, но отстает от MKL PARDISO и MUMPS.

Таблица 3. Сравнение работы последовательной версии решателя авторов с некоторыми сторонними решателями.

Матрица	Решатель авторов		SuperLU		MUMPS		MKL	
	Число элементов в факторе	t, сек	Число элементов в факторе	t, сек	Число элементов в факторе	t, сек	Число элементов в факторе	t, сек
pwtk	61 678 703	18,6	108 904 188	79,8	48 988 990	11,9	52 190 144	5,5
msdoor	180 142 589	91,9	111 245 898	80,5	54 793 824	11,8	57 478 740	6,1
parabolic_fem	26 872 825	9,7	52 361 272	37,6	28496994	8,7	28 125 024	6,5
tmt_sym	45 181 460	28,1	88 165 169	76,9	36156598	17,2	33 075 548	8,7
ecology2	33 203 232	56,0	68 677 613	38,0	32956605	12,6	38 516 392	10,4
G3_circuit	213 111 677	468,4	Ошибка		102828595	76,8	104 692 916	25,5
audikw_1	2006 889 336	20 273,0	Ошибка		1 270 735 946	2856,5	1 270 292 396	793,0

Рассмотрим текущие результаты параллельной версии решателя в сравнении с параллельными версиями SuperLU и MKL PARDISO при работе в 8 потоков (Таблица 4). Для решателя авторов отдельно приведено ускорение численной фазы разложения. Как видно из таблицы, параллельная версия решателя авторов в целом несколько уступаетSuperLUисущественно отстает от MKLPARDISO.

Таблица 4. Сравнение работы параллельной версии решателя авторов с некоторыми сторонними решателями.

Матрица	Решатель авторов				SuperLU			MKL		
	Число элементов в факторе	t, сек	Ускорение численной фазы	Ускорение общее	Число элементов в факторе	t, сек	Ускорение	Число элементов в факторе	t, сек	Ускорение
pwtk	61 678 703	9,14	2,8	2,0	109 716 912	13,6	5,9	51 050 753	1,5	3,8
msdoor	180 142 589	55,69	1,8	1,7	111 395 559	18,7	4,3	57 096 124	1,9	3,2
parabolic_fem	26 872 825	6,27	3,8	1,8	54 270 042	7,4	5,1	27 941 564	2,5	2,6
tmt_sym	45 181 460	15,82	3,9	1,8	88 177 959	14,8	5,2	32 816 445	3,5	2,5
ecology2	33 203 232	56,09	4,4	1,2	68 688 337	8,0	4,7	38 876 742	4,5	2,3
G3_circuit	213 111 677	303,82	2,8	1,5	Ошибка			102 828 595	8,7	2,9

Анализ результатов экспериментов позволяет сделать следующие выводы:

- Время работы определяется стадиями переупорядочивания и численной фазой разложения Холецкого.
- Решатели, существенно обгоняющие по скорости работы реализацию авторов, используют в работе для переупорядочивания библиотеку METIS – мировой лидер в указанной области на протяжении многих последних лет.В нашей вычислительной схеме применяется собственная реализация метода вложенных сечений. На всех тестовых примерах заполнение фактора лучше, чем у SuperLU, но хуже, чем у MKL PARDISO.
- По сравнению с MUMPS, решатель авторов показывает незначительное отставание на матрицах размером до миллиона (parabolic_fem, tmt_sym, pwtk). На матрицах большего размера отставание возрастает, но не превышает одного порядка.
- Отставание решателя по отношению к MKL колеблется в зависимости от размера матрицы и ее заполнения. Большее отставание на ряде матриц (msdoor, G3_circuit, audikw_1) вызвано прежде всего тем, что MKL PARDISO имеет лучшую реализацию переупорядочивателя (METIS), а также более эффективно работает с памятью.

6. Заключение

На данный момент авторами реализована базовая и супернодальнаяпоследовательные версии решателя и ряд их модификаций. Также разработана базовая параллельная супернодальная версия решателя.Анализ результатов вычислительных экспериментов показал, что последовательная версия решателя авторов на большинстве тестовых задач опережает по скорости решатель SuperLU, но проигрывает решателям MUMPS и Intel MKL PARDISO на матрицах порядка $10^5 – 10^6$. Величина отставания зависит от задачи идля некоторых матриц составляет 1,5-5 раз, что наряду со сравнением с SuperLU подтверждает качество текущей реализации и создает предпосылки ее дальнейшего развития. Результаты работы используются в учебном процессе факультета ВМК в рамках курса «Параллельные численные методы».

Целью дальнейшего исследования является повышение производительности решателя. Основные направления работы:

- Реализация многоуровневого параллельного переупорядочивателя на базе метода вложенных сечений с возможным подключением алгоритма минимальной степени на некотором шаге метода.
- Оптимизация последовательной и параллельной версии численной фазы разложения Холецкого.

Литература

1. Ashcraft C., Grimes R. The influence of relaxed supernode partitions on the multifrontal method // ACM Trans. Math. Software. – 1989. – Vol. 15, No. 4. – P. 291-309.
2. Aschcraft C., Liu J.W.H. A partition improvement algorithm for generalized nested dissection // Technical Report BCSTECH-92-020 Boeing computer Services. – 1994.
3. Davis T.A. Direct methods for sparse linear systems. – Philadelphia: SIAM, 2006. – 217 pp.
4. Demmel J.W., Eisenstat S.C., Gilbert J.R., Li X.S., Liu J.W.H. A supernodal approach to sparse partial pivoting // SIAM J. Matrix Anal. Appl. – 1999. – Vol. 20, No. 3. – P. 720-755.
5. Duff I. S., Reid J. K. The multifrontal solution of indefinite sparse symmetric linear equations // ACM Bans. Math. Software, 9 (1983). – P. 302-325.
6. George A., Liu J.W.H. An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems // SIAM J. on Numerical Analysis. – 1978. – Vol. 15, No. 5. – P. 1053–1069.
7. Gilbert J.R., Ng E.G., Peyton B.W. An efficient algorithm to compute row and column counts for sparse Cholesky factorization // SIAM J. Matrix Anal. Appl. – 1994. – Vol. 15, No. 4. – P. 1075-1091.
8. Hogg J.D. Efficient sparse Cholesky factorization – [http://www.maths.ed.ac.uk/~s0455378/EfficientCholesky.pdf].
9. Hogg J.D. Elimination Trees and Up-/Down-dating of Sparse Cholesky Factorizations – [http://www.maths.ed.ac.uk/~s0455378/ETreesUpDown.pdf].
10. Hogg J.D., Reid J.K., Scott J.A. A DAG-based Sparse Cholesky Solver for Multicore Architectures // Tech. report RAL-TR-2009-004, Rutherford Appleton Laboratory. – 2009.
11. Liu J.W.H. The multifrontal method and paging in sparse Cholesky factorization // ACM Trans. Math. Softw. – 1989. – P. 310-325.
12. Liu J. W. H. The multifrontal method for sparse matrix solution: Theory and practice // SIAM Review. – 1992. Vol. 34. – P. 82-109.
13. Liu. J. W. H. The role of elimination trees in sparse factorization // Matrix Anal Appl. – 1990. – P. 134-172.
14. Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. – М.: Мир, 1984.
15. Писсанецки С. Технология разреженных матриц. – М.: Мир,1988.
16. The University of Florida Sparse Matrix Collection –[www.cise.ufl.edu/research/sparse/matrices/].

Система удаленного доступа к Грид-инфраструктуре экспериментов на БАК как инструментальная платформа PaaS для разработки геоприложений космического мониторинга в системах дистанционного зондирования Земли

В.В. Кореньков, В.М. Котов, М.А. Минеев, Н.А. Русакович, А.В. Яковлев

Объединенный институт ядерных исследований

Решение проблемы эффективного и устойчивого функционирования распределённых систем дистанционного зондирования Земли (ДЗЗ), динамика сопровождения и дальнейшего их развития делает невозможным решить эту задачу в “статике” один раз и навсегда. Распределённость системы, изменение условий (числа узлов и каналов связи, их характеристик) существенно влияют на правила и алгоритмы управления системы. Проблема оптимизации распределения ресурсов, обеспечения устойчивости и эффективности системы определяет необходимость разработки инструментальной системы PaaS, как составляющей программного обеспечения в современных системах обработки для отслеживания изменений характеристик системы и требований пользователей на всём жизненном цикле системы.

1. Введение

Сбор, предварительная обработка и анализ данных, полученных при дистанционном зондировании Земли (ДЗЗ) с помощью космических радиолокаторов с синтезированной апертурой (РСА), связаны с накоплением и обработкой информации, объемы которой в сотни тысяч раз превышают средний уровень потребностей и достигают десятков Tb в год.

Европейское Космическое Агентство (European Space Agency (ESA)) начиная с 90-х годов прошлого столетия поставляет пользователям данные космических аппаратов, имеющих РСА (Envisat, Radarsat, TerraSar, Cosmo-Skymed), и уже собран большой архив данных.

В настоящее время и в России в рамках Федеральной космической программы ведется разработка космических аппаратов («Метеор-М №3», «Аркон-2М»), оснащенных многофункциональной РСА с активной фазированной антенной решеткой, характеристики которой соответствуют характеристикам современных РСА [1].

Традиционные подходы к разработке инфраструктуры и обработки таких объемов данных (получивших название Большие данные) не годятся, необходимы новые подходы к обработке и анализу данных для современных РСА, в том числе и реализация интерактивного взаимодействия в режиме удаленного доступа для многоуровневой, распределенной географически вычислительной системе обработки данных больших объемов.

Аналогичная проблема стояла и в области экспериментальной физики высоких энергий еще в 90-е годы прошлого столетия при формировании программы исследований крупнейшего проекта современности в области фундаментальной науки: созданию Большого адронного коллайдера (БАК) в Европейском центре ядерных исследований (ЦЕРН). В результате многолетней работы большого коллектива в ЦЕРНе была разработана, отлажена, запущена и успешно работает второй год в режиме реальных измерений на пучках ускорителя БАК система сбора и обработки данных эксперимента ATLAS, объединяющая в единую систему все компоненты обработки, управления и передачи Больших данных (это передача и фильтрация в реальном времени 100Gb/s и offline обработка и анализ экспериментальных данных объемом в десятки Pb в год) [2].

ОИЯИ является участником программы исследований на БАК. Важной составляющей такого участия в обработке и анализе данных, получаемых в экспериментах на БАК является создание в ОИЯИ системы удалённого доступа реального времени (СУДРВ) и интеграция ее в глобальную сервис-ориентированную архитектуру Грид-системы сбора и обработки данных экспериментов на БАК. [3]

2. Концепция

Опыт разработки и реализации системы сбора и обработки данных экспериментов на БАК может быть использован при создании систем обработки данных в системах ДЗЗ, и в частности для данных, полученных с помощью космических радиолокаторов с синтезированной апертурой.

На сегодня не существует технологии, обеспечивающей необходимую функциональность и эффективность обработки таких потоков и объемов экспериментальных данных в одной системе. Поэтому система сбора и обработки эксперимента ATLAS была разделена на два уровня: online - сбор и предварительная обработка и offline - полная обработка и анализ.

Offline обработка требует необычно больших вычислительных ресурсов (примерно 100000 самых мощных на сегодня процессоров или 150 Kcores) и была реализована в Грид - системе (рис.1).

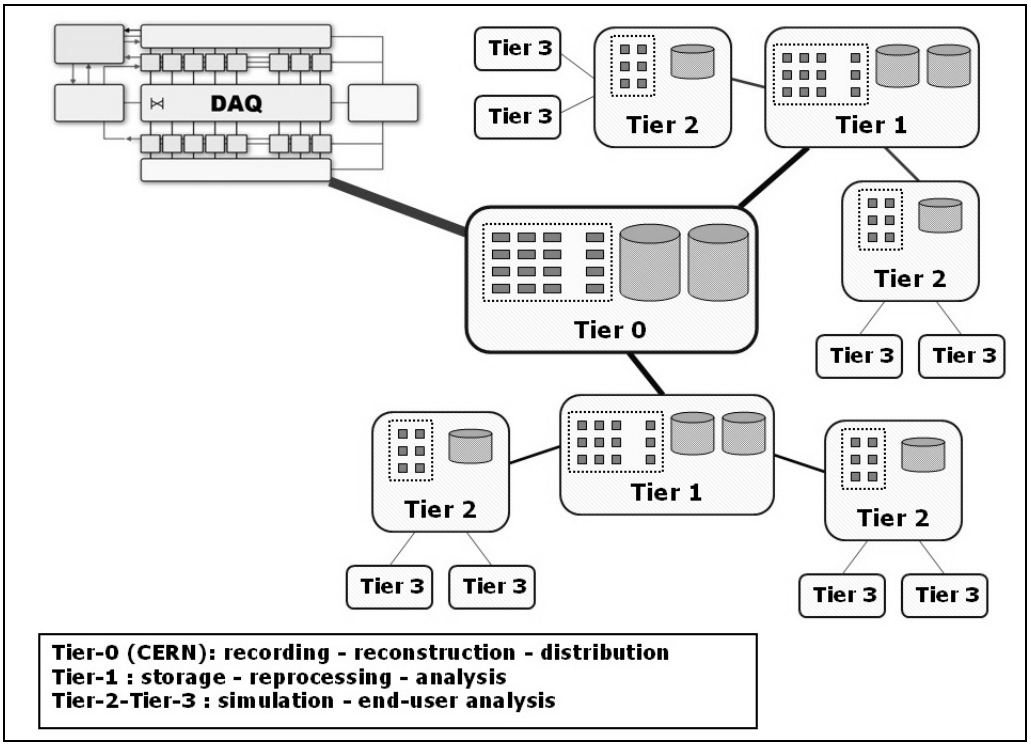


Рис. 1. Система Грид обработки

Показанная в левом верхнем углу на рис. 1. система DAQ и есть online составляющая системы по сбору и предварительной обработке входных данных.

Современная архитектура наземной географически распределенной системы обработки данных SAR (Synthetic Aperture Radar), предлагаемая Европейским космическим агентством, также имеет online уровень обработки входных данных.

Необходимость первичной обработки радиолокационных данных определяется особенностью форматов и структурой данных SAR, требующих предварительной обработки для выполнения в последующем полного цикла обработки информации с SAR. Кроме того, структура радарных данных позволяет проводить эффективную предварительную обработку на уровне первичной обработки радиолокационных изображений, оперируя изображением в целом как образом, для быстрого опознания и анализа объектов и их характеристик.

Радарные данные имеют целый ряд особенностей: сложность обработки из-за геометрических искажений, а также непростая интерпретация изображений. Состав функций предварительной обработки включает следующие возможности обработки данных: фокусировка, корегистрация, удаление спекл-шумов, извлечение характеристик (включая когерентность), геокодирование, радиометрическую калибровку и нормализацию, составление мозаики и классификации.

ESA в октябре 2007 г. заключило договор с канадской фирмой Array Systems Computing на разработку инструментального программного обеспечения и с 2010 г. поставляет комплект программного обеспечения с открытым исходным кодом NEST (Next ESA SAR Toolbox) [4], включающего в себя функциональность всех предыдущих версий инструментального ПО BEST, BEAM и др. Состав пакета NEST, компоненты и потоки данных приведены на рис. 2.

По условиям технического задания ESA инструментарий NEST предназначен для помощи в подготовке элементов системы обработки данных SAR, но не является SAR процессором или готовой системой обработки SAR данных в полном технологическом цикле.

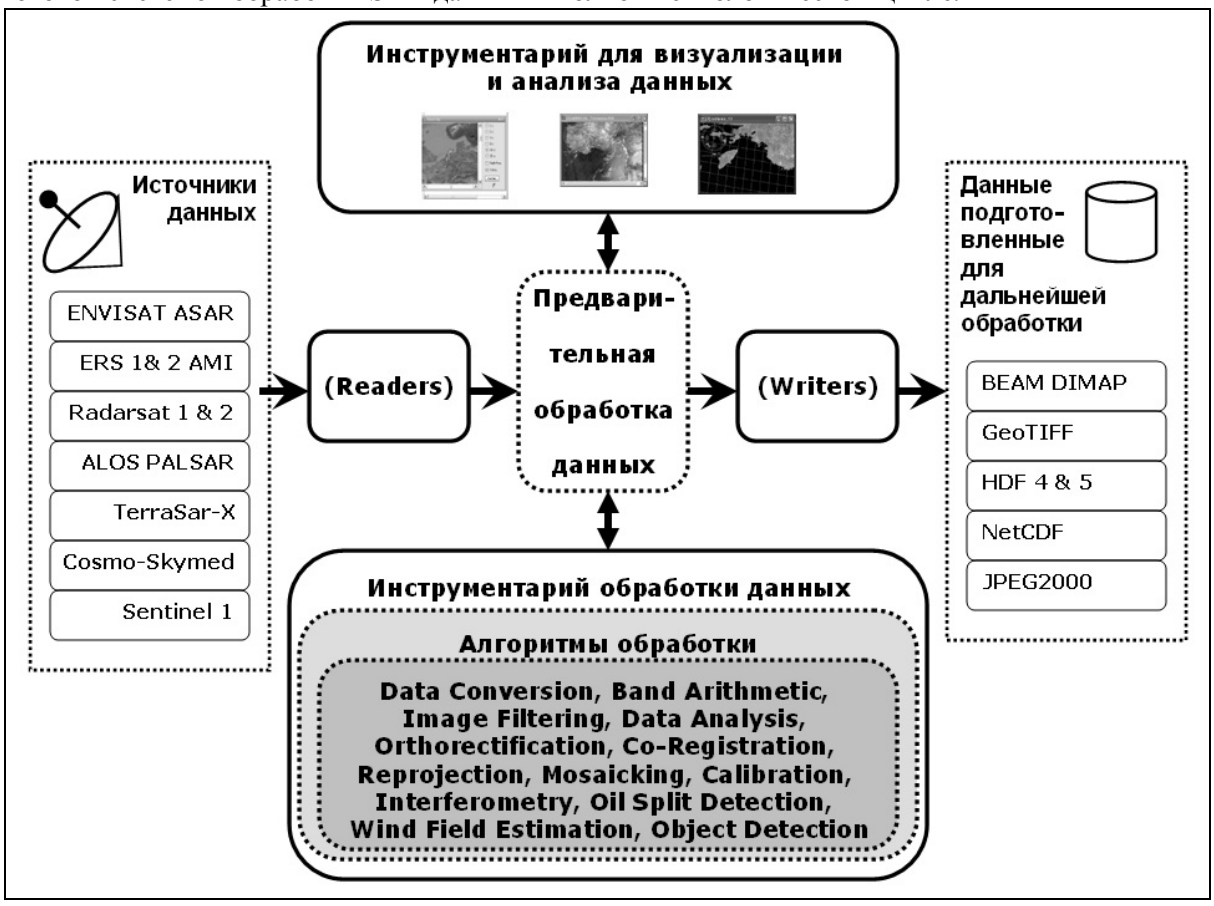


Рис. 2. Компоненты и потоки данных программного комплекса NEST

Вместе с тем, статус пакета NEST как программы с открытым исходным кодом позволяет использовать его в составе PaaS (Platform as a Service) для разработки, тестирования, развертывания и поддержки полномасштабной системы сбора и обработки данных с космических локаторов РСА, реализующей полный технологический цикл обработки радарных данных. Предполагается реализовать набор инструментов для отображения и частичной обработки радарных снимков в среде ArcGIS.

В качестве основы для разработки прототипа подобной PaaS предлагается использовать систему удаленного доступа реального времени (СУДРВ) ОИЯИ. СУДРВ представляет собой композитный сервис Грид-системы и является фрагментом общей системы обработки данных эксперимента ATLAS БАК [5]. Согласно планам развития информационной инфраструктуры экспериментов БАК ЦЕРН и в соответствии с концепцией «облачных вычислений», где все есть Сервис (XaaS), СУДРВ ОИЯИ будет применяться как PaaS для дальнейшего развития системы обработки данных эксперимента ATLAS на БАК.

Кроме того, в соответствии с концепцией «Открытой инновационной лаборатории» эксперимента ATLAS-LAB (ATLAB), на рабочем совещании ЦЕРН-ОИЯИ «Brainstorming workshop on applications from ATLAS using EU-funding for R&D-upgrades» в г. Дубне 24.10.2010 г по обсуждению доклада ОИЯИ «Real Time remote access system for ATLAS» было поддержано предложение о возможности прикладного использования СУДРВ ОИЯИ в области космического мониторинга, проводимых при поддержки ESA совместно с ЦЕРН.

Структура программного обеспечения СУДРВ и NEST использует объектно-ориентированный подход проектирования и соответствует стандарту PSS05 ESA. Объединение NEST и СУДРВ в единую платформу обеспечит интеграцию NEST в общую систему Грид-обработки данных экспериментов БАК, а значит и возможность отладки в последующем и offline режима обработки данных космического мониторинга в географически распределенной вычислительной системе Грид-обработки и партнерство с участием ESA и ЦЕРН.

Следует отметить, что кроме инструментария NEST, отражающего специфику обработки радарных данных, предлагаемая платформа PaaS будет обеспечивать также доступ к сервисам сбора и обработки данных, необходимых для функционирования NEST в составе СУДРВ.

В качестве базового ядра предлагаемой системы удаленного доступа для сбора и обработки космической радиолокационной информации предполагается использовать набор компонентов из Системы сбора и обработки данных в реальном времени эксперимента ATLAS LHC (ATLAS TDAQ) [6].

Базовое программное обеспечение имеет два различных логических представления: представление в виде подсистем и представление в виде уровней ПО.

Разделение на подсистемы произведено в соответствии с основными функциональными возможностями, которые базовое программное обеспечение предоставляет внешним пользователям. Каждая подсистема разделена в свою очередь на пакеты, которые отвечают за отдельные аспекты, соответствующие функциональным возможностям. Каждый пакет реализует функциональные возможности, определенные в API, и предоставляет интерфейс для всех типов пользователей. Основные подсистемы базового программного обеспечения:

Конфигурация - Пакеты подсистемы Конфигурация содержат описание системной конфигурации и обеспечивают доступ для записи информации в конфигурационную базу данных о ходе работы во время сбора и обработки данных..

Управление - содержит пакеты для управления всеми процессами в системе, обеспечивает инициализацию и выключение системы, передачу управляющих команд для запуска и останова процессов внутри системы.

Мониторинг - содержит классы для поддержки мониторинга в системе. Классы системы Мониторинг сообщают о возникающих ошибках, для опубликования информации о состоянии систем, для передачи гистограмм, данных в процессе их сбора.

Система разделена на 3 уровня:

Уровень интерфейсов - это классы API предоставляющие интерфейсы для пользователей базового программного обеспечения.

Уровень сервисов – содержит сервисы и интерфейсы необходимые для их вызова.

Уровень промежуточного программного обеспечения – это пакеты для внутреннего использования другими пакетами. Пакеты промежуточного программного обеспечения не принадлежат какой-либо подсистеме.

Большая часть компонентов базового программного обеспечения является сервисами, которые используются другими системами. Взаимосвязь этих пакетов и подсистем показана на рис 3.

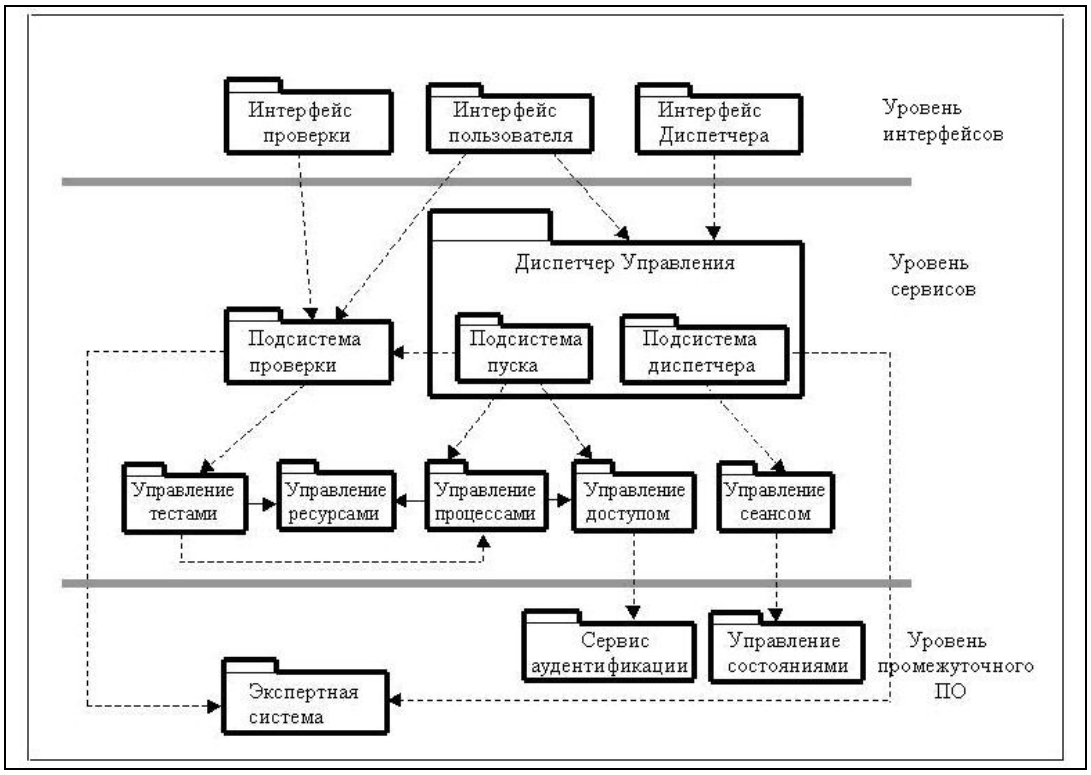


Рис. 3. Компоненты базового программного обеспечения и отношения между компонентами каждого уровня

В состав компонентов программного обеспечения PaaS входят также подсистемы: проверки управления тестами, сервис аутентификации и набор сервисов по интеграции с Грид-системой обработки данных экспериментов на БАК.

3. Заключение

Интеграция инструментального пакета NEST ESA и системы удаленного доступа СУДРВ ОИЯИ позволит создать развитую платформу (PaaS) для разработки прототипов (макетов) систем предварительной обработки радарных данных космических локаторов с синтезированной апертурой высокого разрешения и отладки их в условиях реальной работы в большой географически распределенной вычислительной Грид-системе обработки данных экспериментов на БАК..

Литература

1. Костюк Е.А., Веремчук Ю.А., Денисов П.В. Перспективные технологии обработки космической радиолокационной информации в НКПОР Оператора КС ДЗЗ // V Международная конференци «Космическая съемка — на пике высоких технологий».
2. Mapelli L. Spanning from Data Acquisition to GRID - Today and a view of tomorrow. // XXIII International Symposium on Nuclear Electronics & Computing NEC’2011.
3. В.В. Кореньков, В.М. Котов, Н.А. Русакович, А.В. Яковлев. Система удаленного доступа реального времени (СУДРВ), как композитный сервис распределенной ГРИД-системы обработки данных экспериментов на Большом Адронном Коллайдере (БАК) // Параллельные вычислительные технологии (ПаВТ’2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2010. – с. 668.
4. Software Architecture Document (SAD) for the Next ESA SAR Toolbox (NEST) (ARR-NEST-RS07-016); http://www.array.ca/nest/Software_Architecture_Document_v2.0.pdf

5. В.В. Кореньков, В.М. Котов, Н.А. Русакович, А.В. Яковлев. Модель и технология интеграции online-сервисов эксперимента ATLAS на Большом Адронном Коллайдере (БАК) и сервисов ГРИД-инфраструктуры // Параллельные вычислительные технологии (ПаВТ'2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2011. - с. 516–521
6. ATLAS High-Level Trigger, Data Acquisition and Controls. Technical Design Report (ATLAS TDR-016); http://atlas-proj-hltdaqdcs-tdr.web.cern.ch/atlas-proj-hltdaqdcs-tdr/tdr-v1-r4/PDF/TDR.pdf

Опыт портирования среды для HDR-обработки изображений на GPU и APU

М.А. Кривов[1], М.Н. Притула[2], С.Г. Елизаров[3]

ООО «ТТГ Лабс»[1], ИПМ им. М.В. Келдыша РАН[2],
ООО «ЦИФ МГУ им. М.В. Ломоносова»[3]

В статье описаны проблемы, с которыми столкнулись авторы при переносе открытого пакета LuminanceHDR на современные архитектуры с графическими ускорителями. Результатом переноса стали две версии пакета, разработанные с использованием технологий NVIDIA CUDA и OpenCL и отдельно оптимизированные для использования на GPU от NVIDIA и новых APU от AMD. На примере рассмотренной задачи проводится сравнение как методов оптимизации под соответствующие технологии, так и аппаратных решений от NVIDIA и AMD.

1. Введение

Одной из областей, в которой крайне востребованы современные подходы к параллельной организации вычислений, является обработка изображений. Благодаря тому факту, что практически во всех алгоритмах каждый пиксель изображения обрабатывается независимо, большинство программ из этой области обладают огромным потенциальным параллелизмом, что делает возможным их оптимизацию с применением различных параллельных вычислительных технологий, будь то использование векторных расширений x86-совместимых процессоров, задействование всех ядер SMP-систем, или же применение графических ускорителей.

В связи с тем, что разрешение матриц цифровых фотоаппаратов постоянно растет, поддержка разнообразных возможностей новых многоядерных процессоров и ускорителей становится крайне актуальной для различных графических пакетов типа Adobe Photoshop, GIMP и прочих. Причём если раньше параллельные реализации алгоритмов обработки изображений были просто ещё одним конкурентным преимуществом, то сейчас поддержка параллельных вычислителей становится жизненной необходимостью. Ведь в противном случае пользователи будут постоянно сталкиваться с ситуациями, когда обработка одной фотографии будет занимать от десятков секунд до нескольких минут.

Крупные компании вроде Adobe и Autodesk постоянно совершенствуют свои решения, распараллеливая используемые алгоритмы и внедряя новые технологии типа NVIDIA CUDA. Однако у разработчиков бесплатных графических пакетов, распространяемых под лицензиями типа GPL, обычно отсутствуют ресурсы и возможности для проведения качественной оптимизации своих программ, в результате чего пользователям приходится либо мириться с низкой скоростью их работы даже на современных процессорах, либо использовать различные ухищрения, жертвуя качеством обработки ради повышения производительности.

Одним из подобных открытых пакетов является среда LuminanceHDR [1], разрабатываемая Рафаэлем Мантиуком и содержащая набор алгоритмов для HDR-обработки изображений, суть которой заключается в изменении локального контраста в целях улучшения восприятия фотографии человеком. Благодаря качественным алгоритмам данный пакет завоевал определённую популярность среди фотографов, однако скорость его работы является предметом постоянной критики — обработка одной 12-мегапиксельной фотографии может занять несколько минут. При этом для получения качественного результата требуется как минимум несколько попыток подбора нужных параметров алгоритма, в результате чего обработка только одной фотографии длится десятки минут.

Компании ООО «ТТГ Лабс» было предложено провести оптимизацию данного пакета путём портирования на графические ускорители одного из самых популярных алгоритмов среды LuminanceHDR под названием Mantiuk06. В результате были созданы CUDA- и OpenCL-

версии требуемого алгоритма, что позволило многократно повысить скорость его работы, а также обеспечить эффективное использование всех возможностей новых вычислительных архитектур, будь то GPU, многоядерный CPU или APU.

2. Описание оптимизируемого пакета

LuminanceHDR является полноценным графическим пакетом с пользовательским интерфейсом, в котором реализованы два этапа обработки изображений: (1) построение по нескольким одинаковым фотографиям с разной экспозицией одной HDR-фотографии и (2) изменение локальной контрастности в HDR-снимке с его последующим переводом в обычную фотографию.

Суть первого этапа заключается расширении диапазона цветов. В обычных форматах изображений на каждый цветовой канал (красный, зелёный и синий) отводится по 8 бит, в результате чего удаётся сохранить только 256 градаций для каждого цвета. Все значения, которые не попадают в этот диапазон (например, более яркие цвета) игнорируются, что приводит к уменьшению реалистичности фотографии, т.к. глаз человека отлично замечает недостающие оттенки.

Предложено два способа обойти это ограничение. Первый заключается в использовании профессиональных фотоаппаратов, способных сохранять фотографии в специальных форматах, поддерживающих 10-12 и даже большее число бит на канал. Второй, реализованный в пакете LuminanceHDR, сводится к использованию нескольких фотографий одного и того же объекта с разной выдержкой. Далее, с помощью специального алгоритма эти фотографии «складываются» в одну HDR-фотографию, в которой на каждый цветовой канал выделяется по 32 бита, интерпретируемых как число с плавающей точкой одинарной точности. При этом качество результирующей фотографии уже целиком зависит от мастерства фотографа, подбирающего исходные снимки с подходящими значениями выдержки.

Второй этап обработки, осуществляемой в среде LuminanceHDR, состоит из обратного перевода HDR-фотографии в обычный формат, где на каждый цветовой канал снова приходится по 8 бит. Очевидно, что современные мониторы и дисплеи портативных устройств технически не в состоянии отобразить полноценное HDR-изображение, поэтому для дальнейшей его визуализации требуется сузить диапазон цветов, при этом минимизировав потери информации. Для этого существует множество алгоритмов, известных под общим названием Tone Mapping, которые изменяют локальный контраст отдельных участков фотографии, делая их более светлыми или тёмными в зависимости от изображённых на них объектах. Получившаяся в результате фотография хоть и станет менее правдоподобной, чем набор исходных снимков, но зато будет намного лучше воспринимается человеком, так как в ней не потеряна цветовая информация об объектах.

Для выполнения обратного перевода используются различные алгоритмы, каждый из которых имеет свои плюсы и минусы и лучше подходит для конкретных типов фотографий (города, пейзажи, фотопортреты и т.д.). В пакете LuminanceHDR реализовано девять подобных алгоритмов, наиболее популярным из которых стал метод Mantiuk06 [2], предложенный основным разработчиком пакета.

Возвращаясь к вопросам производительности, стоит отметить, что самым долгим является именно второй этап, в котором не только требуется выполнить множество операций над числами с плавающей точкой, но и подобрать правильные параметры алгоритма, что выливается в необходимость выполнения однотипной обработки несколько десятков раз. Поэтому было решено провести оптимизацию именно этого этапа как основного узкого места пакета.

Для иллюстрации базовых идей, заложенных в алгоритм Mantiuk06, следует описать схему его работы, которая состоит из трех стадий:

1) Выделение контраста из исходного изображения. С этой целью для каждого пикселя HDR-изображения вычисляется яркость, а под контрастом понимается логарифмическое отношение яркостей двух соседних пикселей. Другими словами, для каждой точки исходного изображения генерируется N вспомогательных точек контраста, где число N определяется исходя из количества рассматриваемых точек-соседей. В зависимости от параметров обработки

N может варьироваться от 2 до 20-30 [2].

2) Обработка контраста. Данная стадия является ключевым моментом описываемого алгоритма, и может быть по разному реализован в зависимости от требуемого вида обработки изображения. В оптимизируемом пакете на этой стадии вычисляется градиент от контраста, после чего полученное значение подставляется в некоторую линейную функцию, трактуемую в дальнейшем как градиент от нового контраста. Последующее численное интегрирование полученных значений позволяет восстановить новый контраст, который будет лучше восприниматься глазом человека.

3) Наложение обновлённого контраста на исходное изображение. Последним этапом является обновление исходного изображения с использованием нового контраста. Фактически такое обновление сводится к обратной задаче (по известным соотношениям яркости соседних точек необходимо восстановить исходное изображение), которая в общем случае не имеет единственного решения. В рассматриваемом алгоритме для выполнения этого шага строится специальный функционал, определяющий погрешность между некоторым произвольным и целевым изображениями, после чего с помощью итерационного метода подбирается такое изображение, функционал погрешности от которого не превышает 10^{-3}.

Легко заметить, что на всех стадиях алгоритма Mantiuk06 активно используются численные операции над большими массивами данных. Например, при обработке 10-мегапиксельной фотографии потребуется численное интегрирование дискретных функций, заданных массивами размера более 3000x3000, а итерационный алгоритм минимизации функционала погрешности сведется к построению пирамид изображений, для каждого уровня которого будут применяться такие операции, как дивергенция и градиент.

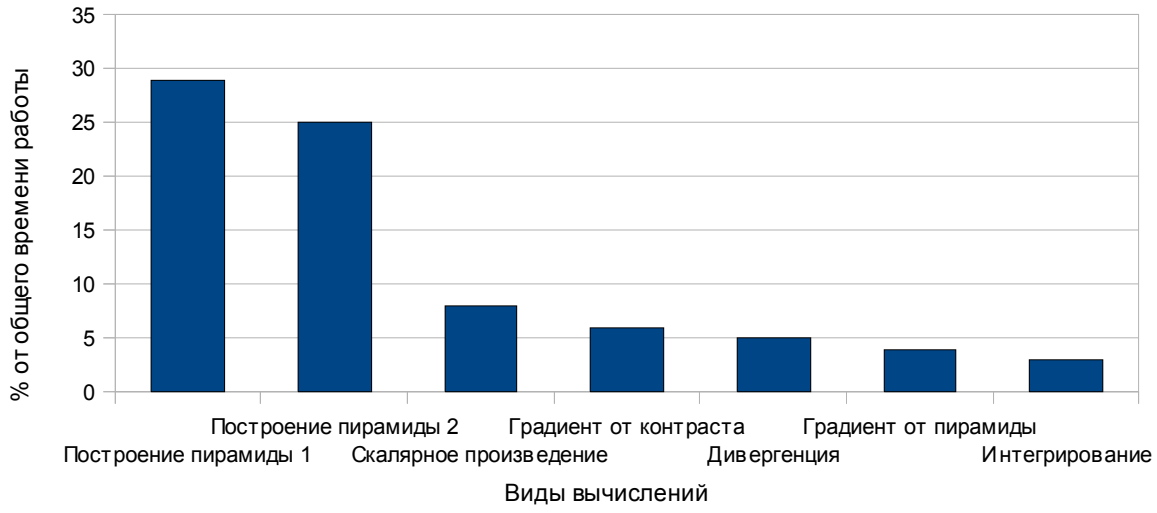


Рис. 1. Распределение времени работы алгоритма Mantiuk06 по основным операциям.

На рис. 1 приведена диаграмма распределения времени работы алгоритма по основным вычислительноёмким операциям, на которые в сумме приходится уходит 85% от общего времени работы программы. Наиболее ресурсоёмким оказался этап восстановления изображения по обновлённому контрасту — он отнимает более половины всего времени счета. Сложность остальных этапов практически равномерно распределена по оставшимся 35%, поэтому при портировании данного алгоритма на графические ускорители необходимо разработать GPU-версии для всех этапов обработки, так как в противном случае (при переносе на GPU только узких мест) потребуются частые пересылки больших объемов данных между CPU и GPU.

3. Проведённые оптимизации

Все изменения, которые были внесены в исходную реализацию алгоритма, логически можно разделить на две категории: (1) проведение общих модификаций, призванных повысить степень параллелизма и уменьшить количество вычислений без учёта особенностей

используемых технологий, и (2) разработка отдельных версий алгоритма с использованием технологий NVIDIA CUDA и OpenCL с последующей оптимизацией под конкретный вычислитель.

Если рассматривать первый тип изменений, то сразу стоит оговориться, что необходимость их проведения была вызвана недостаточным качеством исходного кода, что легко объяснить как бета-версией пакета, так и большим объёмом работ, выполняемых основным разработчиком LuminanceHDR, в результате чего на рефакторинг программы, скорее всего, просто не хватило времени. Основные проведённые модификации:

– Оптимизация работы с памятью. В исходном пакете активно использовались функции динамического выделения и освобождения памяти, в результате чего при обработке только одной фотографии блоки памяти размером в десятки мегабайт создавались и уничтожались в общей сложности более 10 000 раз. В принципе, для последовательной программы это не является особым минусом, так как объём системной памяти достаточно велик, а при её недостатке будет задействован файл подкачки. Однако при использовании графического ускорителя это может привести к нехватке памяти, в результате чего программа будет вынуждена аварийно завершиться. Особенно это важно при использовании современных APU, у которых объём доступной памяти ограничен 128 — 1024 Мбайт.

Авторами создан собственный менеджер памяти, который в начале обработки создаёт набор буферов, используемых в дальнейшем вместо динамически выделяемой памяти, а также переписан ряд функций с целью реализации более оперативного освобождения используемых буферов. Это позволило существенно уменьшить потребление памяти: так, для обработки фотографии размером 1600x1200 теперь достаточно всего 40 блоков памяти. Одновременно удалось избежать утечки памяти, которая наблюдалась в исходной реализации.

– Изменение пропорций пирамид. Как уже отмечалось, в алгоритме Mantiuk06 активно используется построение пирамид изображений, в которых каждый уровень в четыре раза меньше предыдущего. Основой подобных пирамид является исходное изображение, поэтому для некоторых форматов фотографий подобные пирамиды получались не выровненными — некоторые уровни имели нечётную длину или ширину. Как следствие, в коде во многих местах присутствовали проверки, призванные не допустить выход за пределы массивов при работе с такими уровнями, так как на краях массивов требовалась соответствующая логика работы.

При портировании на графический ускоритель подобные ветвления крайне не желательны, поскольку из-за специфики GPU будут выполнены все ветки каждого условного оператора, вследствие чего объём вычислений заметно возрастёт. Более того, из-за часто изменяющегося размера массивов не удастся реализовать выровненный доступ к памяти, что обернётся четырехкратной потерей производительности на ускорителях серии Tesla C1060 и двукратной – на Tesla C2050. Чтобы избежать этих проблем, было решено дополнять каждую обрабатываемую фотографию рамочкой, благодаря которой размер обрабатываемого изображения всегда позволит построить ровные пирамиды. К примеру, при обработке фотографии 500x375 на самом деле вычисления будут выполнены для изображения 512x384, которое потом будет обрезано до исходного формата.

– Оптимизация информационного графа программы. Отдельным видом оптимизации стал анализ информационного графа с целью последующего удаления избыточных обращений к памяти и вычислений. Благодаря этому удалось укрупнить некоторые циклы и функции, уменьшив число обращений к памяти и отказавшись от ряда вспомогательных переменных и промежуточных массивов. В дальнейшем при портировании алгоритма на графический ускоритель это позволило более активно использовать разделяемую/локальную память, так как после укрупнения функций во многих местах программы стало возможным поместить обрабатываемую область в кэш, и тем самым уменьшить количество обращений к глобальной памяти.

Ещё одним интересным моментом оказалось использование в исходном алгоритме быстрой сортировки, которую, как известно, на графическом ускорителе реализовать довольно сложно. При анализе графа выяснилось, что для работы требуются далеко не все значения отсортированного массива, а лишь некоторые из них, поэтому в GPU- и APU-версиях программы требуемую сортировку удалось заменить обычной параллельной свёрткой.

Проведённые модификации второго типа, по сравнению с только что описанными

являются, скорее, техническими, нежели концептуальными. В большинстве своём они сводились к опытному перебору нескольких альтернативных реализаций того или иного подхода и выбору оптимального варианта.

– Подбор оптимальной топологии данных. В зависимости от количества мультипроцессоров на графическом ускорителе требуется использовать различные разбиения данных на блоки, каждый из которых будет выполняться на отдельном мультипроцессоре. При этом не существует заранее известного оптимального размера блока, ведь этот параметр зависит от особенностей используемого GPU и размера обрабатываемых данных. Так, при крупном разбиении повышается скорость вычислений на отдельном мультипроцессоре, а при мелком гарантированно загружаются все ресурсы. Для дополнительного повышения скорости работы оптимальные размеры блоков были подобраны экспериментальным путём. Как показал ряд тестов, в среднем данный вид оптимизации позволил повысить скорость работы на 20-30% как в CUDA-, так и в OpenCL-версии.

– Использование разделяемой/локальной памяти. Одним из основных подходов к оптимизации программ для графических ускорителей является использование расположенной в GPU «быстрой» памяти, которую программист может использовать как буфер для временных результатов. Основная её особенность заключается в доступности записанных в неё значений для всех ядер одного мультипроцессора, что позволяет минимизировать количество чтений из глобальной памяти. В рассматриваемом алгоритме данный механизм оказался применим к четырём наиболее вычислительноёмким функциям. Так, при построении уровней пирамиды разделяемая/локальная память позволяет в четыре раза уменьшить число обращений к глобальной памяти, а в функциях вычисления градиента и дивергенции — в два раза.

– Использования аппаратной интерполяции. Поскольку графические ускорители долгое время разрабатывались исключительно для отображения 3D-объектов, то до сих пор даже в профессиональных вычислителях сохранился ряд атавизмов, изначально разработанных для растеризации геометрических объектов. Одним из таких атавизмов являются блоки аппаратной интерполяции, реализованные как возможность адресации массива дробными индексами. При подобном обращении к памяти результат будет являться линейной интерполяцией двух соседних элементов массива, причём данная операция будет выполнена за один такт. В ходе оптимизации данные блоки были использованы для масштабирования исходного изображения, что позволило отказаться от достаточно долгой операции, просто заменив схему индексации исходного массива.

4. Оценка ускорения на GPU

Версия пакета, оптимизированная для графических ускорителей, тестировалась на двух системах на базе GPU NVIDIA GeForce 580 GTX и GPU NVIDIA GeForce 555M. Оба ускорителя основаны на архитектуре Fermi, одной из основных особенностей которой является наличие кэша L1/L2. В первом GPU содержится 16 мультипроцессоров, суммарно состоящих из 512 CUDA-ядер, что обеспечивает пиковую производительность в 1,5 TFlops. Второй ускоритель разработан для ноутбуков, поэтому обладает пониженной частотой. Содержащиеся в нём 144 CUDA-ядра позволяют достичь 340 GFlops при вычислениях с одинарной точностью. Данные ускорители находятся в разных ценовых категориях, что позволяет оценить ускорение, которые гарантированно получат пользователи при использовании как высокопроизводительных рабочих станций, так и обычных компьютеров или ноутбуков.

На каждой системе проводилась обработка набора эталонных изображений с использованием пяти различных реализаций алгоритма. Две реализации используют технологию NVIDIA CUDA (с разделяемой памятью и без неё), ещё две — технологию OpenCL (с локальной памятью и без неё), а последняя является слегка оптимизированной версией исходного кода, исполняемого на одном ядре центрального процессора и скомпилированного с использованием Visual C++ 2008.

Результаты тестирования системы на базе GPU NVIDIA GeForce 580GTX и CPU Intel Xeon E3-1230 приведены на рис. 2.

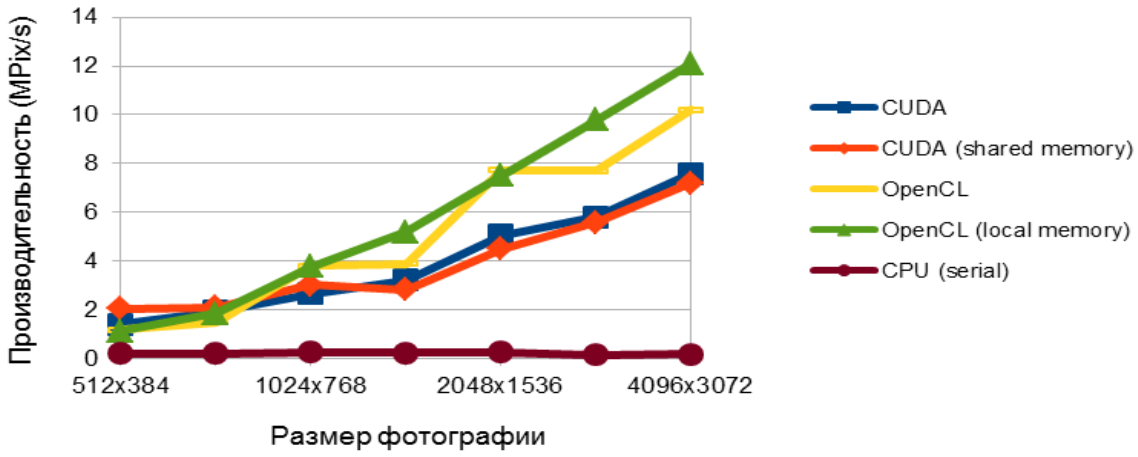


Рис. 2. Тестирование CUDA и OpenCL версий алгоритма на ускорителе NVIDIA GeForce 580 GTX.

Наиболее интересным моментом оказался рост производительности при увеличении размера обрабатываемой фотографии. Если для центрального процессора размер практически не сказывался на скорости работы, то графические ускорители обрабатывали большие изображения примерно в 10 раз быстрее, чем изображения небольших размеров. Это легко объяснить исходя из архитектурных особенностей GPU – чем больше фотография, тем больше порождается параллельных потоков, и тем более эффективно загружаются все вычислительные блоки.

Заслуживает отдельного комментария тот факт, что на данной задаче использование локальной/разделяемой памяти не всегда приводило к повышению производительности. Например, в технологии NVIDIA CUDA при обработке небольших фотографий программное кэширование обращений в глобальную память позволило получить 10-процентное ускорение. Однако на больших изображениях эта оптимизация лишь замедлила программу. В случае OpenCL-версии ситуация оказалась ещё более неоднозначной — в большинстве случаев локальная память давала заметное ускорение, но для некоторых форматов фотографий скорость обработки, наоборот, понижалась.

Также важно подчеркнуть, что оптимизация CUDA- и OpenCL-версий программы проводилась независимо и отчасти разными способами, поэтому более высокая скорость работы в последнем случае, скорее, является результатом больших усилий разработчика, чем отражением особенностей используемой технологии.

В качестве последнего замечания к рис. 2 следует привести итоговое ускорение относительно центрального процессора. Оно находилось в диапазоне от 7 до 60 раз, что является хорошим результатом для задачи, скорость решения которой упирается в пропускную способность глобальной памяти. Также стоит отметить, что на других тестовых данных, имеющих более «правильный» для GPU размер, авторами было достигнуто 100-кратное ускорение. Если перевести озвученные значения во время ожидания пользователем окончания обработки одного изображения, то для 12-мегапиксельной фотографии он уменьшилось с 76 до 1,2 с.

Аналогичное тестирование было проведено на другой системе, состоящей из GPU NVIDIA GeForce 555M и CPU Intel Core i7 2670 (рис. 3). На данном графике наблюдается аналогичное поведение производительности всех версий в зависимости от размера сеток. Единственным отличием является более заметное доминирование CUDA-версий программы над OpenCL-реализациями алгоритма при обработке небольших фотографий (размером до 1024x768). Поэтому в данной задаче при использовании ускорителей от NVIDIA имеет смысл использовать OpenCL-и CUDA-версии совместно, выбирая оптимальный вариант в зависимости от входных данных. Ускорение по сравнению с одним ядром CPU на данной системе составило от 5,5 до 20 раз.

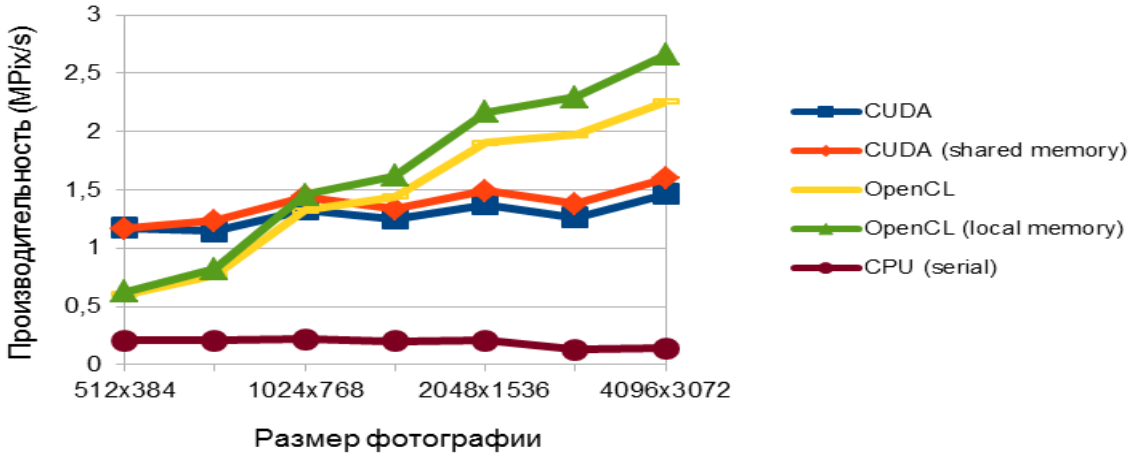


Рис. 3. Тестирование CUDA и OpenCL версий алгоритма на ускорителе NVIDIA GeForce 555M.

5. Оценка ускорения на APU

В настоящее время в качестве одной из альтернатив графическим ускорителям рассматриваются системы на базе гибридных процессоров, в состав которых входят как ядра CPU, так и программируемые потоковые процессоры GPU. Гибридные процессоры разрабатываются всеми крупнейшими производителями: AMD (Fusion), NVIDIA (Project Denver) и Intel (Ivy Bridge).

На момент написания данной статьи в свободном доступе имелись только решения от компании AMD, известные как Accelerated Processing Unit (APU). Основным их достоинством является общая память, в результате чего отсутствует необходимость в пересылке данных по шине PCI-Express, что в перспективе позволит существенно ускорить GPGPU-программы. К сожалению, в текущей версии это оказывается, скорее, недостатком — программная модель OpenCL не позволяет эффективно задействовать общую память при активной записи в неё из каждого типа вычислителя [3], а объём выделенной памяти для GPU-ядер лимитирован 128-1024 Мбайт в зависимости от модели APU и версии BIOS. Другим недостатком является ограниченность пропускной способности оперативной памяти, которая не только ниже, чем у памяти дискретного графического ускорителя, но и совместно используется и CPU- и GPU-ядрами.

Другими словами, хотя с точки зрения программиста архитектура гибридных процессоров практически не отличается от архитектуры обычного графического ускорителя и при этом обладает рядом аппаратных недостатков, ее ожидают неплохие перспективы в случае появления адаптированных под неё программных моделей и интеграции CPU- и GPU-ядер в последующих версиях APU.

Ниже приведены результаты тестирования двух систем на базе подобных процессоров, один из которых имеет 4 ядра CPU и 400 ядер GPU (суммарно порядка 0,5 TFlops), а второй — 2 ядра CPU и 80 ядер GPU (порядка 0,1 TFlops). При тестировании GPU-ядер использовались две OpenCL-версии пакета (с локальной памятью и без неё), а для оценки производительности CPU-ядер были задействованы OpenCL-версия и исходная последовательная реализация алгоритма, скомпилированная при помощи Visual C++ 2008. Результаты тестирования APU AMD A8-3850 приведены на рис. 4.

Стоит отметить, что на системах данного типа использование локальной памяти практически не сказалось на общей производительности при вычислениях на GPU-ядрах. В остальном встроенный графический ускоритель ведёт себя так же, как и его дискретный аналог: чем больше размер фотографий, тем выше скорость обработки. При использовании ядер центрального процессора выигрыш от OpenCL-версии неоднозначен. С одной стороны, она позволяет задействовать все ядра, а с другой, ускорение — по сравнению с последовательной

версией программы оказалось лишь двукратным при использовании четырёх ядер.

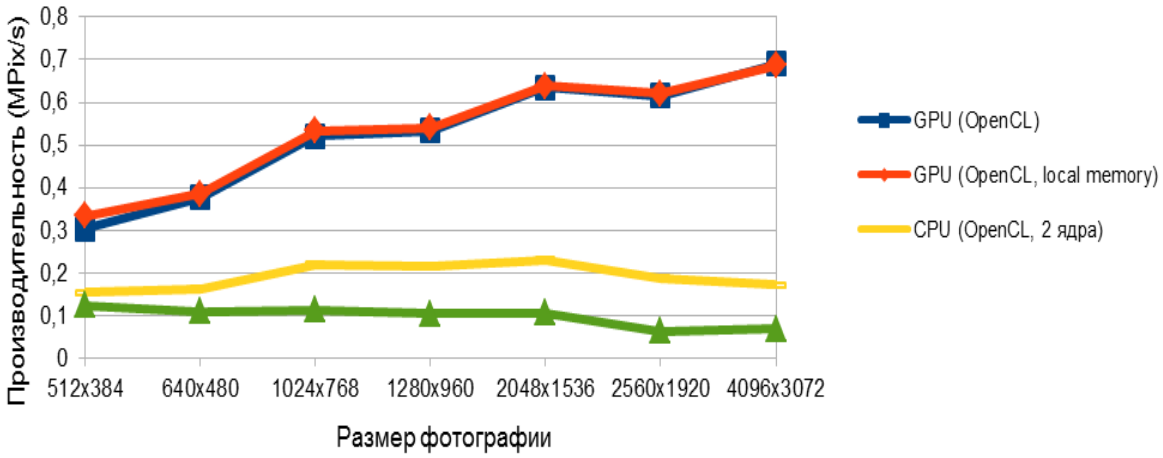


Рис. 4. Тестирование OpenCL версии алгоритма на AMD APU A8-3850.

При проведении аналогичного тестирования на другом APU (AMD E-350) были получены результаты, практически полностью аналогичные предыдущим, за исключением отсутствия возможности провести тестирование GPU-ядер на больших фотографиях. Как было отмечено ранее, максимальный объём памяти APU довольно мал (в данной модели — 128 Мбайт), поэтому для обработки фотографий размера 2048x1536 и более глобальной памяти попросту не хватает. Потенциальным решением является использование общей системной памяти, однако в этом случае скорость доступа к ней будет в десятки раз ниже, чем к глобальной памяти.

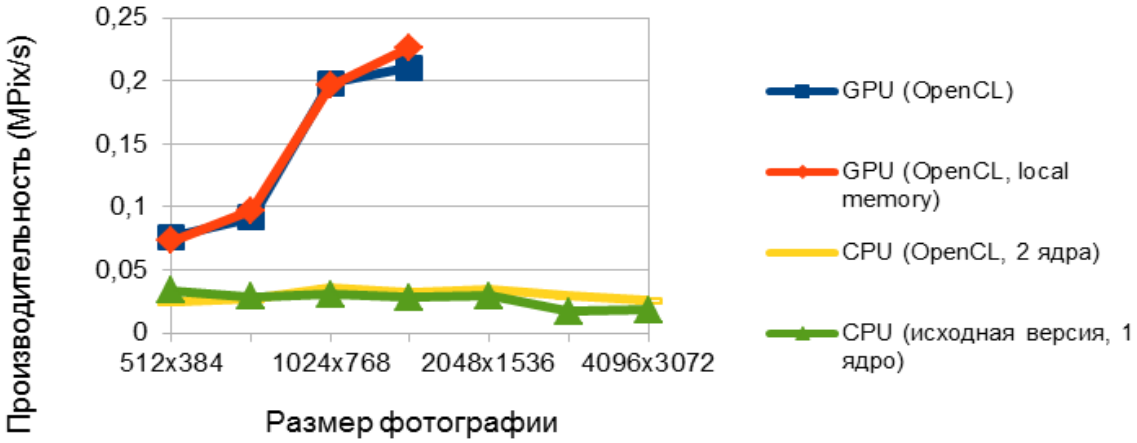


Рис. 5. Тестирование OpenCL версии алгоритма на AMD APU E-350.

Подводя итоги, стоит отметить, что на ускорителях типа APU удалось достичь 3-10-кратного ускорения, причём оно оказалось одинаковым как на самой производительной модели AMD A8-3850, так и на более массовой AMD E-350.

6. Оценка ускорения на многоядерных CPU

Одним из преимуществ технологии OpenCL является тот факт, что написанная на ней программа может быть запущена как на графических ускорителях, так и на многоядерных центральных процессорах. В последнем случае, по заявлению производителей OpenCL-драйверов, которыми являются компании Intel и AMD, будут также задействованы векторные

расширения процессора (SSE или AVX), что принесёт дополнительное ускорение. Впрочем, авторами [4] показано, что даже в синтетических тестах производительность программ на OpenCL, использующих центральный процессор, оказывается крайне низка, что можно объяснить меньшим временем, выделяемым на оптимизацию. В то время как обычный компилятор может потратить практически любое время на анализ исходного кода, драйвер OpenCL должен провести компиляцию практически аналогичной программы, имея на это всего несколько миллисекунд.

На рис.6 приведены результаты сравнения производительности параллельной OpenCL-версии программы и ее последовательной реализации, скомпилированной при помощи Visual C++ 2008. В качестве тестовых машин были выбраны две серверные платформы, одна из которых оснащена двумя 16-ядерными процессорами от AMD, а вторая — 4-ядерным процессором Intel Xeon E3. Так как обе модели вычислителей поддерживают расширения AVX (позволяющие обрабатывать за такт 8 чисел с плавающей точкой одинарной точности), то была использована реализация OpenCL от Intel, являющаяся не только самой быстрой, но и адаптированной для работы с подобными процессорами.

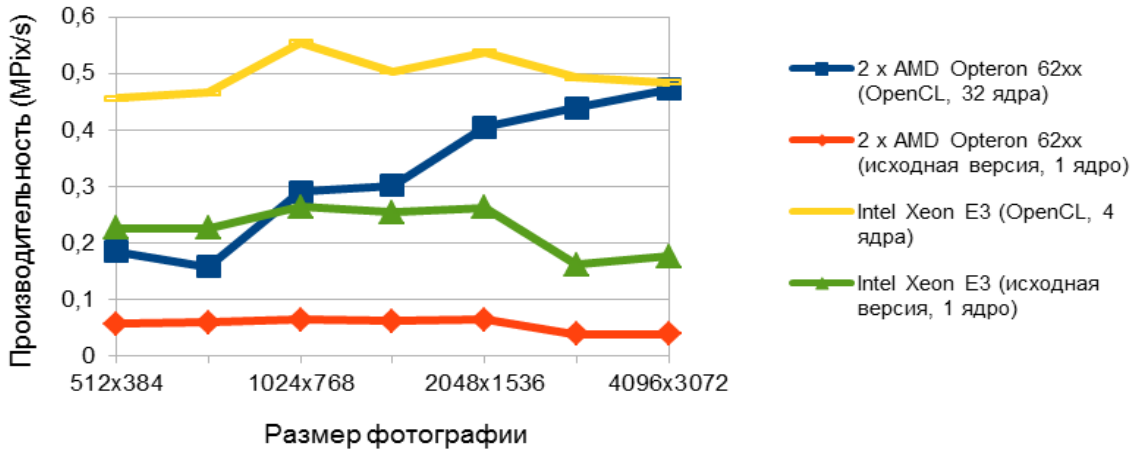


Рис. 6. Тестирование OpenCL-версии алгоритма на многоядерных центральных процессорах.

Результаты тестирования оказались довольно неожиданными. Если сравнивать процессоры от Intel и AMD, то следует объявить о безусловной победе первых. Один 4-ядерный процессор компании Intel с пиковой производительностью порядка 100 GFlops на всех тестовых фотографиях оказался быстрее, чем два 16-ядерных процессора от AMD с пиковой производительностью 560 GFlops. При этом скорость работы программы на 32-ядерной системе практически линейно возрастала с ростом размера картинок, что наводит на мысль о проблемах доступа к памяти при столь большом количестве ядер. Данная система имеет разделённый кэш, поэтому при частых операциях чтениях и записи в один и тот же участок памяти требуется синхронизация кэшей разных процессоров, что, возможно, и объясняет столь низкую скорость работы.

Другим интересным фактом является то, что последовательная версия программы на 4-ядерном процессоре всего в 2-3 раза медленнее, чем параллельная, и в 3,5-10 раз медленнее на 32-ядерной системе. Это лишний раз свидетельствует о том, что на данный момент статические компиляторы создают более быстрые версии программ, чем драйвер OpenCL, компилирующий аналогичные программы динамически.

В заключение стоит отметить, что авторам при ручной оптимизации ряда тестовых программ удавалось на подобных системах достичь 40-50% от пиковой производительности [5], поэтому полученные выше результаты стоит рассматривать лишь как проверку эффективности OpenCL-драйвера для центральных процессоров.

7. Дальнейшее развитие

В результате проведенных работ было создано две оптимизированные реализации алгоритма Mantiuk06, которые на тестовых системах оказались в 5-60 раз быстрее исходной версии. Дальнейшее развитие этой работы планируется осуществить, как минимум, в двух направлениях:

– Разработка гибридной версии алгоритма, которая самостоятельно определяет, на каком вычислителе (CPU или GPU) и какую именно реализацию (CUDA или OpenCL) выгоднее использовать для обработки конкретной фотографии. Как было показано в предыдущих разделах, выбор оптимальной ветки не всегда является очевидным, и в зависимости от архитектурных особенностей динамическое переключение между созданными реализациями позволит повысить суммарную скорость работы на десятки процентов.

– Повышение стабильности и опубликование патча для исходной среды. В настоящий момент во всех созданных версиях алгоритма не реализованы проверки на ряд ошибок времени выполнения типа недостатка памяти, отсутствия требуемых библиотек и OpenCL-драйверов и т.д.. После добавления соответствующих проверок и проведения дополнительного тестирования планируется формализация всех выполненных наработок в виде открытого патча для последней версии среды LuminanceHDR.

Литература

1. Электронный ресурс http://qtpfsgui.sourceforge.net.
2. Mantiuk R., Myszkowski K., Seidel H.-P., A Perceptual Framework for Contrast Processing of High Dynamic Range Images, ACM, 2006.
3. AMD Accelerated Parallel Processing OpenCL™ Programming Guide (v1.3c). Электронный ресурс http://developer.amd.com/sdks/AMDAPPSDK/documentation/Pages/default.aspx
4. Кривов М.А., Зелёно-сине-красная OpenCL // Журнал "Суперкомпьютеры", Осень 2011, с. 47-50.
5. Кривов М.А., Казеннов А.М. Портируем на GPU и оптимизируем для CPU // Журнал "Суперкомпьютеры", Весна 2011, с. 43-45.

Опыт разработки гибридных версий решателей разреженных СЛАУ

М.А. Кривов, С.А. Гризан

ООО «ТТГ Лабс»

В статье описан процесс портирования на графические ускорители двух популярных алгоритмов решения СЛАУ с разреженными матрицами, получаемыми при численном решении ряда уравнений аэродинамики на неструктурированных сетках. Рассмотрены подходы к оптимизации кода, заключающиеся в использовании различных вариантов распараллеливания и представления данных. Приведены результаты тестирования CUDA-версий решателей, а также дана оценка потери производительности при переходе от структурированных сеток к неструктурированным.

1. Введение

С появлением графических ускорителей, имеющих достаточно специфическую архитектуру и накладывающих ряд ограничений на реализуемый алгоритм, разработчики из многих предметных областей оказались перед непростым выбором — применить алгоритмически оптимальный метод и довольствоваться десятыми или сотыми процента от пиковой производительности, или же взять более простой и медленный алгоритм, который на графических ускорителях позволит достичь 10-30% пиковой производительности. Очевидно, что в первом случае вычисления можно ускорить, например, путем использования более высокого порядка аппроксимации, тогда как второй путь обеспечивает гарантированно высокое ускорение от эффективного переноса программы на графические ускорители.

Ещё одной дилеммой может оказаться выбор точности вычислений — даже на последних версиях ускорителей NVIDIA Tesla C2050 переход от одинарной к двойной точности способен заметно уменьшить итоговую производительность. В статье [1] было показано, что при перемножении матриц с помощью популярных GPU-версий BLAS переход от одинарной к двойной точности снизил эффективность использования вычислительных блоков ускорителя с 40% до 10%, что можно интерпретировать как замедление работы программы в восемь раз. Поэтому при использовании, например, абсолютно устойчивых итерационных методов, возможно, стоит ограничиться одинарной точностью вычислений, увеличив число требуемых для сходимости итераций, но при этом уменьшив суммарное время работы программы за счёт более эффективного использования вычислительных блоков GPU.

Озвученные проблемы особенно заметны при использовании сеточных разностных методов для решения ряда уравнений аэродинамики. Так, в зависимости от выбранной разностной схемы алгоритм решения итогового СЛАУ может оказаться не подходящим для использования на GPU, что не только приведет к дополнительным временным затратам на создание его эффективной реализации, но и может потребовать внесения ряда изменений в саму математическую модель.

В данной статье описываются ещё не завершённые работы по портированию существующих численных решателей для уравнения Лапласса, разработанных в рамках пакета SigmaFlow [2]. Целью работ стал поиск ответа на сформулированные выше вопросы, а также портирование существующих решателей на графические ускорители и их последующая оптимизация.

2. Переход к неструктурированным сеткам

При портировании упомянутых алгоритмов на графические ускорители ключевую роль играет выбор оптимальной модели представления данных. Использование

неструктурированных сеток позволяет существенно уменьшить количество узлов на целевой модели, тем самым сократив время вычислений. С другой стороны, при использовании структурированных сеток разработчик получает априорную информацию о расположении соседних узлов, что в дальнейшем позволяет выбрать подходящий формат массивов, а также препросчитать ряд констант.

Для выбора модели представления данных авторами была разработана небольшая тестовая программа, решающая трёхмерное уравнение Лапласа методом Якоби на структурированной и неструктурированной сетках. В обоих случаях использовалось равномерное разбиение параллелепипеда, однако в первом варианте данные представлялись как трёхмерный массив, а во втором – как некоторый граф, каждой вершине которого соответствует значение моделируемой характеристики в заданном узле, а ребру — связь между соседними узлами. Для неструктурированной сетки проводилось два типа тестов — оценка производительности при использовании «упорядоченного» графа, в котором все соседние вершины некоторого узла по возможности располагаются подряд в используемом для их хранения массиве, и «перемешанного» графа, где все вершины произвольным образом размешаны в соответствующем массиве. В соответствии с этим были разработаны версии данной программы для центрального и графического процессоров с использованием технологий OpenMP и NVIDIA CUDA.

Отдельно стоит остановиться на видах оптимизации, применявшихся при разработке GPU-версии программы. Так, при обсчёте с использованием структурированной сетки активно применялась разделяемая память. Легко заметить, что при использовании явной разностной схемы для вычисления значения в одной точке требуется семь обращений к глобальной памяти. Благодаря тому, что все вершины расположены в одном массиве, а для обращения к соседнему узлу требуется обратиться к элементу по некоторому заранее известному адресу, становится возможным загрузить в разделяемую память сразу некоторую подобласть, тем самым снизив количество обращений к глобальной памяти до одного, что значительно повышает скорость вычислений. Другим приёмом стало разбиение основного вычислительного ядра на два мини-ядра. Первое производит вычисления внутри каждой подобласти, помещаемой в разделяемую память, а второе — только на границе таких подобластей. Указанное разбиение позволило достичь выровненного доступа к памяти (в терминологии CUDA именуемого coalesced [3]), а также отказаться от трёх условных операторов, что, к примеру, на сетке размером 256x256x64 повысило скорость вычислений на 40-50%.

В случае неструктурированной сетки возможность использования разделяемой памяти фактически отсутствует, так как для получения индекса соседней вершины требуется прочитать значение из вспомогательного массива индексов, что делает выделение подобластей невозможным. Поэтому пришлось задействовать такие альтернативные механизмы, как текстурная память и L1-кэш глобальной памяти. В первом случае адресация массива со значениями происходила через специальные текстурные блоки графического ускорителя, кэширующие сразу некоторый участок памяти. Во втором предполагалось автоматическое использование кэша L1, который появился в архитектуре Fermi и который, по заявлению компании NVIDIA, позволяет достичь лучших результатов, чем при использовании текстурной памяти.

Результаты тестирования описанных выше программ на системе с ускорителем NVIDIA Tesla C2050 и процессором Intel Core 2 Quad приведены в Табл. 1 (все значения в гигафлопсах).

Таблица 1. Результаты тестирования на сетках различной структуры.

Сетка	Структурированная		Неструктурированная (упорядоченная)			Неструктурированная (перемешанная)		
	GPU	CPU	GPU (L1 кэш)	GPU (textures)	CPU	GPU (L1 кэш)	GPU (textures)	CPU
64x64x16	0,31	1,03	16,13	14,89	0,69	15,52	15,65	0,63
128x128x32	28,28	1,46	20,88	20,59	0,35	20,8	20,53	0,4
192x192x48	46,34	1,56	20,69	20,77	0,53	20,72	20,71	0,69
256x256x64	50,77	1,54	20,69	21,06	0,32	20,97	20,94	0,64
320x320x80	51,3	1,53	21	21,13	0,55	20,9	20,99	0,67

Подводя итоги, стоит отметить, что переход к неструктурированным сеткам вызвал потерю производительности всего в 2,5 раза, которая в дальнейшем может быть легко скомпенсирована уменьшением объёма вычислений посредством укрупнения ряда областей. Стоит отметить также, что графический ускоритель в данной задаче оказался в 20-30 раза быстрее четырёхядерного процессора, на котором были задействованы все ядра, причём независимо от используемой модели представления данных.

Ещё одним интересным результатом оказалось подтверждение гипотезы о том, что на архитектуре Fermi кэш L1 позволяет полностью заменить текстурную память. Оба названных механизма показали одинаковую производительность как на «упорядоченном», так и на «перемешенном» графах. Однако данные результаты верны лишь для ускорителя Tesla C2050: при попытке провести аналогичное сравнение на более старой карте Tesla C1060, в которой кэш L1 отсутствует, было зафиксировано 6-кратное падение производительности.

3. Вариационный решатель

Суть всех алгоритмов, рассматриваемых в данной статье, заключается в численном решении эллиптического уравнения в постановке задачи Дирихле в некоторой трёхмерной области, сетка на которой задана в виде графа, каждый из узлов которого имеет не более N соседей. Идея первого из оптимизируемых алгоритмов состоит в итерационном построении решения путём минимизации вектора сопряжённой невязки. С этой целью на каждой итерации строится и решается соответствующая сопряжённая система с использованием факторизации Якоби, а затем производится обновление приближенного значения искомой величины (в рассматриваемом случае – потенциала) [4].

В исходной реализации данного алгоритма используется весьма нестандартный формат представления сеток. Все вершины хранятся в памяти как один одномерный массив, содержащий значения искомой величины в соответствующих узлах, а для задания связей между вершинами служат два вспомогательных массива, длина которых равна числу рёбер в графе, и в которых содержатся индексы начальной и конечной вершин, соответственно. Таким образом, при проведении одной итерации алгоритма необходимо обойти все рёбра графа, внося изменения в соответствующие массивы вершин по заданным на дугах индексам. Очевидно, что данная реализация не может быть распараллелена, так как при независимой параллельной обработке нескольких рёбер будут возникать конфликты при адресации одних и тех же вершин, известные под названием data race.

При портировании данного алгоритма на графические ускорители были рассмотрено два варианта решения отмеченной проблемы. Первый заключается в использовании такой аппаратной возможности GPU, как атомарные операции. Данный механизм появился в архитектуре 1.1 и позволяет провести некую арифметическую операцию над произвольной переменной в глобальной памяти, гарантируя её атомарность и, как следствие, отсутствие влияния сторонних потоков на результат. К сожалению, данный механизм можно использовать не для всех типов переменных. Так, в архитектуре 2.0 (известной как Fermi) атомарные операции могут быть выполнены для чисел с одинарной и двойной точностью, в то время как в архитектуре 1.3 – только для чисел с одинарной точностью. Результатом применения атомарных операций стала разработка двух реализаций алгоритма, первая из которых оптимизирована под ускоритель NVIDIA Tesla C2050, имеющего архитектуру 2.0, а вторая — под более старую модель NVIDIA Tesla C1060 с архитектурой 1.3 и поэтому не поддерживающий атомарные операции с двойной точностью. Стоит отметить, что в последнем случае поддержка double-чисел всё-таки была добавлена путём замены недостающей команды циклом попыток записи результата до изменения исходной переменной другим потоком.

Альтернативным решением проблемы распараллеливания, очевидно, является выбор другого представления данных. Например, если к каждой вершине добавить в некотором виде список всех её соседей, то потребность в использовании массивов с рёбрами, а, следовательно, и атомарных операций отпадет. Однако здесь возникают два новых препятствия: 1) в исходной реализации используется довольно много массивов со вспомогательными коэффициентами, которые адаптированы под исходный формат данных и которые весьма трудно переделать под

новый, и (2) в общем случае у одной вершины может быть произвольное количество соседей, что заметно усложняет эффективное распараллеливание алгоритма под GPU из-за сильного ветвления. Для обхода указанных трудностей авторами была разработана специальная подсистема кэширования, которая для заданной сетки «на лету» создавала вспомогательные массивы с индексами, расположенные исключительно в памяти GPU и используемые в параллельной версии алгоритма. Поскольку данные преобразования выполнялись только на первой итерации алгоритма, а число итераций обычно оказывалось довольно большим, влияние динамического изменения формата данных на итоговую производительность не превысило одного процента.

На рис. 1 приведены результаты сравнительного тестирования двух описанных подходов на разных стеках при использовании чисел с одинарной точностью и GPU с архитектурой 2.0.

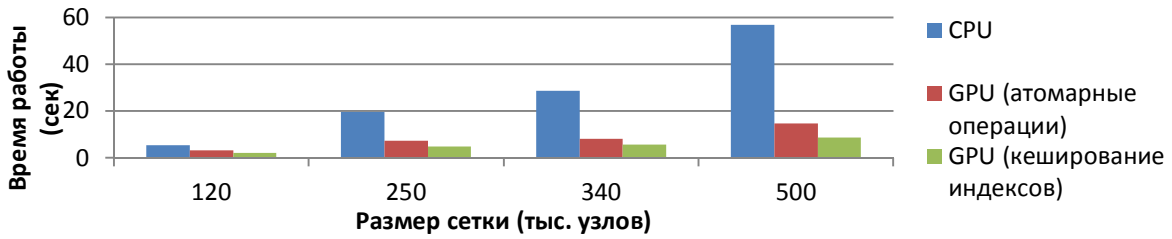


Рис. 1. Сравнение времени работы различных методов распараллеливания под GPU.

Легко заметить, что хотя при кэшировании индексов количество операций и выросло (из-за необходимости обхода дополнительных массивов), суммарная скорость работы увеличилась на 45-65% в зависимости от размера сеток. Поэтому при финальном тестировании ускоренной версии алгоритма, результаты которого приведены на рис. 2 и 3, использовался лишь этот вариант.

Оценка достигнутого ускорения проводилась на системе на базе процессора Intel Xeon E3 с пиковой производительностью 100 GFlops и ускорителя NVIDIA GeForce 580 GTX, обладающего пиковой производительностью 1600 GFlops.

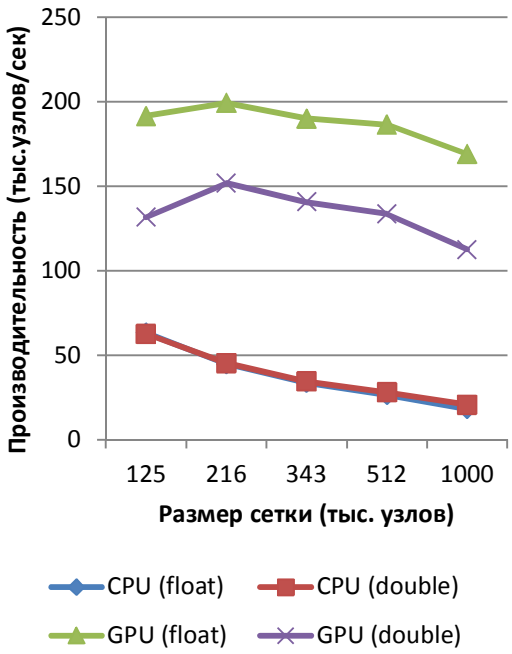


Рис. 2. Скорость вычислений для вариационного решателя.

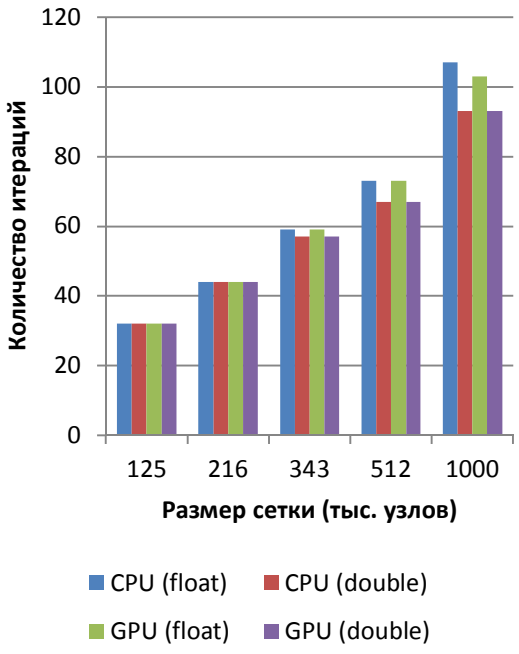


Рис. 3. Скорость сходимости для вариационного решателя.

На рис. 2 проиллюстрирована полезная производительность в пересчёте на количество узлов сетки, обработанных за секунду. На втором — число итераций, потребовавшихся для достижения заданной точности.

Поскольку узким местом данного алгоритма является пропускная способность памяти графического ускорителя, а не нехватка вычислительных ресурсов, то потеря в производительности при переходе от одинарной к двойной точности составила около 35%. В общем случае ускорение GPU-версии относительно CPU было 7-9-кратным, если учитывать пересылки данных, и 10-14-кратным при рассмотрении только времени непосредственных вычислений. Стоит отметить, что в настоящий момент ведутся работы по внесению изменений в архитектуру пакета, в результате чего от ряда пересылок данных удастся отказаться, тем самым приблизив итоговое ускорение к уровню 10-15 раз. Важно подчеркнуть, что все оценки проводились относительно последовательной версии программы, использующей только одно ядро процессора. Авторы считают такое сравнение вполне корректным, так как при разработке параллельной версии программы для CPU потребуется решать аналогичные проблемы синхронизации потоков, в результате чего эффект от многоядерности может оказаться относительно небольшим. Более того, в ряде работ (см., например, [5]) используется именно подобное сравнение, что позволяет сопоставить полученные результаты с уже известными попытками портирования подобных алгоритмов на GPU.

Последним моментом, на котором стоит акцентировать внимание, является более высокая точность float-вычислений на графическом ускорителе. На сетке из одного миллиона узлов GPU-версии потребовалось на четыре итерации меньше, чем исходной реализации для центрального процессора. Это легко объяснить изменением порядка суммирования, в результате чего при вычислении скалярного произведения массивов погрешность накапливается значительно медленнее. Подобное небольшое повышение точности наблюдается довольно часто и является своеобразным «бонусом» от портирования приложений на GPU.

4. Решатель D-ILU

В другом решателе реализован метод разложения матрицы системы на суперпозицию трёх вспомогательных матриц, что в дальнейшем упрощает процесс итерационного построения точного решения [6]. Данный метод активно применяется в популярном пакете OpenFOAM, в результате чего он и стал известен под сокращённым названием D-ILU.

В отличие от рассмотренного ранее вариационного решателя, в данной реализации изначально используется более удобный формат представления данных, в результате чего проблемы обеспечения атомарности операций отсутствовали. Зато трудности возникли при портировании на GPU алгоритма обращения одной из вспомогательных матриц. В изначальной реализации для этого служит паттерн вычислений, известный под названием scan. Он сводится к последовательному обходу массива, для обработки i-го элемента которого требуются результаты обработки всех предыдущих элементов с индексами i-1, i-2, ..., 1, 0. Очевидно, что данный алгоритм в принципе не подходит для переноса на графический ускоритель, поэтому пришлось его заменить итерационным аналогом, который строит приближённое значение искомой обратной матрицы. В результате получилась достаточно интересная реализация, в которой внешние итерации служат для построения приближённого решения искомой величины и, в свою очередь, содержат внутренние итерации для инвертирования вспомогательной матрицы.

Подобная реализация требует не только большего объема вычислений, но и нахождения того порога точности при построении вспомогательной матрицы, который обеспечит достаточную скорость сходимости внешних итераций и минимальное суммарное время работы всего решателя. Путем ряда тестов удалось определить, что 3-4 внутренние итерации гарантированно обеспечивают сходимость для всех тестовых данных, и при этом соответствующая реализация имеет максимальную производительность.

На рис. 4 и 5 приведены результаты тестирования ускоренной версии решателя D-ILU на той же вычислительной системе.

В отличие от вариационного решателя, скорость работы алгоритма D-ILU с увеличением размера сеток заметно понижается при проведении вычислений как на графическом ускорителе, так и на центральном процессоре. Итоговое ускорение от портирования оказалось 4-7-кратным, что объясняется значительно большим объёмом вычислений, обусловленным введением внутренних итераций. Здесь снова следует подчеркнуть, что после проведения ряда работ по

оптимизации пересылок данных между центральным и графическим ускорителем скорость работы GPU-версии удастся повысить более чем на 50%, сделав ускорение 6-12-кратным.

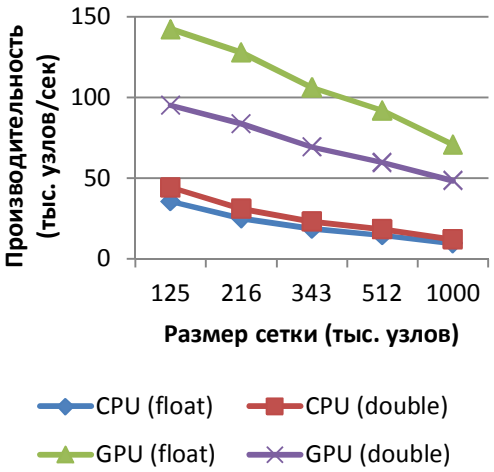


Рис. 4. Скорость вычислений для решателя D-ILU.

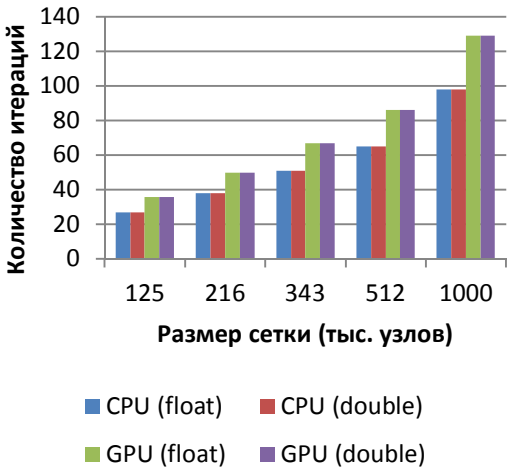


Рис. 5. Скорость сходимости для решателя D-ILU.

Возвращаясь к проблеме обращения вспомогательной матрицы, стоит отметить, что в предложенном варианте для достижения заданной точности потребовалось на 30% больше внешних итераций, что также негативно повлияло на итоговую производительность.

5. Заключение

В работе представлены промежуточные результаты портирования на графические ускорители двух решателей эллиптического уравнения на неструктурированных сетках. Рассмотрены проблемы, которые возникли при разработке параллельных версий соответствующих алгоритмов, а также проведено тестирование созданных реализаций на разных сетках и при вычислениях с двойной и одинарной точностью.

Следующим этапом данных работ является полное портирование на графические ускорители всего пакета SigmaFlow и проведение дальнейшей оптимизации отдельных решателей, а также добавление поддержки систем с несколькими GPU.

Литература

1. Кривов М.А., Казеннов А.М. Сравнение вычислительных возможностей графических ускорителей при решении различных классов задач // Труды Всероссийской научно-практической конференции "Применение гибридных высокопроизводительных вычислительных систем для решения научных и инженерных задач". Н.Н., 2011, с. 18-24.
2. Использование программы SigmaFlow для численного исследования технологических объектов / А.А. Дектерев, А.А. Гаврилов, Е.Б. Харламов, К.Ю. Литвинцев // Вычислительные технологии. 2003. Т. 8. Ч. 1. С. 250–255.
3. CUDA Programming guide: официальный сайт / NVIDIA Corporation - Santa Clara, 2009.
4. Жуков В.Т., Феодоритова О.Б., Янг Д.П. Итерационные алгоритмы для схем конечных элементов высокого порядка. / Математическое моделирование, т. 16, N 7, 2004.
5. M.J. Mawson. Designing numerical solvers for next generation high performance computing / University of Manchester, 2010.
6. T. Behrens. OpenFOAM's basic solvers for linear systems / Technical U. of Denmark, 2009.

Информационная система ГридННС*

А.П. Крюков[1], Л.В. Шамардин[2]

Научно-исследовательский институт ядерной физики имени Д.В.Скобельцына Московского государственного университета имени М.В.Ломоносова (НИИЯФ МГУ), 119991, Москва, Ленинские горы, д.1

Грид-инфраструктура Национальной нанотехнологической сети (ГридННС) предназначена для предоставления ученым, инженерам, аспирантам и студентам, работающим в области нанонаук, унифицированного безопасного удаленного доступа к суперкомпьютерным ресурсам ННС. Одним из главных отличий ГридННС от других грид-инфраструктур является использование архитектурного стиля REST для реализации грид-сервисов. В настоящей работе описана архитектура информационной системы ГридННС, построенная на основе RESTful-веб-сервисов, структура публикуемой информации и ее модель безопасности.

Ключевые слова: высокопроизводительные вычисления, распределенные вычисления, грид, ГридННС, REST, RESTful-веб-сервисы, информационная система.

1. Введение

В 2000 году Р. Филдингом [1] был предложен новый, простой в использовании и гибкий подход к созданию веб-сервисов — архитектурный стиль REST. Важным достоинством RESTful-сервисов по сравнению с подходом основанным на WSRF является простая и ясная семантика запросов, соответствующая интуитивно ясным методам HTTP протокола. Кроме того, RESTful сервисы использует протокол HTTP в своем прямом предназначении как протокол запросов, а не просто транспортный протокол, как это происходит в случае WSRF, где процесс сериализации и десериализации создает дополнительные проблемы, в том числе и с совместимостью реализаций.

В процессе работы на проектом ГридННС [2] авторы использовали архитектурный стиль REST для реализация грид-сервисов в рамках концепции OGSA. В частности, на этой основе был реализован ключевой сервис в ГридННС — сервис управления потоком заданий Pilot [3,4].

Информационная система (ИС) ГридННС первоначально была построена на базе WS-MDS из инструментального набора Globus Toolkit 4. Однако, сложность использования и расширения возможностей WSRF сервисов поставила вопрос о переходе на более удобные в использовании, RESTful-веб-сервисы. Вопрос о новой реализации информационной системы для грида стал еще более актуальным после того как стали известны планы разработчиков Globus Toolkit об отказе от использования WSRF сервисов в последней версии Globus Toolkit 5, а также отсутствия в его составе информационной системы и инструментария для ее создания.

2. Структура ГридННС

Грид-инфраструктура Национальной нанотехнологической сети (ГридННС) предназначена для предоставления ученым, инженерам, аспирантам и студентам, работающим в области нанонаук, унифицированного безопасного удаленного доступа к суперкомпьютерным ресурсам ННС.

* Работа была выполнена при финансовой поддержке Министерства образования и науки РФ (контракт № 07.514.11.4022) и гранта РФФИ № 11-07-00434-а.

[1] kryukov@theory.sinp.msu.ru

[2] shamardin@theory.sinp.msu.ru

Структуру ГридННС в самом общем виде можно представить в виде трех слоев: слой пользовательских интерфейсов, слой грид-шлюзов к ресурсам и слой общих инфраструктурных сервисов.

Слой пользовательских интерфейсов (ПИ) предназначен для запуска пользователями своих заданий в ГридННС.

Слой общесистемных сервисов отвечает за функционирование и управление ГридННС. Основными общесистемными сервисами являются информационная система, система учета, сервис управления и распределения задач, система регистрации сервисов и ресурсов, служба выдачи и управления цифровыми сертификатами, сервис проверки работоспособности ресурсов.

Слой грид-шлюзов обеспечивает доступ к вычислительным ресурсам, подключенным к ГридННС. Грид-шлюз (ГШ) — это комплекс сервисов, который обеспечивает всю функциональность, необходимую для использования кластера или суперкомпьютера из ГридННС. Одним из компонентов ГШ является сервис информационной системы. Другой компонент информационной системы, агрегатор («hub»), является частью общесистемных сервисов. Далее в статье подробно описывается устройство и особенности функционирования информационной системы ГридННС.

Важный принцип, который был положен в основу проектирования ГридННС — это отказ от установки дополнительного ПО на вычислительном ресурсе. Все промежуточное ПО устанавливается только на специальном сервере — грид-шлюзе.

3. Информационная система ГридННС

Архитектура ИС ГридННС изображена на рисунке 1. Основная задача ИС состоит в том, чтобы обеспечить пользователей и сервисы ГридННС информацией о текущем состоянии ресурсов. В качестве источников информации о ресурсе выступают файл со статическим описанием ресурса, а также провайдер динамической информации о ресурсе. Статическая информация включает, например, название ресурса, его местоположение. Первоисточником динамической информации является локальный менеджер ресурсов (ЛМР), которым является система управления пакетной обработки (СУПО) вычислительного ресурса.

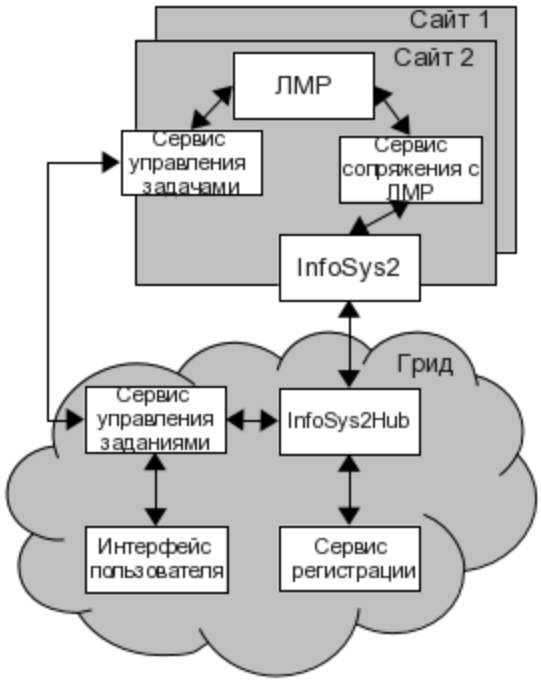


Рис. 1. Архитектура информационной системы ГридННС

Важным моментом построения ИС является вопрос о схеме данных. В качестве прототипа схемы данных использовалась GLUE Scheme 2.0 [5]. В качестве формата для обмена данными используется JSON [6]. Общая структура публикуемой информации по каждому вычислительному ресурсу имеет следующий вид, показанный на рисунке 2.

```
{
...
   «Site»: {
   ...
      «Cluster»: [
      ...
         «SubCluster»: [
          ...
          {
           ...
           «Queue»: [...]
           «Host»: [...]
           «Software»: [...]
          }
         ]
      ]
   }
...
}
```

Рис. 2. Общая структура публикуемой информации

Таким образом, каждый ресурс (сайт) представляет собой список кластеров, которые в свою очередь содержат подкластеры. Разбиение кластера на подкластеры позволяет учесть неоднородную структуру вычислительного ресурса. Структура подкластера предполагается однородной.

Статическая часть информации о ресурсе содержит общее описание ресурса, такие как его наименование, местоположение, административные контакты и так далее.

Информация о кластерах, зарегистрированных на одном ресурсном центре, помещается в секцию «Cluster», которая включает следующую информацию (см. рис. 3).

```
{
    "SubCluster": [
         {
             "Name": "rc.ru/subcluster0",
             "PhysicalCPUs": 16,
             "LogicalCPUs": 16,
             "Queue": [ ... ],
             "Host": [ ... ],
             "Software": [ ... ],
         },
         ...
    ] ...
}
```

Рис. 3. Структура секции «Cluster»

Структура рабочих узлов (серверов) описывается в секции «Host». Сюда относится информация о объеме оперативной памяти, типа ЦПУ, тип ОС и ее версия.

Доступное пользователям ПО публикуется в секции «Software» (рис.4.).

```
"Software": [
    {
        "LocalID": "hpmpi-2.02.05.01-20070708r",
        "Version": "2.02.05.01-20070708r",
        "Name": "hpmpi",
        "InstalledRoot": "/opt/hpmpi"
    },
    {
        "ModuleName": "lmp",
        "LocalID": "lammps-25Sep11",
        "EnvironmentSetup": [
            {
                "softenv": "+lammps-25sep11"
            }
        ],
        "ACL": {
            "Rule": [
                "VOMS:/sysadmin",
                "VOMS:/gridnnn",
                "VOMS:/education",
                "VOMS:/abinit"
            ]
        },
        "Version": "25Sep11",
        "Name": "lammps",
        "InstalledRoot": "/shared/lammps"
    }
] ...
```


Рис. 4. Структура секции «Softwaer»

В секции, описывающей ПО, важную роль играют права доступа (ACL), с помощью которых системный администратор может сообщить о существовании ограничений на использование тех или иных пакетов прикладного ПО членами конкретных ВО. Таким образом, публикуется локальная политика использования прикладного ПО в зависимости от принадлежности к ВО.

Другой важной частью схемы является секция «EnvironmentSetup», в которой указываются метки программного окружения softenv или другие расширения, по наличию которых программное обеспечение грид-шлюза осуществляет специальную настройку среды выполнения для обеспечения доступа к соответствующему ПО некоторым стандартным образом. Например, этой настройкой может быть добавление необходимых путей в переменные PATH и LD_LIBRARY_PATH. Данная необходимость связана с тем, что прикладное ПО на ресурсах может быть установлено различным способом. Простейший пример — использование различных корневых директорий для прикладных пакетов. Так как пользователь в момент запуска задания не знает на каком ресурсе будет выполнена его задача, необходим механизм «настройки» задачи в момент запуска ее на конкретном ресурсе. Использование механизма SoftEnv позволяет пользователю не знать особенности установки ПО на ресурсах грида, что существенно упрощает запуск задач в такой неоднородной среде.

Другим важным атрибутом ресурса, публикуемым в информационной системе является информация о свойствах очереди, и, в частности, об обслуживаемых ВО. Секция Queue содержит так же динамическую информацию о состоянии очередей СУПО каждого подкластера вычислительного ресурса. На рисунке 5 представлено описание параметров доступных очередей. В этой секции публикуется динамическая информация, которая получается от СУПО.

```
"Queue": [
    {
          "CEInfo": "example.com/batch-A",
          "Feature": [ "mpi", "single" ],
          "ACL": {
              "Rule": [
                  "VOMS:/nnn-vo-0",
                  "VOMS:/nnn-vo-1",
                  "VOMS:/nnn-vo-2/group1",
                  "VOMS:/nnn-vo-3/Role=VO-Admin"
              ]
          }
          «MaxWallTime»: 6000,
          «MaxTotalJobs»: 100,
          «MaxRunningJobs»: 50,
          «RunningJobs»: 30,
    }
] ...
```

Рис. 5. Описание параметров доступных очередей

В приведенном примере в качестве динамической информации представлено количество запущенных заданий (RunningJobs).

Данная секция так же как и секция «Software» имеет подсекцию «ACL», которая позволяет указать права доступа к очереди членам ВО. Параметры очередей СУПО устанавливают политику администрации вычислительного ресурса по отношению к потребленным компьютерным ресурсам, таким как время счета, максимальное количество доступных вычислительных ядер. Таким образом, данная секция позволяет информировать пользователей об ограничениях на возможное количество потребляемых ресурсов.

Совместно, секция «Software» и секция «Queue» обеспечивают эффективный и гибкий механизм управления доступом к вычислительным ресурсам.

Информация с каждого ресурса агрегируется специальным RESTful сервисом InfoSys2Hub, который собирает информацию с сайтов, опубликованных в сервисе регистрации грид-сервисов ГридННС, с учетом их режима работы («работает» или «тестируется»). Такая структура позволяет потребителям информационного сервиса иметь дело с единственной «точкой входа» в инфраструктуру. Регистрация нового ресурса автоматически приводит к появлению информации о нем в ИС, изменения состояния сайтов также отражаются в общей информационной системе.

Безопасность ИС построена на использовании цифровых сертификатов стандарта X.509 инфраструктуры публичных ключей (PKI). Доступ к сервису предоставляется конечному набору потребителей в соответствии с конфигурацией сервиса.

4. Заключение

В процессе разработки ГридННС был решен ряд принципиальных вопросов создания грид-инфраструктур на основе RESTful-грид-сервисов.

В частности, была реализована информационная система как RESTful сервис , который обеспечивает публикацию информации о состоянии ресурсов, ее агрегацию со всех доступных для пользователей ресурсов, предоставление ее потребителям, которым она необходима.

В качестве формата обмена информацией был использован JSON. Гибкость данного формата позволила реализовать подмножество публикуемых параметров, которые описаны в GLUE Scheme 2, являющейся стандартом для построения ИС в грид-инфраструктурах.

Использование RESTful сервисов, позволило использовать HTTP протокол не только в качестве транспортного протокола, а в своем прямом назначении, как протокола запросов с ясной

семантикой. Это позволило значительно упростить серелизацию и десерилизацию информации, что улучшило надежность системы.

В настоящее время данная версия ИС успешно прошла тестирование на полигоне ГридННС и в ближайшее время будет внедрена в инфраструктуру в качестве основной.

Литература

1. Fielding R.T. Architectural styles and the design of network-based software architectures // Dotoral dissertation, University of California, Irvine, 2000
2. Грид для Национальной нанотехнологической сети. URL: http://ngrid.ru/ngrid, 2008.
3. Демичев А., Ильин В., Крюков А. и Шамардин Л. Реализация программного интерфейса грид-сервиса Pilot на основе архитектурного стиля REST.// Вычисленные методы и программирование, с.62-65, т.11, 2010.
4. Демичев А., Крюков А. и Шамардин Л. Принципы построения грид с использованием Restful-веб-сервисов.// Программные продукты и системы, №4, 2009, г.Тверь
5. GLUE Scheme. URL: http://forge.ogf.org/sf/projects/glue-wg, 2007.
6. Zyp K. A JSON Media Type for Describing the Structure and Meaning of JSON Documents. // Techniocal report, IETF Network Workong Group, draft-zyp-json-schema-02, March 2010.

Параллельные вычисления при исследовании мышечного сокращения

Ю.Б. Линд[1], Д.С. Казакова[2]

ООО «БашНИПИнефть»[1], Башкирский государственный университет[2]

Изучение механизмов мышечного сокращения является важнейшей задачей биомеханики. Это обусловлено тем, что вся жизнедеятельность человеческого организма связана с мышечной деятельностью. При рассмотрении молекулярных механизмов мышечного сокращения существенную часть исследования составляет математическое моделирование, поскольку современный уровень техники не позволяет проследить за ними непосредственно в мышце. Сокращение саркомера описывается жесткими системами нелинейных дифференциальных уравнений, что приводит к необходимости использования параллельных вычислений при оптимизации параметров системы. Корректность предлагаемых подходов показана также на макроуровне исследования мышцы при решении задачи оценки распределения волокон в мышечной ткани.

1. Введение

На сегодняшний день изучение механизмов мышечного сокращения является одной из наиболее важных и актуальных задач биомеханики. Это обусловлено тем, что вся жизнедеятельность человеческого организма связана с мышечной активностью.

Исследование мышц, как и любого сложного объекта, может быть осуществлено на нескольких уровнях: микроуровень (для рассматриваемого объекта это саркомер – элементарная сократительная единица мышцы), мезоуровень (миофибрилла – структурная составляющая мышечного волокна), макроуровень (отдельная мышца) и мегауровень (организм человека в целом). В данной работе рассматривается механизм мышечного сокращения в отдельном саркомере, то есть на молекулярном уровне – микроуровне. Корректность применяемых алгоритмов была проверена на макроуровне, поскольку для мышцы в целом легче проводить проверочные эксперименты.

Изучение молекулярных механизмов мышечного сокращения связано с большими трудностями, которые, главным образом, заключаются в сложности отслеживания этих механизмов непосредственно в мышце. Исследование же изолированных сократительных белков может дать только косвенную информацию о процессе мышечного сокращения. Таким образом, является актуальным исследование молекулярных механизмов мышечного сокращения путем математического моделирования, основанного на знаниях о молекулярной конструкции саркомера и его физико-химических свойствах.

В настоящее время наиболее распространенной является теория скользящих нитей, согласно которой в основе мышечного сокращения лежит циклическое взаимодействие двух сократительных белков мышцы: актина и миозина. Головки миозина взаимодействуют с мономерами актина за счет энергии, выделяющейся при расщеплении молекулы АТФ до АДФ и фосфорной кислоты. Благодаря этому взаимодействию актиновые и миозиновые нити сдвигаются друг относительно друга, осуществляя сокращение мышцы [1].

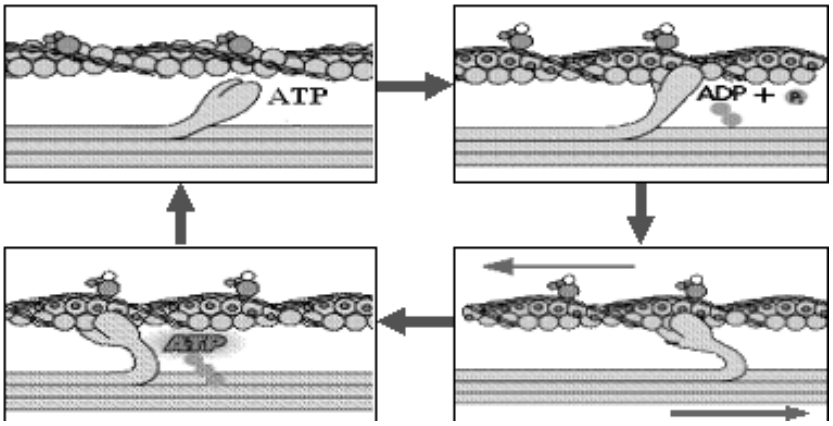


Рис. 1. Механизм мышечного сокращения

2. Кинетическая модель мышечного сокращения

Рассмотренный выше процесс взаимодействия актиновых и миозиновых нитей описывается следующим циклом химических реакций в системе саркомер-раствор [2]:

Таблица 1. Стадии реакции сокращения саркомера

$X_0 + 2X_1 \leftrightarrow 2X_2$	(1)	$X_0 = AM$ – *актинмиозиновый комплекс,*
$2X_2 \leftrightarrow 2X_3 + X_4$	(2)	$X_1 = АТФ$,
$2X_3 \leftrightarrow X_5$	(3)	$X_2 = М.АТФ$ – *комплекс миозина и молекулы АТФ,*
$X_5 \leftrightarrow X_6$	(4)	$X_3 = М.АДФ.Ф$ – *комплекс миозина, АТФ и фосфора,*
$X_6 \leftrightarrow X_7 + 2X_9$	(5)	$X_4 = H^+$ – *ион водорода,*
$X_6 \leftrightarrow X_0 + 2X_8 + 2X_9$	(6)	$X_5 = АМ.АДФ.Ф$, $X_6 = АМ.АДФ.Ф$,
$X_7 \leftrightarrow X_0 + 2X_8$	(7)	$X_7 = АМ.АДФ$, $X_8 = АДФ$, $X_9 = Ф$ – *фосфор.*

Кинетическая модель для данной системы, полученная на основе закона действующих масс, имеет вид системы обыкновенных нелинейных дифференциальных уравнений:

$$
\begin{cases}
\dfrac{dx_0}{dt} = -k_1 x_0 (x_1)^2 + \bar{k}_1 (x_2)^2 + k_6 x_6 - \bar{k}_6 x_0 (x_8)^2 (x_9)^2 + k_7 x_7 - \bar{k}_7 x_0 (x_8)^2; \\
\dfrac{dx_1}{dt} = -2k_1 x_0 (x_1)^2 + 2\bar{k}_1 (x_2)^2; \\
\dfrac{dx_2}{dt} = 2k_1 x_0 (x_1)^2 - 2\bar{k}_1 (x_2)^2 - 2k_2 (x_2)^2 + 2\bar{k}_2 (x_3)^2 x_4; \\
\dfrac{dx_3}{dt} = 2k_2 (x_2)^2 - 2\bar{k}_2 (x_3)^2 x_4 - 2k_3 (x_3)^2 + 2\bar{k}_3 x_5; \\
\dfrac{dx_4}{dt} = k_2 (x_2)^2 - \bar{k}_2 (x_3)^2 x_4; \\
\dfrac{dx_5}{dt} = k_3 (x_3)^2 - \bar{k}_3 x_5 - k_4 x_5 + \bar{k}_4 x_6; \\
\dfrac{dx_6}{dt} = k_4 x_5 - \bar{k}_4 x_6 - k_5 x_6 + \bar{k}_5 x_7 (x_9)^2 - k_6 x_6 + \bar{k}_6 x_0 (x_8)^2 (x_9)^2; \\
\dfrac{dx_7}{dt} = k_5 x_6 - \bar{k}_5 x_7 (x_9)^2 - k_7 x_7 + \bar{k}_7 x_0 (x_8)^2; \\
\dfrac{dx_8}{dt} = 2k_6 x_6 - 2\bar{k}_6 x_0 (x_8)^2 (x_9)^2 + 2k_7 x_7 - 2\bar{k}_7 x_0 (x_8)^2; \\
\dfrac{dx_9}{dt} = 2k_5 x_6 - 2\bar{k}_5 x_7 (x_9)^2 + 2k_6 x_6 - 2\bar{k}_6 x_0 (x_8)^2 (x_9)^2;
\end{cases}
\qquad (1)
$$

с начальными условиями:

$$x_i(t_0)=x_i^0 \tag{2}$$

В системе (1)-(2) x_i, i=1,…,9 – концентрации веществ, участвующих в акте мышечного сокращения, k_i и $\overline{k}_i$, i=1,…,7 – соответственно, кинетические константы скоростей прямой и обратной стадий, t – время протекания реакции; t_0 – начальное время.

3. Исследование механизма мышечного сокращения с применением параллельных вычислительных технологий

Исследование механизма мышечного сокращения включает рассмотрение как прямой задачи (решение системы (1)-(2) при заданных значениях кинетических параметров), так и обратной (восстановление параметров модели по имеющемуся экспериментальному материалу). Прямая задача решается методом Кутты-Мерсона с автоматическим выбором шага интегрирования.

Решение обратной кинетической задачи сводится к рассмотрению серии прямых задач и минимизации критерия отклонения расчета от эксперимента:

$$F = \sum_{i=1}^{N}\sum_{j=1}^{n}\left|x_{ij}^{calc} / x_{ij}^{exp} - 1\right|, \quad x_{ij}^{exp} > 0\,, \tag{3}$$

где x_{ij}^{calc} – расчетные значения; x_{ij}^{exp} – экспериментальные данные; N – количество точек эксперимента; n – количество веществ, участвующих в реакции.

Обратная задача решена для модельного эксперимента, построенного при следующих значениях кинетических параметров, взятых из физико-химических соображений: k_1=10, k_2=0,1, k_3=1, k_4=5, k_5=1, k_6=1, k_7=1, $\overline{k}_1$=1, $\overline{k}_2$=1, $\overline{k}_3$=1, $\overline{k}_4$=0,5, $\overline{k}_5$=1, $\overline{k}_6$=0,5, $\overline{k}_7$=0,5.

Поскольку при решении обратной задачи приходится многократно решать прямую задачу, что предполагает большой объем вычислений, целесообразным является применение параллельных вычислительных технологий.

Рассмотрено несколько методов решения обратной задачи, наиболее эффективным из которых оказался параллельный вариант генетического алгоритма (рис. 2).

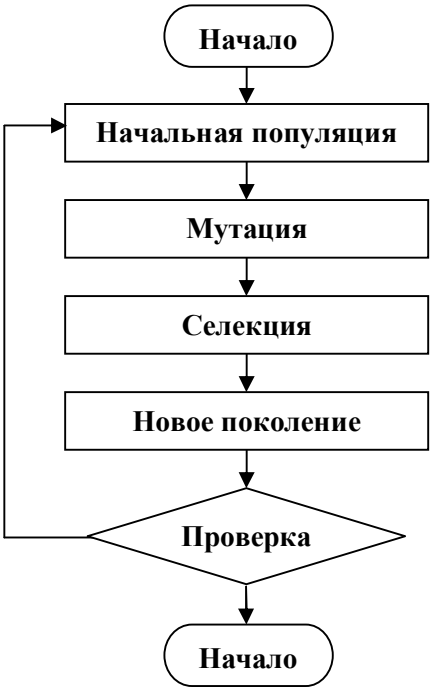


Рис. 2. Генетический алгоритм

На первом шаге алгоритма случайным образом создается начальная популяция, состоящая из N особей (N точек в пространстве кинетических параметров, каждая точка имеет m координат – значений параметров).

На этапе мутации особи популяции изменяются в соответствии с заранее определенной операцией мутации, в качестве которой выступает покоординатный/параболический спуск из точек пространства.

На этапе селекции из всей популяции выбирается определенная ее доля, которая останется «в живых» на этом этапе эволюции. Вероятность выживания особи зависит от значения функции приспособленности для этой особи; в качестве функции приспособленности выступает функционал невязки (3). Доля выживших s является параметром генетического алгоритма, и по итогам отбора из N особей популяции в итоговую популяцию войдут sN особей. В рассматриваемом случае $s=1/2$.

При формировании нового поколения используется скрещивание – чтобы произвести потомка, нужно два родителя. Для формирования новой точки в пространстве параметров в качестве родителей выбирается одна точка из «выживших» и одна из «погибающих», и скрещивание производится путем выбора $m/2$ координат от первой точки и оставшихся – от второй; при этом потомок наследует черты обоих родителей. Особи для размножения выбираются из всей популяции, а не из выживших на первом шаге элементов, с целью исключения возможности деградации популяции.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма, в качестве которого выступает любое из условий:

– нахождение глобального, либо субоптимального решения;

– исчерпание числа поколений, отпущенных на эволюцию;

– исчерпание времени, отпущенного на эволюцию.

Распараллеливание вычислительного процесса производится на стадии начального заполнения, когда заданные псевдослучайно точки в пространстве параметров равномерно распределяются по процессам МВС. Мутация осуществляется каждым процессом независимо; обмен данными производится на этапе селекции. При этом время автономной работы процессов значительно превышает время межпроцессорных взаимодействий, что обусловливает эффективность данного алгоритма.

Оценка эффективности распараллеливания при тестировании программы на вычислительном кластере Башкирского государственного университета (18 процессорных ядер AMD Opteron, пиковая производительность 144 GFlops, объем оперативной памяти 20 Gb, объем дискового пространства 4,2 Tb) показала, что параллельная программа работает достаточно эффективно при увеличении числа процессов (рис. 3).

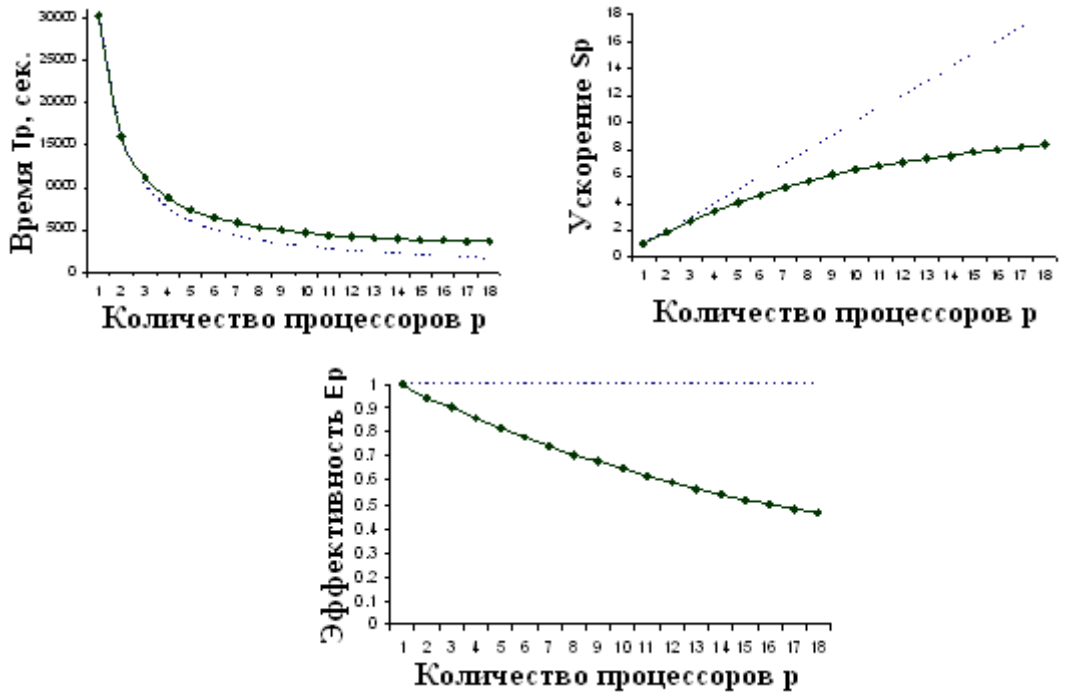


Рис. 3. Анализ эффективности параллельной программы:
а) время выполнения программы; **б)** ускорение; **в)** эффективность

В результате работы алгоритма был получен следующий набор кинетических констант: k_1=2,1, k_2=0,1, k_3=1,1, k_4=4,3, k_5=2,2, k_6=2,3, k_7=0,8, $\bar{k}_1$=0,2, $\bar{k}_2$=1, $\bar{k}_3$=1,2, $\bar{k}_4$=0,4, $\bar{k}_5$=2,3, $\bar{k}_6$=1,2, $\bar{k}_7$=0,4. Несмотря на обеспечение требуемой точности по функционалу (3), полученные значения кинетических параметров некоторых стадий (а именно, стадий (1), (5), (6)) значительно отличаются от исходных. Поэтому было решено, прежде всего, проверить сходимость предложенного генетического алгоритма. При подтверждении его сходимости на следующем этапе исследования необходимо рассматривать не отдельный набор кинетических констант, а области их неопределенности, т.е. области, вариация параметров внутри которых сохраняет требуемое качество описания измерений [3].

4. Оценка распределения мышечных волокон по характеристическим скоростям

Сходимость генетического алгоритма было решено проверить на макроуровне, что обеспечило возможность проводить проверочные эксперименты. Для этого была рассмотрена задача оценки распределения мышечных волокон по максимальным скоростям укорочения.

Основоположник мышечной биомеханики А Хилл, проводя опыты на портняжной мышце лягушки, вывел зависимость между развиваемым мышцей усилием и скоростью ее сокращения [4]. Также на основании ряда опытов он сделал вывод о том, что мышца состоит из волокон, существенно различающихся по своим характеристическим скоростям (максимальным скоростям укорочения при нулевой нагрузке). Он выбрал симметричное распределение как наиболее вероятное (для 82 волокон – табл. 2)

Таблица 2. Симметричное распределение волокон по характеристическим скоростям

v/ℓ_0, c^{-1}	2,4	2,2	2	1,8	1,6	1,4	1,2	1	0,8	0,6
n	1	3	7	13	17	17	13	7	3	1

Была поставлена задача проверки этой гипотезы. С помощью описанного выше генетического алгоритма было найдено наилучшее распределение мышечных волокон по характеристическим скоростям (табл. 3).

Таблица 3. Наилучшее распределение волокон по характеристическим скоростям

ν/ℓ_0 , c^{-1}	2,4	2,2	2	1,8	1,6	1,4	1,2	1	0,8	0,6
n	1	1	1	1	1	26	48	1	1	1

Сравнение полученного наилучшего распределения и гипотезы Хилла с экспериментом представлено на рис. 4. P/P_0 – относительное напряжение в мышце, ν/ℓ_0 – приведенная скорость сокращения мышцы (l_0 – стандартная длина, м [4]).

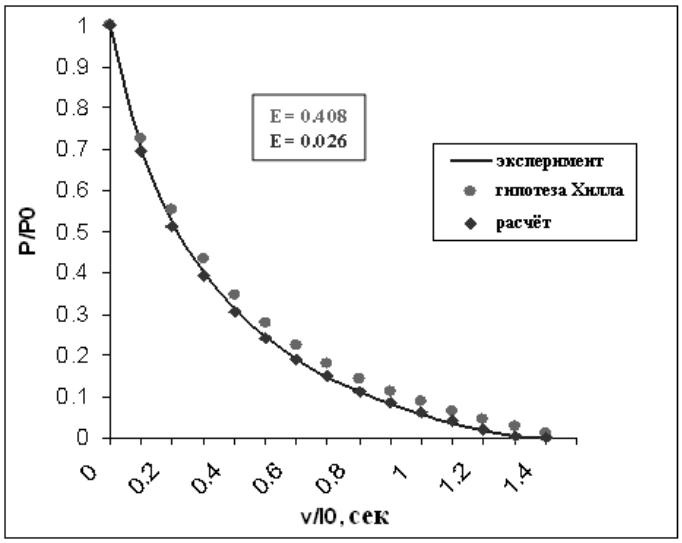


Рис. 4. Сравнение расчета и гипотезы Хилла с экспериментом

Специфика данной задачи состоит в том, что общее количество возможных вариантов конечно, то есть, используя современные информационные технологии, можно перебрать все возможные варианты. Тем не менее, перебирая все значения 10 переменных от 0 до 82, необходимо рассмотреть $83^{10} \approx 1{,}5 \cdot 10^{19}$ вариантов, для которых расчет на ПК займет порядка $3 \cdot 10^{13}$ сек > 1,5 миллиона лет! Для решения этой проблемы авторами был разработан алгоритм «перебрасывания», позволяющий перебирать только нужные варианты (по сумме дающие 82) с минимизацией отклонения кривой от экспериментальной (рис.5).

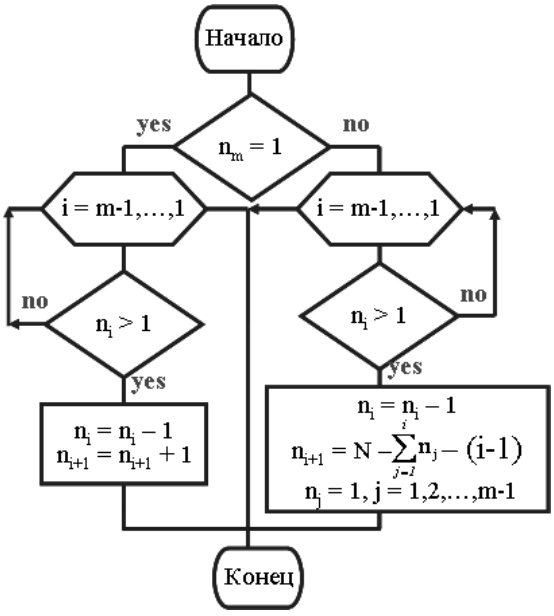


Рис. 5. Алгоритм «перебрасывания»

При реализации указанного алгоритма общее количество вариантов составляет

$$R = \frac{(M-1)!}{(M-m)!(m-1)!} = \frac{(83-1)!}{(83-10)!(10-1)!} \approx 3 \cdot 10^{11}, \qquad (4)$$

что требует, тем не менее, более 5 суток счета на ПК. При использовании всего вычислительного ресурса кластера БашГУ время расчета составило менее 4 часов.

С помощью данного алгоритма найдено распределение волокон, наилучшим образом описывающее экспериментальные данные. Оно совпадает с распределением, полученным при помощи генетического алгоритма, что говорит о корректности последнего. При этом посредством вычислительного эксперимента показано, что большую часть мышцы составляют волокна, обладающие небольшими характеристическими скоростями, однако присутствуют в ней и более быстрые волокна, скорость которых достигает 2,4 l_0/сек.

Таким образом, показана целесообразность использования технологии параллельных вычислений при исследовании молекулярных механизмов мышечного сокращения и установлена сходимость предлагаемого варианта генетического алгоритма. Для дальнейшего исследования кинетики мышечного сокращения на микроуровне планируется выделить области неопределенности кинетических параметров системы и на их основе уточнить схему химических превращений, происходящих в саркомере.

Литература

1. Волков Н.И., Несен Э.Н., Осипенко А.А., Корсун С.Н. Биохимия мышечной деятельности. – Киев, «Олимпийская литература», 2000. – 503с.
2. Быстрай Г.П., Охотников С.А. Термодинамика нелинейных биологических процессов. Переход к хаосу. − Екатеринбург: Изд-во Урал ун-та, 2008, – 154 с.
3. Линд Ю.Б., Аристархов А.В. Параллельные вычисления при построении кинетической модели реакции гидроалюминирования олефинов // Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи». – М.: Изд-во МГУ, 2010. – С. 231-237.
4. Хилл А. Механика мышечного сокращения. – М.: Мир, 1972. – 183 с.

Параллельные вычисления при проектировании профилей горизонтальных скважин

Ю.Б. Линд[1], Л.Р. Миникева[2], Э.И. Зайруллина[2]

ООО «БашНИПИнефть»[1], Башкирский Государственный Университет[2]

Бурение горизонтальных скважин позволяет увеличить дебит нефти из продуктивного горизонта в условиях истощения месторождений и необходимости освоения трудноизвлекаемых запасов. При этом оптимальное сочетание горизонтальных стволов с различными типами профилей позволяет минимальным количеством скважин и кустовых площадок достичь необходимого охвата месторождения. При составлении проекта на строительство куста многозабойных скважин важной задачей является одновременное проектирование и оптимизация сетки профилей для всего куста, что обусловливает целесообразность использования технологии параллельных вычислений. Оптимизация расположения кустовых площадок на месторождениях ОАО АНК "Башнефть" обеспечивает экономию до 30% стоимости проекта.

1. Введение

В настоящее время большинство нефтегазовых месторождений на территории РФ находится на поздних стадиях разработки, в связи с чем возникает проблема интенсификации добычи нефти и освоения трудноизвлекаемых запасов. Однако прирост запасов достигается за счёт освоения сложнопостроенных месторождений. Таким образом, актуальной является работа по поиску технологических решений для максимального использования потенциальных возможностей каждого месторождения. Основным способом решения данной проблемы является бурение горизонтальных и многозабойных скважин, за счёт чего достигается увеличение дебита нефти из продуктивного горизонта и одновременно снижение издержек на обустройство буровых площадок, максимальное сохранение сельскохозяйственных угодий. Оптимальное сочетание горизонтальных стволов с различными типами профилей позволяет минимальным количеством скважин и кустовых площадок достичь необходимого охвата месторождения. При составлении проекта на строительство куста многозабойных скважин важной задачей является также одновременное проектирование и оптимизация сетки профилей как для отдельной многозабойной скважины, так и для всего куста, что накладывает временные ограничения на проведение расчетов. В связи с этим целесообразным является использование для решения поставленной задачи технологии параллельных вычислений.

2. Теоретическая часть

Одной из составных частей технических проектов на строительство наклонно направленных скважин является проектирование профилей скважин, которое состоит в выборе типа и конфигурации профиля, расчёте и построении траектории оси скважин [1].

При проектировании профиля исходят из конкретных горно-технологических условий бурения и целевого назначения скважины. При этом расчёт должен обеспечивать выполнение следующих технических условий:

1) вхождение скважины в продуктивный пласт в конкретной заданной точке;
2) наличие минимального количества участков с разными радиусами искривления, величина которых не превышает допустимые значения;
3) возможность выбора любого типа профиля (плоскостной или пространственный, J- или S-образный, 3- или 5-интервальный, одно- или многосекционный);
4) качественное строительство скважины при минимальных затратах времени и средств;
5) выбор подходящего бурового оборудования на основе расчета осевого нагружения бурильной колонны.

Существуют две группы типов профилей наклонно направленных скважин (рис. 1). Первые называются профилями плоскостного типа и представляют собой кривую линию, расположенную в одной вертикальной плоскости, вторые – пространственную кривую. В данной работе рассмотрены профили, относящиеся к первой группе.

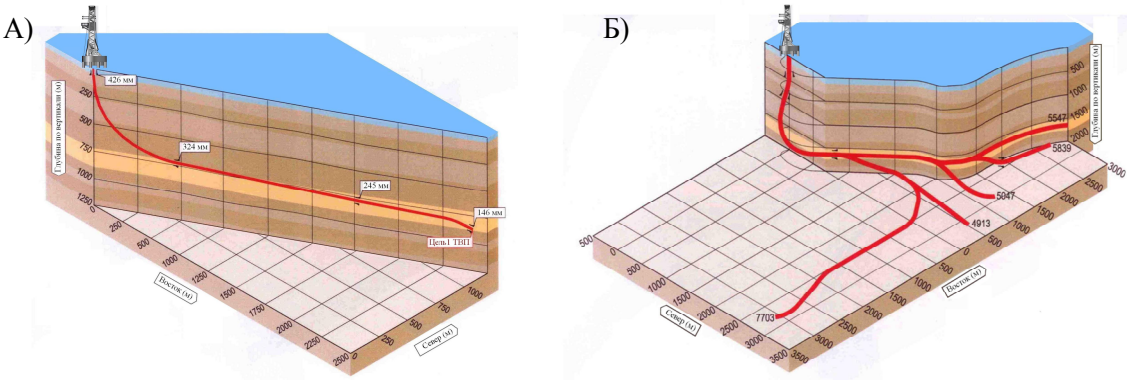


Рис. 1. Графическое представление профиля горизонтальной скважины
А) плоскостного типа; Б) пространственного типа

Проектирование профилей горизонтальных скважин производится на основе расчета геометрических параметров (рис. 2), определяющих пространственное положение элементов профиля (вертикальных, наклонно-стабилизированных и искривленных участков) и графическое построение профиля ствола [2].

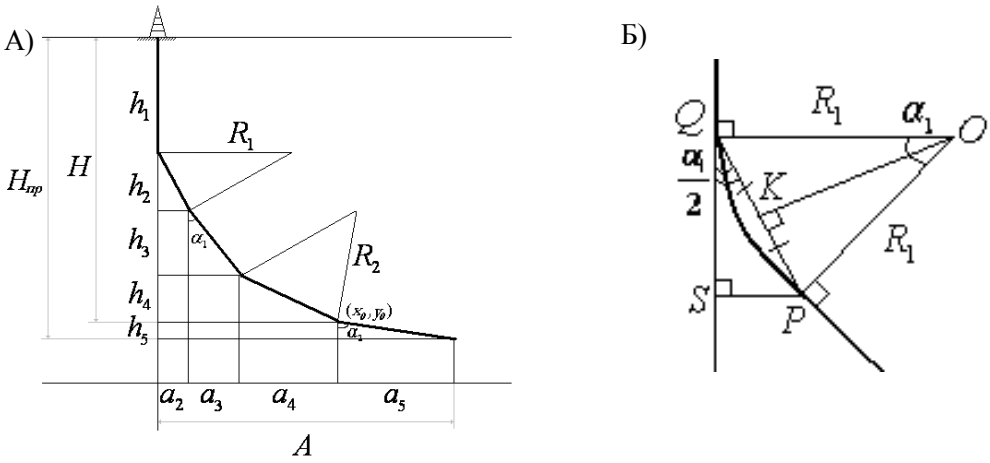


Рис. 2. Геометрическое представление:
А) J-образного 5-ти интервального типа профиля; Б) участка набора угла наклона ствола.

Разработанные алгоритмы обеспечивают при построении профиля отдельной скважины выполнение условий оптимальной проводки трассы в процессе строительства, оптимизации компоновки бурильной колонны и выбора бурового оборудования (на основе расчета осевого нагружения спроектированной колонны [3]), а также других технологических и экономических требований.

3. Программная реализация

3.1 Основные положения

На основе описанных выше алгоритмов был разработан программный комплекс Plane Profile проектирования оптимального профиля горизонтальной скважины плоскостного типа и определения осевого нагружения колонны.

Используются следующие входные данные, формирующиеся непосредственно пользователем в ходе интерактивной работы программы:

- плотность бурового раствора;

- приведенная плотность трубы;
- коэффициент трения о стенки скважины;
- диаметр долота;
- диаметр забойного двигателя;
- тип профиля (S- или J- образный);
- количество интервалов;
- вид используемых труб;
- параметры для расчёта траектории (зависят от типа профиля).

На основе этих данных производится расчёт траектории ствола, который включает в себя определение глубины по стволу, вертикальной и горизонтальной проекций для каждого участка и скважины вцелом.

В качестве выходных данных пользователь получает графическое и аналитическое представления рассчитанной траектории ствола скважины (рис.3), значение осевых нагрузок буровой колонны, а также рекомендации по выбору бурового оборудования.

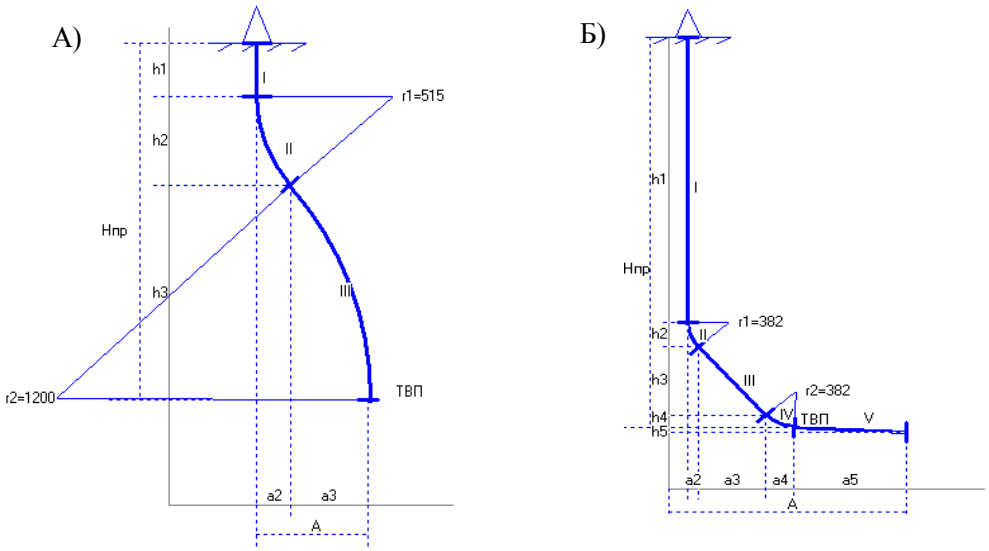


Рис. 3. Примеры графического представления рассчитанных траекторий ствола скважины: А) S-образный 3-х интервальный профиль; Б) J-образный 5-ти интервальный профиль

3.2 Распараллеливание

Распараллеливание вычислительного процесса осуществляется с использованием интерфейса MPI и основано на том, что расчет для кустовых площадок месторождения, для скважин куста, а также для горизонтальных отходов одной многозабойной скважины можно вести независимо друг от друга (рис .4), контролируя выполнение технологических требований по геометрии их расположения (условие непересечения стволов и т.д.). Т.о., время выполнения задачи проектирования одним процессом значительно превышает время межпроцессорного взаимодействия на стадии проверки выполнения этих условий, что обеспечивает эффективность распараллеливания при количестве процессов по числу проектируемых стволов.

Рис. 4. Уровни и количество расчетных задач для месторождения

3.3 Вычислительный эксперимент

С использованием разработанных параллельных алгоритмов проведена оптимизация расположения кустовых площадок на месторождениях ОАО АНК «Башнефть». На рис.5 показано проектное (А) и оптимизированное (Б) расположение кустовых площадок для Воядинского месторождения. В качестве технологического критерия оптимизации выбрана суммарная стоимость строительства площадок при фиксированном охвате месторождения.

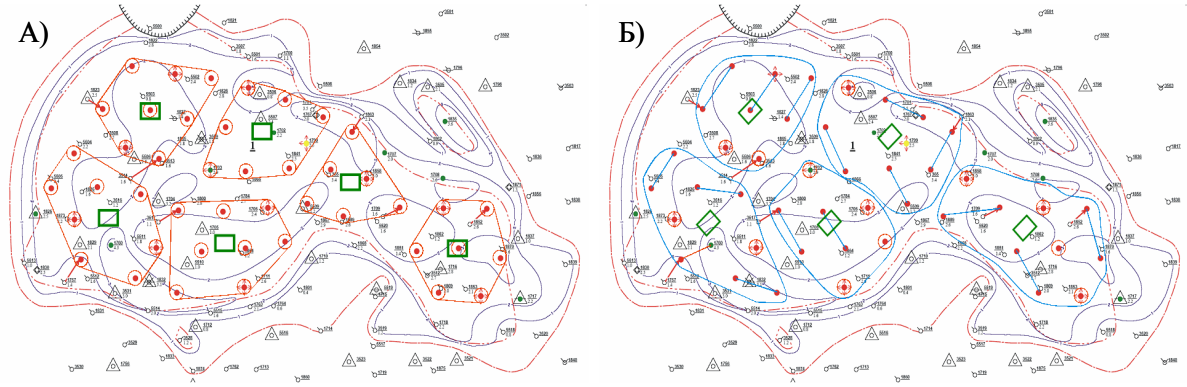


Рис. 5. Расположение кустовых площадок на Воядинском месторождении:
А) проектное (45 скв., 6 площадок); Б) оптимизированное (28 скв., 5 площадок)

Проведен вычислительный эксперимент по расчету расположения кустовых площадок с использованием 1, 2, 4, 8 и 16 процессорных ядер вычислительного кластера Башкирского государственного университета (18 процессорных ядер AMD Opteron, пиковая производительность 144 GFlops, объем оперативной памяти 20 Gb, объем дискового пространства 4,2 Tb) (рис. 6). Анализ полученных результатов показал, что наибольшая эффективность на данном этапе достигается при использовании 8-10 процессорных ядер.

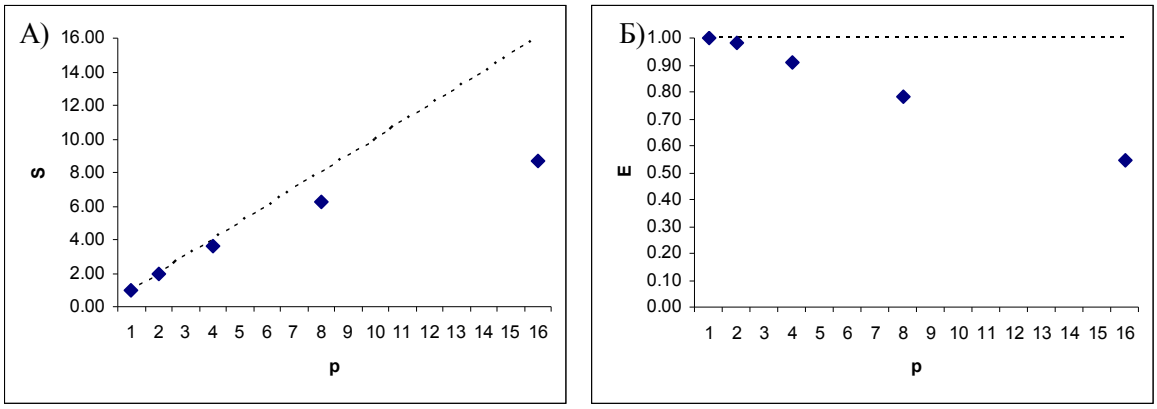


Рис. 6. Анализ эффективности работы параллельной программы:
А) ускорение; Б) эффективность

Результаты вычислительного эксперимента показали, что оптимизированное расположение кустовых площадок позволяет обеспечить охват месторождения меньшим количеством скважин (на 37%) и меньшим числом площадок (на 17%), что дает экономию до 28% от стоимости проекта на строительство скважины в целом.

4. Заключение

В данной работе рассмотрено применение параллельных технологий в задачах проектирования строительства нефтегазовых скважин плоскостного типа. В дальнейшем предполагается разработка соответствующих алгоритмов для скважин пространственного типа. Спецификой строительства этих скважин является постоянное отслеживание условий непересечения стволов и оптимальной траектории скважины, что позволит эффективно использовать распараллеливание на большем числе процессоров МВС.

Литература

1. Овчинников В.П., Двойников М.В., Герасимов Г.Т., Иванцов А.Ю. // Технологии и технологические средства бурения искривленных скважин: Учебное пособие для вузов. /Тюмень: изд-во ТюмГНГУ, 2008 – 152 с.
2. М.Т. Абдурахманов, Н.Ф. Кагарманов. Проектирование профилей горизонтальных скважин// Технология строительства и эксплуатации скважин в осложненных условиях. Сборник научных трудов. – Уфа: Башнипинефть, 1991. – С. 98-102.
3. В.М. Валов, О.Д. Даниленко и др. Инструкция по расчету бурильных колонн для нефтяных и газовых скважин. – М.: ВНИИТнефть, 1997. – 156 с.

Прогнозирование осложнений в процессе бурения с использованием технологии параллельных вычислений

Ю.Б. Линд[1], А.Р. Кабирова[2], Л.Ф. Нурисламова[3]

ООО «БашНИПИнефть»[1], Институт нефтехимии и катализа РАН[2],
Башкирский государственный университет[3]

Разработана методика построения прогноза осложнений в бурении на основе использования искусственных нейронных сетей и технологии параллельных вычислений. Методика реализована для прогнозирования поглощений буровых растворов на месторождениях Республики Башкортостан. Разработанное ПО включает систему управления базой данных по поглощениям на месторождениях РБ, а также модули построения карт интенсивностей поглощений и прогнозирования интенсивности поглощений при бурении новых скважин. Использование нейросетей позволяет осуществить прогнозирование поглощений на основе минимума входной информации, а применение параллельных технологий – сократить временные затраты на проведение расчета и увеличить точность прогноза.

1. Введение

Осложнения при строительстве скважин сопровождаются значительными затратами времени и средств на ликвидацию их последствий, что резко снижает технико-экономические показатели бурения, поэтому задача прогнозирования и предупреждения возможных осложнений становится важной и актуальной при проектировании строительства скважин. При составлении проектов на строительство новых скважин необходимо обоснованно подбирать состав и свойства буровых растворов, за счет которых можно резко снизить вероятность возникновения осложнений или предотвратить их полностью. А это, в свою очередь, возможно только на основе адекватной математической модели и составления прогноза по информации о ранее пробуренных скважинах.

Для прогнозирования поглощений предлагается использование искусственной самообучающейся нейронной сети. Хорошо обученная сеть обладает способностью моделировать функцию, связывающую значения входных и выходных переменных, на основе чего появляется возможность прогнозирования ситуации с неизвестными выходными значениями.

Необходимость обработки большого объема данных из экспериментальной базы и использование нейронной сети порождает значительную вычислительную сложность задачи, что обусловливает целесообразность использования для ее решения параллельных вычислений.

2. Поглощения бурового раствора

Поглощения бурового раствора являются основным видом осложнений при бурении нефтяных и газовых скважин на месторождениях Республики Башкортостан (на долю поглощений приходится более 75% всех осложнений, возникающих при бурении нефтегазовых скважин). Поглощение бурового раствора препятствует выносу из скважины разбуренной горной породы, способствует возникновению обвалов стенок скважины и прихватов бурильного инструмента, что может привести к авариям и необходимости ликвидации скважины. Поэтому необходимо своевременное проведение мероприятий по предупреждению поглощений на основе их прогнозирования по промысловым данным с ранее пробуренных скважин. Методы прогнозирования поглощений буровых растворов, предлагаемые в данной статье, полностью применимы к остальным видам осложнений при строительстве скважин.

Существуют различные причины возникновения поглощений, которые относятся к двум группам [1]:

– геологические факторы (тип поглощающего пласта, его мощность и глубина залегания, пористость и проницаемость, недостаточность сопротивления пород гидравлическому разрыву, величина пластового давления, характеристика пластовой жидкости и т.п.);

– технологические факторы (объем и технологические параметры подаваемого в скважину бурового раствора, способ и режим бурения, скорость проведения спуско-подъемных операций и т.п.).

Геологические факторы характеризуют априорное состояние горных пород, технологические – т.н. «шумы», изменения, вносимые вмешательством человека. Учет геологических факторов является первостепенным и необходимым условием эффективного и оперативного прогнозирования поглощений при бурении новых скважин. Геологические факторы во многом определяются пространственным расположением скважины; при определенных допущениях формирование горных пород на отдельных месторождениях можно считать протекающим в идентичных условиях.

Для составления прогноза на основе вероятностных моделей или регрессионных уравнений необходимо знать градиенты давлений в скважине, характеристики бурового раствора и спуско-подъемных операций и множество других параметров, однако не всегда есть возможность оперативно получить эти данные. Поэтому актуальной является задача построения прогноза на основе минимума информации по ранее пробуренным скважинам. С этой целью для прогнозирования поглощений предлагается использование искусственных нейронных сетей, которые обладают способностью предсказания ситуаций с неизвестным видом связей между входными и выходными параметрами.

3. Прогнозирование поглощений

Для оценки пространственного расположения скважин и отслеживания тенденций распространения поглощений авторами разработано программное построение карт интенсивностей поглощений. В качестве исходных данных при построении карты выступают следующие данные, объединенные в базу данных, разработанную авторами: 1) название скважины; 2) месторождение, к которому относится скважина; 3) условные координаты скважины; 4) сведения о наличии и интенсивности поглощений; 5) глубина залегания и стратиграфическое подразделение, к которому относится поглощающий пласт. На основе базы данных производится построение карты интенсивностей для каждого объекта поглощения на данном месторождении, которая представляет собой совокупность маркеров, нанесенных на плоскость согласно условным координатам скважин и соответствующих максимальной интенсивности поглощения в данной скважине. Т.о., все скважины делятся на 4 класса: без поглощений, с поглощениями небольшой интенсивности – до 40 $м^3$/час, с поглощениями средней интенсивности – от 40 до 80 $м^3$/час, с катастрофическими поглощениями – более 80 $м^3$/час) (рис. 1).

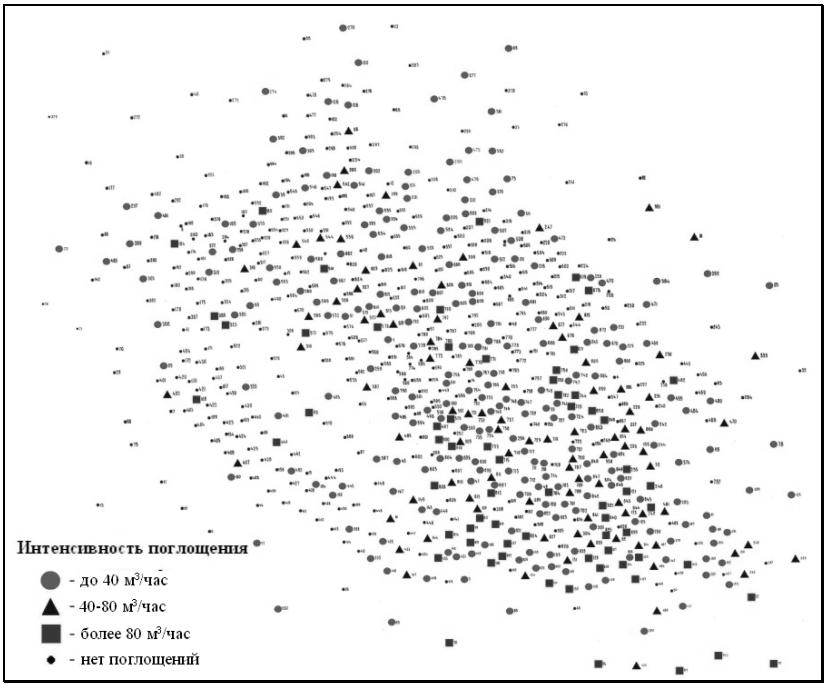


Рис. 1. Карта интенсивности поглощений (Шкаповское месторождение)

Авторами был проведен анализ существующих типов искусственных нейронных сетей, по результатам которого для достижения поставленной цели была выбрана самообучающаяся нейронная сеть, называемая картой Кохонена [2]. Выбор продиктован тем, что такая сеть учится понимать структуру данных, что и требуется при анализе карт интенсивностей поглощений. Т.о., по построенной карте интенсивностей с использованием нейросети Кохонена для каждого объекта поглощения производится анализ расположения и классификация скважин, относящихся к данному месторождению, по интенсивности поглощений и достройка узлов каждого класса. На рис. 2 показан пример достройки узлов класса поглощений высокой интенсивности (маркеры в виде больших квадратов – существующие скважины данного класса, маленьких – достроенные узлы).

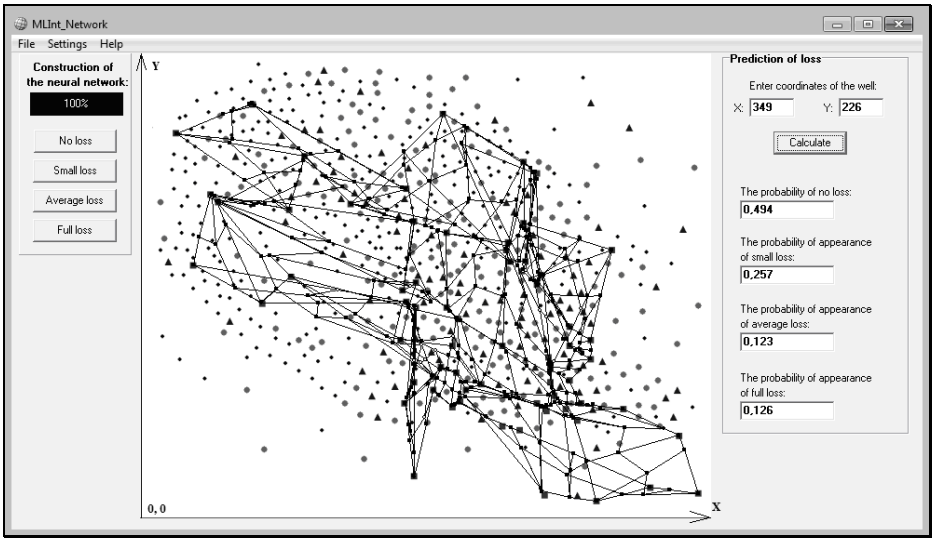


Рис. 2. Достройка нейросетью узлов карты

На основе данной процедуры производится кластеризация всей карты по введенным классам, в результате чего появляется возможность отнесения любой вновь проектируемой скважины к одному из этих классов (рис. 3).

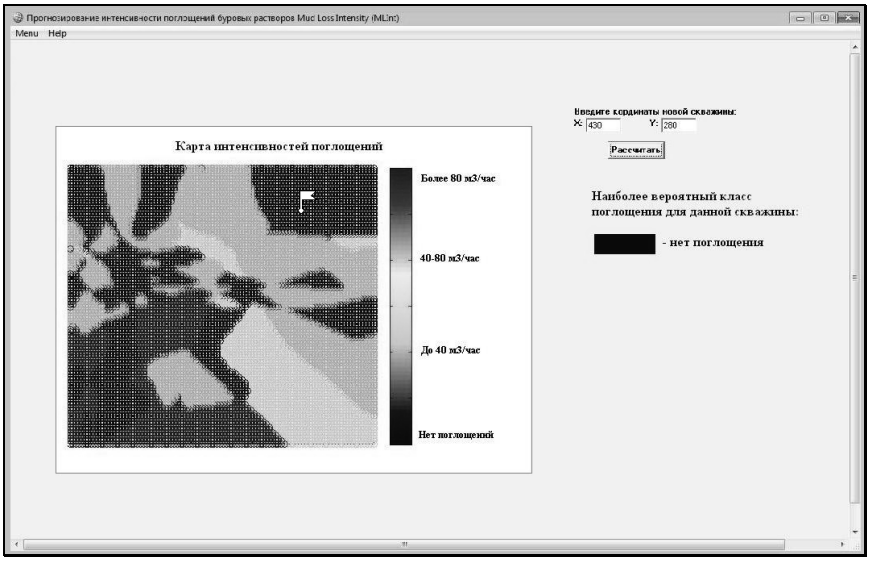


Рис. 3. Кластеризованная карта интенсивности поглощений

Т.о., на основе проектирования карты интенсивностей поглощений в пробуренных скважинах на месторождениях РБ с использованием искусственной нейронной сети производится отслеживание тенденции распространения поглощений каждого класса интенсивности в зависимости от геологических факторов, на основе чего составляется вероятностный прогноз возникновения поглощений при бурении новых скважин. Опираясь на данный прогноз, производится выдача рекомендаций по типу и свойствам бурового раствора и параметрам технологических операций при бурении [3].

4. Распараллеливание вычислительного процесса

Для распараллеливания вычислительного процесса при решении задачи прогнозирования предложена четырехуровневая модель распараллеливания (рис. 4).

Рис. 4. Модель распараллеливания вычислительного процесса

Данная модель включает основные принципы разработанной авторами методологии распараллеливания [4]: использование внутреннего параллелизма задачи, распараллеливание по экспериментальной базе и декомпозиция метода решения задачи. Распараллеливание вычислительного процесса осуществляется с использованием интерфейса MPI.

Внутренний параллелизм задачи состоит в том, что, хотя разные типы осложнений и могут быть обусловлены одними и теми же причинами, но имеют разную физическую природу и приводят к разным последствиям, в связи с чем их можно рассматривать независимо друг от друга. В данной работе реализовано построение прогноза для поглощений буровых растворов, а в дальнейшем планируется разработать также алгоритмы прогнозирования остальных типов осложнений – флюидопроявлений, осыпей и обвалов, прихватов и провалов бурового инструмента, и реализовать этот этап распараллеливания. В настоящее время организация вычисли-

тельного процесса осуществляется комбинацией распараллеливания по экспериментальной базе и декомпозиции метода решения задачи.

Распараллеливание по экспериментальной базе состоит из двух уровней: распараллеливание по месторождениям и по горизонтам месторождения. Поскольку геологические факторы возникновения поглощений для каждого горизонта месторождения специфичны, то становится возможным при прогнозировании для одной скважины проведение расчетов для всех горизонтов независимо друг от друга.

Декомпозиция метода решения задачи прогнозирования основана на том, что при кластеризации карты интенсивности поглощений с использованием нейронной сети достройку узлов по каждому классу интенсивности можно проводить независимо. Время работы каждого процесса при этом значительно превосходит время межпроцессорного взаимодействия, что обусловливает эффективность работы параллельной программы.

5. Вычислительный эксперимент и выводы по исследованию

Программный комплекс MLInt успешно протестирован на данных нескольких месторождений РБ. Расчет проводился для уже пробуренных скважин с целью сравнения фактических показателей с рассчитанными (табл. 1).

Таблица 1. Результаты вычислительного эксперимента

Месторождение	**Номер скважины**	**Горизонт**	**Интенсивность поглощения, м^3/час**	
			Фактическая	**Рассчитанная**
Арланское	1461	P2_kz	0	40 - 80
		C1_s	15	< 40
		D3_fm	0	0
Арланское	1462	P2_kz	0	40 - 80
		C1_s	15	< 40
		D3_fm	0	0
Арланское	1463	P2_kz	нет данных	40 - 80
		C1_s	0	< 40
		D3_fm	0	0
Арланское	1465	P2_kz	> 80	> 80
		C1_s	18	< 40
		D3_fm	0	0
Арланское	1467	P2_kz	> 80	> 80
		C1_s	15	< 40
		D3_fm	0	0
Шкаповское	6195	C	25	< 40
		C1_s2	0	0
		C1_v	0	< 40
Шкаповское	1704	C	0	0
		C1_s2	0	< 40
		C1_v	0	0
Шкаповское	6889	C	> 80	> 80
		C1_s2	> 80	> 80
		C1_v	0	0
Шкаповское	7538	C	79	40 - 80
		C1_s2	15	< 40
		C1_v	18	< 40
Шкаповское	1927	C	0	0
		C1_s2	0	0
		C1_v	0	0

Анализ полученных результатов показал, что прогнозируемый класс поглощения в 78-80% случаев совпадает с реальной интенсивностью поглощения в скважине. Погрешность про-

гноза обусловлена влиянием на возникновение осложнений таких «шумов» как различия в технологических параметрах буровых растворов, скоростях спуско-подъемных операций и т.п., и компенсируется оперативностью получения прогноза (минимумом входных данных).

Вычислительный эксперимент проведен с использованием вычислительного кластера Башкирского государственного университета (18 процессорных ядер AMD Opteron, пиковая производительность 144 GFlops, объем оперативной памяти 20 Gb, объем дискового пространства 4,2 Tb). Анализ полученных результатов показал, что параллельная программа работает достаточно эффективно при использовании всех процессорных ядер кластера (рис. 5).

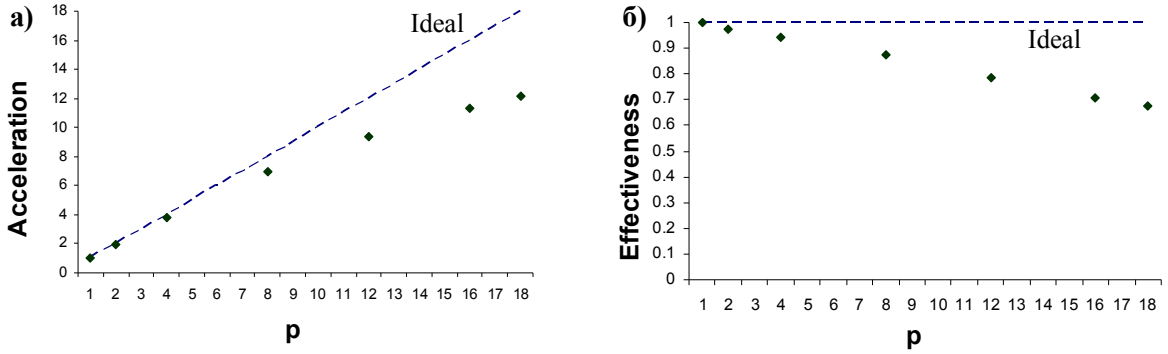


Рис. 5. Анализ эффективности распараллеливания:
а) ускорение; б) эффективность

В настоящее время при составлении проектов на строительство новых скважин тип и состав буровых растворов подбирается с учетом прогноза по поглощениям. Планируется создать единый программный комплекс прогнозирования всех видов осложнений и внедрить его в работу проектных отделов ООО «БашНИПИнефть» и буровых предприятий ООО «Башнефть-Бурение». Полученные рекомендации позволят сократить время на ликвидацию осложнений в условиях буровой и сэкономить дорогостоящие химические реагенты, что, в свою очередь, приведет к повышению технико-экономических показателей бурения. Предварительная оценка экономической эффективности программного комплекса показала, что экономия при его использовании составляет более 650 тыс. руб./скв. (≈ 4% стоимости бурения).

Литература

1. Ясов В.Г., Мыслюк М.А. Предупреждение поглощений при разбуривании трещиноватых пластов. – М.: ВНИИОЭНГ, 1982. – 39 с.
2. Головко В.А. Нейронные сети: обучение, организация и применение – М.: ИПРЖР, 2002. – 256 с.
3. Линд Ю.Б., Клеттер В.Ю., Мулюков Р.А., Губайдуллин И.М. Применение современных информационных технологий для оптимизации состава и оперативного управления технологическими параметрами буровых растворов // «Территория нефтегаз». – №10, 2010. – С. 18-22.
4. Линд Ю.Б., Губайдуллин И.М., Мулюков Р.А. Методология параллельных вычислений для решения задач химической кинетики и буровой технологии // «Системы управления и информационные технологии». – № 2(36), 2009. – С. 44-50.

KernelGen – система компиляции для программной модели «центральный графический процессор – периферийная хост-система»*

Н.Н. Лихогруд[1], Д.Н. Микушин[2], А.В. Адинец[3]

Факультет Вычислительной математики и кибернетики МГУ им. М.В. Ломоносова[1], НОЦ «Параллельные вычисления»[2], Научно-исследовательский вычислительный центр МГУ им М.В. Ломоносова[3]

В работе рассматривается метод автоматической генерации CUDA-кода из программ на языках Си и Фортран, отличающийся полным переносом исполнения программы на GPU, включая последовательные части. Данный экспериментальный метод позволяет сократить количество передаваемых по обменной шине данных за счёт исполнения бóльшего объёма кода на GPU. Преобразование и компиляция программ проводится с помощью DragonEgg, LLVM, Polly/LLVM и nvopencc. В настоящее время система позволяет компилировать корректный и работоспособный PTX-ассемблер для нескольких тестовых приложений.

1. Введение

Массовое использование GPU в кластерных вычислительных системах требует массовой адаптации множества сложных приложений. Программная модель CUDA достаточно хорошо подходит для небольших программ с ярко выраженным вычислительным ядром. Однако для сложных приложений, состоящих из множества отдельных блоков, таких как математические модели, сложность настройки взаимодействия оригинального кода и кода для GPU многократно возрастает. По этой причине для упрощения программирования GPU активно развиваются гибридные программные модели, позволяющие с помощью директив компиляции разметить участки кода исходной программы для работы на GPU, по аналогии с OpenMP. Для стандартизации набора директив основными разработчиками подобных коммерческих технологий создан консорциум OpenACC [1]. Существуют аналогичные открытые системы F2C-ACC [2] и KernelGen 0.1 [3] для преобразования исходного кода на языке Fortran в аналогичную гибридную форму, но они не могут автоматически оценивать параллельность переносимых на GPU вычислительных циклов. Открытые трансляторы внутреннего представления вычислительных циклов компилятора GCC в OpenCL-код [5] и компилятора Clang в CUDA-код [6] производят преобразования с учётом зависимостей данных на основе модели CLooG [7].

Программные модели с выделением фрагментов кода для акселерации предполагают расстановку необходимых аннотаций вручную. По этой причине редко удаётся полностью портировать достаточно большое приложение на GPU, что приводит к наличию обменов данными, связывающих CPU- и GPU-части. Например, при портировании только одного блока WSM5 модели WRF с помощью директив PGI Accelerator, время обменов данными составляет 40-60% общего времени [12]. Такая же ситуация возможна, если для выполнения на CPU оставлять все непараллельные фрагменты кода.

В существующем стандартном процессе компиляции CUDA-кода *nvcc* вспомогательный процессор $cudafe$ работает только с C++-кодом, разделяя каждый входной исходный файл на CPU-часть и GPU-часть, дополнительно преобразуя некоторые конструкции языка. Затем CPU-часть обрабатывается CPU-компилятором (gcc, icc, cl.exe и др.), а GPU-часть – компилятором *nvopencc* [8] (вариант $open64$ [9] с бэкендом для генерации ptx-ассемблера).

*Работа поддержана контрактами НОЦ «Параллельные вычисления» 12-2011 и 13-2011, тестирование велось на оборудовании, установленном на факультете ВМК МГУ, НИВЦ МГУ и компании NVIDIA.

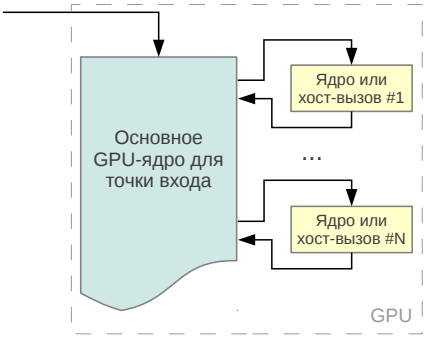


Рис. 1. Схема исполнения «центральный графический процессор – периферийная хост-система»

Теоретически, обе части можно было бы компилировать с помощью *open*64, но, вероятно, разработчики желали сохранить привязку приложений к оригинальному CPU-компилятору для лучшей совместимости. Таким образом, несмотря на то, что компиляторы обоих ветвей помимо C++ поддерживают и другие языки, $cudafe$ и сам способ организации расщепления кода делают их использование невозможным.

При разработке гибридных систем компиляции также необходимо предусмотреть возможность применения к кластерным приложениям с множеством GPU, которые уже распараллелены по вычислительным элементам. Если используется многопроцессное распределение (например, MPI), то необходимо переключение GPU, если многопоточное (OpenMP, POSIX Thread), то также необходима потоковая безопасность (thread-safety) управляющей системы (runtime-библиотеки).

Настоящая работа является развитием существующей системы компиляции KernelGen. На данном этапе были поставлены и решены следующие основные задачи:

Минимизировать обмен данными между памятью системы и GPU за счёт переноса всего кода на GPU. Во всех существующих в настоящее время системах компиляции разметка исходного кода определяет отдельные участки для выполнения на GPU, что делает неизбежным явный обмен данными. Более того, и в интегрированных системах типа AMD Fusion (если их поддержка будет включена в OpenACC), где память CPU и GPU используют один и тот же физический носитель, адресные пространства логически разделены и требуют копирования данных. Данная разработка впервые использует противоположную технику: по умолчанию на GPU переносится *весь* код приложения, предполагая, что даже последовательные части кода может быть более выгодно исполнять на GPU, чем обмениваться данными и выполнять их на CPU. Тем не менее, полностью перенести весь код на GPU невозможно, например, из-за наличия системных вызовов или операций ввода-вывода. Для этого случая необходимо предусмотреть режим обратного вызова, при котором исполнение GPU-ядра останавливается до завершения хост-вызова. Таким образом, элементы привычной модели «основная хост-система и периферийный графический процессор» меняются местами: «центральный графический процессор – периферийная хост-система» (рис. 1).

Обеспечить поддержку широкого множества языков программирования. Как показал опыт разработки KernelGen версии 0.1, создать препроцессор разделения кода для другого языка, аналогичный $cudafe$, возможно, но не слишком рационально по необходимым ресурсам. Подходящей альтернативой разделению исходного кода могут быть преобразования на уровне внутреннего представления компилятора. KernelGen использует LLVM [10] (инфраструктура компилятора с простым внутренним представлением LLVM IR

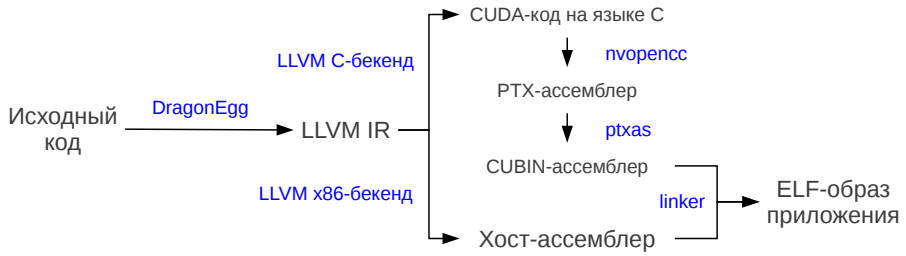


Рис. 2. Схема компиляции кода KernelGen

SSA) и DragonEgg [11] (плагин gcc, генерирующий LLVM IR для любого поддерживаемого входного языка), как показано на рис. 2

Производить автоматическую оценку параллельности и преобразование вычислительных циклов. Для определения параллельных участков кода задействован CLooG, но не напрямую, а посредством LLVM Polly [4], который встраивает выпуклый анализ циклов в структуру оптимизирующих преобразований (passes) LLVM. В настоящее время LLVM Polly поддерживает генерацию эффективного кода только для CPU и OpenMP. В данной работе он был изменён таким образом, чтобы распараллеливать код с учётом специфики GPU: отображение многомерных параллельных циклов на многомерную вычислительную решётку и запуск ядер средствами CUDA API.

В следующих разделах настоящей статьи дана харатеристика этапов преобразования исходного кода и генератора параллельных циклов.

2. Этапы преобразования исходного кода

KernelGen работает напрямую с оригинальным приложением, не требуя каких-либо изменений ни в исходном коде, ни в системе сборки. Для наилучшей поддержки больших приложений это свойство чрезвычайно важно и часто недооценивается. Чтобы обеспечить обычный процесс сборки, код для GPU сначала добавляется в отдельную секцию объектных файлов, затем объединяется и снова разделяется на отдельные ядра на этапе линковки. Окончательная компиляция GPU-ядер в ассемблер происходит при необходимости, уже во время работы приложения (JIT, just-in-time compilation).

В результате работы компилятора, исходное приложение преобразуется во множество GPU-ядер: одно или несколько *основных* ядер и множество *вычислительных* ядер. Основные ядра исполняются на GPU в одном потоке. Их задача – хранить данные, исполнять последовательные участки кода и производить вызовы вычислительных ядер и отдельных CPU-функций, которые невозможно перенести на GPU (рис. 1). Вычислительные ядра исполняются на GPU множеством параллельных нитей с полной загрузкой мультипроцессоров. Таким образом, максимальная доля кода выполняется на GPU, а CPU лишь координирует исполнение. В частности, при работе MPI-приложения каждый рабочий процесс в данном случае будет представлять собой GPU-ядро с небольшим числом CPU-вызовов MPI. Использование MPI дополнительно облегчается за счёт поддержки GPU-адресов в командах обмена данными [13].

Компиляция. При компиляции отдельных объектов, генерируется как x86-ассемблер (таким образом, приложение по-прежнему работоспособно при отсутствии GPU), так и представление LLVM IR. Для разбора исходного кода используется компилятор GCC, чьё внутренее представление *gimple* преобразуется в LLVM IR с помощью плагина DragonEgg. Затем в IR-коде производится выделение тел циклов в отдельные функции, вызываемые

через универсальный интерфейс вида

```
__device__ int kernelgen_launch(
    unsigned char* name, unsigned long long szarg, unsigned int* arg);
```

где name – имя или адрес функции (вместо имён в начале работы программы подставляются адреса), arg – структура, агрегирующая аргументы вызова, szarg – её размер.

Стандартный механизм выделения циклов в функции LLVM *CodeExtractor* расширен, так чтобы цикл не заменялся, а дополнялся вызовом функции по условию:

```
if (kernelgen_launch(name, szarg, arg) == -1)
{
    // Lauch original loop.
}
```

Таким образом, *kernelgen_launch* запускает параллельное ядро только при выполнении определённых условий, в противном случае (например, если в цикле слишком мало итераций) оригинальный цикл выполняется последовательно в основном GPU-ядре.

Далее, существовавшие в оригинальном коде функции подставляются друг в друга (inline). Подстановка необходима, т.к. ни CUDA, ни OpenCL не реализуют для GPU систем линковки (и, вообще говоря, неясно как её реализовывать при статическом распределении регистров и разделяемой памяти), а значит полученные ядра должны быть как можно более самодостаточными. Если ядро имеет неразрешимую внешнюю зависимость, то такой вызов преобразуется в вызов вида

```
__device__ void kernelgen_hostcall(
    unsigned char* name, unsigned long long szarg, unsigned int* arg);
```

при котором GPU-приложение останавливает свою работу и передаёт данные и адрес функции для выполнения на CPU. Функции *kernelgen_launch* и *kernelgen_hostcall* работают в GPU-ядре и вызывают остановку его выполнения. После завершения работы другого ядра или CPU-функции, основное ядро продолжает работу.

Линковка. При линковке отдельных объектов в результирующее приложение или библиотеку, в IR-коде производится окончательная подстановка и оптимизация. В результате предыдущих преобразований, единый IR-модуль содержит код main-функции (или код множества точек входа в случае динамической библиотеки) и код функций с отдельными вычислительными циклы.

Отдельной обработки требуют глобальные переменные. С одной стороны, глобальные переменные размещаются в глобальной памяти (а значит могут совместно использоваться различными ядрами), с другой – *nvopencc* компилирует их в неинициализированные символы вместо внешних из-за отсутствия линковки. Таким образом, синхронизацией глобальных переменных между несколькими ядрами необходимо управлять вручную. Вместо этого на данном этапе была реализована локализация: определения включены в тело основного ядра и в аргументы вычислительных ядер. Однако, такое решение некорректно в случае нескольких основных ядер.

Модель исполнения. Основное ядро запускается в самом начале выполнения приложения и работает на GPU постоянно. Во время работы вычислительного ядра или CPU-функции основное ядро переходит в состояние активного ожидания и продолжает работу после завершения внешнего вызова. Для реализации данной схемы GPU должно поддерживать одновременное исполнение нескольких ядер (concurrent kernel execution) или временную выгрузку активного ядра (kernel preemption). Одновременное исполнение ядер доступно в GPU NVIDIA, начиная с Compute Capability 2.0, в GPU AMD такой возможности нет, но есть вероятность появления kernel preemption в одной из следующих версий OpenCL. По этой причине в данный момент KernelGen работает только с CUDA.

Вызовы *kernelgen_launch* и *kernelgen_hostcall* состоят из двух частей – device-функции

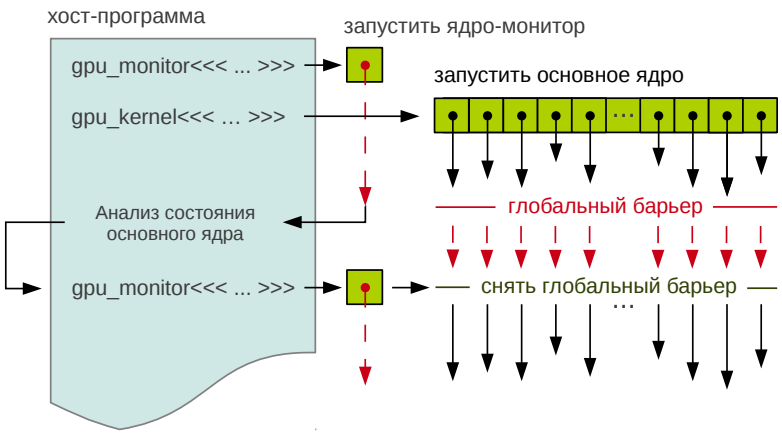


Рис. 3. Схема системы активной синхронизации

на GPU и одноимённого вызова в CPU-коде, который выполняет, соответственно, окончательную генерацию кода и запуск вычислительного ядра или загрузку данных с GPU и запуск CPU-функции средствами Foreign Function Interface (FFI). Взаимодействие между часятми может быть организовано посредством глобальной памяти GPU или pinned-памяти хоста. Однако для гарантированной передачи корректного значения необходимо обеспечить *атомарный* режим операций чтения и записи, доступность которого является определяющим фактором. По этой причине был реализован метод, использующий глобальную память.

В глобальную память GPU помещается целый маркер с состояниями 0 и 1, начальным состоянием 1. В момент начала работы приложения запускается ядро-монитор, атомарно меняет значение маркера на 0 и ожидает его изменения на 0. Следом запускается основное ядро, которое при наступлении события заполняет данными управляющую структуру, атомарно меняет значение маркера на 1 и переходит в состояние ожидания значения 0, позволяя при этом завершиться ядру-монитору. Синхронизация при завешении ядра-монитора на стороне CPU и является сигналом начала взаимодействия: производится чтение управляющей структуры и требуемый вызов. После его окончания, снова запускается ядро-монитор, которое разблокирует основное ядро (рис. 3).

Дополнительное препятствие взаимодействию GPU-ядра с другим ядром или CPU состоит в том, что данные нити (CUDA thread) хранятся в регистрах или локальной памяти. Это означает, что аргументы, переданные из основного GPU-ядра не могут быть использованы где-либо, кроме как в нём самом. Для преодоления этого ограничения, код компилятора *nvopencc* был дополнен реализацией опции $-CG : auto\_as\_static = 0$. В таком режиме компиляции локальные переменные помещаются не в .local-секцию, а в .global, делая их доступными всем GPU-ядрам и хосту.

3. Генерация CUDA-ядер для параллельных циклов с помощью LLVM/Polly

Polly [4] (от polyherdal analysis – выпуклый анализ) – это оптимизирующее преобразование циклов, основанное на CLooG и инфраструктуре LLVM. Оно способно распознавать параллельные циклы, оптимизировать кеширование за счёт добавления блочности, оптимизировать доступ к памяти за счёт перестановки циклов и генерировать код, использующий OpenMP. Для заданного кода CLooG строит *абстрактное синтаксическое дерево* (AST), проводя расщепление циклов по некоторым измерениям. Благодаря возможности расщепления частично-параллельных измерений, для исходного цикла может быть найдено экви-

валентное представление из одного или нескольких циклов, часть которых параллельна. Подобный подход используется довольно редко, большинство современных компиляторов ограничиваются проверкой параллельности измерений существующих циклов без глубокого анализа.

Элементами AST являются *статические части потока управления* (static control parts – SCoPs) – части программы, в которых поток управления и шаблоны доступа к памяти могут быть вычислены во время компиляции. Часть программы представляет собой SCoP при выполнении следующих условий:

1. Единственными операторами управления являются циклы-счётчики (for) и условные операторы.

2. Какждый цикл-счётчик имеет только одну целочисленнную индексную переменную, которая изменяется в теле цикла с константным шагом. Верхняя и нижняя границы цикла заданы афинными выражениями, зависящими от параметров и индексных переменных внешних циклов, где под параметрами понимаются любые целочисленные переменные, не изменяющиеся внутри SCoP.

3. Условные операторы сравнивают только значения двух афинных выражений.

4. Помимо управлющих конструкций присутствуют только операторы присваивания, выражения и индексный доступ к массивам. Выражениями могут быть операторы или вызовы функций без побочных эффектов, аргументами которых являются параметры, индексные переменные или элементы массивов.

5. Обращения к массивам проводятся по индексам, являющимися афинными выражениями от параметров и переменных.

При генерации LLVM IR высокоуровневые конструкции исходных языков преобразуются в последовательности низкоуровневых инструкций. Так, циклы описываются условными переходами, формирующими эквивалентный поток управления, обращения к массивам выражаются адресной арифметикой, афинные выражения расщепляются на последовательности трёхадресных инструкций. Для восстановления высокоуровневой информации из промежуточного представления Polly использует средства LLVM. Преимуществом такого подхода является возможность извлечения высокоуровневой информации из низкоуровневой, даже если она присутствует в высокоуровневом коде программы лишь неявно. Другими словами, Polly может оптимизировавать не только циклы, присутствующие в явном виде, но и любой код семантически эквивалетный циклам, например программы на языке Fortran, использующие goto.

В KernelGen из выделенных на этапе компиляции функций с тесно-вложенными циклами средствами LLVM/Polly выбираются параллельные по одному и более измерениям. С помощью класса *polly* :: *clastExpGen* для каждого цикла генерируется код расчёта числа итераций, используемый при задании вычислительной сетки ядра. К исходным циклам также применяются стандартные упрощающие преобразования LLVM и пространственно-временные оптимизации Polly. Данные этапы могут быть применены как на этапе компиляции, так и во время исполнения приложения. В первом варианте можно сразу же провести оптимизацию и отбросить циклы, определённые как непараллельные. В то же время, второй вариант позволяет выполнить дополнительную подстановку значений скалярных переменных из контекста исполнения, что в некоторых случаях улучшает качество анализа за счёт того, что неафинные выражения становятся афинными.

В Polly имеется генератор кода для OpenMP. Если внешний цикл является параллельным, то его содержимое перемещается в отдельную функцию, с добавлением вызовов функций библиотеки libgomp – GNU релизацией OpenMP. При этом распределение итераций по

ядрам производит среда выполнения и распараллеливается только внешний цикл. Адаптация этого метода для генерации GPU-ядер потребовала следующих изменений:

1. Отображение пространства итераций на нити GPU, с учётом необходимости объединения запросов в память нитей варпа (coalescing transaction).

2. Рекурсивная обработка вложенных циклов с целью использования возможностей GPU по созданию многомерных сеток нитей.

Пусть в заданной группе циклов можно распараллелить N тесно-вложенных циклов. Тогда ядро может быть запущено на решётке с числом измерений N (для CUDA $N \leq 3$). Для каждого измерения, распределяемого между нитями GPU, генерируется код, рассчитывающий положение нити в блоке и блока в сетке. Каждому параллельному циклу ставится во взаимно однозначное соответствие измерение решетки, причём в обратном порядке – внутреннему циклу соответствует измерение X (это позволяет объединять запросы в память). Для каждого параллельного цикла генерируется код, определяющий нижнюю и верхнюю границы части пространства итераций, которая должна быть выполнена нитью. Затем генерируется последовательный код цикла с изменёнными границами и шагом.

4. Заключение

В настоящее время компилятор KernelGen способен производить описанные этапы преобразования и компилировать тестовые программы в корректный PTX-ассемблер. Прежде чем приступить к тестированию более сложных приложений и исследованию производительности, необходимо реализовать общий метод синхронизации данных CPU и GPU, оценочную функцию для переключения между параллельными и последовательными версиями циклов, а также систему сравнения результатов с контрольными версиями циклов. Исходный код системы доступен на сайте проекта: `http://hpcforge.org/projects/kernelgen/`.

Литература

1. The OpenACC^{TM}Application Programming Interface. Version 1.0, November, 2011, `http://www.openacc-standard.org`.

2. Govett M. Development and Use of a Fortran -> CUDA translator to run a NOAA Global Weather Model on a GPU cluster. // Path to Petascale: Adapting GEO/CHEM/ASTRO Applications for Accelerators and Accelerator Clusters. April 2-3, 2009, National Center for Supercomputing Applications. University of Illinois at Urbana-Champaign.

3. Mikushin D. KernelGen – naïve GPU kernels generation from Fortran source code.. COSMO General Meeting, September, 2011, Rome.

4. Grosser T., Zheng H., Aloor R., Simbürger A., Größlinger A., Pouchet L.-N. Polly – Polyhedral Optimization in LLVM. IMPACT 2011 (at CGO 2011), Charmonix France, April 2011.

5. Kravets A., Monakov A., Belevantsev A. GRAPHITE-OpenCL: Automatic parallelization of some loops in polyhedra representation. // GCC Developers' Summit, GCC Developers' Summit. October 25-27, 2010, Ottawa, Canada.

6. Verdoolaege S., et al. PPCG – C to CUDA processor

7. Bastoul C. Code Generation in the Polyhedral Model Is Easier Than You Think. // PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques. September, 2004, Juan-les-Pins, France.

8. Murphy M. NVIDIA's Experience with Open64. // Open64 Workshop at CGO 2008, April 6, 2008, Boston, Massachusetts.

9. Open64 compiler.

10. Chris Lattner LLVM: An Infrastructure for Multi-Stage Optimization. Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Dec. 2002.

11. Sands D. Reimplementing llvm-gcc as a gcc plugin. // Third Annual LLVM Developers' Meeting. October 2, 2009, Apple Inc. Campus, Cupertino, California.

12. Wolfe M., Toepfer C. The PGI Accelerator Programming Model on NVIDIA GPUs Part 3: Porting WRF.

13. Jeff Squyres J., Bosilca G., Sumimoto S., vandeVaart R. Open MPI State of the Union. // Open MPI Community Meeting, Supercomputing 2011.

Молекулярно-динамическое моделирование наномасштабного пузырька пара в воде*

В.Л. Малышев[1], К.И. Михайленко[1,2], Е.Ф. Моисеева[1]

Центр микро- и наномасштабной динамики дисперсных систем, Башгосуниверситет, Уфа[1], Институт механики Уфимского научного центра РАН, Уфа[2]

Записана математическая модель полярной жидкости (воды) с использованием молекулярной динамики. Межмолекулярное взаимодействие осуществляется по двум механизмам: на основе потенциала Леннарда-Джонса и электромагнитного взаимодействия между отдельными атомами. Структура молекулы воды построена по модели TIP4P. Для указанной модели написан параллельный код.
Верификация программы производилась посредством сравнения расчетных данных для жидкости и пара после установления равновесного состояния с известными экспериментальными результатами.
Проведены вычислительные эксперименты по возникновению и динамике наномасштабных паровых пузырьков под действием знакопеременного давления в рассматриваемой области.

1. Введение

В настоящее время наметилась тенденция использования методов молекулярной динамики при математическом моделировании микро- и наномасштабных явлений и объектов. Такой подход определяется тем, что характерные размеры исследуемых объектов или областей не превышают нескольких десятков нанометров, то есть их размер сравним с длиной волны света видимого спектра, что создаёт серьёзные трудности для непосредственного экспериментального наблюдения. Также для таких малых областей невозможно и адекватное применение классических моделей сплошной среды, что связано с малым количеством молекул, заполняющих наномасштабную область. Поэтому численное исследование методами молекулярной динамики объектов и явлений с характерными размерами менее 100 нм нередко является единственно возможным прямым методом исследований.

В представленной работе делается попытка практической верификации метода молекулярной динамики для моделирования водяного пара и воды при термодинамическом равновесии и малых отклонениях от него. Рассмотрены состояния воды в достаточно широком диапазоне плотностей и температур, допускающем существование как жидкой воды, так и пара. Показаны процессы испарения и конденсации, приводящие к восстановлению термодинамического равновесия при малых отклонениях от него. Проведено сравнение полученных результатов с теоретическими и экспериментальными данными.

Со времени своего появления, метод моделирования, основанный на молекулярной динамике, отличается особой простотой модели, что, с другой стороны, вызывает определённые сомнения в вопросах границ его применимости. В этой связи существует большое количество работ по верификации молекулярной динамики.

В 1979 году Николас с коллегами [1] в своей работе рассчитали и представили таблицы значений конфигурационной энергии и давления для широкого диапазона значений плотности и температуры, а так же получили кривые насыщения для системы жидкость—пар, содержащей 256 атомов. В качестве исследуемой системы рассматривался аргон.

В статье Брауна и Кларка [2] приведен сравнительный анализ динамических и статических свойств жидкости при плотности и температуре, близких к тройной точке для 4х различных ансамблей.

Одна из последних работ по данной теме, это статья Ландри и др. [3], в которой ис-

*Работа выполнена при поддержке Министерства образования и науки РФ (грант 11.G34.31.0040).

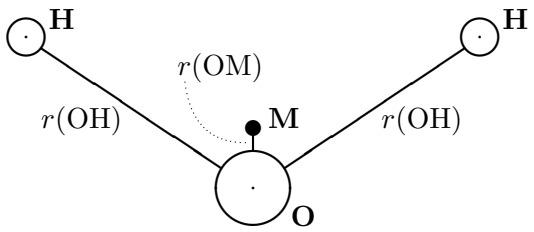


Рис. 1. Схема модели TIP4P молекулы воды.

следовался процесс испарения капли жидкого аргона, окруженного газообразным аргоном при температуре и давлении, заданных в широком диапазоне значений.

2. Математическая модель

Для моделирования воды нами использована модель TIP4P, впервые опубликованная в 1983 году [4]. Данная модель имеет дополнительный четвертый безмассовый узел (точка M), обладающий лишь зарядом, расположенным вблизи кислорода на биссектрисе угла HOH, как это показано на рис. 1. Такое расположение зарядов позволяет улучшить электростатическое распределение вокруг молекулы и хорошо воспроизводит экспериментально установленные свойства воды.

В описанной модели используется четыре потенциала. Взаимодействие кислород—кислород описывается потенциалом Леннарда-Джонса:

$$O - O: \quad u = 4(r^{-12} - r^{-6})$$

кроме этого имеются чисто электростатические взаимодействия точка M—точка M, точка M—водород и водород—водород:

$$M - M: \quad u = 4b/r,$$

$$M - H: \quad u = -2b/r,$$

$$H - H: \quad u = b/r.$$

Такое распределение взаимодействий определяется тем, что основная масса молекулы воды сосредоточена в точке, принятой за центр атома кислорода, тогда как заряды располагаются в точке M (отрицательный) и в центрах атомов водорода (положительные, равные по модулю половине заряда точки M каждый).

3. Алгоритм решения

Существует несколько равноправных алгоритмов расчёта движения молекулы под действием суперпозиции потенциалов окружающих молекул. Для решения математической модели динамики воды нами выбран численный метод предиктор — корректор (Adams—Bashforth—Moulton), описанный в [5].

Идея метода заключается в использовании ранее вычисленных значений на предыдущих шагах по времени $t, t-h, \ldots, t-kh$ для нахождения нового значения на следующем шаге по времени $t + h$. Изначально, выбирая число k, мы устанавливаем глубину используемых значений с предыдущих шагов, повышая точность алгоритма. Наиболее часто используется значение $k = 2$.

Метод состоит из первоначального вычисления значений координаты и скорости (предиктор)

$$P(x): \quad x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k+1} \alpha_i f(t + [1-i]h),$$

$$P(\dot{x}): \quad h\dot{x}(t+h) = x(t+h) - x(t) + h^2 \sum_{i=1}^{k+1} \alpha'_i f(t + [1-i]h)$$

и их последующей корректировки

$$C(x): \quad x(t+h) = x(t) + h\dot{x}(t) + h^2 \sum_{i=1}^{k+1} \beta_i f(t + [2-i]h),$$

$$C(\dot{x}): \quad h\dot{x}(t+h) = x(t+h) - x(t) + h^2 \sum_{i=1}^{k+1} \beta'_i f(t + [2-i]h).$$

Здесь α_i, α'_i, β_i, β'_i — некоторые коэффициенты метода, характерные для заданной величины k; а функция f описывает характер зависимости значений рассчитываемых величин на новом шаге от предыдущих.

Аналогично описанному выше двухэтапному методу вычисления поступательного движения молекул, вычисляется и их вращение. В этом случае в качестве координат частиц выступают кватернионы.

В качестве начальных условий для моделирования традиционно используется куб с сеточным расположением частиц. Количество молекул определяется размерами куба и заданной плотностью, каждая молекула случайным образом поворачивается в пространстве. Также частицам придаются некоторые случайные значения поступательной и угловой скоростей, определяемые заданной температурой. Когда такая система «отпускается», уже через небольшое количество расчётных шагов частицы в достаточной мере хаотизируются, чтобы полученные результаты можно было считать независимыми от начальных координат.

Граничные условия выбираются исходя из постановки задачи. Наиболее распространённый вариант — периодические граничные условия, при которых покидающая область молекула возвращается с противоположной стороны области с той же скоростью и ускорением (модель бесконечной области). Также нами использовались термостатирующие граничные условия, при которых достигшая границы молекула зеркально отражается от неё по правилам соударения, однако приобретает скорость, по модулю соответствующую заданной на границе температуре. Использование термостатирующих граничных условий приводит к тому, что через некоторое время система приобретает среднюю температуру, определяемую температурой стенки.

4. Методы увеличения производительности

Расчёты по методу молекулярной динамики являются чрезвычайно ресурсоёмкими. Особенно значительные требования к ресурсам предъявляют модели, в которых используются дальнодействующие потенциалы, например, потенциал электростатического взаимодействия, имеющий порядок r^{-1}. В этом случае подход с введением некоторого радиуса обрезания, за пределами которого потенциал принимают равным нулю и не учитывают в расчётах, не будет работать. Более того, для наномасштабных областей моделирования электростатический потенциал будет иметь заведомо значимую величину в пределах всей расчётной области. Таким образом, моделирование полярных молекул приводит к значительному росту вычислительной сложности до $O(N)$.

Существует несколько равноправных подходов к увеличению производительности программ, предназначенных для молекулярно-динамического моделирования веществ, содержащих полярные молекулы.

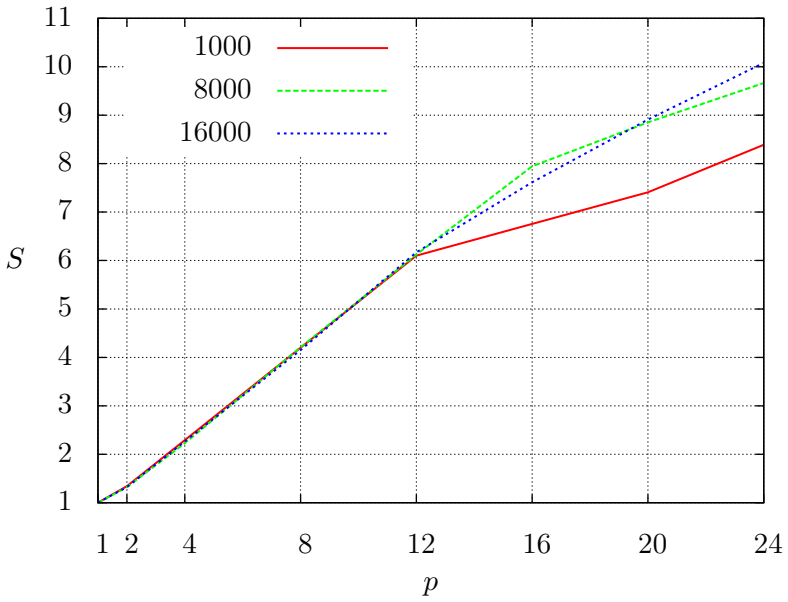


Рис. 2. Ускорение вычислительного процесса в зависимости от количества используемых вычислительных ядер p для задач разного размера.

В первую очередь следует отметить стандартные экстенсивные методы ускорения вычислений, такие как использование OpenMP и MPI. В случае описываемой в работе модели с электростатическим потенциалом, использование OpenMP весьма просто, удобно и оправдано. Использование MPI наоборот, затруднено, если в используемом потенциале отсутствует радиус обрезания. Именно поэтому нами для ускорения вычислений в настоящее время выбран OpenMP.

Вычисления производились на компьютере, содержащем два процессора Intel Xeon X5660 и 24 Gb оперативной памяти. Таким образом, при включенном гипертредининге в нашем распоряжении имелось до 24 вычислительных ядер. Компьютер работает под управлением 64-разрядной системы ALT Linux, компиляция OpenMP-приложения производилась компилятором gcc (gfortran) версии 4.5.1.

На рис. 2 представлены результаты исследования производительности параллельного приложения. Приведены ускорения вычислительного процесса S для различного числа используемых вычислительных ядер p для трёх задач разного размера, моделирующих движение 1000, 8000 и 16000 молекул воды соответственно. На графике можно заметить перегиб, вохникающий при использовании бóльшего, чем 12 количества ядер. Такой результат легко объясняется подключением ядер, основанных на гипертредининге и возникновения соответствующих задержек. Однако при увеличении вычислительной системы и, соответственно, при увеличении времени параллельной работы, эффективность использования такой многоядерной системы возрастает.

При дальнейшем развитии вычислительного кода предполагается использование ещё двух методов ускорения вычислений. Во-первых, планируется воспользоваться современными GPU, такими как nVidia Tesla 2070. Кроме того, планируется использование интенсивного метода ускорения вычислений, в частности, FMM [6]. Использование FMM для моделирования задачи многих тел позволяет снизить вычислительную сложность до величины $O(N \log N)$. Именно такой задачей многих тел является молекулярно-динамическая модель с электростатическим потенциалом.

Совместное использование FMM и нескольких вычислителей nVidia Tesla 2070 позволит производить моделирование нескольких миллионов молекул воды, то есть пространственные области и объекты уже не наномасштабных, а микронных размеров.

5. Результаты численного моделирования

На рис. 3 показан один из моментов численного моделирования процесса возникновения пузырька пара. Для этого в области, заполненной жидкостью в состоянии термодинамического равновесия постепенно понижается давление. Каждый шаг изменения давления производится после установления термодинамического равновесия в области. Через некоторое время достигается состояние перегретой жидкости. На рис. 4 описанный процесс хорошо виден. Здесь синяя кривая описывает процесс изменения безразмерной плотности жидкости ρ_ℓ^*, зависящей от давления в зависимости от времени. Красная кривая описывает изменение плотности пузырька ρ_v^*. Для жидкости хорошо видны два участка: наклонный участок, определяемый постепенным снижением давления в области и горизонтальный, когда снижение давления прекращено, жидкость находится в метастабильном состоянии.

Через некоторое время в жидкости возникает достаточная для образования зародыша парового пузырька флуктуация плотности. С этого момента плотность жидкости слегка увеличивается, так как часть области теперь занята пузырьком. Также с этого момента на рисунке показана плотность в пузырьке (красная кривая). По причине весьма незначительного размера парового пузырька, в нем постоянно наблюдаются процессы конденсации и испарения жидкости внутрь пузырька. Эти процессы можно наблюдать в виде непрерыв-

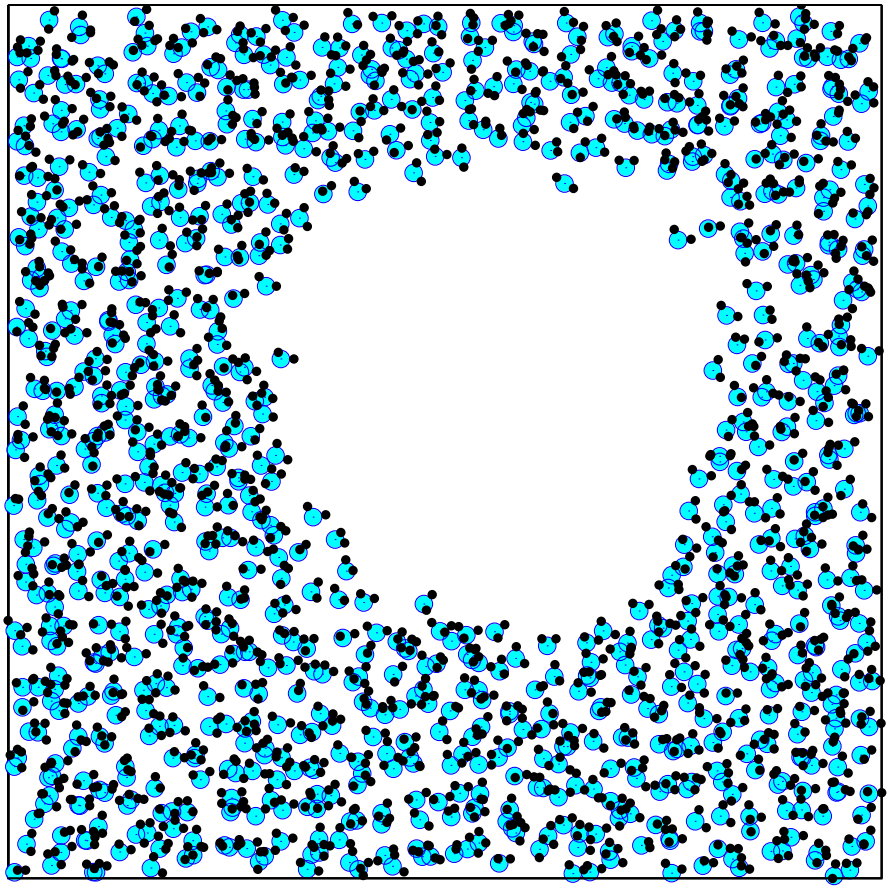


Рис. 3. Зарождение пузырька в воде (диаметр зародыша 3 нм).

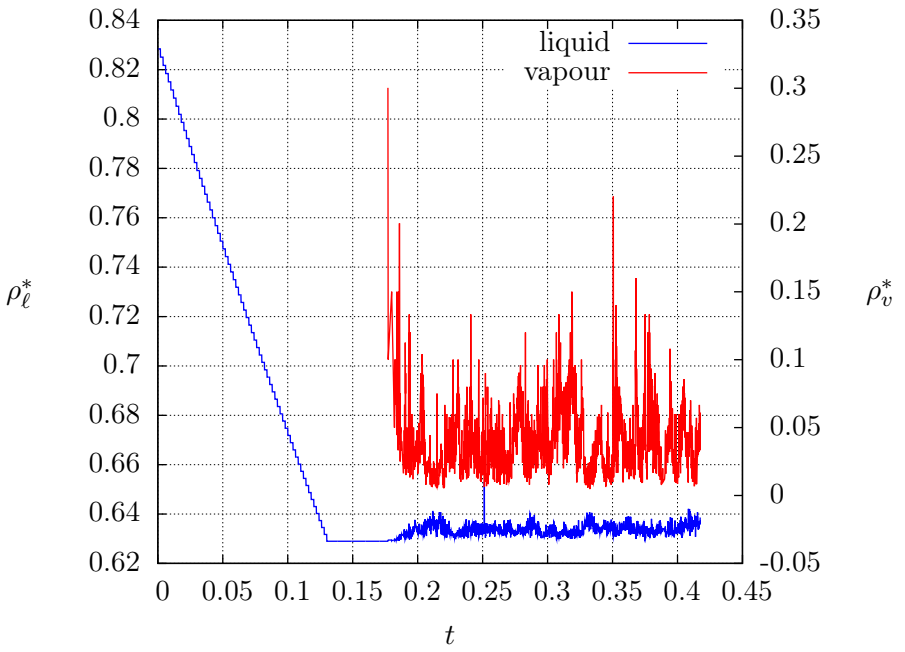


Рис. 4. Моделирование зарождения парового пузырька в жидкости. Синяя кривая показывает изменение безразмерной плотности жидкости, красная — изменение безразмерной плотности пара.

ного изменения плотности как жидкости, так и пара, хорошо заметные на графике в виде хаотического колебания графиков.

6. Заключение

В работе показано, что методы молекулярной динамики позволяют с высокой точностью моделировать процессы, связанные с фазовыми переходами в микро- и наномасштабных пространственных областях. Также из результатов работы можно видеть, что возможно не только моделировать взаимное стохастическое движение отдельных молекул, но и на основе этого движения восстанавливать термодинамические макропараметры, такие как температура, давление, плотность, энергия, скорость среды. В дальнейшей работе планируется воспроизвести образование, динамику и кавитацию нанопузырька на поверхности твёрдого тела, влияние гидрофобности на указанные процессы.

Список литературы

1. Nicolas J. J., Gubbins K. E., Streett W. B., Tildesley D. J. Equation of state for the Lennard-Jones fluid // Molecular Phys/: An Int. J. at the Interface Between Chem. and Phys. 1979. Vol. 37. N. 5. P. 1429–1454.

2. Brown D., Clarke J. H. R. A Comparison of Constant Energy, Constant Temperature and Constant Pressure Ensembles in Molecular-Dynamics Simulations of Atomic Liquids // Molecular Phys. 1984. Vol. 51. P. 1243–1252.

3. Landry E. S., Mikkilineni S., Paharia M., McGaughey A. J. H. Droplet evaporation: A molecular dynamics investigation // J. of Appl. Phys. 2007. Vol. 102. N. 12. P. 124301–7.

4. Jorgensen W. L., Chandrasekhar J., Madura J. D., Impey R. W., Klein M. L. Comparison of simple potential functions for simulating liquid water // J. Chem. Phys. 1983. Vol. 79. P. 926-935.

5. Rapaport D. C. The Art of Molecular Dynamics Simulation. Cambridge University Press, 1995. 549 p.

6. Gumerov N. A., Duraiswami R. Fast multipole methods on graphics processors // J. of Computational Phys. 2008. Vol. 227. P. 8290-8313.

Технология фрагментированного программирования*

В.Э. Малышкин

Институт вычислительной математики и математической геофизики СО РАН

Новосибирский государственный университет

Кратко представлена технология фрагментированного программирования и реализующие её язык и система фрагментированного программирования LuNA, разрабатываемые в ИВМиМГ СО РАН. Технология ориентирована на поддержку разработки параллельных программ, реализующих большие численные модели, и их исполнения на суперкомпьютерах. Система LuNA автоматически обеспечивает такие динамические свойства параллельных программ как динамическая настройка на все доступные ресурсы, динамическая балансировка нагрузки, учет динамики поведения моделируемого явления и т.п.

1. Введение

В течении последних 15 лет в ИВМиМГ СО РАН велись работы по созданию методов и средств параллельной реализации больших численных моделей на суперкомпьютерах, а также параллельной реализации таких моделей. Накопленный опыт позволил сформировать идеи технологии фрагментированного программирования. Основная проблема параллельной реализации больших численных моделей состоит в том, что необходимая сложность программирования заметно превосходит квалификацию программистов, работающих обычно в численном моделировании, так как в программе моделирования понадобилось реализовывать динамические системные функции.

Чтобы преодолеть эту проблему, нужна поддержка процесса параллельного программирования таких моделей. Необходимая технология – технология фрагментированного программирования и реализующие её язык и система программирования LuNA - разработаны в ИВМиМГ СО РАН.

Поддерживая технику разработки программ численного моделирования также хотелось бы обеспечивать:

- должную степень непроцедурности [1] представления алгоритма в параллельной программе (не знать MPI, свойства вычислителя и его коммуникационной сети, методы и средства параллельного программирования и т.д.),
- неизменность алгоритма, его независимость от оборудования (раздельное описание алгоритма и реализующей его программы), автоматическая генерация фрагментированной программы, что требуется для обеспечения её переносимости,
- автоматическое включение реализации динамических свойств в прикладные параллельные программы, такие как динамическая балансировка нагрузки, настраиваемость на все доступные ресурсы вычислителя, динамическое распределение ресурсов, выполнение коммуникаций на фоне счета, учет поведения моделируемого явления.

Теоретическую базу проекта LuNA составляет теория синтеза параллельных программ на вычислительных моделях [2]. В проекте системы LuNA учтен опыт разработки как больших численных моделей [3,4, 13, 14], так и различных систем сборочного программирования в мире [5-8]. Текущие технологические результаты представлены в настоящей статье. Теоретические аспекты проекта не рассматриваются. Более ранние результаты опубликованы в [9-11].

* Работа выполнена при частичной поддержке РФФИ, грант 10-07-00454-а

2. Идеология системы фрагментированного программирования LuNA

Перечисленные мотивы, и ряд других, после общего анализа трансформировались в следующие исходные проектные решения в системе фрагментированного программирования LuNA.

2.1 Основные проектные решения

1). Технология фрагментированного программирования поддерживает процесс сборки целой программы из фрагментов вычислений (модулей, процедур, их входных/выходных фрагментов данных и т.п.) и ее исполнение.
2). Каждый фрагмент вычислений - независимая единица программы (рис. 1), содержит описание входных/выходных переменных и кода (модуля, процедуры) фрагмента.
3). Фрагментированная программа – это рекурсивно перечислимое множество фрагментов вычислений и их входных/выходных переменных.

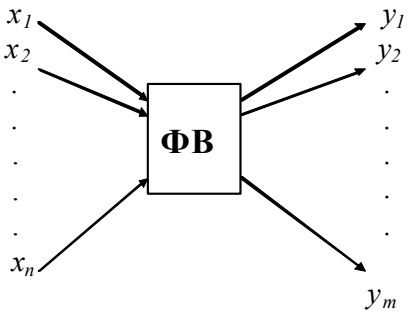


Рис. 1. Фрагмент вычислений

Таким образом, фрагментированная программа определяется как множества переменных (фрагментов данных) и фрагментов вычислений. К фрагменту вычислений можно обращаться по-разному, например, как к обычной процедуре в последовательном языке программирования

4). В отличие от технологии модульного программирования, в ходе исполнения фрагментированная структура программы сохраняется. Каждый фрагмент вычислений определит в ходе исполнения независимо исполняющийся процесс программы, взаимодействующий с другими процессами.
5). Следуя С.Клини [12], в качестве базового представления алгоритма взято рекурсивно перечислимое множество функциональных термов. Фрагментированный алгоритм при необходимости извлекается алгоритмом вывода из множества фрагментов вычислений.

2.2 Исполнение фрагментированной программы

Базовый алгоритм:

- Фрагмент вычислений исполняется, если все его входные переменные получили значения
- После выполнения фрагмента вычислений получают значения его выходные переменные.
- Алгоритм может реализоваться либо управлением в сгенерированной программе, либо run-time системой. В системе LuNA для обеспечения необходимой степени асинхронности выбрано исполнение фрагментированной программы run-time системой.

Эти правила определяют асинхронное исполнение фрагментированной программы, в которой порядок исполнения фрагментов вычислений определяется лишь информационными зависимостями между ними. При исполнении фрагментированной программы run-time система ищет лучшие способы исполнения фрагментированного алгоритма.

3. Шаги разработки ФП

Фрагментированная программа разрабатывается в несколько последовательных этапов.

- Разработка исходного алгоритма решения задачи.
- Фрагментация исходного алгоритма. Фрагментация рассматривается здесь как универсальный способ распараллеливания алгоритмов, что позволяет поддержать его реализацию системой программирования (примеры фрагментации приведены в разделе 4). Фрагментация нередко является очень сложной работой. Например, фрагментация алгоритма прогонки [15] заняла 2 года и завершилась защитой кандидатской диссертации. На тестах получено ускорение исполнения алгоритма прогонки в 6000 раз на 8000 процессоров [15].
- Описание фрагментированной программы на языке LuNA. Входной язык системы LuNA устроен просто, в него в основном включены средства для определения множеств фрагментов данных и вычислений, синтаксис языка не очень интересен. Пример фрагментированной программы на языке LuNA можно видеть в разделе 4.1.
- Компиляция и анализ фрагментированного алгоритма. Генерация платформо-ориентированной фрагментированной программы.
- Исполнение фрагментированной программы.

4. Примеры фрагментированных алгоритмов

Несколько примеров поясняют технологию фрагментированного программирования и проблемы разработки системы LuNA.

4.1 Исходный алгоритм умножения квадратных матриц

На рис.2 представлен исходный алгоритм умножения квадратных матриц. Вычисления проводятся по формулам:

$$c_{i,j} = \sum_{k=1}^{N} a_{i,k} \times b_{k,j}, \quad i, j = 1, 2, \ldots, N.$$

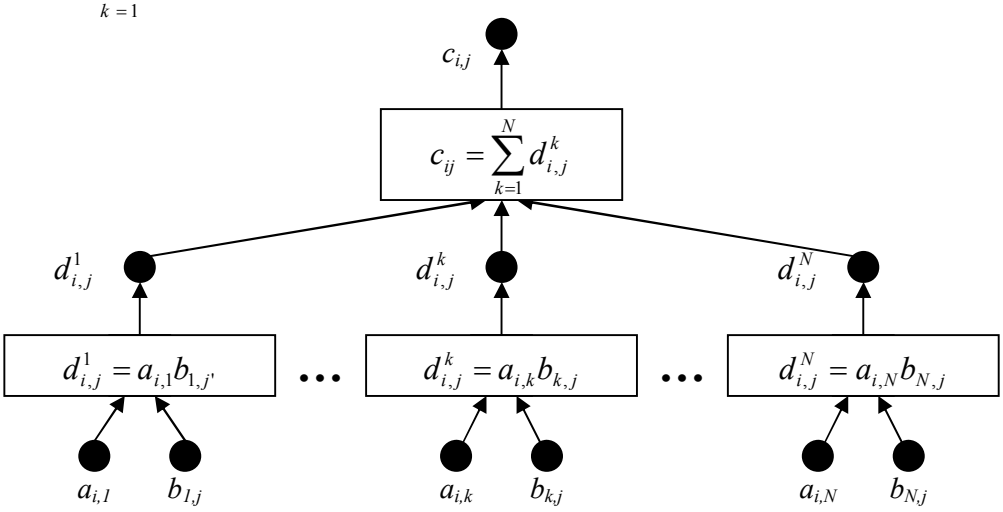


Рис. 2. Множество функциональных термов, представляющих исходный алгоритм

Так как множество функциональных термов хранить нехорошо, то в языке LuNA вместо множества функциональных термов определяются множества фрагментов вычислений и фрагментов данных, а нужные термы конструируются из них по мере необходимости алгоритмом вывода.

4.2 Фрагментированный алгоритм умножения квадратных матриц

В системе LuNA исходный алгоритм может быть запрограммирован как фрагментированный, т.е. каждая операция $c_{ij} = \sum_{k=1}^{N} d_{i,j}^{k}$ объявлена фрагментов вычислений (и будет реализо-

ваться как независимый процесс программы), переменные a_{il}, b_{lj}, c_{ij}, d_{ij} объявлены фрагментами данных. Но такая фрагментированная программа будет исполняться с большим замедлением, примерно с 1000 кратным, по сравнению с программой с использованием MPI из-за больших расходов на реализацию управления. Поэтому для численных алгоритмов, отличающихся высоко регулярностью, в процессе фрагментации проводится агрегация и переменных и операций. На рис. 3 представлена схема фрагментации алгоритма умножения матриц, которая показывает способ агрегации данных и вычислений, а на рис. 4 – фрагментированный алгоритм. Здесь:

$$C_{I,J} = \sum_{L=1}^{K} A_{I,L} \times B_{L,J}, \qquad C_{I,J}, A_{I,L}, B_{L,J}, \quad I,J,L = 1,2,...,K$$

Даже в таком простом примере, как алгоритм умножения матриц, информационные зависимости не определяют хорошего исполнения фрагментированной программы. Например, если для всех I и J выполнить сначала все, кроме K-го, фрагменты вычислений $D^L_{I,J} = A_{I,L}B_{L,J}$, а потом исполнить все K-ые фрагменты, то память вычислителя должна будет хранить все выработанные, но своевременно не потребленные, промежуточные данные. Объём хранимых промежуточных данных в K раз большие чем может понадобиться при хорошей организации вычислений. В результате будет ограничен размер решаемой задачи и замедлится исполнение программы. Поэтому основными задачами, решаемыми run-time системой, являются распределение ресурсов и выбор наиболее подходящего очередного фрагмента вычислений на исполнение.

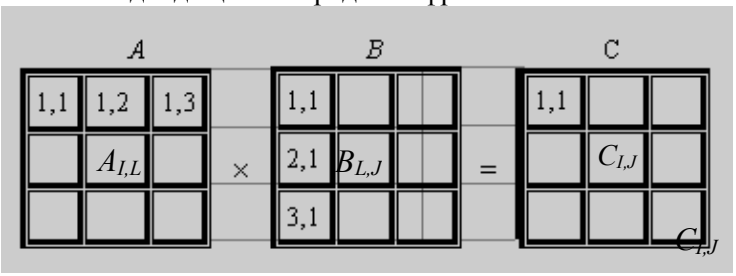


Рис. 3. Агрегация данных и фрагментов вычислений исходного алгоритма

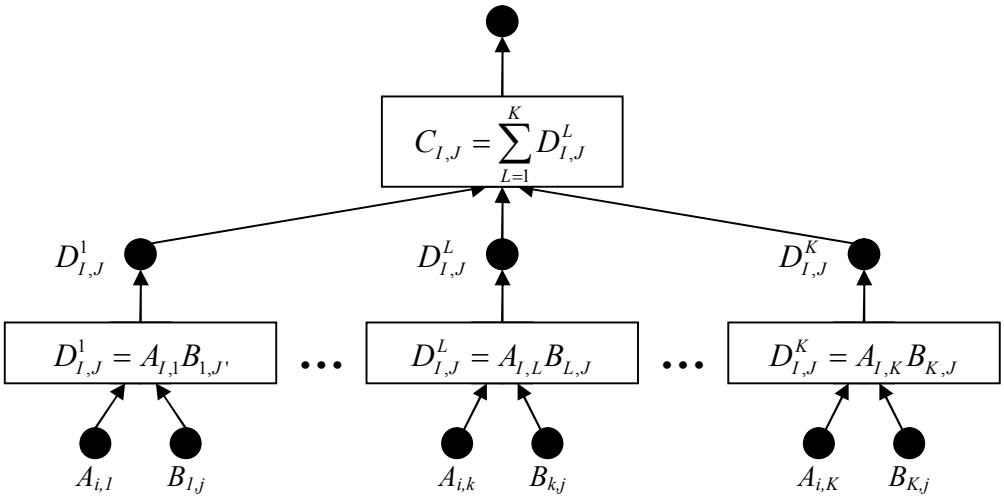


Рис. 4. Фрагментированный алгоритм

В качестве переменных фрагментированной программы используются агрегаты переменных исходного алгоритма, составляющих подматрицы исходной матрицы. Аналогично, код фрагмента вычислений является агрегатом кодов операций исходного алгоритма. В системе LuNA конструируются программы, в которых размер фрагментов данных является входным параметром. Ниже в качестве примера приведена часть фрагментированной программы умножения матриц, записанная в языке LuNA:

Множество фрагментов данных:

```
df a[i,k] := block(4*M*M) | i=0..K-1, k=0..K-1;
df b[k,j] := block(4*M*M) | k=0..K-1, j=0..K-1;
```

```
df c[i,j] := block(4*M*M) | i=0..K-1, j=0..K-1;
df d[i,j,k] := block(4*M*M) | i=0..K-1, j=0..K-1, k=0..K-1;
```

Множество фрагментов вычислений

```
cf initc[i,j] := proc_zero<M,M> (out: c[i,j])
| i=0..K-1, j=0..K-1;
cf mul[i,j,k] := proc_mmul<M,M,M> (in: a[i,k],b[k,j];
out: d[i,j,k]) | i=0..K-1, j=0..K-1, k=0..K-1;
cf sum[i,j,k] := proc_add<M,M> (in: d[i,j,k],c[i,j]; out: c[i,j])
| i=0..K-1, j=0..K-1, k=0..K-1;
```

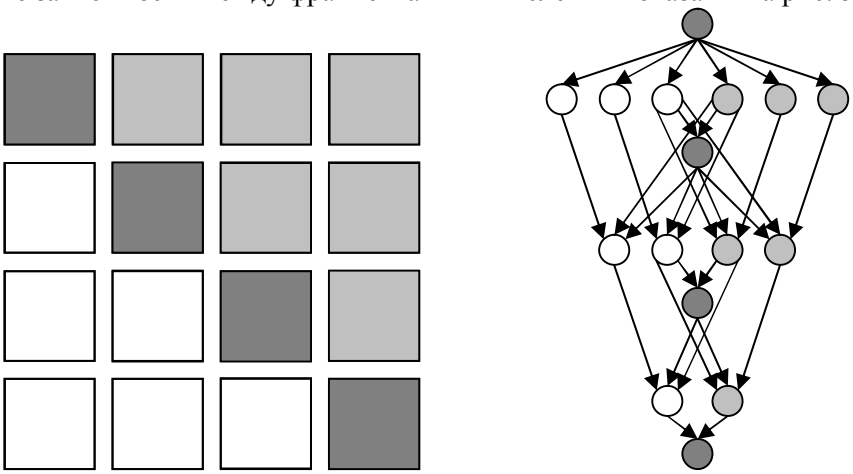
4.3. LU-разложение

LU-разложение преобразует квадратную матрицу A по формулам $l_{i,j} = a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j}$,

$u_{i,j} = \frac{1}{l_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j} \right)$ к виду $A=LU$, где:

$$L = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \ldots \\ l_{31} & l_{32} & l_{33} \\ \ldots & & \end{pmatrix} \qquad U = \begin{pmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \ldots \\ 0 & 0 & 1 \\ \ldots & & \end{pmatrix}$$

Матрица A делится на фрагменты данных (рис. 5*a*), каждый фрагмент данных – подматрица матрицы A, для обработки каждого фрагмента данных формируется фрагмент вычислений. Фрагменты вычислений должны выполняться в следующем порядке (таковы информационные зависимости): вначале исполняется фрагмент (1,1), затем могут выполняться все фрагменты первого столбца и первой строки, затем может исполняться фрагмент (2,2) и т.д. Информационные зависимости между фрагментами вычислений показаны на рис. 5.*b*.



a) *b)*

Рис. 5. Фрагментация вычислений *a)* алгоритма LU-разложения и информационные зависимости между фрагментами вычислений *b)*

Этот порядок нехорош тем, что создает неравномерную нагрузку на вычислитель: есть моменты времени, когда только один фрагмент вычислений готов к исполнению, и есть моменты времени, когда много более одного фрагмента готовы исполняться.

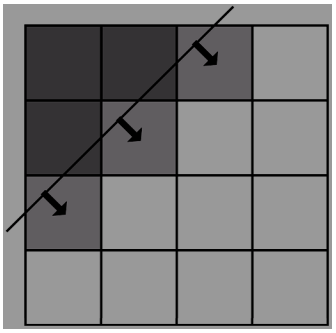


Рис. 6. Исполнение фрагментов вычислений гиперплоскостями

Лучше было бы организовать исполнение фрагментов вычислений по гиперплоскостями (рис. 6), при котором формируется более равномерная нагрузка процессоров вычислителя.

4.3. Неравномерность загрузки процессоров в модели эволюции облака пыли

Параллельная реализация модели эволюции протопланетного диска описана в [13, 14]. На рис. 7 показана неравномерность распределения нагрузки на узлы мультикомпьютера в процессе моделирования. Каждый прямоугольник изображает фрагмент вычислений и потребляемые им ресурсы. Динамическая балансировка нагрузки узлов вычислителя планируется run-time системой и реализуется миграцией фрагментов вычислений с перегруженного узла на соседние, менее загруженные. На этом основана автоматическая реализация динамических свойств фрагментированной программы.

Планирование, точный расчет балансировки нагрузки не делается, а моделируется физический процесс диффузии в жидких и/или газообразных средах. Средой здесь является множество фрагментов вычислений. Точный расчет балансировки нагрузки в условиях динамически меняющегося состояния системы взаимодействующих процессов фрагментированной программы не имеет смысла, и, как минимум, не технологичен.

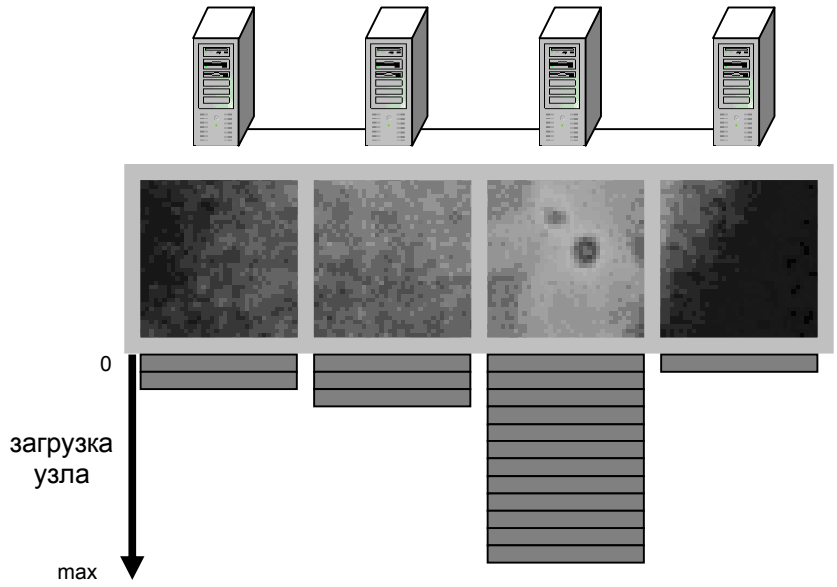


Рис. 7. Неравномерная нагрузка узлов вычислителя

Завершая обсуждение фрагментации численных алгоритмов, необходимо указать на одну важную качественную характеристику исполнения фрагментированной программы (рис. 8).

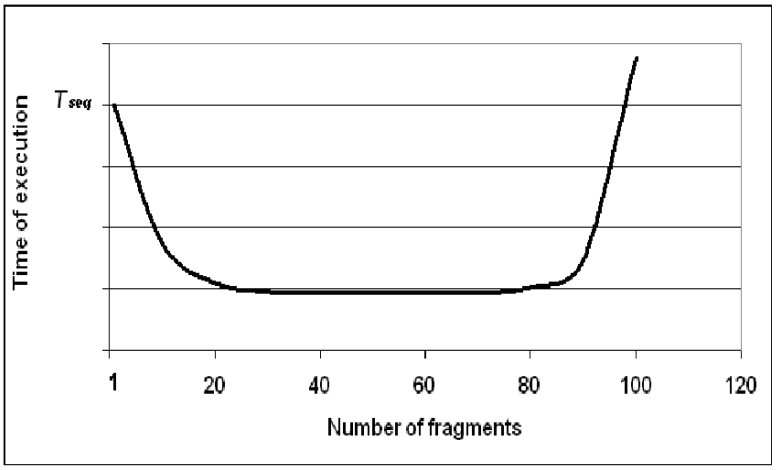


Рис. 8. Качественный график времени исполнения фрагментированной программы

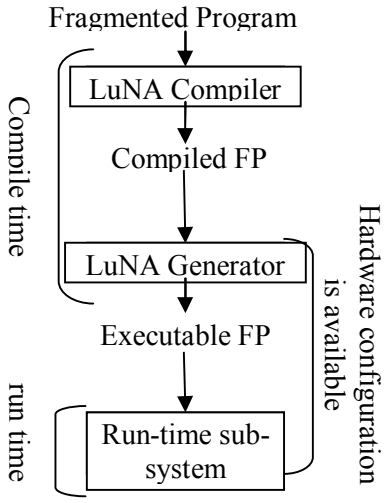


Рис. 9. Компоненты системы LuNA

Фрагментированная программа исполнялась в одном узле, сравнение производилось с последовательной программой. На рисунке T_{seq} – это время исполнения фрагментированной программы, состоящей из одного фрагмента, т.е. это время исполнения последовательной программы. Затем число фрагментов данных увеличивалось (соответственно, размер фрагментов данных уменьшался), при этом наблюдалось уменьшение времени исполнения программы. Минимальное время исполнения программы получалось, когда фрагмент вычислений со всеми обрабатываемыми им фрагментами данных попадал целиком в кэш-память. Увеличение времени исполнения программы начиналось с ростом числа фрагментов данных и вычислений, что приводило к увеличению накладных расходов на реализацию управления и динамического распределения ресурсов.

5. Язык и система фрагментированного программирования LuNA

Входной язык LuNA - теоретико-множественный, единственного присваивания и единственного исполнения фрагментов вычислений. Фрагменты данных и вычислений задаются рекурсивно перечислимыми множествами с использованием индексных выражений. Управление в LuNA-программе задается отношением частичного порядка на множестве фрагментов вычислений.

Отношение соседства на множестве фрагментов данных определяет, какие фрагменты данных должны размещаться в одном либо в соседних узлах. Дополнительно имеются операторы-рекомендации по распределению ресурсов вычислителя, по определению требуемого порядка исполнения фрагментов вычислений. Средства задания приоритетов исполнения фрагментов вычислений используются run-time системой для выбора наиболее подходящего фрагмента на исполнение.

Функциональная структура системы фрагментированного программирования LuNA представлена на рис. 9. Язык, некоторые теоретические и технологические аспекты системы программирования LuNA описаны в ряде публикаций [10, 11] .

LuNA имеет три уровня преобразования фрагментированного алгоритма в программу: Компиляция, Генерация и Исполнение:

- Компилятор принимает решения (и вносит их в формируемую программу), которые можно принять, используя только информацию о свойствах алгоритма.

- Генератор принимает решения, которые зависят от свойств конкретного вычислителя (количество и типы доступных ресурсов, производительность ресурсов, нумерация узлов и т.п.), на котором программа должна исполняться.

- Run-time система принимает те решения, которые могут быть сделаны только динамически, в ходе вычислений. В их числе выбор фрагмента вычислений на исполнение, динамическая балансировка нагрузки узлов вычислителя, динамическое распределение ресурсов, включая назначение процессора для исполнение фрагмента вычислений и многое другое.

В системе LuNA есть еще один компонент – профилировщик, который собирает реальную информацию о ходе исполнения фрагментированной программы. Эта информация используется затем для улучшения последующих исполнений программы.

6. Родственные работы

Общее представление о работах, посвященных сборке программ из готовых фрагментов, дают проекты [5-8]. Проект *Charm* [5] за долгое время развития стал хорошо известной системой программирования. Её основным недостатком является отсутствие глобальной оптимизации исполнения и не соответствующие входной язык и технология программирования, не позволяющие использовать в полной мере достоинства системы программирования, что свойственно и другим проектам

Литература

1. Вальковский В.А., Малышкин В.Э. К уточнению понятия непроцедурности языков программирования. // Кибернетика, № 3, 1981г. стр. 55
2. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. Издат. "Наука", Сибирское отделение, Новосибирск, 1988г. 129 стр.
3. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. // International Journal on Future Generation Computer Systems. Vol. 17 (2001), No. 6, pp. 755-765.
4. V.Malyshkin. Assembling of Parallel Programs for Large Scale Numerical Modeling. IGI Global, USA, 2010, 1021 pp. The Handbook of Research on Scalable Computing Technologies. Chapter 13, pp. 295 – 311. ISBN 978-1-60566-661-7
5. Charm++**.** URL: http://charm.cs.uiuc.edu/papers
6. ProActive Parallel Suite. URL: http://proactive.inria.fr/
7. S-Net home page. — URL: http://www.snet-home.org/
8. Berzins M., Meng O., Schmidt J., Sutherland J.C. DAG-Based Software Frameworks for PDEs. URL: http://www.csafe.utah.edu/pdf/papers/2011_Berzins_Meng_Schmidt_Sutherland_(DAG-Based_Software_Frameworks_for_PDEs).pdf
9. Kireev S., Malyshkin V. Fragmentation of numerical algorithms for parallel subroutines library // The Journal of Supercomputing, Springer, Volume 57, Number 2, August 2011, pp. 161-171
10. Kireev S., Malyshkin V., Fujita H.. The LuNA Library of Parallel Numerical Fragmented Subroutines // PaCT-2011 proceedings, Springer, LNCS 6873 (2011), pp. 290-301
11. Malyshkin V., Perepelkin V. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System. // The Journal of Supercomputing. pp. 1-14. DOI: 10.1007/s11227-011-0649-6
12. Клини С. Введение в метаматематику. Иностранная литература, М.: 1957
13. Киреев С.Е. Параллельная реализация метода частиц в ячейках для моделирования задач гравитационной космодинамики // Автометрия. 2006. №3. с. 32-39.
14. Kireev S.E. A Parallel 3D Code for Simulation of Self-gravitating Gas-Dust Systems // PaCT-2009 Proceedings, Springer, LNCS 5698 (2009), pp. 406–413.
15. Terekhov A.V. Parallel Dichotomy Algorithm for solving tridiagonal system of linear equations with multiple right-hand sides // Parallel Computing, Volume 36, Issue 8, 2010. P. 423-438.

Библиотека PARMONC для решения «больших» задач по методу Монте-Карло*

М.А. Марченко

Институт вычислительной математики и математической геофизики СО РАН

В работе представлена библиотека PARMONC (сокращение от «PARallel MONte Carlo»), предназначенная для распараллеливания широкого круга приложений метода Монте-Карло, обладающих большой вычислительной трудоемкостью. Цель разработки библиотеки - создать простой в использовании программный инструмент для реализации распределенных вычислений, не требующий от пользователя знания языка MPI. «Ядром» библиотеки является тщательно протестированный, быстрый и надежный длиннопериодный генератор псевдослучайных чисел. С использованием библиотеки распараллеливание сложных программ статистического моделирования производится достаточно просто. В результате вычислительная нагрузка автоматически распределяются по процессорам кластера.

1. Введение

В настоящее время ведущими специалистами по вычислительной математике высказывается убеждение в том, что в ближайшем будущем в области компьютерного моделирования будут широко применяться вероятностные имитационные модели и методы Монте-Карло (методы численного статистического моделирования). С одной стороны, это убеждение основано на том, что вероятностные имитационные модели дают адекватное описание физических, химических, биологических и др. явлений при их рассмотрении «из первых принципов». С другой стороны, алгоритмы метода Монте-Карло, реализующие вероятностные модели, допускают возможность эффективного распараллеливания в виде распределенного статистического моделирования. Методы Монте-Карло перспективны в связи с появлением супер-ЭВМ экзафлопсного уровня в ближайшем будущем [1].

Упомянем одно из важных применений методов статистического моделирования. С течением химически реагирующих газов часто приходится сталкиваться при рассмотрении физических явлений, связанных с повышением температуры газовых смесей выше критической. К таким явлениям относятся процессы горения газовых топлив в камерах сгорания, соплах и горелках различных геометрических форм. Численное моделирование на компьютерах течений химически реагирующих газов по методу Монте-Карло является весьма трудоёмким, поскольку приходится учитывать многие физические и химические особенности протекающих при горении процессов. Как показывает анализ литературы, в последние десятилетия понимание химии горения (по крайней мере, с участием небольших молекул) и возможности моделирования процессов горения на больших компьютерах достигли такого уровня, который обеспечивает необходимую надежность результатов. Численное моделирование процессов горения требует значительных вычислительных ресурсов и относятся к числу задач, для которых необходимо использование супер-ЭВМ. Такие потребности обусловлены большим средним временем расчета каждой реализации ансамбля тестовых частиц, большим числом независимых реализаций ансамбля (которое определяется из необходимой точности расчетов) и большим объемом используемой машинной памяти. Для такого рода задач применение описываемой в настоящей работе библиотеки PARMONC позволит существенно сократить трудоемкость расчетов.

*Работа проводилась в рамках реализации федеральной целевой программы "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы", государственный контракт 07.514.11.4016, а также при финансовой поддержке грантов РФФИ №№ 10-07-00454, 12-01-00034, 12-01-00727.

В зависимости от условий задачи и параметров алгоритма статистического моделирования применяются разные методики параллельной реализации. Метод «распределенного статистического моделирования» состоит в распределении независимых реализаций ансамбля тестовых частиц по отдельным процессорам с периодическим осреднением полученных результатов по статистически эффективной формуле. При этом обмен между процессорами сводится к минимуму и в самом простом случае состоит лишь в начальном и финальном обмене данными. Распределенное статистическое моделирование является примером идеально масштабируемого параллельного алгоритма. Возможные отказы компонент вычислительной системы и связанные с этим потери данных приводят лишь к отсутствию соответствующих смоделированных выборочных значений, что компенсируется расчетами на других вычислительных узлах [2].

Следует упомянуть широкий класс приложений, например, методы прямого статистического моделирования в задачах газовой динамики и теории дисперсных систем, в которых ресурсов одного вычислительного узла недостаточно для моделирования реализации ансамбля и требуется распределение нагрузки между процессорами. При решении пространственно неоднородных кинетических уравнений для моделирования особо крупных ансамблей тестовых частиц целесообразно применять способы распараллеливания, основанные на пространственной декомпозиции расчетной области («параллельные алгоритмы статистического моделирования»). При этом следует применять способы динамической балансировки вычислительной нагрузки, основанные на специальных формулах прогноза времени вычислений каждым процессором [3].

2. Распределенное статистическое моделирование

Под численным статистическим моделированием обычно понимают реализацию с помощью компьютера вероятностной модели некоторого объекта с целью оценивания изучаемых интегральных характеристик на основе закона больших чисел [4]. При этом чем больше объем выборки, составленной из смоделированных независимых реализаций, тем выше точность оценивания, причем статистическая погрешность убывает обратно пропорционально квадратному корню от числа реализаций.

Как правило, при параллельной реализации необходимый объем выборки базовых случайных чисел очень велик, поэтому целесообразно использовать «длиннопериодные»псевдослучайные последовательности. А именно, для решения т.н. «больших задач»по методу Монте-Карло предлагается использовать генератор следующего вида [5]:

$$u_0 = 1,\ u_n \equiv u_{n-1}A \pmod{2^{128}}, \qquad \alpha_n = u_n 2^{-128}, \qquad n = 1, 2, \ldots,$$

где

$$A \equiv 5^{100109} \pmod{2^{128}}.$$

Последовательность чисел $\{\alpha_n\}$ является периодической, длина ее периода равна $L = 2^{126} \approx 10^{38}$.

Для распределения псевдослучайных чисел между разными вычислительными ядрами предлагается следующий порядок их использования. Последовательность $\{u_n\}$ предварительно разбивается на подпоследовательности длины μ, начинающиеся с чисел $u_{m\mu}$, $m = 0, 1, 2, \ldots$. Разные подпоследовательности используются на разных ядрах. Значение «прыжка» генератора μ должно выбираться из соображений, чтобы μ псевдослучайных чисел хватало для моделирования на каждом ядре. Ясно, что для метода вычетов начальные значения $u_{m\mu}$ указанных подпоследовательностей получаются по формуле

$$u_{(m+1)\mu} \equiv u_{m\mu}A_\mu (\mathrm{mod}\ 2^{128}), A_\mu \equiv A^\mu (\mathrm{mod}\ 2^{128}), \alpha_{m\mu} = u_{m\mu} \cdot 2^{-128}. \qquad (1)$$

Этот параллельный генератор был проверен с помощью содержательных статистических тестов [2] и успешно используется в ряде институтов СО РАН на протяжении последних десяти лет.

С целью понижения трудоемкости моделирование независимых выборочных значений можно распределить по разным вычислительным ядрам. Очевидно, что главным критерием осуществимости такой параллельной реализации является возможность «поместить» вычислительную программу моделирования выборочного значения в память каждого ядра.

При распределении вычислений по ядрам следует допускать возможность реализации различных объемов выборки на различных ядрах с использованием статистически оптимального способа осреднения результатов по формулам вида

$$\bar{\zeta} = \sum_{m=1}^{M} n_m \bar{\zeta}_m \Big/ \sum_{m=1}^{M} n_m,$$

где M – число ядер, n_m – объем выборки для m-го процессора, $\bar{\zeta}_m$ – соответствующее среднее значение. Таким образом, можно добиться обратно пропорциональной зависимости величины трудоемкости случайной оценки от числа ядер (при условии, что используемые ядра имеют одинаковую производительность) [2].

Описанную методику параллельного моделирования можно легко модифицировать с целью эффективной оценки функционалов, зависящих от параметра $x \in X$:

$$\varphi(x) \approx \mathrm{E}\zeta(x;\omega).$$

При этом моделируемое распределение случайной величины ω может зависеть, а может и не зависеть от параметра x. Аналогичную методику можно также использовать для оценки функционалов, возникающих в результате осреднения базовых функционалов по случайному параметру σ задачи, например по случайной плотности среды. Вероятностное представление при этом имеет вид двойного математического ожидания

$$\varphi \approx \mathrm{EE}\zeta(\omega,\sigma).$$

Примеры подобных задач приведены в [2].

3. Библиотека PARMONC

С целью унификации реализации распределенного статистического моделирования при решении широкого круга задач методом Монте-Карло разработана и внедрена в широкое использование программная библиотека PARMONC (сокращение от PARallel MONte Carlo). Библиотека и примеры ее использования описаны в работах [6–8].

Возможность применения библиотеки PARMONC для распараллеливания широкого круга задач определяется естественной модульностью программ статистического моделирования. Общая структура такого рода программ, в упрощенном виде, следующая (в нотации языка C):

```
TypeRO RO, SUBT;
SUBT={0.0, .., 0.0};
for(i=0;i<L;i++){
        realization(RO);
        SUBT= SUBT+RO;
}
SUBT=SUBT/L;
```

Здесь L – общее число независимых реализаций случайного объекта, задаваемого композитным типом данных $TypeRO$ (допускается поэлементное суммирование переменных такого типа). Реализации моделируются в результате работы процедуры (или блока операторов) $realization$. Полученная таким образом реализация RO (статистически независимая от других моделируемых реализаций) добавляются к «счетчику» $SUBT$ и впоследствии осредняются, что дает статистическую оценку искомого математического ожидания случайного

объекта. В процессе вычислений используются потоки псевдослучайных чисел, получаемых в результате работы подпрограммы, которая в общем случае вызывается следующим образом:

```
a = rand();
```

Здесь a – очередное псевдослучайное число, равномерно распределенное в интервале от нуля до единицы. Для использования PARMONC определяется процедура *realization* (моделирующая подпрограмма), возвращающая реализацию RO. При этом считается, что моделирующая подпрограмма использует потоки псевдослучайных чисел, генерируемых внешней по отношению к ней подпрограммой. Описанный выше цикл по независимым реализациям и финальное осреднение заменяются вызовом библиотечной процедуры следующего вида:

```
parmoncc (realization, L, ...);
```

Здесь имя моделирующей подпрограммы и общее число независимых реализаций передаются в библиотечную процедуру *parmoncc* в качестве аргументов; для простоты остальные аргументы библиотечной процедуры опущены. Процедура *parmoncc* автоматически распределяет моделирование независимых реализаций по процессорам используемой многопроцессорной вычислительной системы. Все остальные операторы пользовательской программы остаются без изменений.

Библиотека PARMONC предназначена для использования на кластерах Сибирского суперкомпьютерного центра (ССКЦ КП СО РАН) для распараллеливания широкого круга трудоемких приложений метода Монте-Карло. «Ядром» библиотеки является тщательно протестированный, быстрый и надежный длиннопериодный генератор псевдослучайных чисел (1), разработанный в Лаборатории методов Монте-Карло ИВМиМГ СО РАН.

Библиотечные подпрограммы могут быть использованы в пользовательских программах, написанных на языках C, C++ и Fortran, причем от пользователя не требуется знание языка MPI. В процессе счета происходит автоматическое получение выборочных средних и границ погрешностей для статистических оценок, алгоритм моделирования которых задается в пользовательской подпрограмме. Имя такой подпрограммы передается в качестве аргумента в соответствующую библиотечную подпрограмму. В процессе счета результаты вычислений периодически сохраняются на жестком диске в удобном для дальнейшей обработки виде. Библиотечные подпрограммы автоматически распределяют вычислительную нагрузку по процессорам кластера. С помощью библиотеки PARMONC можно легко организовать продолжение ранее проведенных расчетов с автоматическим учетом их ре-

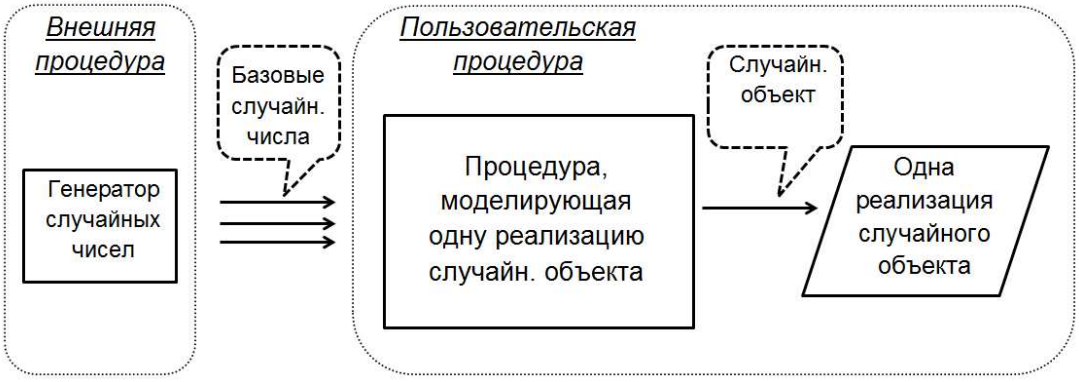


Рис. 1. Взаимосвязь объектов при статистическом моделировании

зультатов. Также с помощью библиотеки можно получать коррелированные статистические оценки различных функционалов.

Библиотека установлена на кластере НКС-30Т ССКЦ КП СО РАН в директории */ifs/apps/parmonc/*. Библиотека PARMONC состоит из следующих подпрограмм и исполняемых файлов:

- *rnd*128 - функция для получения одного случайного числа, равномерно распределенного в интервале от 0 до 1, с помощью параллельного генератора случайных чисел (1);
- *parmoncf* - процедура, осуществляющая распределенное статистическое моделирование (для программ на Fortran-е);
- *parmoncc* - процедура, осуществляющая распределенное статистическое моделирование (для программ на C);
- *manaver* - программа для осреднения выборочных средних, независимо рассчитанных на разных процессорах;
- *genparam* - программа для расчета параметров параллельного генератора случайных чисел.

Здесь *rnd*128, *parmoncf* и *parmoncc* являются библиотечными подпрограммами для использования в пользовательских программах на Fortran-е, C или C++, *genparam* и *manaver* являются исполняемыми файлами для запуска из командной строки. Объектные файлы PARMONC упакованы в статическую библиотеку *libparmonc.a*. Пользователь вставляет вызовы процедур *rnd*128 и *parmoncf*/*parmoncc* в свои программы. Главная пользовательская программа, в которой находится вызов *parmoncf*/*parmoncc*, рассматривается компилятором как MPI-программа, несмотря на то, что в самой пользовательской программе нет явных вызовов директив MPI. Это означает, что такая программа должна компилироваться и собираться с использованием команд *mpiicc* или *mpiifort*. Результаты расчетов, выполненных с использованием процедур *parmoncf*/*parmoncc*, сохраняются в файлы, которые находятся в специальной поддиректории в рабочей директории пользователя. Все эти файлы обновляются всякий раз, когда центральный процессор получает данные с других процессоров, осредняет их и сохраняет на диск.

В файл *.bashrc*, который находится в домашней директории пользователя, необходимо добавить следующую строку:

```
source $/ifs/apps/parmonc/bin/parmoncvars.sh$
```

Тем самым объявляются три переменные окружения

```
$PRMCBIN, $PRMCLIB, $PRMCINC.
```

Эти переменные используются при компиляции и сборке приложений с помощью PARMONC, а также при запуске команд.

Для компиляции и сборки пользовательских программ с использованием библиотеки PARMONC следует использовать следующие команды (пример для программ на C):

```
$ mpiicc -o test -L$PRMCLIB -I$PRMCINC test.c -lparmonc
```

Здесь *test* - имя исполняемого файла, *test.f*90 и *test.c* - пользовательские программы.

4. Заключение

Упомянем некоторые напрвления дальнейшего развития библиотеки PARMONC. Библиотека может стать базовым программным уровнем для масштабируемых параллельных приложений метода Монте-Карло, реализующих сложные вероятностные модели естествознания. Кроме того, целесообразно адаптировать библиотеку к современным супер-ЭВМ с гибридной архитектурой.

Литература

1. Глинский Б.М., Родионов А.С., Марченко М.А., Подкорытов Д.В. Винс Д.И. Агентно-ориентированный подход к имитационному моделированию суперЭВМ экзафлопсной производительности в приложении к распределенному статистическому моделированию // Вестник ЮУрГУ, серия "Математическое моделирование и программирование" – 2012 – 12 С. (принято в печать)

2. Марченко М.А., Михайлов Г.А. Распределенные вычисления по методу Монте-Карло // Автоматика и телемеханика. – 2007 – Вып. 5 – С. 157—170.

3. Marchenko M.A. Majorant frequency principle for an approximate solution of a nonlinear spatially inhomogeneous coagulation equation by the Monte Carlo method // Russ. J. Numer. Anal. Math. Modelling. — 2008. -– Vol. 21, N. 3. – P. 199-218.

4. Михайлов Г.А., Войтишек А.В. Численное статистическое моделирование. Методы Монте-Карло – Москва: Издательский центр «Академия», 2006.

5. Marchenko M.A. Parallel Pseudorandom Number Generator for Large-scale Monte Carlo Simulations // LNCS -– 2007 – Vol. 4671 — P. 276-282.

6. Marchenko M. PARMONC – A Software Library for Massively Parallel Stochastic Simulation // LNCS -– 2011 -– Vol. 6873 – P. 302-315.

7. Страница библиотеки PARMONC на сайте ССКЦ КП СО РАН: http://www2.sscc.ru/SORAN-INTEL/paper/2011/parmonc.htm.

8. Ссылка на документацию к библиотеке PARMONC на сайте ССКЦ КП СО РАН: http://www2.sscc.ru/SORAN-INTEL/paper/2011/parmonc.pdf.

Ускорение расчета процессов молекулярной динамики при помощи GPU *

Д.Ф. Марьин[1,2]

Центр микро- и наномасштабной динамики дисперсных систем, Башкирский государственный университет[1],
Институт механики УНЦ РАН[2]

В работе представлены результаты по производительности и эффективности использования GPU при моделировании процессов молекулярной динамики. Моделировался обрезанный потенциал Леннарда–Джонса при помощи вычислительной схемы leapfrog.

1. Введение

Задачи динамики дисперсных систем в микро- и наномасштабе возникают во многих отраслях науки и промышленности: механической, химической, нефтяной, экологической и др.. Проведение физических экспериментов над процессами, происходящими на микро- и наноуровнях сильно затруднено тем фактом, что размеры исследуемых элементов и структур зачастую оказываются на порядок меньше размеров длины волны видимого света. Это означает, что фиксация происходящих процессов либо достаточно сложна и требует сложного дорогостоящего оборудования, либо невозможна, так как коротковолновое рентгеновское и гамма излучение характеризуются высокой энергией квантов излучения, то есть их использование может в значительной степени искажать реальную картину наблюдаемых процессов.

Для решения вышеописанных сложностей используется вычислительный эксперимент, который позволяет описывать и измерять мельчайшие детали.

Однако и при проведении вычислительного эксперимента имеется ряд проблем, которые связаны с тем, что при достаточно подробном математическом описании проблемы, учитывающем многомерность и многопараметричность, а также с использование при моделировании большого числа частиц, серьезно возрастают требования к производительности как используемого программного кода, так и вычислительной системы в целом.

Для проведения вычислительного эксперимента в разумные сроки требуется удовлетворить ряд важных условий. Первое условие заключается в снижении вычислительной сложности используемых алгоритмов. Так сложность прямого алгоритма пропорциональна числу всех парных взаимодействий в системе размера N, то есть равна $O(N^2)$. И использование алгоритмов, которые имеют меньшую вычислительную сложность, является задачей чрезвычайно важной и актуальной.

Наряду с использованием алгоритмов с низкой вычислительной сложностью существует ещё одно направление повышения производительности вычислений — использование высокопроизводительных вычислительных систем. В настоящее время наиболее эффективными для задач динамики многих тел являются гетерогенные системы, представляющие собой вычислительные кластеры, узлы которых содержат как CPU (центральный процессор), так и GPU (графический процессор). Однако эффективное использование описанных вычислительных систем требует как значительной модификации существующих алгоритмов, так и разработки новых.

Следует отметить ещё один факт. Он заключается в том, что на протяжении масштабной шкалы от микро- до наноуровней располагается условная точка, после которой исполь-

*Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации, грант 11.G34.31.0040.

зование классических континуальных моделей многофазных систем оказывается недопустимым, и актуальность приобретают кинетические модели, используемые в методах молекулярной динамики.

Таким образом, проведение исследований процессов динамики дисперсных систем, происходящих на микро- и наноуровнях, требует реализации молекулярно-динамических моделей при помощи численных методов, ориентированных на использование современных высокопроизводительных алгоритмов и терафлопсных гетерогенных вычислительных систем, содержащих как CPU, так и GPU.

2. Теоретическая часть

2.1. Математическая модель

Математическая модель среды, описываемой в терминах молекулярной динамики для случая неполярных молекул, основана на предположении о том, что среда состоит из сферических частиц, которые взаимодействуют друг с другом по определённому закону.

Функция (потенциальная функция, потенциал), описывающая такое взаимодействие, может принимать различные формы и зависит от поставленной задачи. Самой известной такой функцией является потенциал Леннарда—Джонса (эта модель достаточно реалистично передаёт свойства реального взаимодействия сферических неполярных молекул и поэтому широко используется в расчётах и при компьютерном моделировании) [1]

$$U_{LJ}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right],$$

где r — расстояние между частицами; ε — глубина потенциальной ямы; σ — расстояние, на котором энергия взаимодействия становится равной нулю. Параметры ε и σ являются характеристиками молекул соответствующего вещества.

Для ускорения расчётов потенциал Леннарда—Джонса обрывается на расстоянии $r_c = 2,5\sigma$. И, чтобы избежать нефизичной ситуации, такой, что при пересечении сферы радиуса r_c какой-то молекулой энергия системы меняется скачкообразно, потенциал сдвигается, чтобы выполнялось условие $U(r_c) = 0$. Таким образом, обрезанный потенциал Леннарда—Джонса принимает следующий вид

$$U_{LJ_{trunc}}(r) = \begin{cases} U_{LJ}(r) - U_{LJ}(r_c), & r \leqslant r_c\,, \\ 0, & r > r_c\,. \end{cases}$$

Кинетические уравнения движения атомов следуют из второго закона Ньютона:

$$m\ddot{\mathbf{r}}_\mathbf{i} = \mathbf{f}_\mathbf{i} = \sum_{\substack{j=1 \\ (j \neq i)}}^{N} \mathbf{f}_\mathbf{ij},$$

где $\mathbf{f_{ij}} = -\nabla U_{LJ_{trunc}}(r_{ij})$. Макроскопические параметры (температура, давление, плотность среды и др.) могут быть получены, исходя из положений молекулярно-кинетической теории.

2.2. Численный метод

Для интегрирования уравнений движения используется простая численная схема — метод чехарды (leapfrog) [1], который имеет вид

$$\mathbf{v_i}(t + h/2) = \mathbf{v_i}(t - h/2) + h\mathbf{a_i}(t),$$

$$\mathbf{r_i}(t + h) = \mathbf{r_i}(t) + h\mathbf{v_i}(t + h/2),$$

где $\mathbf{r_i}$ — координаты i–ой частицы; $\mathbf{v_i}$ — её скорость; $\mathbf{a_i}$ — её ускорение; t — текущий временной шаг; h — шаг по времени.

Если для оценки необходимо значение скорости на шаге по времени соответствующем шагу, на котором вычислены координаты, то можно использовать следующую формулу

$$\mathbf{v_i}(t) = \mathbf{v_i}(t - h/2) + (h/2)\mathbf{a_i}(t).$$

3. Реализационная часть

В работе представлены результаты для прямого расчёта, когда для каждой частицы рассматривается взаимодействие со всеми другими частицами с проверкой по радиусу обрезания, на CPU и на GPU. Версия на GPU реализована при помощи программно-аппаратной архитектуры CUDA.

Так же в работе представлены результаты расчета на CPU с использованием структуры данных и списка соседей. Использование структуры данных позволяет снизить вычислительную сложность всего алгоритма с $O(N^2)$ до $O(NM)$, где N — общее число частиц; M — среднее число частиц в соседних боксах. Построение структуры данных основано на использовании гистограммы распределения частиц по боксам и bucket–сортировки частиц [2]. Вычислительная сложность алгоритма построения структуры данных равна $O(N)$. Таким образом общая вычислительная сложность снижается при оптимальном подборе числа боксов в зависимости от N.

4. Экспериментальная часть

Тестовые расчёты проводились на вычислительной системе с CPU Intel Xeon 5660, 2.8GHz, GPU NVIDIA Tesla C2050, операционной системой Linux 64bit, компиляторами GCC v.4.4, CUDA v.4.0. Размер блока при проведении расчетов выбирался исходя из оптимальности и, начиная с некоторого числа частиц, равнялся 256 потокам на блок.

4.1. Результаты прямого расчёта

На рис. 1 показано время расчёта в зависимости от числа частиц в рассматриваемой системе для чисел с плавающей точкой одинарной точности. Как видно из рисунка графики выходят на постоянный тренд и имеют одинаковый наклон. Аналогичную картину можно наблюдать и для чисел с плавающей точкой двойной точности (см. рис. 2).

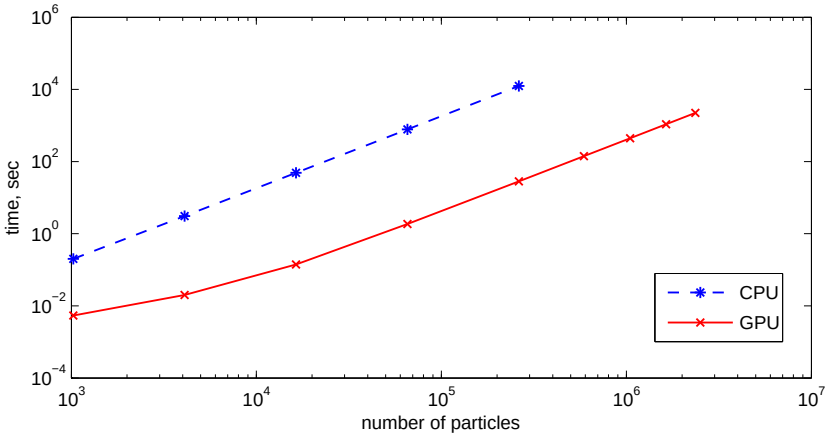


Рис. 1. Время выполнения в зависимости от числа частиц для чисел с плавающей запятой одинарной точности

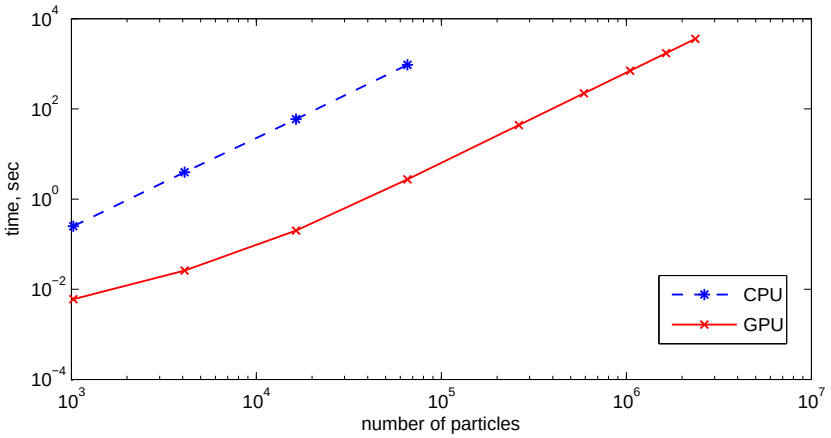


Рис. 2. Время выполнения в зависимости от числа частиц для чисел с плавающей запятой двойной точности точности

На рис. 3 показано ускорение полученное на GPU в сравнении с запуском на одном ядре CPU для чисел с плавающей точкой одинарной и двойной точности. Оно достигает 490 и 380 раз для чисел с одинарной и двойной точностью соответственно. Ускорение получено по времени вычислений без учета времени коммуникаций. Важно отметить, что сравнение ведется с оптимизированным, но не распараллеленным кодом на CPU (была включена автоматическая оптимизация компилятором как для CPU–кода, так и для GPU–кода). Целью является не соревнование между CPU и GPU, а использование кода на CPU в качестве основы для разработки кода на GPU.

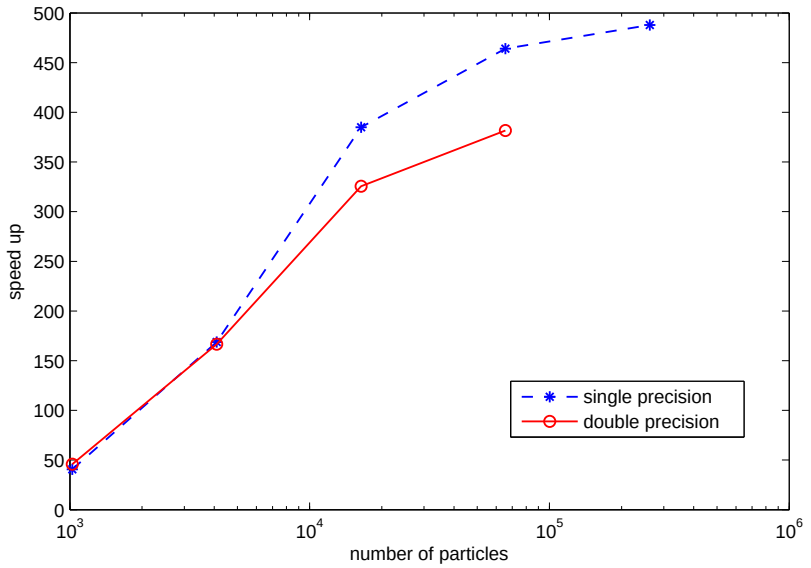


Рис. 3. Ускорение в зависимости от числа частиц

Об эффективности использования GPU можно судить по получаемой производительности, графики которой для чисел с плавающей точкой одинарной и двойной точности показаны на рис. 4. Производительность достигает 525 и 330 GFLOPS для чисел с одинарной и двойной точностью соответственно, что превышает половину пиковой производительности GPU C2050[1]. При расчёте производительности арифметические операции и стандартные

[1] пиковая производительность для чисел с плавающей точкой одинарной точности — 1030 GFLOPS, двой-

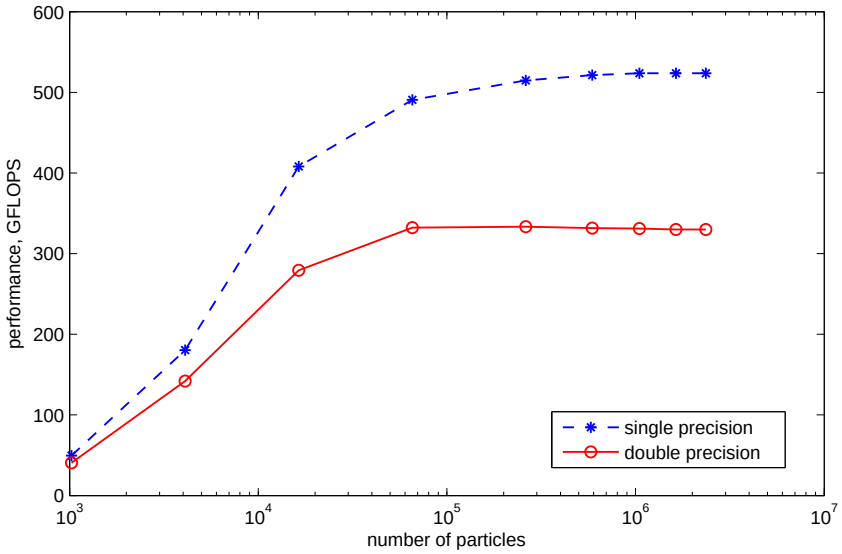


Рис. 4. Производительность в зависимости от числа частиц

функции (например, модуль числа) по занимаемому числу тактов приравнивались к операции сложения. Так же стоит отметить, что в связи со спецификой реализации на GPU, при расчёте сил взаимодействия на GPU не учитывался третий закон Ньютона, в отличии от расчёта на CPU, то есть GPU произвело примерно в два раза больше вычислений, чем CPU.

4.2. Результаты расчёта с использованием структуры данных на CPU

На рис. 5 показано время расчёта в зависимости от числа частиц в рассматриваемой системе для чисел с плавающей точкой одинарной точности.

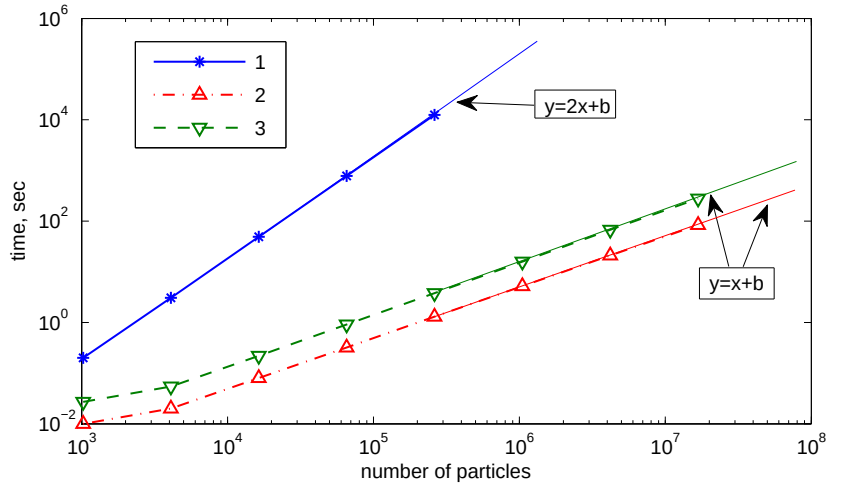


Рис. 5. Время выполнения на CPU, одинарная точность: (1) — время прямого расчёта, (2) — время расчёта с использованием структуры данных, (3) — (2) вместе со временем генерации структуры данных для каждой итерации

На графике так же показаны аппроксимирующие линии и их уравнения в логарифми-

ной точности — 515 GFLOPS

ческой системе координат. Как видно из рисунка, графики выходят на постоянный тренд, и угол наклона графиков (2, 3) в логарифмической системе координат, соответствующий вычислительной сложности $O(N)$, где N — число частиц, меньше угла наклона графика (1) из предыдущего пункта, соответствующего вычислительной сложности $O(N^2)$. Для чисел с плавающей точкой двойной точности картина аналогична.

Для динамичных систем, когда положение частиц меняется на каждой итерации, время генерации структуры данных превышает время одной итерации и, как видно из рис. 6, растёт с ростом числа частиц.

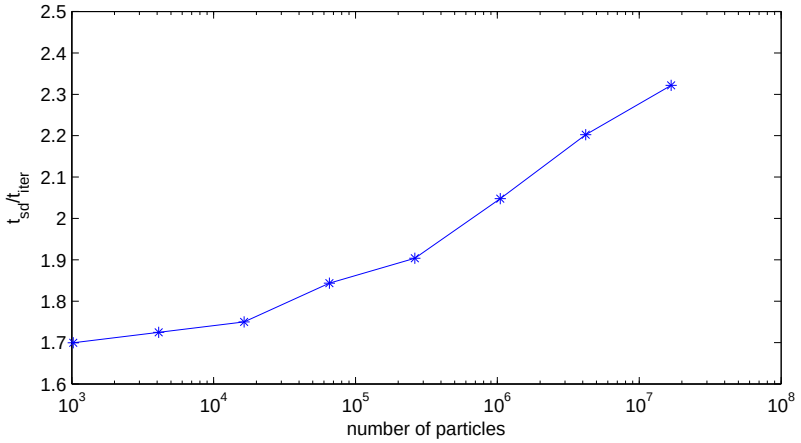


Рис. 6. Отношение времени генерации структуры данных ко времени одной итерации в зависимости от числа частиц

Но в зависимости от характерных параметров рассматриваемой среды, числа частиц и согласно условию Куранта можно подобрать число боксов так, чтобы проводить генерацию структуры данных один раз на 5–10 итераций.

5. Заключение

Таким образом можно сделать вывод, что достигнута очень хорошая производительность на GPU, благодаря чему удалось достичь ускорения проведения расчётов по сравнению с CPU в 490 и 380 раз для чисел с плавающей точкой одинарной и двойной точности соответственно.

Использование структуры данных позволяет достичь хорошего ускорения, путём снижения общей вычислительной сложности алгоритма. И реализация структуры данных на GPU является перспективным направлением для ускорения проведения расчётов рассматриваемой задачи.

Литература

1. Rapaport D.C. The art of molecular dynamics simulation. 2004. p. 400.

2. Q. Hu, N.A. Gumerov, and R. Duraiswami Scalable fast multipole methods on distributed heterogeneous architectures // in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, (New York, NY, USA), ACM, 2011.

3. NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Version 3.2. 2010.

Численное решение 2.5-D динамической задачи сейсмики с использованием алгоритмов распараллеливания*

Б.Г. Михайленко, А.А. Михайлов

Институт Вычислительной Математики и Математической Геофизики СО РАН

В настоящем докладе рассматривается эффективный алгоритм распараллеливания численного решения системы линеаризованных уравнений для 2.5-D динамической задачи сейсмики. Предлагаемый алгоритм основан на использование комплексирования конечных интегральных преобразований Фурье и Лагерра с конечно-разностным методом решения преобразованной задачи. Исходная система записывается в виде гиперболической системы первого порядка в терминах вектора скоростей и тензора напряжений для трёхмерной Декартовой системы координат. Предлагаются два варианта распараллеливания расчетов и анализируются особенности их численной реализации. В докладе представлены численные результаты моделирования сейсмических волновых полей, полученные в результате расчётов на многопроцессорном вычислительном комплексе.

1. Введение

При моделировании сейсмических волновых полей в неоднородных средах оказалось эффективным использование алгоритмов комплексирования интегральных преобразований по одной или двум пространственным переменным и временной координате с конечно-разностным методом решения полученных в результате преобразований задач. В настоящем докладе рассматривается эффективный алгоритм численного решения системы линеаризованных уравнений для 2.5-D динамической задачи сейсмики основанный на применении интегрального преобразования Лагерра по временной координате. Этот метод можно рассматривать как аналог известного спектрального метода на основе Фурье преобразования, где вместо частоты ω мы имеем параметр m - степень полиномов Лагерра. Однако, в отличие от Фурье, применение интегрального преобразования Лагерра по времени позволяет свести исходную задачу к решению системы уравнений, в которой параметр разделения присутствует только в правой части уравнений и имеет рекуррентную зависимость. Данный метод решения динамических задач теории упругости был впервые рассмотрен в работах [1, 2], а затем развит для задач вязкоупругости [3, 4] и теории пористых сред [5]. В указанных работах рассмотрены отличительные особенности данного метода от принятых подходов и обсуждаются преимущества применения интегрального преобразования Лагерра в отличии от разностного метода и Фурье преобразования по времени. Целью данной статьи является представление алгоритма эффективного распараллеливания расчетов на основе данного подхода к решению 2.5-D динамической упругой задачи сейсмики.

В рассматриваемой постановке задачи исходная система записывается в виде гиперболической системы первого порядка в терминах вектора скоростей и тензора напряжений для трёхмерной Декартовой системы координат. При этом полагается, что параметры среды (плотность и скорости продольных и поперечных волн) имеют зависимость только по двум координатам , а по третьей координате среда однородна. Данную постановку задачи принято называть 2.5-D задачей. Для решения поставленной задачи используются конечные интегральные косинус и синус преобразования Фурье по координате с постоянными параметрами среды и интегральное преобразование Лагерра по времени. В результате исходная задача сводится к решению N независимых (N - количество гармоник преобразования Фурье по пространственной координате) систем дифференциальных уравнений. Для их решения используется конечно-разностная аппроксимация производных по двум пространственным координатам на сдвинутых сетках с 4-

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 11-05-00937)

ым порядком точности. В итоге, решение получаемых N независимых систем линейных алгебраических уравнений находится с помощью метода сопряжённых градиентов.

2. Постановка задачи

Рассмотрим Декартовую систему координат (x_1,x_2,x_3) в полупространстве $x_2 \geq 0$. Связь между компонентами напряжений и скоростями смещений для задачи распространения сейсмоакустических колебаний в упругой изотропной среде записывается как:

$$\begin{cases} \dfrac{\partial u_i}{\partial t} + \dfrac{\partial \sigma_{ij}}{\partial x_j} = F_i\, f(t) \\ \dfrac{\partial \sigma_{ij}}{\partial t} + \mu\left(\dfrac{\partial u_j}{\partial x_i} + \dfrac{\partial u_i}{\partial x_j}\right) + \lambda \delta_{ij}\, div\,\vec{u} = 0 \end{cases} \tag{1}$$

Здесь δ_{ij} - символ Кронекера, $\lambda(x_1,x_2)$, $\mu(x_1,x_2)$ - упругие параметры среды, $\rho(x_1,x_2)$ - плотность среды, $\vec{u} = (u_1,u_2,u_3)$ - вектор скорости смещений, σ_{ij} - компоненты тензора напряжений. Здесь F_1,F_2,F_3 - составляющие силы $\vec{F}(x_1,x_2,x_3) = F_1\vec{e}_1 + F_2\vec{e}_2 + F_3\vec{e}_3$, описывающие распределение локализованного в пространстве источника, а $f(t)$ - заданный временной сигнал в источнике.

Задача решается при нулевых начальных данных

$$u_i\big|_{t=0} = \sigma_{ij}\big|_{t=0} = 0 \tag{2}$$

и граничных условиях на свободной поверхности в плоскости $x_2 = 0$

$$\sigma_{12}\big|_{x_2=0} = \sigma_{23}\big|_{x_2=0} = \sigma_{22}\big|_{x_2=0} = 0. \tag{3}$$

Полагаем, что функции $u_i(x_1,x_2,x_3)$ и $\sigma_{ij}(x_1,x_2,x_3)$ обладают достаточной гладкостью для применения всех последующих преобразований.

3. Алгоритм решения

На первом этапе решения для сведения поставленной задачи к серии задач меньшей размерности, воспользуемся конечными интегральными косинус-синус преобразованиями Фурье по пространственной координате, в направлении которой среда считается однородной. Для каждой компоненты системы введем соответствующее косинус или синус преобразование [6]:

$$\overrightarrow{W}(x_1,x_2,n,t) = \int_0^a \overrightarrow{W}(x_1,x_2,x_3,t)\begin{Bmatrix}\cos(k_n x_3)\\ \sin(k_n x_3)\end{Bmatrix} d(x_3), \tag{4}$$

с соответствующей формулой обращения

$$\overrightarrow{W}(x_1,x_2,x_3,t) = \frac{1}{\pi}\overrightarrow{W}_0(x_1,x_2,t) + \frac{2}{\pi}\sum_{n=1}^{N}\overrightarrow{W}(x_1,x_2,n,t)\cos(k_n x_3) \tag{5}$$

или

$$\overrightarrow{W}(x_1,x_2,x_3,t) = \frac{2}{\pi}\sum_{n=1}^{N}\overrightarrow{W}(x_1,x_2,n,t)\sin(k_n x_3), \tag{6}$$

где $k_n = \dfrac{n\pi}{a}$.

В результате данного преобразования получим $N+1$ независимых двухмерных по пространству нестационарных задач. Далее к полученным таким образом задачам применим интегральное преобразование Лагерра по времени вида [1, 2]:

$$\overrightarrow{W}_m(x_1,x_2,n)=\int_0^{\infty}\overrightarrow{W}(x_1,x_2,n,t)(ht)^{-\frac{\alpha}{2}}l_m^{\alpha}(ht)d(ht)\,, \qquad (7)$$

с формулой обращения

$$\overrightarrow{W}(x_1,x_2,n,t)=(ht)^{\frac{\alpha}{2}}\sum_{m=0}^{\infty}\frac{m!}{(m+\alpha)!}\overrightarrow{W}_m(x_1,x_2,n)l_m^{\alpha}(ht)\,, \qquad (8)$$

где $l_m^{\alpha}(ht)$ - функции Лагерра.

В результате преобразования по времени решение исходной задачи сводится к решению $N+1$ двумерных дифференциальных задач в спектральной области. Для решения каждой такой задачи применяем конечно-разностную аппроксимацию производных по двум пространственным координатам на сдвинутых сетках с 4-ым порядком точности [7].

Для этого в расчетной области введем в направлении координаты $z=x_1$ сетки ωz_1 и $\omega z_{1/2}$ с шагом дискретизации Δz, сдвинутые относительно друг друга на $\Delta z/2$:

$$\omega z_1=(x,j\Delta z,t),\quad \omega z_{1/2}=(x,j\Delta z+\frac{\Delta z}{2},t),\ j=0,...,K\,.$$

Аналогично, введем в направлении координаты $x=x_2$ сетки ωx_1 и $\omega x_{1/2}$ с шагом дискретизации Δx, сдвинутые относительно друг друга на $\Delta x/2$:

$$\omega x_1=(i\Delta x,z,t),\quad \omega x_{1/2}=(i\Delta x+\frac{\Delta x}{2},z,t),\ i=0,...,L\,.$$

На данных сетках введем операторы дифференцирования D_x и D_z, аппроксимирующие производные $\frac{\partial}{\partial x}$ и $\frac{\partial}{\partial z}$ с четвертым порядком точности по координатам $z=x_1$ и $x=x_2$:

$$D_xu(x,z)=\frac{9}{8\Delta x}\left[u(x+\frac{\Delta x}{2},z)-u(x-\frac{\Delta x}{2},z)\right]-\frac{1}{24\Delta x}\left[u(x+\frac{3\Delta x}{2},z)-u(x-\frac{3\Delta x}{2},z)\right],$$

$$D_zu(x,z)=\frac{9}{8\Delta z}\left[u(x,z+\frac{\Delta z}{2})-u(x,z-\frac{\Delta z}{2})\right]-\frac{1}{24\Delta z}\left[u(x,z+\frac{3\Delta z}{2})-u(x,z-\frac{3\Delta z}{2})\right].$$

Определим искомые компоненты вектора решения в следующих узлах сеток:

$u_y^m(x,z),\sigma_{xx}^m(x,z),\sigma_{yy}^m(x,z),\sigma_{zz}^m(x,z)\in\omega x_{1/2}\times\omega z_1$,

$\sigma_{xy}^m(x,z),u_x^m(x,z)\in\omega x_1\times\omega z_1$,

$\sigma_{yz}^m(x,z),u_z^m(x,z)\in\omega x_{1/2}\times\omega z_{1/2}$,

$\sigma_{xz}^m(x,z)\in\omega x_1\times\omega z_{1/2}$.

Тогда конечно разностные уравнения, аппроксимирующие исходную задачу с четвертым порядком точности по пространственным переменным, запишутся в виде:

$$\frac{h}{2}u_{x,(i,j)}^m-\rho_{(i,j)}^{-1}\left(D_z\sigma_{xz}^m+D_x\sigma_{xx}^m+k_n\sigma_{xy}^m\right)_{(i,j)}=-h\sum_{p=0}^{m-1}u_{x,(i,j)}^p\,,$$

$$\frac{h}{2}u_{y,(i+1/2,j)}^m-\rho_{(i+1/2,j)}^{-1}\left(D_z\sigma_{yz}^m+D_x\sigma_{xy}^m-k_n\sigma_{yy}^m\right)_{(i+1/2,j)}=-h\sum_{p=0}^{m-1}u_{y,(i+1/2,j)}^p\,,$$

$$\frac{h}{2}u_{z,(i+1/2,j+1/2)}^m-\rho_{(i+1/2,j+1/2)}^{-1}\left(D_z\sigma_{zz}^m+D_x\sigma_{xz}^m+k_n\sigma_{yz}^m\right)_{(i+1/2,j+1/2)}=-h\sum_{p=0}^{m-1}u_{z,(i+1/2,j+1/2)}^p\,,$$

$$\frac{h}{2}\sigma^m_{xx,(i+1/2,j)} - \lambda_{(i+1/2,j)}\left(D_z u^m_z + k_n u^m_y\right)_{(i+1/2,j)} - (\lambda+2\mu)_{(i+1/2,j)} D_x u^m_{x,(i+1/2,j)} = -h\sum_{p=0}^{m-1}\sigma^p_{xx,(i+1/2,j)},$$

$$\frac{h}{2}\sigma^m_{yy,(i+1/2,j)} - \lambda_{(i+1/2,j)}\left(D_z u^m_z + D_x u^m_x\right)_{(i+1/2,j)} - k_n(\lambda+2\mu)_{(i+1/2,j)} u^m_{y,(i+1/2,j)} = -h\sum_{p=0}^{m-1}\sigma^p_{yy,(i+1/2,j)},$$

$$\frac{h}{2}\sigma^m_{zz,(i+1/2,j)} - (\lambda+2\mu)_{(i+1/2,j)} D_z u^m_{z,(i+1/2,j)} - \lambda_{(i+1/2,j)}\left(D_x u^m_x + k_n u^m_y\right)_{(i+1/2,j)} = -h\sum_{p=0}^{m-1}\sigma^p_{zz,(i+1/2,j+1/2)},$$

$$\frac{h}{2}\sigma^m_{xy,(i,j)} - \mu_{(i,j)}\left(D_x u^m_y + k_n u^m_x\right)_{(i,j)} = -h\sum_{p=0}^{m-1}\sigma^p_{xy,(i,j)},$$

$$\frac{h}{2}\sigma^m_{xz,(i,j+1/2)} - \mu_{(i,j+1/2)}\left(D_z u^m_x - D_x u^m_z\right)_{(i,j+1/2)} = -h\sum_{p=0}^{m-1}\sigma^p_{xz,(i,j+1/2)},$$

$$\frac{h}{2}\sigma^m_{yz,(i+1/2,j+1/2)} - \mu_{(i+1/2,j+1/2)}\left(D_z u^m_y + k_n u^m_z\right)_{(i+1/2,j+1/2)} = -h\sum_{p=0}^{m-1}\sigma^p_{yz,(i+1/2,j+1/2)},$$

где приняты следующие обозначения $u^m_{x,(i,j)} = u^m_x(x_i, z_j)$. Для других компонент аналогично.

В результате конечно-разностной аппроксимации получим $N+1$ систем линейных алгебраических уравнений. Представим искомый вектор решения $\vec{W}$ в следующем виде:

$\vec{W}(m) = (\vec{V}_0(m), \vec{V}_1(m), ..., \vec{V}_{K+L}(m))^T$,

$\vec{V}_{i+j} = (\ u_x^{i,j}, u_y^{i+1/2,j}, u_z^{i+1/2,j+1/2},\ \sigma_{xx}^{i+1/2,j}, \sigma_{yy}^{i+1/2,j}, \sigma_{zz}^{i+1/2,j}, \sigma_{xy}^{i,j}, \sigma_{xz}^{i,j+1/2}, \sigma_{yz}^{i+1/2,j+1/2}\ )^T$.

Тогда для каждой n-той гармоники ($n = 0,...,N$) система линейных алгебраических уравнений в векторной форме может быть записана как:

$$\left(A_\Delta + \frac{h}{2}E\right)\vec{W}(m) = \vec{F}_\Delta(m-1). \qquad (9)$$

Последовательность компонент волнового поля в векторе решения $\vec{V}$ выбирается с учетом минимизации количества диагоналей в матрице A_Δ. При этом на главной диагонали матрицы специально располагаются компоненты, входящие в уравнения системы как слагаемые, имеющие в качестве сомножителя параметр h (параметр преобразования по Лагерру). Следует отметить, что за счет выбора параметра h имеется возможность существенно улучшать обусловленность матрицы системы. Решив систему линейных алгебраических уравнений (9) можно определить спектральные значения для всех компонент волнового поля $\vec{W}(m,n)$. Затем, воспользовавшись формулами обращения для Фурье преобразования (5),(6) и преобразования Лагерра (8), получим решение исходной задачи (1)-(3).

4. Аспекты численной реализации

В аналитических преобразованиях Фурье и Лагерра при определении значений функций по их спектру используются формулы обращения в виде сумм с бесконечным пределом. При численной реализации необходимым условием является определение требуемого количества членов суммируемого ряда для построения решения с заданной точностью. Так, например, количество гармоник в формулах обращения преобразования Фурье (5), (6) зависит от минимальной пространственной длины волны в моделируемой среде и размеров расчётной пространственной области восстанавливаемого поля, которая задаётся конечными пределами интегрального преобразования. Кроме того, скорость сходимости суммируемого ряда зависит от гладкости функций моделируемого волнового поля.

Количество гармоник по Лагерру, необходимых для определения функций по формуле (8), зависит от задаваемого сигнала в источнике $f(t)$, выбора параметра h и значения временного

интервала восстанавливаемого волнового поля. Как можно определить требуемое количество гармоник и выбрать оптимальное значение параметра h, подробно рассмотрено в работе [5].

Анализ численных расчетов показывает, что основная погрешность вычислений в представленном алгоритме решения поставленной задачи связана с численной аппроксимацией пространственных производных. Поэтому для разностной аппроксимации использовался четвертый порядок точности. Для более точного описания производных вблизи границ раздела сильноконтрастных слоев среды, а также более точного учета граничных условий лучше использовать разностную сетку с переменным шагом дискретизации. Таким образом можно уменьшать шаг разбиения сетки при аппроксимации производных на определенных участках среды, что позволяет получить решение с требуемой точностью при меньшем количестве узлов разностной сетки.

Для решения системы линейных алгебраических уравнений (9) наиболее эффективным оказалось использование итерационного метода сопряженных градиентов. В этом случае для матриц систем большой размерности не требуется хранение всей матрицы в машиной памяти. Преимуществом этого метода является также быстрая сходимость к решению задачи при условии хорошей обусловленности матрицы системы. Наша матрица как раз обладает этим свойством за счет введенного параметра h. Задав нужное значение h, можно существенно ускорить сходимость итерационного процесса. Выбор оптимального значения h, в этом случае, осуществляется исходя из минимизации количества гармоник Лагерра и уменьшения количества итераций требуемых для нахождения решения для каждой гармоники.

Использование преобразования Фурье по одной из пространственных координат для решения поставленной задачи позволяет реализовать эффективное распараллеливание решения. В этом случае на каждом процессоре будет решаться независимая задача для каждой гармоники Фурье-преобразования. Дополнительно, при проведении расчётов на кластерных вычислительных комплексах с малым объёмом оперативной памяти доступной одному процессору, для решения больших пространственных задач (более 100 длин волн) осуществлено распараллеливание решения двухмерной пространственной задачи. На этом этапе проведения вычислений была реализована распараллеленная версия метода сопряженных градиентов для решения системы алгебраических уравнений для каждой гармоники Фурье. На уровне входных данных при задании модели среды это равносильно декомпозиции исходной области на несколько подобластей двухмерной задачи. Такой подход даёт возможность распределения памяти, как при задании входных параметров модели, так и при дальнейшей численной реализации алгоритма в подобластях. Схематическое разбиение области решаемой задачи показано на **Рис. 1.**

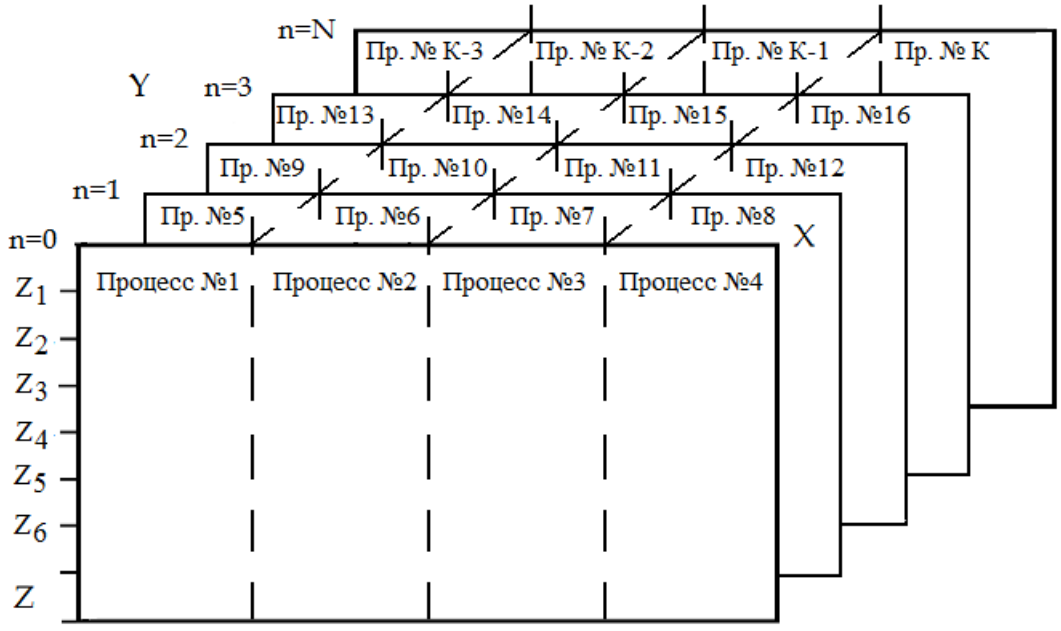


Рис. 1. Схема распределения расчётной области (n, x, z) по процессорам

5. Результаты численных расчетов

Для тестовых расчетов при моделировании волнового поля использовалась ограниченная область среды размерностью $(x, y, z) = (2\,км, 2\,км, 1.5\,км)$. Заданная модель среды состоит из двух однородных слоёв разделенных криволинейной границей в плоскости XZ в виде углового выступа. Глубина залегания границы на отрезке $[x_1, x_2] = [0\,км, 1\,км]$ - 0.75 км. Глубина залегания границы на отрезке $[x_1, x_2] = [1\,км, 2\,км]$ - 0.5 км. Физические характеристики слоёв были заданы следующими:

1) верхний слой - $c_p = 1.6$ км/сек, $c_s = 1.2$ км/сек, $\rho = 1.2$ г/см^3;
2) нижний слой - $c_p = 2$ км/сек, $c_s = 1.5$ км/сек, $\rho = 1.5$ г/см^3.

Волновое поле моделировалось от группы источников, состоящей из 10 точечных источников типа центра расширения, расположенных на линии в направлении оси X. Координаты первого источника $x_0 = 1$ км, $y_0 = 1$ км, $z_0 = 0.005$ км. Интервал между источниками $\Delta x = 0.01$ км. Каждый источник начинает работать с задержкой по времени $\Delta t = 0.006$ секунды относительно соседнего. Временной сигнал в источниках задавался в виде импульса Пузырёва:

$$f(t) = exp\left(-\frac{2\pi\, f_o (t - t_0)^2}{\gamma^2}\right) sin(2\pi\, f_0 (t - t_0)),$$

где $\gamma = 4$, $f_0 = 30$ Гц, $t_0 = 0.05$ сек.

Результаты численных расчетов волнового поля для заданной модели среды представлены на **Рис. 2-4**. На **Рис. 2** и **Рис. 3** изображены мгновенные снимки волнового поля для u_z компоненты скорости смещений в момент времени T = 0.4 и T = 0.6 секунды. Слева - в плоскости Y=1 км, справа - в плоскости X=1 км. Граница раздела слоёв показана сплошной линией. Из представленных рисунков видно отличие в распространении волнового поля в плоскости расположения источников и в плоскости перпендикулярной линии расположения источников. На **Рис. 4** изображён мгновенный снимок волнового поля для $u_z(x, y, z)$ в момент времени T = 0.6 секунды. Тестовые расчёты проводились на основе двух алгоритмов распараллеливания описанных выше. Сравнение результатов расчётов и времени счёта показывают хорошую эффективность распараллеливания в случае использования алгоритма разбиения исходной области модели при разностной аппроксимации на подобласти. Так как при увеличении количества разбиений время расчётов пропорционально уменьшается и имеет практически линейную зависимость.

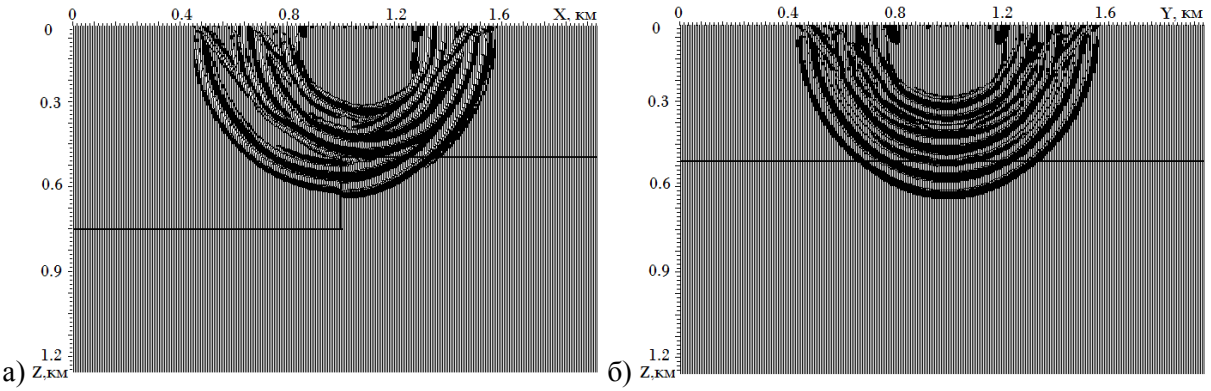


Рис. 2. Мгновенные снимки волнового поля в момент времени T = 0.4 секунды для u_z компоненты скорости смещений: а) в плоскости Y=1 км; б) в плоскости X=1 км

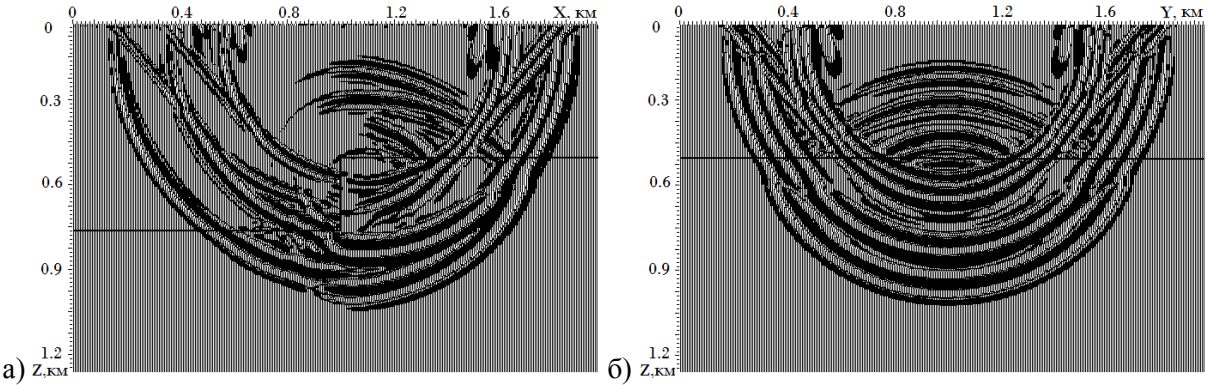


Рис. 3. Мгновенные снимки волнового поля в момент времени T = 0.6 секунды для u_z компоненты скорости смещений: а) в плоскости Y=1 км; б) в плоскости X=1 км

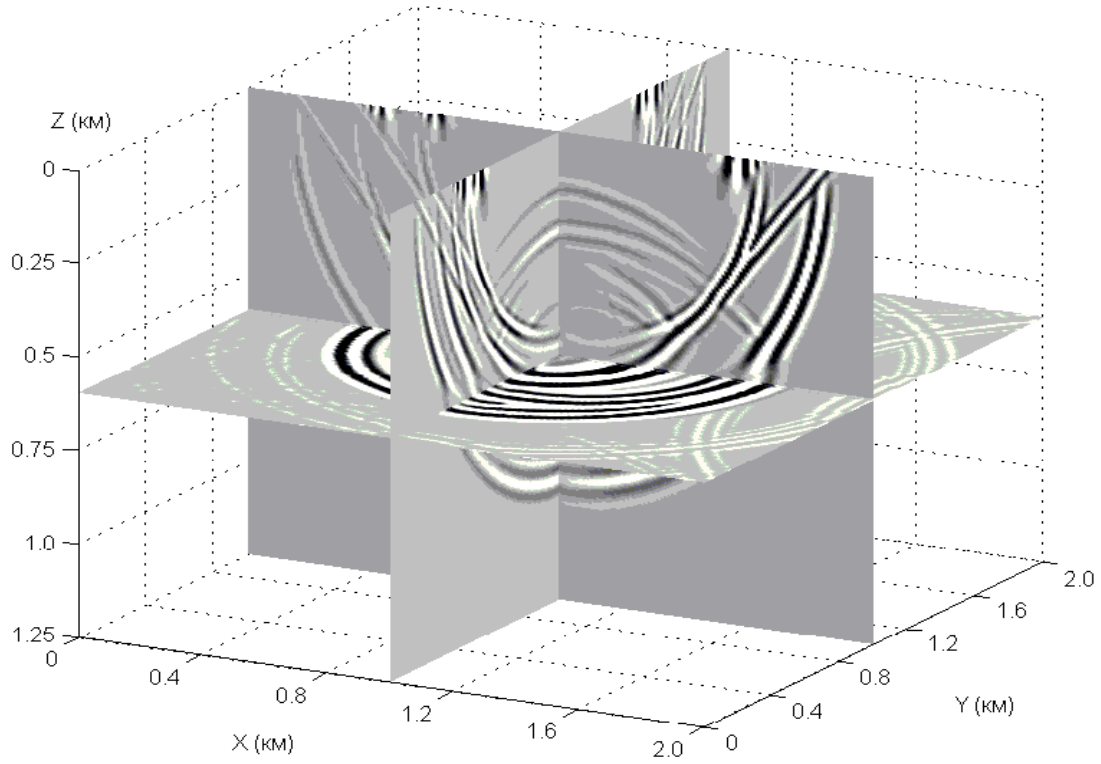


Рис. 4. Мгновенный снимок волнового поля для $u_z(x, y, z)$ в момент времени T = 0.6 секунды

Результаты численных расчетов волнового поля для более сложной и большей модели среды (~ 150 минимальных длин волн) представлены на **Рис. 6**. На данном рисунке изображён мгновенный снимок волнового поля для $u_z(x, y, z)$ в случае распространения сейсмических волн в модели среды представленной на **Рис. 5**. Моделировалось волновое поле от точечного источника типа центр давления с координатами $(x_0, y_0, z_0) = (7.5км, 3.5км, 0.01км)$. Временная зависимость в источнике задавалась по формуле Пузырёва. Основная частота 10 Гц. Плотности начиная со второго слоя рассчитываются по формуле Гарднера - ρ = 1.745*$V_p^{0.25}$, где ρ - плотность (г/см^3), V_p –скорость продольных волн (км/сек). В первом слое – вода ρ = 1 г/см^3.

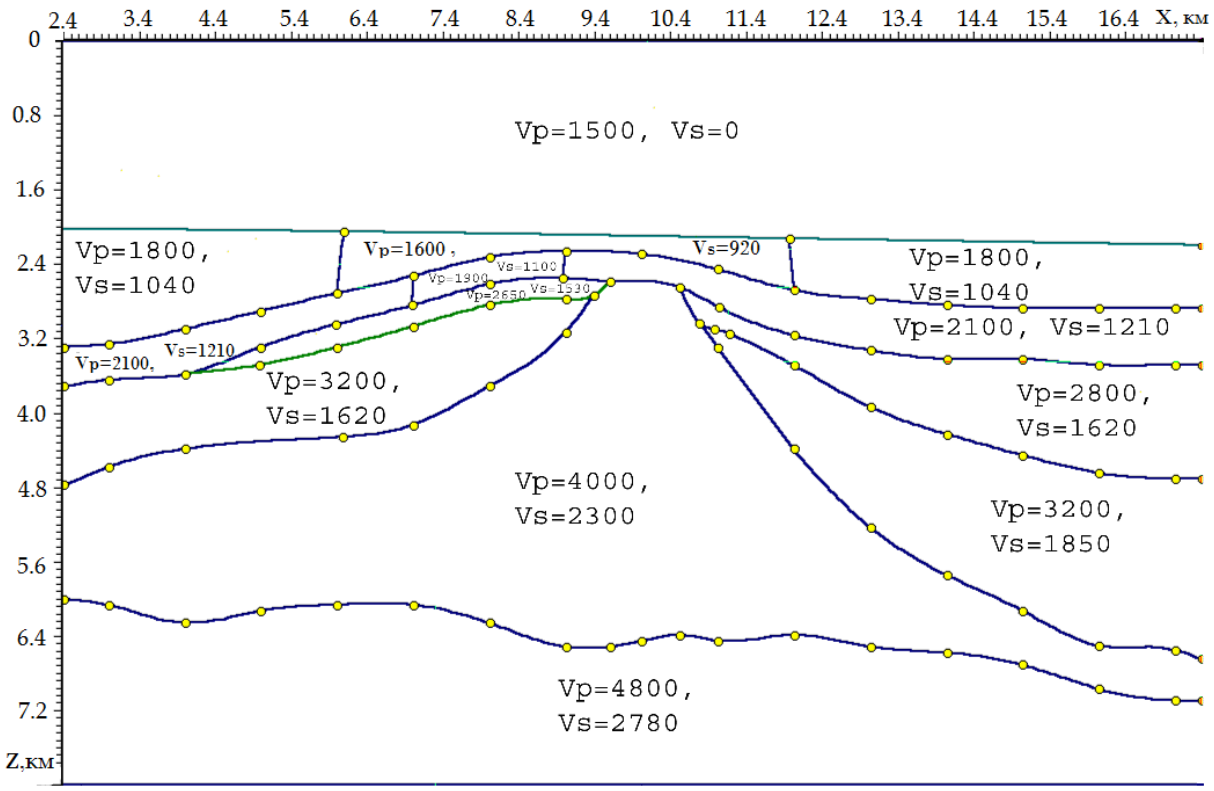


Рис. 5. Модель среды в плоскости XZ

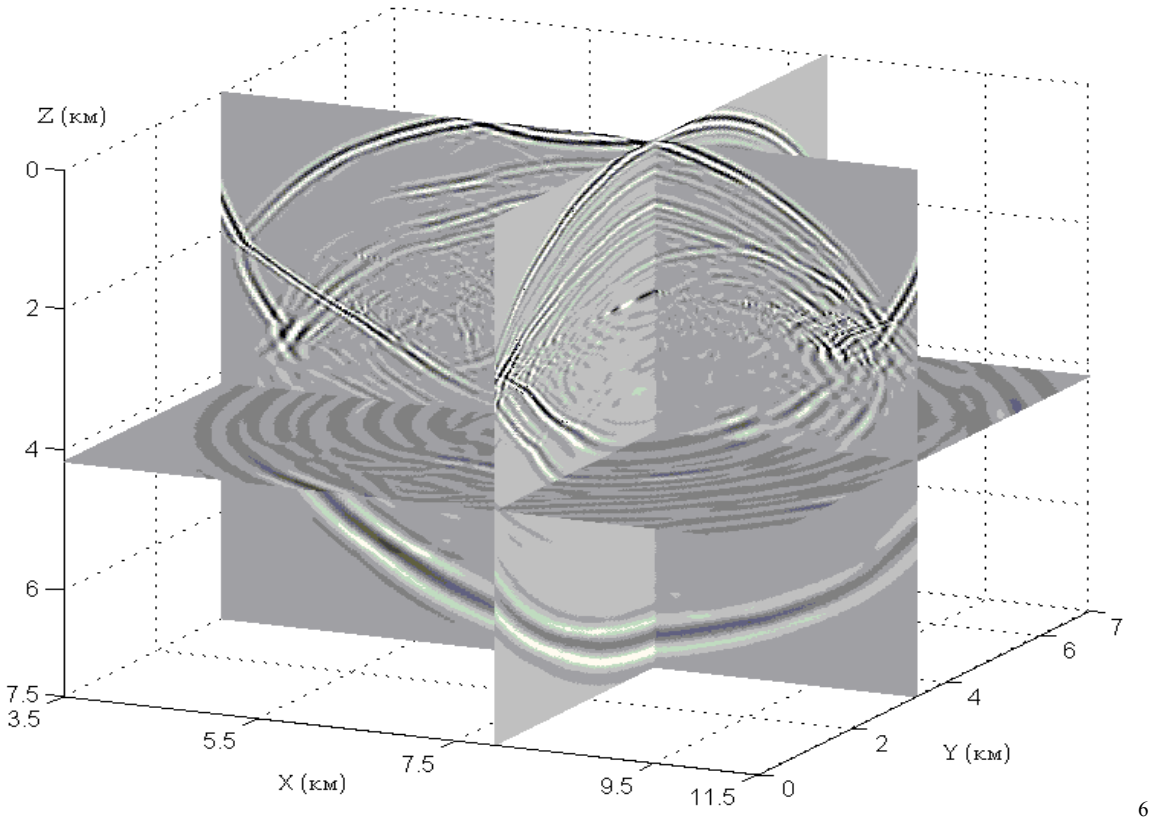


Рис. 6. Мгновенный снимок волнового поля для $u_z(x, y, z)$ в момент времени $T = 3$ секунды

6. Заключение

Анализ проведённых исследований и численные эксперименты показывают, что рассмотренный подход к решению поставленной задачи на основе комплексного использования аналитических преобразований и численных методов позволяет построить эффективные алгоритмы организации распараллеливания расчётов, а также осуществлять гибкое изменение алгоритма в зависимости от заданной модели среды.

Литература

1. Mikhailenko B.G. Spectral Laguerre method for the approximate solution of time dependent problems // Applied Mathematics Letters. 1999. № 12. P. 105–110.
2. Konyukh G.V., Mikhailenko B.G., Mikhailov A.A. Application of the integral Laguerre transforms for forward seismic modeling // Journal of Computational Acoustics. 2001. Vol. 9, № 4. P. 1523-1541.
3. Mikhailenko B.G., Mikhailov A.A., Reshetova G.V. Numerical modeling of transient seismic fields in viscoelastic media based on the Laguerre spectral method // Journal Pure and Applied Geophysics. 2003. № 160. P. 1207–1224.
4. Mikhailenko B.G., Mikhailov A.A., Reshetova G.V. Numerical viscoelastic modeling by the spectral Laguerre method // Geophysical Prospecting, 2003. № 51. P. 37–48.
5. Имомназаров Х.Х., Михайлов А.А. Использование спектрального метода Лагерра для решения линейной двумерной динамической задачи для пористых сред // Сибирский журнал индустриальной математики. 2008. Т. 11, № 2(35). С. 86-95.
6. Михайлов А.А. Моделирование сейсмических полей для 2.5D неоднородных вязкоупругих сред // Труды международной конференции "Математические методы в геофизике". Новосибирск. 2003. Часть 1. С. 146-152.
7. Levander A.R. Fourth order velocity-stress finite-difference scheme // Proc. 57-th SEG Annual Meeting. New Orleans. 1987. P. 234 – 245.*-

Децентрализованная самодиагностика распределённых вычислительных систем*

О.В. Молдованова

Сибирский государственный университет телекоммуникаций и информатики

Предложен децентрализованный алгоритм самодиагностики распределённых вычислительных систем (ВС), характеризующийся параллельным выполнением фаз тестирования и распространения диагностической информации. Проведено моделирование алгоритма с использованием библиотеки дискретного моделирования YACSIM. Приведены результаты моделирования для распространённых топологий распределённых ВС.

1. Введение

В последнее время постоянно увеличивается число трудоёмких задач, решаемых на распределённых вычислительных системах (ВС) [1]. Основным функциональным элементом таких систем является элементарная машина (ЭМ). Распределённые ВС характеризуются большемасштабностью – количество ЭМ в их составе может достигать $10^5 – 10^6$. Несмотря на высокую надёжность микроэлектронной базы, вероятность возникновения отказов в распределённых ВС повышается с ростом количества элементарных машин. Следовательно, организация отказоустойчивого функционирования таких систем требует создания алгоритмических и программных средств самоконтроля и самодиагностики.

В течение нескольких десятилетий был предложен ряд методик диагностики отказов в вычислительных системах [2–9]. Большинство работ базируется на стратегии, описанной в [2], согласно которой ЭМ способны тестировать друг друга. При этом исправные ЭМ могут безошибочно определить состояние тестируемых ими элементарных машин. А результат тестирования неисправными ЭМ может быть произвольным. Все результаты тестов собираются на высоконадёжной управляющей элементарной машине (центральном обозревателе), которая и определяет диагностический образ системы.

Опыт разработок в области самодиагностики большемасштабных распределённых вычислительных систем показывает, что централизованный подход ведёт к снижению производительности ВС и нарушает важный принцип их построения: отказ одной ЭМ влечёт за собой отказ всей системы. Эти проблемы могут быть решены при децентрализации процесса диагностирования.

Методология децентрализованной самодиагностики была сформулирована в работах [3, 4]. Её основой является предположение, что каждая исправная ЭМ может определить корректный диагностический образ всей системы, базируясь на результатах взаимных тестов других исправных элементарных машин этой ВС. Таким образом, передача тестовой информации осуществляется только между исправными компонентами системы, что гарантирует изоляцию неисправных и препятствует их влиянию на формирование диагностического образа распределённой ВС. В дальнейшем эта идея получила развитие в работах других исследователей [5–9].

Среди основных недостатков ранее предложенных алгоритмов следует отметить:

- накладываемые ограничения на тестовую топологию (например, в [7] используется тестовая топология в виде дерева);
- изменение состояния ЭМ не может происходить во время фазы тестирования [5];

* Работа выполнена при поддержке Совета по грантам Президента РФ (ведущая научная школа НШ 5176.2010.9), Российского фонда фундаментальных исследований (гранты 11-07-00109-а, 09-07-00095-а) и в рамках государственного контракта № 07.514.11.4015 с Минобрнауки РФ.

- использование диагностической модели сравнения, позволяющей установить лишь факт неисправности, т.е. выполнить самоконтроль ВС, без возможности диагностировать, какая конкретно ЭМ неисправна [9];
- последовательная передача диагностических сообщений.

В работе предлагается децентрализованный алгоритм самодиагностики распределённых ВС, обладающий следующими характеристиками:

- каждая исправная ЭМ тестируется только одной другой машиной;
- передача диагностической информации происходит только при изменении состояния тестируемой ЭМ;
- изменение состояния ЭМ может происходить во время фазы тестирования;
- фазы тестирования и распространения диагностической информации выполняются параллельно.

Для исследования алгоритма используется программное средство дискретного моделирования – YACSIM [10].

2. Постановка задачи

Рассматривается распределённая вычислительная система, состоящая из N элементарных машин (в дальнейшем узлов), соединённых друг с другом каналами связи.

Каждый узел может находиться в исправном или неисправном состоянии. Исправный узел обладает информацией о том, какие узлы являются его соседями. Кроме того, он способен инициировать тестирование соседнего узла и отвечать на тестовые запросы своих соседей. В качестве теста используются сообщения вида «Are you alive?». Исправный узел всегда отвечает на тестовый запрос в течение определённого периода времени (тайм-аута), передаёт диагностическую информацию своим соседям и может запрашивать их стать его тестерами.

Узел, находящийся в неисправном состоянии, не способен отвечать на любые сообщения от своих соседей, передавать им диагностическую информацию или запросы стать его тестерами. Таким образом, узлы в системе не могут информировать друг друга о своей неисправности.

Если в течение тайм-аута ответ на тестовый запрос от узла не получен, то узел-тестер делает заключение о неисправности тестируемого им узла. Величина тайм-аута определяется как функция задержки каналов связи.

Узел распределённой ВС в любой момент времени может перейти в неисправное состояние. В свою очередь неисправный узел может быть восстановлен и снова введён в эксплуатацию. При этом он получает всю необходимую информацию, касающуюся своих соседей, но не обладает информацией о текущем диагностическом образе системы.

Контроль и диагностика неисправности каналов связи алгоритмом не предусматривается. Поэтому не делается различия между неисправностью тестируемого узла и канала связи, соединяющего его с тестером.

Предполагается, что в начальный момент времени все узлы ВС исправны.

3. Децентрализованный алгоритм самодиагностики распределённых вычислительных систем

В работе предлагается децентрализованный алгоритм самодиагностики DSLD (Distributed System-Level Diagnostics) распределённых ВС. Алгоритмом предусматривается, что узлы обнаруживают изменение состояния своих соседей, а затем передают эту диагностическую информацию всем узлам системы. Диагностическая информация состоит из событий двух видов: переход узла из исправного состояния в неисправное и наоборот.

Для хранения и сбора диагностической информации в ходе выполнения алгоритма DSLD на каждом узле j вычислительной системы используются следующие структуры данных:

- массив $events_j[N]$ счётчиков событий, где N – количество узлов в диагностируемой системе. Если $events_j[i]$ – чётное число, то узел i исправен, иначе – не исправен. Начальное значение для элементов массива – 0;
- массив $testnbs_j[N]$, где N – количество узлов в диагностируемой системе; $testnbs_j[i] = 0$,

если узлы i и j не являются соседями; $testnbs_j[i] = 1$, если узел i тестируется узлом j; $testnbs_j[i] = 2$, если узел i тестирует узел j; $testnbs_j[i] = 3$, если узлы i и j являются соседями, но не тестируют друг друга; $testnbs_j[i] = 4$, если узлы i и j тестируют друг друга.

Обнаружение изменения состояния узлов в вычислительной системе осуществляется путём их периодического тестирования (сообщение типа «TestReq»). Каждый узел тестируется только одним исправным узлом. Если в течение определённого тайм-аута ответ (сообщение типа «TestResp») получен, узел диагностируется как исправный, иначе – как неисправный. Сразу после диагностирования изменения состояния тестирующий узел начинает передачу диагностической информации (сообщение типа «NewEvent») своим соседям, а те в свою очередь своим и т.д.

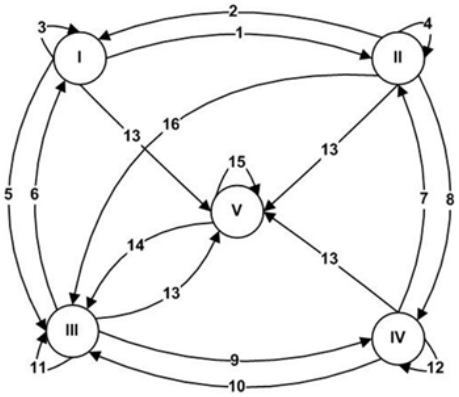
После того как узел i отправил диагностическое сообщение соседнему с ним узлу j, он ожидает от j подтверждения (сообщение типа «AckNewEvent») получения сообщения в течение определённого тайм-аута. Если такое подтверждение не поступает, i начинает распространение информации о неисправности узла j. Таким образом, удаётся получить сведения об изменении состояния узлов, не дожидаясь следующего раунда тестирования.

Если j исправен, то после получения диагностического сообщения от i он проверяет, является ли информация в этом сообщении новой, старой или такой же, какой обладает и он. Для этого используется сравнение значений из локальной версии массива $events_j$ и значений из массива $events_i$, полученного от узла i.

Если $events_j[k] = events_i[k]$ для всех $k \in \{1, 2, ..., N\}$, i и j имеют одинаковые данные о состоянии всех узлов в ВС. И узел j не передаёт полученное им сообщение от i далее своим соседям.

Если $events_j[k] > events_i[k]$ хотя бы для одного $k \in \{1, 2, ..., N\}$, то j обладает более новой информацией о состоянии некоторых узлов распределённой ВС. В этом случае j передаёт i свои данные о диагностическом образе вычислительной системы.

Если $events_j[k] < events_i[k]$ хотя бы для одного $k \in \{1, 2, ..., N\}$, то j содержит более старую информацию о состоянии некоторых узлов системы. В таком случае j обновляет свои данные и передаёт их далее своим соседям.



Состояния узлов:

I – исправный, бездействующий
II – исправный, ожидающий ответа на тестовый запрос
III – исправный, "брошенный", бездействующий
IV – исправный, "брошенный", ожидающий ответа на тестовый запрос
V – неисправный

→ – отправка сообщения
← – получение сообщения
↓ – сообщение не получено в течение тайм-аута

События:

1	E1: TestReq →	9	E9: TestReq →
2	$E2_1$: ← TestResp $E2_2$: ↓ TestResp $E2_3$: ← TestMeRepair от тестируемого узла $E2_4$: ← Repair от тестируемого узла	10	$E10_1$: ← TestResp $E10_2$: ↓ TestResp $E10_3$: ← TestMeRepair от тестируемого узла $E10_4$: ← Repair от тестируемого узла
3	$E3_1$: ← TestReq $E3_2$: ← NewEvent $E3_3$: ← TestMe $E3_4$: ← AckNewEvent $E3_5$: ← TestMeRepair $E3_6$: ← Repair $E3_7$: ↓ AckNewEvent	11	$E11_1$: ↓ AckTestMe $E11_2$: ← NewEvent $E11_3$: ← TestMe $E11_4$: ← AckNewEvent $E11_5$: ← TestMeRepair $E11_6$: ← Repair $E11_7$: ↓ AckTestMeRepair $E11_8$: ← TestReq $E11_9$: ↓ AckNewEvent
4	$E4_1$: ← TestReq $E4_2$: ← NewEvent $E4_3$: ← TestMe $E4_4$: ← AckNewEvent $E4_5$: ← TestMeRepair $E4_6$: ← Repair $E4_7$: ↓ AckNewEvent	12	$E12_1$: ← NewEvent $E12_2$: ← TestMe $E12_3$: ← AckNewEvent $E12_4$: ← TestMeRepair $E12_5$: ← Repair $E12_6$: ↓ AckTestMe $E12_7$: ← TestReq $E12_8$: ↓ AckTestMeRepair $E12_9$: ↓ AckNewEvent
5	$E5_1$: ← NewEvent $E5_2$: ← TestMe от тестера $E5_3$: ← TestMeRepair от тестера $E5_4$: ↓ AckNewEvent от тестера	13	E13: Узел не исправен
6	$E6_1$: ← AckTestMe $E6_2$: ← NewEvent $E6_3$: ← AckTestMeRepair	14	E14: Узел восстановлен
7	$E7_1$: ← AckTestMe $E7_2$: ← AckTestMeRepair	15	E15: Узел не исправен
8	$E8_1$: ← NewEvent $E8_2$: ← TestMe от тестера $E8_3$: ← TestMeRepair от тестера $E8_4$: ↓ AckNewEvent от тестера	16	E16: ↓ TestResp

Рис. 1. Диаграмма состояний узла при выполнении алгоритма.

Таким образом, алгоритм DSLD не использует время формирования диагностического образа на конкретном узле ВС для определения актуальности этого образа, а значит дополнительная синхронизация часов в узлах распределённой вычислительной системы не требуется.

В результате выполнения описанного выше алгоритма тестирования один или несколько узлов в системе могут быть «брошены», т.е. они перестают тестироваться другими узлами.

Все неисправные узлы являются «брошенными», поскольку после диагностирования их неисправности узел-тестер перестаёт их тестировать. Только после восстановления и нового ввода в эксплуатацию, т.е. изменения состояния на исправное, такие узлы будут снова тестироваться. Восстановленный узел обращается ко всем своим соседям по очереди с запросом на тестирование (сообщение типа «TestMeRepair»), пока не получит сообщение, подтверждающее согласие стать его тестером (сообщение типа «AckTestMeRepair»). После этого узел-тестер начинает раунд тестирования. Кроме этого, вновь исправный узел передаёт всем своим соседям информацию о своей исправности (сообщение типа «Repair»).

Исправный узел также может стать «брошенным», в случае если его узел-тестер поменял своё состояние на неисправное. Так же как и неисправный «брошенный» узел, он должен обратиться ко всем своим соседям по очереди с запросом на тестирование (сообщение типа «TestMeRepair»).

На рис. 1 приведена диаграмма состояний, описывающая поведение узла распределённой ВС при выполнении децентрализованного алгоритма самодиагностики системы.

4. Результаты моделирования

Моделирование алгоритма DSLD проводилось с использованием средства дискретного моделирования YACSIM [10]. Это событийно- и процессно-ориентированный инструментарий моделирования, позволяющий создавать взаимодействующие друг с другом процессы.

Программная реализация децентрализованного алгоритма самодиагностики распределённой ВС включает в себя следующие процессы:

- Init, выполняющий начальную инициализацию и создающий процессы MsgHandler и Lost;
- MsgHandler, отвечающий за получение и обработку всех видов сообщений (см. рис. 1);
- Lost, выполняющий рассылку сообщений «TestMe» для поиска соседнего узла-тестера;
- Test, реализующий рассылку тестовых запросов;
- Event, выполняющий сравнение присылаемой диагностической информации с имеющейся на узле;
- SendEvent, рассылающий диагностическую информацию соседним узлам.

При моделировании каждый узел ВС переключался между диагностическим алгоритмом и обычной рабочей нагрузкой. Время выполнения рабочей нагрузки на узле являлось экспоненциально распределённой случайной величиной со средним значением, равным 1 единице времени. По истечении этого времени сразу же выполнялся повторный запрос на использование процессора для выполнения рабочей нагрузки. Эти запросы обрабатывались в порядке очереди FIFO. Фоновые процессы, реализующие алгоритм самодиагностики, осуществляли запросы процессора через ту же самую очередь FIFO. Общее время моделирования составляло 1000 единиц времени.

Следующие виды задержек использовались при моделировании предложенного алгоритма:

- время формирования тестового запроса (2 единицы времени);
- время формирования ответа на тестовый запрос или подтверждающего сообщения (1 единица времени);
- время обработки ответа на тестовый запрос (1 единица времени);
- время обработки диагностического сообщения и обновления информации на узле (2,5 единицы времени);
- время определения номера соседнего узла для отправки ему диагностического сообщения (0,1 единицы времени);
- время формирования диагностического сообщения (2,5 единицы времени);

- время, затрачиваемое на пересылку любого сообщения от одного узла другому (1 единица времени);
- интервал между тестами (500 единиц времени).

Исследование разработанного алгоритма самодиагностики распределённых вычислительных систем проводилось для топологий: двумерная решётка (4 x 4), двумерный тор (4 x 4), четырёхмерный гиперкуб и трёхмерная решётка (4 x 4 x 4). Моделировалась ситуация, когда узел 1 переходил в неисправное состояние в момент времени 9 при первом раунде тестирования. При этом использовались 100 различных начальных значений для генерации случайных величин.

На рис. 2 показаны схема тестирования (а) и передача диагностических сообщений при обнаружении неисправности в узле 1 (б) для топологии двумерная решётка (4 x 4). Неисправность узла 1 обнаруживается его тестером, узлом 2, после чего неисправный узел полностью исключается из процесса диагностирования системы. Сплошными линиями на рис. 2, б показаны новые диагностические сообщения, после получения которых узел-приёмник начинает передачу диагностического образа системы в своей локальной окрестности. Сообщения, представленные пунктирными линиями (рис. 2, б), содержат данные, не отличающиеся от локальных данных узлов-приёмников.

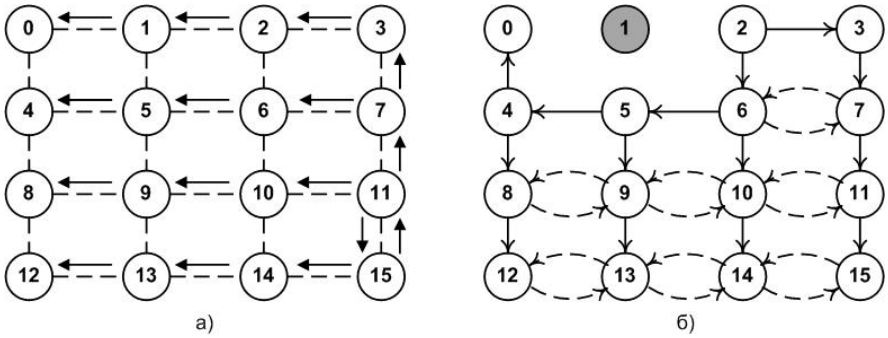


Рис. 2. Выполнение алгоритма самодиагностики на топологии двумерная решётка (4 x 4):
а) схема тестирования; б) обнаружение неисправности в первом узле;
○ – исправный узел; ● – неисправный узел;
← – направление тестирования (от тестера к тестируемому узлу);
⟵ – передача новой диагностической информации;
⇠ – передача диагностической информации, уже имеющейся на узле-приёмнике.

В табл. 1 приведены средние значения полученных оценок. Проанализировав результаты, можно сделать вывод, что увеличение размерности топологии системы не приводит к большому увеличению времени информирования всех узлов распределённой ВС о текущем диагностическом образе.

Таблица 1. Результаты моделирования

Топологии / Полученные оценки	Двумерная решётка (4 x 4)	Двумерный тор (4 x 4)	Четырёхмерный гиперкуб	Трёхмерная решётка (4 x 4 x 4)
Обнаружение неисправности	9,5	9,2	9,76	9,36
Время, когда последний узел системы узнаёт о неисправности	78,2	58,9	60,89	126,47
Количество передаваемых диагностических сообщений	26	39	38	204

Для топологии трёхмерная решётка (4 x 4 x 4) получены также оценки загрузки системы при выполнении предложенного алгоритма в единицах времени и в процентном соотношении от общего времени моделирования (табл. 2). Результаты показывают, что загрузка системы при выполнении алгоритма в отсутствии неисправностей очень мала и незначительно

увеличивается при наличии одной неисправности.

Таблица 2. Результаты моделирования для топологии трёхмерная решётка

Загрузка в отсутствии неисправностей	15,1 (1,51%)
Загрузка при одной неисправности	66,47 (6,647%)

5. Заключение

В работе предложен децентрализованный алгоритм самодиагностики DSLD распределённых ВС, характеризующийся параллельным выполнением фаз тестирования и распространения диагностической информации.

Проведено моделирование разработанного алгоритма с использованием средства дискретного моделирования YACSIM. Исследование проводилось для распространённых топологий распределённых вычислительных систем. Результаты моделирования показывают, что загрузка системы при выполнении алгоритма в отсутствии неисправностей очень мала, и она увеличивается незначительно при наличии неисправностей.

В дальнейшем планируется провести исследования алгоритма при наличии нескольких событий, в том числе событий восстановления узла после отказа, и реализовать предложенный алгоритм как часть системного программного обеспечения самодиагностики пространственно-распределённой мультикластерной вычислительной системы Центра параллельных вычислительных технологий федерального государственного образовательного бюджетного учреждения высшего профессионального образования «Сибирский государственный университет телекоммуникаций и информатики» и Института физики полупроводников им. А.В. Ржанова СО РАН.

Литература

1. Хорошевский В.Г. Архитектура вычислительных систем. – М.: МГТУ им. Н.Э. Баумана, 2008. – 520 с.
2. Preparata F.P., Metze G., Chien R.T. On the connection assignment problem of diagnosable systems // IEEE Trans. Electron. Comput. Dec. 1967. – vol. EC-16. – no. 6. – pp. 848–854.
3. Евреинов Э.В., Хорошевский В.Г. Однородные вычислительные системы. – Новосибирск: Наука, 1978. – 319 с.
4. Kuhl J.G., Reddy S.M. Fault-diagnosis in fully distributed systems // Proc. 11th Int. Symp. Fault-Tolerant Computing, June 1981. – pp. 100–105.
5. Bagchi A., Hakimi S.L. An optimal algorithm for distributed system level diagnosis // Proc. 21st Int. Symp. Fault-Tolerant Computing, June 1991.
6. Stahl M., Buskens R., Bianchini R. On-line diagnosis in general topology networks // IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, July 1992. – pp. 114–121.
7. Bianchini R., Stahl M., Buskens R. The Adapt2 on-line diagnosis algorithm for general topology networks // Proc. Globecorn, 1992. – pp. 610–614.
8. Bartha T. Efficient system-level fault diagnosis of large multiprocessor systems: thesis for the degree of Doctor of Philosophy. – Budapest, 2000. – 157 p.
9. Albini L.C.P., Duarte, Jr. E.P., Ziwich R.P. A generalized model for distributed comparison-based system-level diagnosis // J. Braz. Comp. Soc., 2005. – vol.10. – no. 3. – pp. 44–56. – ISSN 0104-6500.
10. Jump R.J. YACSIM: Reference Manual, V. 2.1. – March, 1993. – Режим доступа: http://oucsace.cs.ohiou.edu/~avinashk/classes/ee663/yac.ps (14.02.2012).

Реализация параллельного метода LU-SGS для задач газовой динамики на кластерных системах с графическими ускорителями

П.В. Павлухин

Московский Государственный Университет им М.В. Ломоносова

В статье предлагается эффективный алгоритм параллельного расчета газодинамических течений на вычислительных кластерных системах с графическими ускорителями. В основе алгоритма лежит неявная схема, приводящая к большим линейным системам с сильно разреженной матрицей, которые решаются методом приближенной факторизации LU-SGS (Lower-Upper Symmetric Gauss-Seidel). Параллельный алгоритм точно воспроизводит работу последовательного и обладает высокой степенью масштабируемости.

1. Введение

Рост производительности вычислительных систем в последние годы все больше и больше определяется увеличением числа вычислительных ядер в узле системы (и числа самих узлов в системе) и во все меньшей степени – увеличением производительности одного ядра. Кроме того, все большее распространение получают системы с новыми SIMD-архитектурами (в частности, системы с использованием графических ускорителей), которые требуют от прикладного программиста дополнительных знаний по устройству таких систем для написания эффективных программ под них. Ясно, что эффективность использования систем с массивно-параллельными архитектурами будет определяться масштабируемостью параллельных алгоритмов и их конкретными реализациями для этих систем. Поэтому особую важность на сегодняшний день приобретают задачи построения таких алгоритмов.

Численные методы газовой динамики делятся на два класса – явные и неявные. Основное достоинство явных схем – относительно простое вычислительное ядро и относительная простота распараллеливания. Но существенным их недостатком является жесткое условие устойчивости, ограничивающее выбор временного шага, и низкая скорость сходимости в схемах с высоким порядком аппроксимации. Это значительно сужает применимость явных схем в вычислительной практике. Неявные методы в большей части лишены этих недостатков и позволяют для ряда задач (в частности, стационарных) значительно сократить время счета. Однако основная проблема, связанная с использованием этих методов, - сложность построения для них эффективных параллельных алгоритмов, особенно под архитектуры современных графических ускорителей. Распространен подход, при котором работа неявного метода в точности соблюдается не на всей расчетной области, а лишь на параллельно обсчитывающихся частях. При этом для улучшения сходимости решения на каждом временном шаге применяются дополнительные итерации. С увеличением числа процессоров, доступных для решения задачи, области, где будет соблюдаться точная работа метода, становятся все меньше. В получаемом решении будет накапливаться все больше ошибок, что может значительно снизить скорость сходимости решения. Ясно, что масштабируемость таких алгоритмов сильно ограничена.

В настоящей работе предлагается параллельный алгоритм для неявного метода LU-SGS (Lower-Upper Symmetric Gauss-Seidel) и его реализация с помощью технологий CUDA и MPI для кластерных систем, основанных на множестве графических ускорителей. Основная его особенность – точное соблюдение работы исходного последовательного алгоритма для всей расчетной области. В отличие от [1], где предлагаемый алгоритм следует работе последовательного лишь в определенных подобластях исходной расчетной области, предлагаемый подход приводит к одинаковому решению не зависимо от того, используется ли одно процессорное ядро, или же расчет ведется на нескольких графических ускорителях.. Следует отметить, что параллельные алгоритмы для рассматриваемого метода, предназначенные для реализации на

традиционных многопроцессорных системах были предложены, в частности, в [6] и [7]. Эффективность LU-SGS относительно других методов в частности оценивается в [2]. Показано, в частности, что данный метод в некоторых задачах обеспечивает скорость сходимости, более чем на порядок превосходящую таковую для явной 3-хэтапной схемы Рунге-Кутты.

2. Метод приближенной факторизации LU-SGS

Опишем коротко метод LU-SGS. Детали можно найти в работах [3, 4, 5]. Рассмотрим систему уравнений Навье-Стокса для сжимаемой жидкости в декартовой системе координат, записанную в форме законов сохранения, которая дискретизируется по пространственным переменным методом конечных объемов. Применяя затем неявную схему интегрирования по времени, в результате получаем систему дискретных уравнений, которая решается методом установления по псевдо-временной переменной с использованием неявной дискретизации и ньютоновских итераций. В результате необходимо решить линейную систему уравнений для определения итерационного инкремента $\delta^s \vec{q}$:

$$(D+L+U)\delta^s \vec{q}_i = -\vec{R}_i^{n+1,s} \tag{1}$$

Где D – блочно-диагональная матрица, а L и U - блочно-диагональные нижняя и верхняя треугольные. Уравнение (1) преобразуется к следующему виду:

$$(D+L)D^{-1}(D+U)\delta^s \vec{q}_i = -\vec{R}_i^{n+1,s} - LD^{-1}U \tag{2}$$

Метод LU-SGS сводится к приближенной факторизации левой части уравнений (1), которая получается, если в этих уравнениях (2) пренебречь последним слагаемым правой части. Следует заметить, что этот член пропорционален Δt^2, и такое приближение видится вполне оправданным. Факторизованные таким образом уравнения распадаются на две подсистемы:

$$\begin{aligned} (D+L)\delta^s \vec{q}_i^{\,*} &= -\vec{R}_i^{n+1,s} \\ (D+U)\delta^s \vec{q}_i &= D\delta^s \vec{q}_i^{\,*} \end{aligned} \tag{3}$$

Первая система уравнений в (3) имеет нижне-треугольную блочную матрицу, каждый элемент которой представляется блоком размерностью (5 x 5), а вторая – соответственно верхнетреугольную. Это позволяет эффективно вычислить их решения за два расчетных цикла по ячейкам: первый - в прямом направлении (от первой ячейки к последней), а второй - в обратном. При этом необходимо обращать только диагональные блоки, т. е., матрицы размером (5 x 5). Получающаяся при этом итерационная невязка $\delta^s \vec{q}$ служит для обновления итерационного вектора, после чего процедура (3) повторяется.

Таким образом, если рассматривать алгоритм LU-SGS как объект для распараллеливания, то его последовательный вариант выглядит как два прохода (прямой и обратный) по ячейкам расчётной области. Оба прохода осуществляются по одной последовательности ячеек, но в разных направлениях.

В общем случае значения искомых функций в ячейке на следующем временном слое зависят от всех ячеек расчётной области на текущем временном слое, что затрудняет построение параллельного алгоритма.

3. Параллельная версия LU-SGS

Построение параллельного алгоритма будет проводиться для двумерной расчетной области со структурированной сеткой. При этом данный алгоритм достаточно просто обобщается и на случай трехмерного пространства. В исходной последовательной версии метода фиксируется порядок ячеек (элементов сеточного разбиения), в соответствии с которым происходит прямой и обратный обход расчетной области (т.е. задается очередность вычислений в последовательности ячеек). Для вычисления изменения вектора решения в некоторой ячейке требуются данные от геометрически соседних ячеек, причем вычислительные операции с данными от ячеек-соседей, которые стоят в очереди обхода раньше текущей ячейки, отличаются от таковых для ячеек-соседей, которые расположены в очереди обхода позже. Для корректной работы метода необходимо, чтобы порядок обхода фиксировался неизменным, т.е. чтобы положение в очереди

обхода всех геометрически соседних ячеек («до/после») не менялось относительно текущей ячейки. Однако сам порядок обхода можно определять любым способом – не обязательно, чтобы следующая ячейка была геометрическим соседом предыдущей. По сути, задача построения параллельного алгоритма сводится к выбору такого порядка обхода в параллельном счете, который будет эквивалентен некоторому обходу в последовательном счете.

Поскольку алгоритм строится для системы многих графических ускорителей, распределенных по узлам кластера, можно выделить два уровня параллелизма: inter-GPU – согласованная работа между графическими ускорителями – и intra-GPU – параллельный счет внутри графического ускорителя. Параллельный счет на уровне inter-GPU полностью аналогичен версии многопроцессорного счета, представленной в [6], а именно, исходная расчетная область разбивается на блоки (по числу графических ускорителей), расположенные стык в стык и образующие на плоскости структуру двумерной решетки. Блоки делятся на два множества: два геометрически соседних блока всегда принадлежат разным множествам – таким образом, эти множества образуют «шахматное» разбиение блоков, каждый из них (блоков) считается на отдельном GPU. Внутри каждого блока выполняется обсчет его частей в соответствии с заданным порядком для множества (назовем их “black” и “white”), к которому принадлежит блок.

Для блоков из множества “black” вычисления проводятся в следующем порядке:

1. Из блоков отсылаются соседям в white копии К граничных частей ячеек (Рис 1. Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками., слева), которые являются соответствующими геометрическими соседями для ячеек из блоков white, при этом отсылаемые ячейки для white не являются обсчитанными. В это же время в блоках black также запускается обход по всем ячейкам, кроме граничных и кроме ячеек, являющихся соседними к граничным внутри блока, т. е. обход по всем ячейкам, кроме «двойного» периметра ячеек (поскольку модифицируются данные в необсчитанных еще ячейках, находящихся позже в очереди обхода).

2. В блоках black граничные ячейки обновляются из присланной копии К (операция корректна, т.к. никаких действий над граничными ячейками в black еще не производилось).

3. В блоках black по флагу проверяется, присланы ли ячейки из соседних white-блоков, которые уже являются обсчитанными, и обновились ли граничные ячейки в блоке. После подтверждения этого в блоках black выполняется обход по оставшимся приграничным ячейкам из «двойного» периметра блока с обращениями к полученным ячейкам из white.

Для множества “white” счет идет в таком порядке:

1. Во всех блоках из white запускается обход по первой половине внутренних ячеек, то есть обход будет сделан только по части внутренних ячеек. В момент счета из соседних блоков black присылаются необсчитанные ячейки, соседние к граничным в блоках white.

2. Затем в white по флагу проверяется, получены ли ячейки из соседних блоков (к этому моменту они будут доставлены, если время счета указанной выше части ячеек в white будет больше времени передачи ячеек из black), и выполняется обход по граничным ячейкам, причем используется копия К ячеек полученных из black, в этой копии также обновляются ячейки (полученные из black, они еще не обсчитаны). После завершения этого обхода из white соседям в black отправляются граничные ячейки блока white, которые являются соседями для соответствующих ячеек из black, и обновленная копия К ячеек из black.

3. Наконец, в блоках white выполняется обход оставшейся второй половины внутренних ячеек. Обратный обход в блоках строится в соответствии с прямым: в black он начинается с "двойного периметра" приграничных ячеек в обратном порядке и заканчивается на внутренней оставшейся их части (так же в обратном порядке); в white сначала выполняется обход второй половины внутренних ячеек, затем граничных и, наконец, оставшейся первой части (в обратном порядке).

Построенный выше алгоритм эффективен при параллельном счете блоков с примерно равным количеством ячеек, поскольку пересылка ячеек совмещена со счетом: передача данных идет одновременно с вычислениями. Корректность алгоритма подтверждает эквивалентный обход всей расчетной области, для которого результаты счета последовательного алгоритма совпадают с таковыми для рассмотренного параллельного. Он строится следующим образом: сначала выполняется обход внутренней части (без «двойного периметра») блоков “black”; затем в каждом блоке “white” обходится сначала первая половина внутренних ячеек, а потом гранич-

ная часть; далее, во всех блоках “black” обходятся «двойные» граничные части и, наконец, в блоках “white” обходятся оставшиеся вторые половины внутренних ячеек. Пример такого обхода для 2x2 блоков показан на Рис 1**.** Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками., справа (числами обозначен порядок в очереди обхода).

Выше был изложен алгоритм распределения вычислений между графическими ускорителями, следующий уровень – intra-GPU – счет внутри одного ускорителя. Архитектура GPU подразумевает одновременный счет многих ячеек, но здесь возникает проблема: как организо-

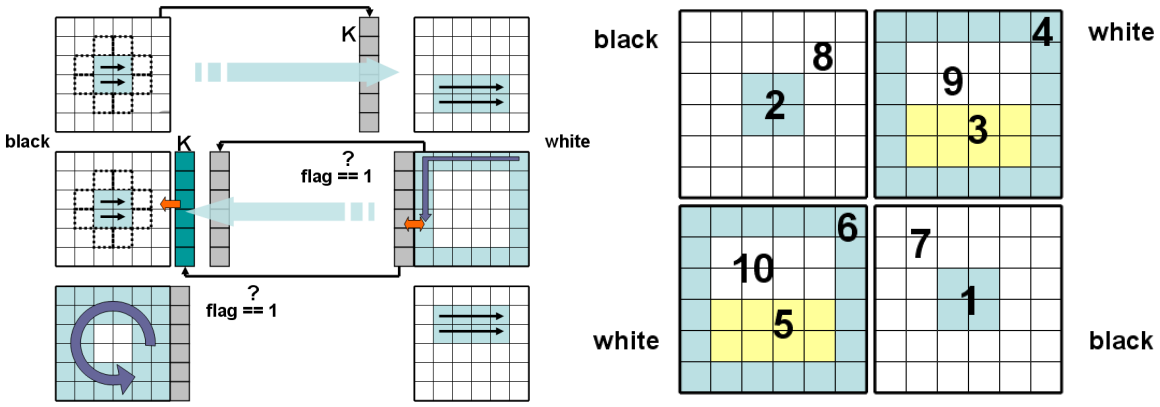


Рис 1. Слева - последовательность вычислений в блоках из множеств “black” и “white”, справа - обход области с 2x2 блоками.

вать очередь обсчета ячеек, чтобы положение в ней геометрически соседних ячеек оставалось неизменным относительно текущей, т.е. нужно гарантировать, что при обращении к каждой соседней ячейке она всегда была бы уже обсчитана (находится в очереди раньше текущей) или еще не обсчитана (в очереди – позже текущей). Средств глобальной синхронизации, которые помогли бы решить данную проблему, в современных GPU нет. Чтобы обеспечить корректность счета, можно разбить параллельный обсчет подобласти на два последовательных – сначала по одной части подобласти, затем – по другой. Ответ на вопрос о нужном разбиении подобласти по сути являет собой свойство автомодельности параллельного алгоритма – разделение ячеек внутри подобласти (внутренней или граничной части блока) на два множества происходит аналогично разбиению блоков исходной расчетной области: две геометрически соседних ячейки всегда принадлежат разным множествам. Таким образом, два этих множества вновь образуют «шахматное» разбиение ячеек. В итоге, счет каждой подобласти блока – внутренней или граничной его части – выполняется в два последовательных этапа: сначала на GPU запускается счет ячеек из “black” (при этом все соседние ячейки будут из множества “white”, которые в данной подобласти будут всегда еще не обсчитанными), затем, после его завершения, запускается счет ячеек из “white” (все соседние ячейки будут уже из “black”, которые в данной подобласти будут всегда уже обсчитанными). Обращение к ячейкам из другой подобласти всегда будет выполняться как к обсчитанным или необсчитанным – порядок в этом случае определяется порядком обсчета подобластей в блоке.

4. Реализация

Программный код был написан с использованием MPI и CUDA ToolKit 4.0. Для увеличения производительности счета и скорости копирования данных между графическими ускорителями внутренние ячейки блока хранились в виде двумерного массива, а граничные – в виде одномерного (в направлении обхода ячеек – сначала по «нижней», затем по «правой», «верхней» и «левой» границам блока). Для совмещения счета и передачи ячеек во времени использовались неблокирующие функции приема/передачи MPI и несколько CUDA-«потоков» (Stream): в одном из них выполнялось вычислительное ядро (kernel), в других параллельно шло копирование

данных. Для синхронизации вычислений использовались функции MPI_Wait и функции работы с событиями (event) в API CUDA. Все вычисления проводились на GPU, ядро центрального процессора использовалось лишь для копирования данных между оперативной памятью и памятью ускорителя и для обмена данными между процессами посредством MPI.

5. Результаты тестов

В качестве тестовой была выбрана задача о коническом теле, мгновенно помещенном в однородный сверхзвуковой поток газа с числом Маха M=1.6 (Рис. 2**.** Распределение плотности в задаче о коническом теле при t = 0.1 с.). Решалась нестационарная

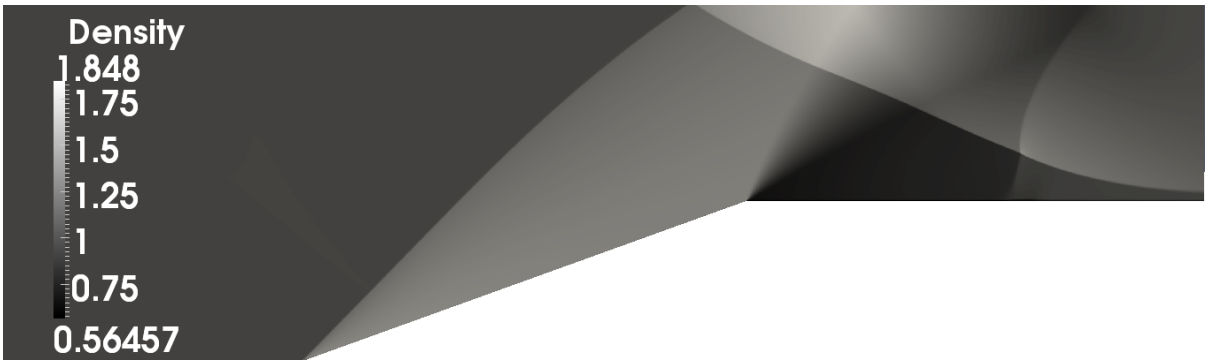


Рис. 2. Распределение плотности в задаче о коническом теле при t = 0.1 с.

задача об образовании ударно-волновой структуры в результате взаимодействия потока с поверхностью тела. Измерения проводились на суперкомпьютере К-100 в ИПМ им М. В. Келдыша. В каждом его узле установлены 3 ускорителя Nvidia Tesla C2070 и 2 процессора Intel Xeon X5670. Во всех измерениях использовалась двойная точность вычислений.

В первом тесте сравнивалась скорость работы одного ядра CPU и одного графического ускорителя. Расчет задачи с сеточным разрешением 3600x1080, 100 шагов по времени, длился на одном ядре 3580с, на одном GPU – 250с. Таким образом, использование одного ускорителя дает прирост производительности в 14.3 раза. Анализ кода с помощью дизассемблера (cuobjdump) и профилировщика CUDA показал, что производительность на GPU ограничена пропускной способностью памяти. Дело в том, что вычислительное ядро программы довольно сложное, и доступных 63 регистра на поток (thread) в архитектуре Fermi недостаточно, чтобы хранить все локальные переменные, поэтому возникает registers spilling – часть временно не используемых переменных записываются (а позднее считываются) в медленную глобальную память. В данной задаче spilling занимал бóльшую пропускную способность памяти GPU.

В следующем тесте исследовалась масштабируемость времени счета на одном GPU при использовании разных разрешений сеток. Базовое разрешение – 450x135 (≈ 60 тыс ячеек) заменялось на другие, которые содержали в 4^n (n = -2, ... 2, 3)раз больше/меньше ячеек. Результаты измерений представлены на Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU., слева. Видно, что с увеличением разрешения относительно базового время счета растет линейно с увеличением числа ячеек расчетной области, однако эта линейность нарушается, когда разрешение сетки уменьшается: время счета уменьшается лишь в ≈ 2.5 раза при уменьшении разрешения в 4 раза (1/4 x – 15390 ячеек), при дальнейшем уменьшении разрешения (1/16 x – 3944 ячейки) счет сократился только в 1.75 раза. Это связано с тем, что в расчетных блоках с малым числом ячеек значительный вклад при счете граничной части начинает вносить время доступа к соседним ячейкам из внутренней части блока – запросы к ним в глобальную память GPU не объединяются в меньшее число транзакций (non-coalesced access), поскольку они происходят из соседних (в одномерном массиве) ячеек граничной части к геометрически соседним, но разреженным (разнесенным) в двумерном массиве ячейкам внутренней части.

Основной тест, демонстрирующий масштабируемость алгоритма с увеличением числа GPU, показан на Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной

сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU. справа. Расчет производился на сетке 3600x1080, всего 100 шагов по времени. Непропорциональное уменьшение времени счета при переходе от одного к двум GPU связано с появляющимися накладными расходами по обмену данными между графическими ускорителями. При дальнейшем же увеличении числа GPU наблюдается практически линейная масштабируемость времени счета. Так, например, при задействовании в 32 раза большего числа GPU (64 вместо 2) время счета сократилось в 26 раз.

6. Заключение

В работе предложен новый параллельный алгоритм, реализующий метод LU-SGS для задач газовой динамики на кластерной вычислительной системе распределенных графических ускорителей. Показана корректность этого алгоритма, описана его реализация с помощью технологий MPI и CUDA. Результаты вычислительных экспериментов подтвердили эффективность алгоритма на задачах с большими расчетными сетками, вплоть до таких, когда на один GPU

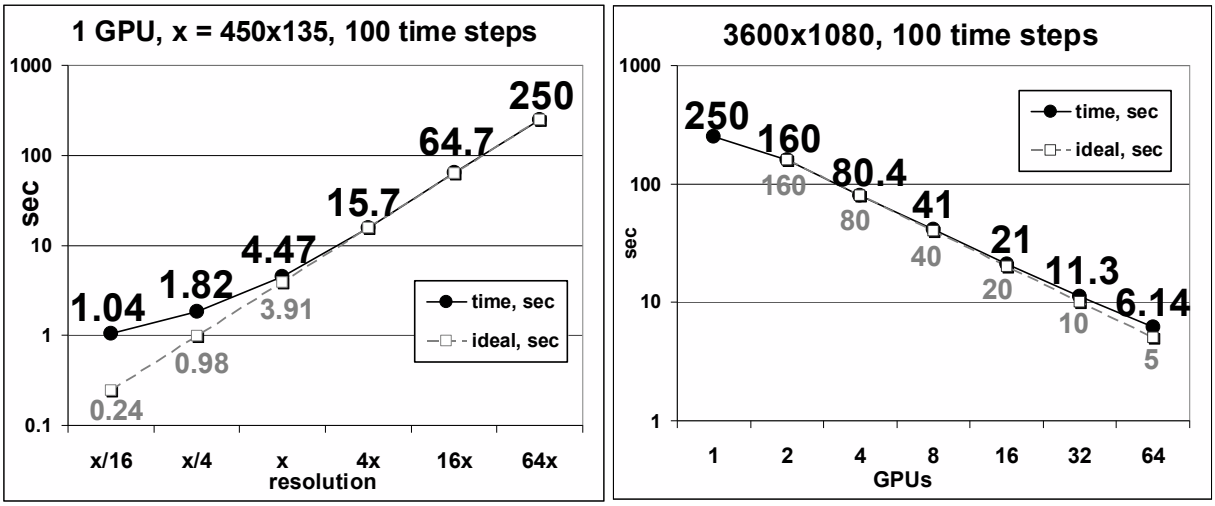


Рис. 3 Слева - время счета на одном GPU с разными разрешениями расчетной сетки, справа - время счета с расчетной сеткой 3600x1080 с разным числом GPU.

приходится порядка 60 тыс ячеек. Масштабируемость счета на таких задачах оказалась близкой к линейной. Такая эффективность достигается за счет совмещения счета и передачи данных между ускорителями: при достаточно больших разрешениях расчетной сетки время обмена данными полностью перекрывается временем счета. По результатам тестов производительность GPU более чем на порядок превосходит производительность процессорного ядра, что свидетельствует о целесообразности применения графических ускорителей в рассматриваемых задачах. В дальнейшем планируется реализовать алгоритм для трехмерного случая и провести оптимизацию самого вычислительного ядра под архитектуру GPU.

Литература

1. D. Sharov, H. Luo, J.D. Baum, R. Loehner Implementation of unstructured grid GMRES-SGS method on shared-memory, cache-based parallel computers // AIAA-2000-927 – 38th Aerospace Sciences Meeting and Exhibit. – 2000.
2. Y. Sun, Z.J. Wang, Y. Liu, C.L. Chen Efficient Implicit LU-SGS Algorithm for High-Order Spectral Difference Method on Unstructured Hexahedral Grids // AIAA 2007-313 – 45th Aerospace Sciences Meeting and Exhibit – 2000.
3. A. Jameson, E. Turkel Implicit schemes and LU decomposition // Math.of Comp., v.37, № 156, pp.385-397, 1981.

4. I. Menshov, Y. Nakamura On implicit Godunov's method with exactly linearized numerical flux //Computers & Fluids, 29 (6), pp. 595 – 616, 2000.
5. I. Menshov, Y. Nakamura Hybrid Explicit-Implicit, Unconditionally Stable Scheme for Unsteady Compressible Flows //AIAA Journal,vol. 42, № 3, pp. 551-559, 2004.
6. П.В. Павлухин, И.С. Меньшов. Эффективная реализация метода LU-SGS для задач газовой динамики. //Научный вестник МГТУ ГА, Т.165, 3, с.46 - 55, 2011
7. Семёнов И.В., Ахмедьянов И.Ф., Уткин П.С. Разработка вычислительного комплекса для решения двух- и трёхмерных задач газодинамики реагирующих течений на многопроцессорных ЭВМ // Материалы 6 Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах». – 2007. – Т. 2. – С. 138 – 145.

Реализация базовых операций целочисленной арифметики в гетерогенных системах*

А.В. Панюков, С.Ю. Лесовой

Национальный исследовательский университет ЮУрГУ

В работе рассмотрены вопросы масштабируемости выполнения операций целочисленной арифметики с операндами произвольной длины. Отмечено, что классический алгоритм деления «столбиком», в отличие от остальных, не является масштабируемым. Предложено решение проблемы масштабируемости операции деления посредством применения метода Ньютона. Даны оценки сложности и масштабируемости.

1. Введение

При программировании вычислительных задач, современные компьютеры позволяют использовать не только ресурсы центрального процессора, но и графического ядра, представляя собой гетерогенную систему с высоким вычислительным потенциалом. Воспользоваться этим потенциалом позволяет, например, высокоуровневая платформа OpenCL, предоставляющая средства управления ресурсами графического ядра в дополнение к центральному процессору.

Длинные числа являются попыткой преодолеть ограничение платформ на диапазон представления целых чисел, определяемый типами данных, поддерживаемых стандартами языка. Необходимость в очень больших целых числах возникает при использовании криптографических алгоритмов или при решении задач, требующих точных дробно-рациональных вычислений. Проблема увеличения производительности выполнения операций над длинными числами остается актуальным уже длительное время. Основные результаты алгоритмического подхода подробно представлены в классическом труде Д. Кнута [1]. С появлением многопроцессорных систем стало возможным дальнейшее повышение эффективности таких операций на основе параллельных версий классических алгоритмов.

Возможности различных параллельных реализаций операции умножения: алгоритм Шенхаге – Штрассена, алгоритм Карацубы, алгоритм Тоома –Кука, – рассмотрены, например, в статье Е.Г. Качко [2], где отмечена конкурентоспособность классического алгоритма умножения «столбиком» при большом числе параллельных процессов.

В работе [3] предложены реализации целочисленных операций с применением классических алгоритмов. В частности рассмотрены операции сложения–вычитания, умножения на цифру (под цифрой понимается минимальный неделимый элемент чисел произвольной длины) и умножение чисел произвольной длины. Отмечено, что классический алгоритм деления «столбиком», в отличие от остальных, не является масштабируемым, т.е. наличие многопроцессорной среды не повышает его производительность.

В данной работе, являющейся развитием работы [3], предложено решение проблемы масштабируемости операции деления посредством применения метода Ньютона. Даны оценки сложности и масштабируемости предложенных алгоритмов.

2. Теоретические оценки возможности реализации

Приведенные ниже оценки сложности выполнения операций алгоритмами, предложенными в работе [3], были сделаны при ряде условностей и допущений, а именно:

а) количество доступных вычислительных элементов GPU не менее чем разрядность меньшего из операндов в случае операции сложения, а в случае операции умножения – не менее произведения разрядности операндов.

* Работа поддержана РФФИ (проект 10-07-96003-р_урал_а)

б) объем доступной оперативной/регистровой памяти GPU позволяет разместить исходные числа, результат и все промежуточные значения.

в) не учитывается пересылка данных между оперативной памятью и памятью GPU, предполагается, что исходные данные уже находятся в видеопамяти и результаты остаются в ней же.

Приведенные допущения позволяют получить оценки времени выполнения основных арифметических операций, приведенные в табл. 1.

Таблица 1. Оценки среднего времени выполнения операций с помощью классических алгоритмов

Операция	Оценка времени выполнения	Количество параллельных процессов
Сложение	$2t_A$	$L + 1$
Умножение на цифру	t_M+2t_A	$U + 1$
Умножение	$t_M+2t_A + 2t_A\log_2 L$	$L + U + 1$
Деление на цифру	$t_D U$	1
Деление	$(U – L + 1)(t_M+4t_A)$	$L+1$

В табл. 1 приняты следующие обозначения:

1) t_A – время выполнения обычной операции сложения над разрядом длинного числа одним вычислительным элементом,
2) t_M - время выполнения обычной операции умножения над разрядом длинного числа одним вычислительным элементом,
3) t_D - время выполнения обычной операции деления над разрядом длинного числа одним вычислительным элементом,
4) L – длина наименьшего из операндов (количество значащих цифр в используемой позиционной системе счисления),
5) U – длина наибольшего из операндов.

Из таблицы 1 видно, что классический алгоритм деления «столбиком» на цифру, в отличие от остальных, не является масштабируемым (при увеличении числа доступных процессов, время выполнения не изменяется, дополнительные процессы не могут быть использованы), а время его выполнения линейно зависит от разрядности чисел. При делении на многоразрядное число время также линейно зависит от разрядности чисел, но появляется возможность использования дополнительных процессов.

В работе [1, с. 304 – 425] показан способ реализации в двоичной системе счисления операции деления через операцию умножения с помощью модифицированного алгоритма Ньютона. В следующем разделе приведена реализация указанного способа в качестве метода фундаментального класса `overlong [4]`, дана оценка требуемых вычислительных ресурсов при его использовании.

3. Операция деления

Чтобы разделить число $\boldsymbol{u}$ на число $\boldsymbol{v}$, можно сначала найти достаточно точное приближение к числу $1/\boldsymbol{v}$, затем умножить его на $\boldsymbol{u}$, что даст приближение к $\boldsymbol{u}/\boldsymbol{v}$. Если L_u и L_v – длины операндов, то длина целочисленного ответа будет не более $L_u – L_v + 1$. Число $1/\boldsymbol{v}$ содержит не менее $L_v + 1$ значащих нулей в старших разрядах, кроме того для получения правильного результата деления оно должно содержать еще не менее $L_u – L_v + 1$ значащих цифр. Таким образом, необходимая и достаточная точность вычисления величины $1/\boldsymbol{v}$ составляет величину $b^{-(L_u+2)}$, где b – основание системы счисления.

Применение метода Ньютона к задаче нахождения корня уравнения $\boldsymbol{f(x) = v – 1/x}$ состоит в последовательном вычислении $\boldsymbol{x_{k+1} = (2 – v\, x_k)x_k}$, $k = 0. 1, 2, \ldots$, где $\boldsymbol{x_0}$ – начальное приближение, вычисленное с достаточной точностью. При $x \geq 1$ функция $\boldsymbol{f(x)}$ является дважды непрерывно дифференцируемой и строго выпуклой. В этом случае метод Ньютона обладает квадратичной скоростью сходимости, т.е. количество значащих разрядов после выполнения очередной итерации будет удваиваться.

Легко проверить, что $x_0 = \left\lfloor \left(b^5 - 1\right) / \left(b^2 v_1 + b \cdot v_2 + v_3\right) \right\rfloor \cdot b^{-L_v - 6}$, где v_1, v_2, v_3 – старшие разряды числа $\boldsymbol{v}$, является приближением величины $1/\boldsymbol{v}$ с точностью не хуже $b^{-(L_v+2)}$. Таким образом, потребуется выполнить по методу Ньютона не более $\left\lceil \log_2 \left(\frac{L_u + 3}{L_v + 2} \right) \right\rceil$ итераций.

Изложенное выше позволяет предложить редакцию метода Ньютона для получения обратной величины с использованием только операций целочисленной арифметики. Реализация алгоритма как метода фундаментального класса overlong [4] представлена на рис. 1.

```
#include "overlong.h"

overlong overlong::
        Division(const overlong& b){
if (b.leng<3)
          return operator/(*this,b);

R1: \\Начальное приближение
overlong z;
z.leng = 6;
z.d = new unsigned short[z.leng];
z.d[0] = 0;  z.d[1] = 0;
z.d[2] = 0;  z.d[3] = 0;
z.d[4] = 0;  z.d[5] = 1;
overlong temp;
temp.leng = 3;
temp.d = new unsigned
                   short[temp.leng];
for (int i=0;i<3;i++)
 temp.d[2-i] = b.d[b.leng-i-1];

z/=temp; int newL = z.nsd();

if (newL<z.leng){
   unsigned short* newD =
            new unsigned short[newL];
   for (int i=0;i<newL;i++)
                   newD[i] = z.d[i];
   delete[] z.d;  z.d = newD;
   z.leng = newL;
   }

R2: \\Итерация по Ньютону
int k = 1, n = b.leng;
while (k<n) {
  overlong z_sq = z*z;
  overlong vk;
  vk.leng = 2*k+3;
  vk.d = new unsigned
                      short[vk.leng];
  if (vk.leng<=n)
   for (int i=0;i<vk.leng;i++)
      vk.d[vk.leng-i-1]=b.d[n-i-1];
else {
    for (int i=0;i<n;i++)
       vk.d[vk.leng-i-1]=b.d[n-i-1];
    for (int i=n;i<vk.leng;i++)
                vk.d[vk.leng-i-1]=0;
   }

  vk*=z_sq;
  overlong newZ;
  newZ.leng = z.leng+k;
  newZ.d = new unsigned
                     short[newZ.leng];

  for (int i=0;i<z.leng;i++)
              newZ.d[newZ.leng-i-1]=
                    z.d[z.leng-i-1];

  for (int i=z.leng;i<newZ.leng;i++)
           newZ.d[newZ.leng-i-1] = 0;

  newZ*=(short unsigned)2;
  unsigned short* vk_old_ptr = vk.d;
  int vk_old_leng = vk.leng;
  vk.d += vk_old_leng - newZ.leng;
  vk.leng = newZ.leng;
  newZ-=vk;
  vk.d = vk_old_ptr;
  vk.leng = vk_old_leng;
  z = newZ;
  k*=2;
 }

R3: \\Завершение
 overlong res;
 temp = (*this)*z;
 int resL = temp.leng-k-1-n;
 res.leng = resL;
 res.d =
        new unsigned short[res.leng];
 for (int i=0;i<res.leng;i++)
  res.d[res.leng-i-1]=
              temp.d[temp.leng-i-1];

     return res;
}
```

Рис. 1. Масштабируемый метод целочисленного деления

Оценим необходимые вычислительные ресурсы. Каждая операция алгоритма с длинными числами является либо умножением либо вычитанием ($L_u - L_v + 1$)-разрядных чисел. Как следует из табл. 1 для этого потребуется не более $2(L_u - L_v + 1)$ параллельных процессов. Каждая итерация требует одно вычитание и два умножения чисел с разрядностью не более $(L_u - L_v + 1)$.

Следовательно в соответствии с табл. 1, время выполнения одной итерации не будет превосходить $2(t_M + 3t_A + 2t_A \log_2 L)$, а время выполнения всей операции –

$$2\left\lceil \log_2\left(\frac{L_u+3}{L_v+2}\right)\right\rceil (t_M + 3t_A + 2t_A \log_2 L).$$

4. Заключение

Таким образом, для выполнения всех низкоуровневых операций длинной арифметики возможно эффективное использование многопроцессорных гетерогенных систем. Это позволит освободить центральный процессор для программирования высокоуровневых проблем, требующих доказательных вычислений.

Литература

1. Кнут, Д. Искусство программирования для ЭВМ. т.2. Получисленные алгоритмы / Д. Кнут, пер. с англ. – М:Наука. – 1985. – С. 250-268.
2. Качко Е.Г./Распараллеливание алгоритмов умножения чисел многократной точности / Е.Г. Качко // Параллельные вычислительные технологии (ПаВТ'2011): труды международной научной конференции – Челябинск: Издательский центр ЮУрГУ, 2011. – 730 с. – URL:http://omega.sp.susu.ac.ru/books/conference/PaVT2011 c.509-515.
3. Панюков А.В., Лесовой С.Ю. Применение массивно-параллельных вычислений для реализации основных операций целочисленной арифметики / А.В. Панюков, С.Ю. Лесовой // Высокопроизводительные параллельные вычисления на кластерных системах: материалы X международной конференции: Изд-во ПГТУ, 2010. – С. 77-84.
4. А.В. Панюков, М.И. Германенко, В.В. Горбик. Библиотека классов "Exact Computational" / Свидетельство о государственной регистрации программы для ЭВМ № 2009612777 от 29 мая 2009г. // Программы для ЭВМ, базы данных, топологии интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам. – № 3. – 2009. – С. 251.

Применение новых вычислительных технологий для повышения эффективности расчетов в грид-средах*

А.В. Пивушков[1], В.М. Волохов[1], Д.А. Варламов[1,2], А.В. Волохов[1], Н.Ф. Сурков[1]

Институт проблем химической физики РАН [1],
Институт экспериментальной минералогии РАН [2]

В статье сделан обзор различных технологий, позволяющих существенно повысить эффективность проведения грид-расчетов (в том числе в области вычислительной химии). Описаны особенности применения технологий виртуализации грид-ресурсов и грид-сервисов, способов работы с большими пулами независимых грид-заданий на равномерных «сетках» данных или параметров, реализация в грид-средах версий прикладных пакетов с поддержкой CUDA. Сформулированы перспективы применения описанных технологий для проведения грид-вычислений.

1. Введение

После создания в России постоянно действующих грид-полигонов и интеграции в их состав значительных вычислительных ресурсов, перед администраторами ресурсных грид-сайтов и пользователями грид-ресурсов встали задачи по повышению эффективности проведения вычислений в распределенных средах (включая снижение расходов на эксплуатацию, понижение трудоемкости администрирования), увеличение степени совместимости прикладного ПО, повышения скорости проведения расчетов, разработка способов решения «объемных» задач на больших «сетках» равномерных данных или входных параметров.

В качестве предлагаемых технологий для решения подобных задач здесь рассмотрены следующие: технологии виртуализации грид-ресурсов, грид-сервисов и прикладных грид-приложений; способы работы в грид-среде с большими пулами независимых заданий, формируемых на равномерных сетках данных или входных параметров, работа в грид-средах с версиями прикладных пакетов, ориентированных на использование расчетов с поддержкой GPU (Graphical Processor Units) устройств.

Данная статья может представлять интерес для администраторов ресурсных грид-сайтов, заинтересованных в увеличении эффективности использования своих ресурсов, системных программистов, конечных пользователей-химиков, заинтересованных в интенсификации своих расчетов. Сразу заметим, что данные технологии могут быть применены для решения широкого круга и других задач, не связанных с химией.

2. Применение технологий виртуализации ресурсов и приложений

2.1 Виртуализация грид-ресурсов и грид-сервисов на основе виртуальных машин

В 2010-2011 годах в ИПХ РАН было проведено детальное изучение различных технологий виртуализации ресурсов для повышения функциональности ресурсных центров грид-полигонов, снижения трудоемкости и затрат на их настройку и поддержку [3,5]. Часть созданных виртуальных машин переведена в режим постоянной эксплуатации в качестве поставщиков грид-сервисов как для нужд самого узла, так и вовне, на используемых грид-полигонах.

Для изучения особенностей использования виртуальных ресурсов в грид-средах были протестированы свободно распространяемые гипервизоры виртуальных машин, среди которых среды Xen (http://xen.org, разработка XenSource, Inc.), VirtualBox (http://www.virtualbox.org, разработка Innotek, затем подразделение Sun Microsystems, далее Oracle Corporation), VMware (http://www.vmware.com, раз-работка VMware Inc. – в настоящее время предоставляется бесплатно распространяемая версия VMware Server), Linux-KVM (Kernel-based Virtual Machine,

* Работа выполнена при поддержке гранта РФФИ № 11-07-00686-а

http://www.linux-kvm.org, фирма Qumranet, подразделение RedHat), oVirt (http://ovirt.org), а также эмуляторы виртуальных машин – QEMU (http://bellard.org/qemu).

Проведенное тестирование и исследование функционалов пакетов показали близость их характеристик для решения поставленных в проекте задач и возможность их применения в распределенных средах. Все указанные гипервизоры позволяют производить как аппаратную, так и паравиртуализацию для ОС семейств Linux и Windows (а так-же других типов ОС типа FreeBSD), осуществлять запуск нескольких изолированных виртуальных машин (ВМ) разного типа на одном физическом узле, адекватно воспринимать различные физические устройства host машины (сетевые карты, диски), проводить резервирование и миграцию ВМ между физическими узлами, высокую скорость запуска и т.д. Нами оценивалась прежде всего возможность их применения к работе в распределенных средах (удобство использования, простота администрирования гипервизора и ВМ, возможность работы на гетерогенных ресурсах и т.п.).

Опыт исследований и применения показал, что виртуальные машины в качестве серверных компонентов (ресурсы, сервисы) в первую очередь должны быть использованы тогда, когда на одном физическом узле желательно выполнение приложений и сервисов, не отличающихся высокой степенью загрузки физического ресурса и высоким сетевым трафиком, но требующих выполнения следующих условий: (а) наличия различных платформ (ОС) для выполнения специфического системного или прикладного ПО; (б) присутствия несовместимых между собой конфигураций одной и той же платформы; (в) изоляции сервисов друг от друга (невозможность выполнения их на одном узле – использование одних и тех же портов и сокетов, конфликт сетевых ресурсов и т.п.).

В качестве окончательного варианта используемого средства виртуализации ресурсов и сервисов был выбран гипервизор KVM (Kernel-based Virtual Machine, http://www.linux-kvm.org.). Выбор основывается на простоте администрирования, устойчивости работы под нагрузкой, независимостью (в отличие от большинства прочих) от стороннего коммерческого разработчика, наибольшей ориентированностью на Linux архитектуру (на которой основано подавляющее большинство российских грид-ресурсов), интеграцией в Linux ядром (т.е. простотой использования), устойчивостью в работе, достаточно высокой эффективностью использования машинных ресурсов (низкая степень накладных «расходов»).

Для полномасштабного тестирования данного гипервизора и создаваемых им ВМ в условиях ресурсного центра грид-полигонов на 2-х управляющих машинах ресурсного сайта ИПХФ были установлены следующие сервисные виртуальные машины:

- для среды Globus Toolkit 4 (http://www.globus.org, полигон ГридННС): ОС CentOS 5.4, сервисы MDS, GRAM, GridFTP, RFT, User Interface
- для среды Unicore 6.2 (http://www.unicore.eu, СКИФ-Полигон) – ОС Ubuntu 9.10, сервисы: шлюз (Gateway); серверный контейнер (Unicore/X), интерфейс к целевой системе (TSI), авторизационный сервис и пользовательская база данных – XUUDB$, пользовательский интерфейс (UI);
- для Computing Element среды gLite (http://glite.web.cern.ch, российский сегмент EGI-RDIG) были размещены ОС (операционная система) ScientificLinux 4.5 и соответствующие сервисы типа lcg-ce, сервисов авторизации и мониторинга.

Выбор соответствующих ОС для управляющих узлов был обусловлен либо требованиями дистрибутивов распределенного ПО (как, например, для gLite), либо рекомендациями разработчиков, либо простотой администрирования. На все управляющие машины были установлены серверные компоненты свободно распространяемого пакета управления заданиями PBS/Torque (http://www.clusterresources.com), тогда как на расчетные узлы (под управлением ОС ScientificLinux 5.4 Boron) была установлена клиентская часть данного пакета. Поскольку все распределенные middleware требуют наличия своих собственных очередей PBS (Portable Batch System), было принято решение о настройке трех одновременно работающих экземпляров pbs_mom на расчетных узлах с соответствующим набором очередей заданий. В таком варианте каждая управляющая машина связывается с расчетными узлами по уникальному порту и имеет дело только со своими заданиями.

Недостатком данного подхода является невозможность (пока) правильного учета ресурсов, используемых расчетным узлом, однако, для экспериментальных и исследовательских работ,

отладки прикладного ПО и проведения в меру ресурсоемких расчетов это вполне приемлемо. Явными преимуществами же являются:

- необходимость однократных установки и настройки ПО для решения входящих задач (особенно прикладных, поскольку часто установка прикладных пакетов оборачивается непредвиденными трудозатратами);
- простота администрирования расчетных узлов;
- экономия ресурсов (включая прежде всего электроэнергию);
- повышенный коэффициент загрузки за счет лучшей утилизации CPU.
- повышенная надежность ресурсного центра. В случае поломки физического узла копия виртуальной машины (размещенная на другом узле – файл-сервере емкостью в несколько терабайт) может быть запущена в считанные минуты (максимум в первые часы).

При наличии достаточного объема ресурсов (особенно оперативной памяти) все управляющие виртуальные машины могут быть размещены на одном физическом узле.

Отметим, что попутно на управляющих машинах можно также разместить дополнительные виртуальные машины, отвечающие за различные нересурсоемкие сервисы сети (например, web- или ftp-сервер, клиентские интерфейсы распределенных сетей, сервер баз данных и т.п.), поскольку их влияние на основные сервисы незначительно, при этом данные службы желательно изолировать от грид-узлов.

В качестве эксперимента на машине с 8 Гб оперативной памяти были размещены управляющие сервисы всех трех вышеуказанных ресурсных узла совместно с двумя машинами, обслуживающими внутрисетевые сервисы ИПХФ, что нисколько заметно не сказалась на качестве выполнения и доступности грид-сервисов, часть сервисов (для сред Globus Toolkit и Unicore) на базе ВМ продолжают эксплуатироваться в составе ресурсного центра ИПХФ.

Таким образом, в результате работ продемонстрирована возможность установки, запуска и работы виртуальных машин с поддержкой различных грид-сервисов на существующих физических узлах без вмешательства в рабочее пространство распределенных и/или параллельных сред ресурсных узлов. Это дает возможность разворачивать распределенные вычислительные полигоны на уже существующих вычислительных кластерах без кардинальной перенастройки их аппаратно-программной конфигурации, что особенно важно для постоянно работающих центров класса «production farm».

Наиболее важной проблемой использования виртуальных машин в качестве ресурсных становится правильный учет доступных для каждой грид среды ресурсов физического узла и мониторинг выполняемых ресурсным узлом задач (на всех входящих в него ВМ плюс хост-система). Учет ресурсов, востребованных ВМ, пока проводится на уровне гипервизора (т.е. только уровень загрузки ЦП) и не оценивается адекватно внешним мониторингом распределенной среды со стороны агентов мониторинга (собственных и внешних), что может вести к недоучету ресурсов и завышенным ожиданиям со стороны входящих задач. В планах развития большинства гипервизоров показана возможность решения этих проблем в дальнейшем, направленная на интеграцию аккаунтинга виртуальных машин для физического узла.

Таким образом, выбран и рекомендован к использованию свободно распространяемый гипервизор виртуальных машин – KVM, разработана методика одновременного размещения нескольких ресурсных сайтов разных распределенных полигонов (среды Globus Toolkit, gLite, Unicore) на едином физическом пространстве одного кластера. Подобная методика позволяет проводить обширные вычислительные эксперименты и разрабатывать сложные грид приложения для различных грид полигонов без вовлечения основных вычислительных ресурсов организации. Введение подобных вариантов использования кластеров с размещенными на нем виртуальными машинами позволяет участвовать в работе нескольких грид-полигонов, значительно повышает надежность управляющих узлов ресурсных сайтов, снижает трудоемкость администрирования сайтов, понижает расходы на эксплуатацию и поддержку грид-инфраструктуры.

2.2 Виртуализация грид-приложений

Другой стороной применения технологий виртуализации стала виртуализация исполняемых грид-заданий на основе прикладных программных пакетов, т.е. создание динамически формируемых образов исполняемых сред, или виртуальных «контейнеров», позволяющих про-

изводить запуски прикладных пакетов ПО вычислительной химии без предустановки и настройки на удаленных узлах распределенных сетей.

Данная технология была ранее успешно реализована авторами в грид-средах gLite и Unicore для бинарных приложений и квантово-химического прикладного программного пакета (ППП) GAMESS и детально описана в ряде статей [1,4,5], поэтому здесь приводится лишь краткое описание.

В 2009-2010 годах авторами был разработан метод создания виртуальных перемещаемых «контейнеров» (или, другими словами, – динамически формируемых исполняемых сред), для которых постулировано следующее содержание: «персональные» копии необходимых системных файлов и библиотек (в том числе, например, математических), скрипты по настройке операционной системы, необходимые файловые «деревья», собственно приложение, файлы данных и т.п.. В результате разработанной авторами оригинальной методики вычислений конечный грид-пользователь получил возможность формировать «виртуальное» (т.е. временное, создаваемое на время выполнения) приложение, которое в виде «виртуального контейнера» доставляется на ресурсный узел вместе со всеми конфигурационными настройками, относящимися к операционной системе, и поддержкой необходимых параллельных протоколов, не требуя процедуры предварительной установки и настройки. Далее «виртуальный контейнер» самостоятельно разворачивается на всех выделенных узлах грид-ресурса, подготавливая среду для исполнения параллельного приложения с последующим его запуском. При этом отсутствуют конфликты приложения с другими, уже установленными на узле программами и даже с другими экземплярами этого же приложения. Суть виртуализации приложения заключается в создании персональной копии необходимой части системных файлов и настроек операционной системы и доставке приложения совместно с этой информацией с последующим запуском в изолированном «контейнере». Проведенные авторами эксперименты в этой области показали, что так могут быть решены проблемы установки, настройки, несовместимости с операционной системой и другими программами, разрешаются конфликты одинаковых приложений.

В настоящее время этот метод применим для исполнения на узлах, поддерживающих ОС системы Linux, т.е. типичных кластерах, интегрированных в грид-среды.

В 2010 году программный пакет для работы с «виртуальными» контейнерами была адаптирован для работы с 64-битными вычислительными архитектурами, а также 64-битными версиями прикладных пакетов (для части которых разработчиками существенно была изменена схема параллелизации расчетов). В 2011 году было проведено успешное тестирование описанного способа виртуализации приложений в рамках среды Globus Toolkit на узлах полигона ГридННС (и ряда аналогичных по функциональности) с использованием «контейнера» для прикладного пакета GAMESS-US, что показало эффективность данного метода.

Серия первичных запусков (с использованием внутренних тестовых примеров собственно пакета GAMESS) вплоть до получения положительного результата тестов (равнозначность поведения сокетных и MPI вариантов) была проведена на ресурсных сайтах ИПХФ РАН для среды Globus Toolkit 4 (сайты использовались как удаленные, т.е. запуск задач шел через соответствующие инфраструктуры грид-полигона). Дальнейшее успешное тестирование было проведено на удаленных ресурсных узлах ГридННС в рамках ВО «NanoChem» (использовались узлы НИИЯФ МГУ и Курчатовского РНЦ). Были проведены успешные запуски пакета GAMESS-US с применением данной технологии. В качестве прикладной задачи с применением данной технологии были (по аналогии с предыдущими этапами) рассчитаны тестовые примеры молекулярных структур из дистрибутива GAMESS, например, серия ab initio расчетов по оптимизации геометрии в 15-атомной системе ($P_3O_9H_3$) на уровне HF/6-31G), подтвердившие полную работоспособность разработанной технологии.

В настоящее время технология формирования «виртуальных контейнеров» для GAMESS-US интегрирована в GRID портал ИПХФ в высокоуровневый web-интерфейс по работе с GAMESS в распределенных средах Globus Tools, gLite, Unicore на пространстве всех доступных вычислительных полигонов, проведен успешное тестирование запусков «контейнеров» через web-портал на узлы ГридННС и аналогичных по функционалу грид-полигонах.

3. Технология работы с «пулами» (pools) независимых грид-заданий на равномерных «сетках» данных или параметров

В естественных науках (в том числе химии) существует огромное количество задач, которые представляют собой на самом деле совокупность независимых расчетных заданий, решение которых основано на переборе исходных данных или входных параметрах, а число зависит от количества параметров задачи или от «сетки» разбиения искомой области данных. Таковыми являются многопараметрические задачи вычислительной химии и химической физики, требующие последовательного перебора большого количества входных параметров. При этом полная задача разбивается на огромное количество независимых подзадач (каждая определяется группой значений совокупности параметров). Как вариант – решение задач на больших областях исходных данных, когда результат в каждой решаемой точке не зависит от «соседей». Задача автоматизации процесса разбиения полной задачи на фрагменты («нарезка») во многом аналогична таковой для многих кластерных задач и определяет детальность и полноту получаемого решения и удобство пользования системой. Типичный пример – туннельные реакции под воздействием электромагнитного излучения, где параметрами являются разные характеристики излучения. Задача имеет высокую вычислительную сложность, однако вычисления в каждой точке сетки в ней происходят независимо друг от друга, поэтому возможно разбить область вычислений на множество непересекающихся подобластей и каждую из них рассчитывать на разных узлах. Схожим примером могут служить траекторные расчеты химических реакций, например, реакция $H_2 + O_2$. Расчет ее представляет собой компьютерное моделирование с использованием классических траекторий элементарного акта столкновения. Для расчета полного сечения реакции для набора необходимой статистики следует рассчитать до десятков миллионов траекторий с учетом углов взаимной ориентации молекул, начальных колебательных и вращательных квантовых чисел, параметров столкновения и т.п. Как правило, расчет независимого задания в одном из узлов расчетной многомерной «сетки» занимает от нескольких минут до первых часов, однако, общее время расчета становится нереальным для одиночной вычислительной системы даже при высокой степени параллелизации выполнения задачи.

В последнее время помимо весьма простых заданий как вышеописанные, стало актуальным использование «тяжелых» прикладных пакетов типа GAMESS для расчета больших массивов точек (например, многомерных энергетических поверхностей), для которых нужно производить вычисления сотен и тысяч «точек» по «сетке» с закономерно изменяющимися параметрами.

Для решения подобных задач авторами ранее [2] была создана технология запуска «пучков» (или пулов) независимых заданий для использования всех доступных ресурсов распределенной среды с использованием грид технологий.

Технология предназначена для работы с большими заданиями на базе ППП или многопараметрических задач, которые могут быть представлены в виде объединения большого количества параллельно выполняемых независимых друг от друга задач. Созданный авторами программный комплекс обеспечивает разбиение первичной задачи по равномерным «сеткам» областей данных или параметров, формирование пула готовых к исполнению грид-заданий, их параллельного запуска в грид-среду, мониторинг и контроль выполнения участников пула, сборка результатов в конечный обобщающий результат.

Основные функции, выполняемые комплексом:

- разбиение первичной задачи на равномерных областях данных или входных параметров;
- автоматическое формирование пула независимых друг от друга грид-заданий вкупе с автоматически созданными файлами;
- параллельный запуск независимых заданий из пула в грид-среду;
- мониторинг и контроль выполнения заданий в рамках пула, включая останов и перезапуск по тайм-аутам;
- сборка результатов расчетов заданий с грид-ресурсов в конечный обобщающий результат.

Комплекс предоставляет средства одновременного запуска большого числа грид-заданий, в том числе с использованием адаптированных ППП (на примере пакета GAMESS) и широкого класса многопараметрических задач для проведения вычислений с использованием равномерных «сеток» данных или входных параметров ППП. Выполнение всех работ с пулами задания

должны происходить с использованием внутренней базы данных web-портала (MySQL или PostgreSQL) и внешних данных аккаунтинга и мониторинга, предоставляемых централизованными средствами грид-полигона.

Для разных распределенных сред разработана методика запуска подмножества независимых заданий (составляющих в совокупности вычислительную задачу) и получения результатов с удаленных ресурсных узлов. Первоначально метод был реализован в локальной гетерогенной вычислительной среде с использованием middleware Condor, что показало высокую эффективность метода. Затем метод был испытан на ряде задач в средах Nimrod и X-Com на географически распределенных вычислительных ресурсах. Далее, на языке Perl был написан комплекс скриптов для формирования «пулов» заданий, их запуска и получения результатов счета с использованием пользовательских интерфейсов (UI) сред gLite и Unicore. Для решения многопараметрических задач квантовой химии были разработаны методы формирования «пулов» независимых заданий с варьирующими параметрами – до 10^4, в перспективе до 10^7 «атомарных» заданий на задачу. Для выбранных областей данных авторскими скриптами производится «нарезка» областей данных или входных параметров, формирование пулов независимых заданий, создание очередей запуска и отправки заданий на брокер ресурсов. После запуска периодически запускаемые средствами ОС (например, по cron) скрипты UI ведут мониторинг выполнения заданий (опрашивая брокер ресурсов или непосредственно удаленные ресурсные узлы), контроль тайм-аутов, перезапуск неудачных заданий и сбор результатов (с использованием базы данных и таблиц в ней, контролирующих состояние заданий – «ожидание», «запуск», «выполнение» и т.д.). По окончании расчетов проводится сборка «атомарных» результатов в единый выходной файл. Для части задач (требующих значительного числа параллельных независимых расчетов) в настоящее время создаются механизмы по разбиению областей данных (или расчетов) на большие независимые «подсетки» в форме независимых заданий, передачи всех их интерфейсам распределенных сред с последующим запуском на параллельных узлах и «сборки» финальных результатов из множества полученных независимых. Это позволяет достигнуть разумного баланса между собственно временем счета заданий и накладными расходами по передаче заданий по распределенной среде. Метод был протестирован на распределенных ресурсах ВО RGSTEST, СКИФ-полигона, ВО «Nanochem» полигона ГридННС в среде Globus.

Функционал комплекса работы с «пулами» заданий интегрирован также в Грид-портал ИПХФ РАН (http://grid.icp.ac.ru), пока только для стандартных многопараметрических задач.

4. Реализация вычислительных грид-сервисов на основе прикладных пакетов с поддержкой GPU (Graphical Processor Units) устройств

В последние годы произошел взрывной скачок интереса к параллельным расчетам с использованием высокоскоростных GPU (Graphical Processor Units) устройств, в том числе для прикладных пакетов в области квантовой химии и молекулярной динамики. Применение GPU вкупе со сменой парадигмы программирования прикладных пакетов во многих случаях ведет к убыстрению расчетов от десятков процентов до первых порядков по времени. В настоящее время резко интенсифицировался перевод программного кода прикладных пакетов (в том числе – квантово-химических) на параллельные технологии с использованием технологий программирования CUDA^{tm} и аналогичных. Это позволяет проводить высокоинтенсивные вычисления с применением встраиваемых в расчетные узлы кластеров высокопроизводительных графических процессоров (Nvidia, Tesla, AMD-ATI), что резко повышает эффективность использования параллельных методов расчетов и снижает как требования к расчетным центрам, так и операционную стоимость расчетов, особенно при использовании большого числа расчетных узлов. Во вновь создаваемых кластерах (в том числе ориентированных на использовании в качестве грид-ресурсов) большинство расчетных узлов оснащается графическими адаптерами, что ставит задачу по использованию CUDA-ориентированных прикладных пакетов на грид-ресурсах весьма актуальной.

Использование массивно параллельных архитектур NVIDIA и Radeon GPU позволяет получать превосходные результаты при работе с приложениями в сфере квантовой химии и молекулярной динамики. По различным оценкам (приведена оценка Nvidia Group), выигрыш в во

времени расчетов для единичного узла составляет (указаны ППП, уже реализованные в ИПХФ в качестве грид-сервисов): GAMESS (метод самосогласованного поля) – до 50-60 раз, Gaussian (расчет потенциала Кулона) – 12-15 раз, VASP – 3-6 раз, NWChem – 3-8 раз, GROMACS (метод стице-ячеечной суммы Эвальда) – от 2 до 5 раз, LAMMPS (потенциалы Леннард-Джонса, Гей-Берне) – в 6 раз, NAMD (расчет невалентных взаимодействий) – от 2 до 7 раз. Для прикладных пакетов же, которые создавались именно для CUDA вычислений (например, TeraChem и PetaChem - http://www.petachem.com или 'Schrodinger Core Hopping') выигрыш по времени может составлять 50-5000 раз.

В ИПХФ РАН начаты методические исследования возможности использования новых GPU-оптимизированных программных вариантов различных прикладных квантово-химических пакетов как на локальных кластерах ИПХФ, так и в рамках грид-сайтов различных грид-сред. Проводится тестирование и оптимизация имеющихся вариантов ППП с поддержкой CUDA вычислений с последующей адаптацией их к проведению расчетов в грид-средах. Проводится оценка повышения эффективности расчетов в условиях грид-ресурсов. Начата разработка методики запуска грид-заданий, ориентированных на поиск и использование ресурсов, поддерживающих использование CUDA технологий.

5. Заключение

Применение новых технологий применительно к грид-вычислениям с использованием прикладных программных пакетов в области квантовой химии и молекулярной динамики позволит существенно повысить эффективность использования грид-ресурсов (включая экономию затрат на эксплуатацию и трудоемкость обслуживания), повысить совместимость прикладного ПО с гетерогенными ресурсами грид-полигонов, заметно ускорить решение многих задач, вплоть до того, что возникает возможность ставить и решать вычислительные задачи фундаментального и прикладного характера в области химических наук, ранее не доступные из-за ограниченности возможностей вычислительных ресурсов. Основные научные области применения – химическая физика, квантовая химия, исследование наноструктур, молекулярная динамика, биохимия, фармацевтика, разработка топливных элементов и близкие отрасли наук.

Литература

1. В.М. Волохов, Д.А. Варламов, А.В. Пивушков, Н.Ф. Сурков, А.В. Волохов Динамически формируемые параллельные среды в условиях грид-полигонов, проблемы и решения // «Вычислительные методы и программирование: Новые вычислительные технологии», М.: МГУ, 2011, т.12, № 1, с.39-45
2. Волохов В.М., Варламов Д.А., Пивушков А.В., Волохов А.В. Способы решения прикладных многопараметрических задач вычислительной химии на пета- и экзафлопных системах // Международная суперкомпьютерная конференция «Научный сервис в сети Интернет: экзафлопсное будущее», 19-24 сентября 2011,Новороссийск – М.: Изд-во МГУ,2011 с.382-384
3. Волохов В.М., Пивушков А.В., Волохов А.В., Варламов Д.А. Реализация нескольких независимых ресурсных грид-сайтов на едином физическом пространстве кластера // X международная конференция «Высокопроизводительные параллельные вычисления на кластерных системах» HPC - 2010, Пермь, ноябрь 2010; изд-во ПГТУ, том 1, стр. 119-124
4. Волохов В.М., Варламов Д.А., Сурков Н.Ф., Пивушков А.В. Виртуальные вычислительные среды: использование на GRID полигонах // Вестн. ЮУрГУ, серия «Математическое моделирование и программирование», 2009, № 17 (150), вып. 3, с.24-35
5. Варламов Д.А., Сурков Н.Ф., Волохов В.М., Пивушков А.В. Виртуализация вычислительной среды в ГРИД // «Параллельные вычислительные технологии 2010» ПаВТ-2010, (Уфа, март 2010), Челябинск, изд-во ЮУрГУ, с.63-70

Исследование влияния параметров функций параллельного ввода-вывода MPI на скорость файловых обменов в суперкомпьютерах*

А.В. Путилин, А.Н. Сальников

факультет Вычислительной Математики и Кибернетики Московского Государственного Университета

Многие научные приложения широко используют параллельный ввод-вывод, например через использование библиотеки Parallel-NetCDF. Часто для научных приложений (например сборка генома с использованием, mpiBLAST) ввод-вывод становится узким местом, существенно влияющим на время исполнения программы. В настоящее время практически все суперкомпьютеры используют специально выделенную подсистему для хранения данных. В данной работе авторы ищут тонкости, связанные с использованием файлового хранилища через комунникационную среду суперкомпьютера. Исследование ведется путем создания специальной системы тестов на основе MPI-IO. С помощью разработанной системы тестов был исследован вычислительный кластер СКИФ-МГУ «Чебышев».

1. Введение

Для любой параллельной программы важны следующие три параметра: скорость выполнения последовательного кода, объем коммуникаций и объем ввода-вывода. Многие научные библиотеки, например Parallel-NetCDF [1] широко используют параллельный ввод-вывод, для некоторых научных приложений (например, mpiblast) ввод-вывод становится узким местом [2].

При наличии данных о производительности отдельных узлов можно предсказать время работы линейных участков кода последовательной программы. Имея данные о скорости и латентности сетевого обмена между процессами можно оценить время коммуникаций. Но если программа каким-то образом использует ввод-вывод, то для точной оценки времени выполнения необходимо иметь данные о производительности дисковых операций.

В настоящее время существует множество многопроцессорных вычислительных систем, таких как СКИФ МГУ «Чебышев» (НИВЦ МГУ), IBM Blue Gene/P (ВМК МГУ), суперкомпьютер «Ломоносов» (НИВЦ МГУ). Эти вычислительные системы используют специально выделенную программно/аппаратную подсистему для хранения данных различного характера. Это могут быть результаты экспериментов и вычислений, резервные копии и так далее. В тексте эту подсистему мы будем называть «файловым хранилищем». В случае, когда вычислительная система имеет кластерную архитектуру, на самих вычислительных узлах аппаратура для хранения может отсутствовать вовсе, или ее объем может быть сильно ограничен. Как правило доступ к общему для различных вычислительных узлов файловому хранилищу осуществляется по сети. Сами файловые хранилища могут иметь кластерную архитектуру и на них могут использоваться специальные распределенные файловые системы для достижения высокой скорости чтения и записи.

Одной из наиболее популярных технологий для разработки параллельных программ на данный момент является MPI — Message Passing Interface (интерфейс передачи сообщений). MPI предоставляет программисту набор функций MPI-IO — эффективный и переносимый интерфейс для осуществления параллельного ввода-вывода. Однако, несмотря на внешнюю простоту этого интерфейса, он скрывает под собой много тонких мест в плане своей реали-

*данная работа частично поддержана грантами РФФИ 11-07-00756-а, 11-07-00614-а и госконтрактами ФЦП «Научные и научно-педагогические кадры инновационной России» П1317, П1367.

зации — это приводит к тому, что в общем случае очень сложно делать априорные предсказания производительности параллельной программы с интенсивным вводом-выводом. В частности, для эффективного ввода-вывода важно понимать, как влияют на время работы такие параметры как: используемая функция ввода-вывода в MPI-IO, размер файла, размер блока записи, представление файла на диске. Определение степени влияния указаных выше параметров позволит выдавать рекомендации по использованию MPI-IO пользователям и программистам современных суперкомпьютеров.

2. Краткое описание MPI-IO

Поскольку работа существенно использует возможности MPI-IO авторы считают необходимым провести краткое описание архитектуры библиотеки MPI-IO. Это важно также в связи с тем, что MPI-IO является одной из наиболее популярных библиотек для организации параллельного ввода-вывода в параллельной программе.

Типичный сеанс работы с файлом в MPI включает в себя следующие пункты.

1. Вызов MPI_File_open для получения внутреннего файлового дескриптора MPI с помощью MPI_File_open.

2. Вызов MPI_File_set_view для установки «представления» (view в терминологии MPI) файла — описание способа отображения Участков файла в логическое представление файла в MPI. Данные и в оперативной памяти, и в файле могут быть представлны с пропусками – «дырками».

3. Непосредственно осуществление ввода/вывода. MPI для действий подобного рода предоставляет богатое множество функций.

4. Вызов MPI_File_close для закрытия файла и освобождения связанных с ним ресурсов.

Наибольший интерес с точки зрения времени исполнения программы представляют пункты 2 и 3: установка представления и функции непосредственного чтения-записи.

Все функции, осуществляющие ввод-вывод, имеют похожий набор параметров, в который входят:

- Файловый дескриптор MPI
- Адрес блока для чтения или записи
- Размер блока

Но между ними есть и существенные отличия. Во-первых, их можно разделить на три группы по типу позиционирования в файле:

1. Функции, работающие с точными смещениями (explicit offset), которые необходимо передавать явно в качестве параметра. Примером такой функции является MPI_File_write_at.

2. Функции, работающие с индивидуальными файловыми указателями. Для таких функций MPI самостоятельно, без вмешательства программиста поддерживает текущее смещение, которое увеличивается на размер блока после чтения/записи. Все процессы имеют независимые смещения. Пример: MPI_File_write.

3. Функции, работающие с общим файловым указателем. MPI поддерживает общее для всех процессов смещение, которое меняется после операции ввода-вывода, происходящей в любом процессе. Пример: MPI_File_write_shared.

Также функции можно подразделить на индивидуальные и коллективные. Разница между ними состоит в том, что коллективные операции должны быть вызваны всеми процессами, котрые вызывали MPI_File_open — это позволяет MPI выполнять разного рода оптимизации доступа.

Наконец, все функции можно разделить на блокирующие и неблокирующие. Блокирующие функции завершаются только после окончания соответствующей операции ввода-вывода, а неблокирующие функции позволяют программисту выполнять вычисления во время выполнения ввода-вывода, и только затем проверить успешность операции.

Функция MPI_File_set_view интересна тем, что она позволяет программисту установить представление файла — то, как он будет выглядеть для процесса. Эта функция добавляет гибкости, позволяя устанавливать начало представления, тип элементов, при этом этот тип может содержать «дыры», что бывает полезно, например, при записи и чтении матриц. Общий принцип проиллюстрирован на изображении 1.

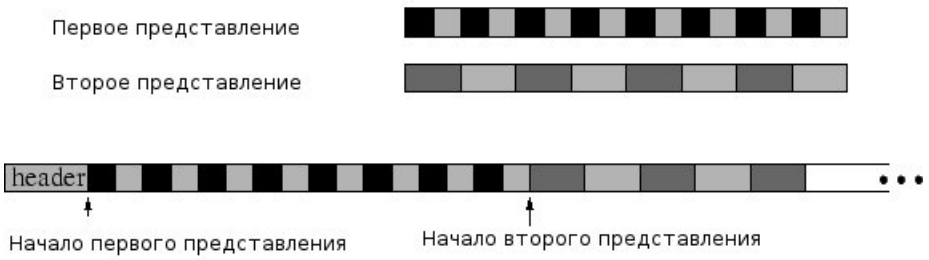


Рис. 1. Задание отображения в MPI-IO

За дополнительной и более подробной информацией можно обратиться к стандарту MPI [3].

3. Некоторые системы тестирования параллельного ввода-вывода

В этом разделе будут рассмотрены существующие системы тестирования ввода-вывода, а также обсуждены их достоинства и недостатки с точки зрения предсказания времени выполнения параллельной прогаммы.

3.1. MADBench2

Существует приложение для решения прикладной задачи космологии MADspec. На основе этого приложения был построен тест ввода-выода MADBench2 [5]. MadBench2 сохраняет всю вычислительную сложность оригинального приложения, но при этом использует автоматически сгенерированные данные. Таким образом, MADBench2 настроен на тестирование общей производительности коммуникационной, вычислительной и дисковой подсистем в контексте данной задачи. То есть MADBench2 не тестирует производительность ввода-вывода отдельно от коммуникаций и вычислений, что с одной стороны является плюсом, поскольку этот тест позволяет оценить общую производительность суперкомпьютера, а с другой – минусом, поскольку невозможно получить оценку производительности только для операций ввода-вывода.

3.2. b_eff_io

b_eff_io [4] преследует две цели: получить некоторую общую среднюю оценку производительности дисковой подсистемы суперкомпьютера, и получить более детализированную оценку производительности при разных образцах поведения программы при доступе к файлам, таким, как первая запись, перезапись, чтение, запись больших областей. Ввод-вывод

здесь тестируется при разных размерах блока, который может равняться 1 КБ, 32 КБ, 1 МБ, максимому из 2 МБ и memory_available_to_one_node/128. Тестирование также производится на размерах блока, не кратных 2. Эти размеры получаются путем добавления 8 байт к предыдущим рассмотреным размерам файла.

Результатом работы теста является число – общая оценка производительности дисковой подсистемы, а также файл с более детальными результатами для каждого образца поведения при доступе к файлу.

3.3. noncontig

Основной задачей noncontig [6] является получение данных о производительности дисковых операций, в четырех случаях: когда данные не расположены непрерывно ни в памяти, ни в файле; когда данные расположены непрерывно в памяти, но «разрывно» в файле; когда данные расположены «разрывно» в памяти, но непрерывно в файле; когда данные расположены непрерывно, как в файле, так и в памяти.

Для каждого из этих случаев печатаются средние данные о производительности.

3.4. mpi-tile-io

mpi-tile-io [7] — это еще один тест от авторов noncontig. В этом тесте файл с данными представляет собой матрицу, над которой выполняются операции чтения и записи. Тест позволяет менять размеры матрицы по вертикали и горизонтали, размер ее элементов, позволяет указать что лучше использовать на данном действии коллективные операции, или индивидуальные. После завершения теста выводятся данные о лучшем, худшем и среднем временах доступа. Также выводится некоторая общая оценка производительности, основанная на худшем времени доступа.

3.5. LANL MPI-IO Test [8]

Тест написан исследователями из Лос-Аламосской национальной лаборатории(Loss-Alamos National Laboratory, LANL), тест измеряет скорость ввода-вывода при разных патернах доступа, таких как:

- N процессов пишут в N файлов
- N процессов пишут в один файл
- N процессов отсылают данные M процессам, которые пишут в M файлов
- N процессов отсылают данные M процессам, которые пишут в один файл

При этом тест позволяет указать, являются ли данные «перемешанными» между собой. На рисунке 2 приведена иллюстрация этого принципа (серым цилиндром изображен диск, на нем разными цветами отмечены области файла, а стрелками показано соответствие между представлением файла в каждом из процессов и представлением файла на диске).

Все перечисленные системы обладают следующими недостатками: отсутствует возможность проводить серию экспериментов с разными параметрами, отсутствует возможность тестировать производительность конкретных функций MPI-IO. Для некоторых из описанных систем тестов написание новых тестов затруднительно всвязи со сложившейся в них организацией кода. С целью минимизации этих недостатков было принято решение разработать собственную систему тестов.

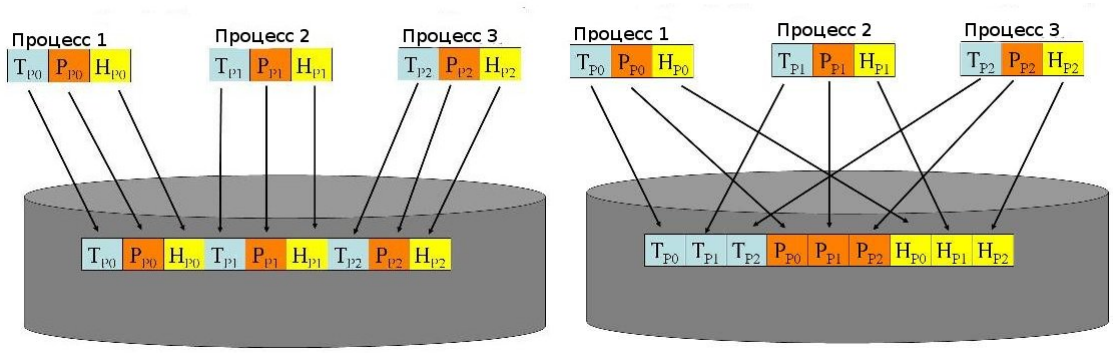


Рис. 2. Установка представления

4. Описание разрабатываемой системы тестов

Было решено разработать систему тестов для MPI-IO, позволяющую определять и оценивать производительность дисковой подсистемы вычислительных кластеров в зависимости от функции, применяемой для ввода-вывода, а также параметров этих функций: размер блока для единовременного чтения/записи, размера файла, а также представления. Запустить полученную систему тестов на суперкомпьютере СКИФ-МГУ «Чебышев» и попытаться определить характеристики, влияющие на эффективностьопераций ввода-вывода.

Разрабатываемая система тестов нацелена на исследование эффективности ввода-вывода при использовании различных функций MPI-IO, а также при различных параметрах, таких как размер файла, и размера блока данных, атомарно передаваемого в функцию MPI. Процесс тестирования производительности отдельно взятой функции ввода-вывода разбит на следующии стадии:

1. Сбрасывание буферов файловой системы. Во многих случаях после чтения файла, этот файл оказывается сохранен в свободной оперативной памяти вычислительного узла, поэтому при повторном чтении этого файла обращение к диску выполняться не будет. Такой эффект полезен в реальных приложениях, но нежелателен при тестировании производительности файлового хранилища. Сбрасывание выполняется следующим образом: файл полностью переписывается другими данными.

2. Открытие файла в режиме для чтения и записи (MPI_MODE_RDWR).

3. Установка метода отображения файла из процесса, что требуется для корректной работы MPI-IO. Для тестирования были предложены три режима. В первом, каждому процессу достается отдельный непрерывный участок файла. Во втором, каждому процессу достается файл целиком (данный режим подходит только для тестов чтения). В третьем режиме участки файлов оказываются перемежены: первому процессу достается 1-й, (NP+1)-й, (2*NP+1)-й и т.д. байты файла, второму 2-й, (NP+2)-й, (2*NP + 2)-й байты файла и так далее до NP-го процесса, которому достается (NP-1)-й (2*NP-1)-й ... (NP – обозначает количество процессов, занимающихся вводом-выводом). Для обеспечения последнего способа отображения создается новый тип данных MPI, содержащий «дыры» (своего рода маска).

4. Непосредственное осуществление ввода-вывода. На данный момент тестируются все функции из MPI-IO. Для этого в каждом процессе вычисляется количество байт, которое необходимо записать или считать, а затем исследуемая функция последовательно вызывается в каждом процессе до тех пор, пока требуемое число байт не будет прочитано или записано. Замеры эффективности выполняются с использованием функции MPI_Wtime.

5. Закрытие файла с использованием MPI_File_close.

Поддерживается проведение серии экспериментов, когда тест каждой функции выполняется с разными размером файла и размером блока для ввода или вывода. Для каждой пары параметров выдается средняя, максимальная и минимальная достигнутая скорость выполнения (в секундах, а также в КБ/с). Во время тестирования для каждого набора параметров выводятся следующие результаты тестирования (средняя, максимальная и минимальная скорости записи). После завершения выводятся матрицы результатов (матрицы собирают скорость ввода-вывода и общее время выполнения ввода-вывода). Этот режим упрощает выяснение зависимости эффективности ввода-вывода от различных параметров.

5. Исследование производительности на кластере СКИФ-МГУ «Чебышев»

Разработанная система тестов запускалась на кластере СКИФ-МГУ «Чебышев». На этом кластере используется параллельная файловая система Panasas, а объем дисковой подсистемы составляет 60 ТБ. Тестирование проводилось со следующими параметрами: размер файла составлял 100 ГБ, количество процессов – 128, размер блока, передаваемого в функции ввода-вывода менялся от 1 МБ до 99 МБ с шагом в 2 МБ.

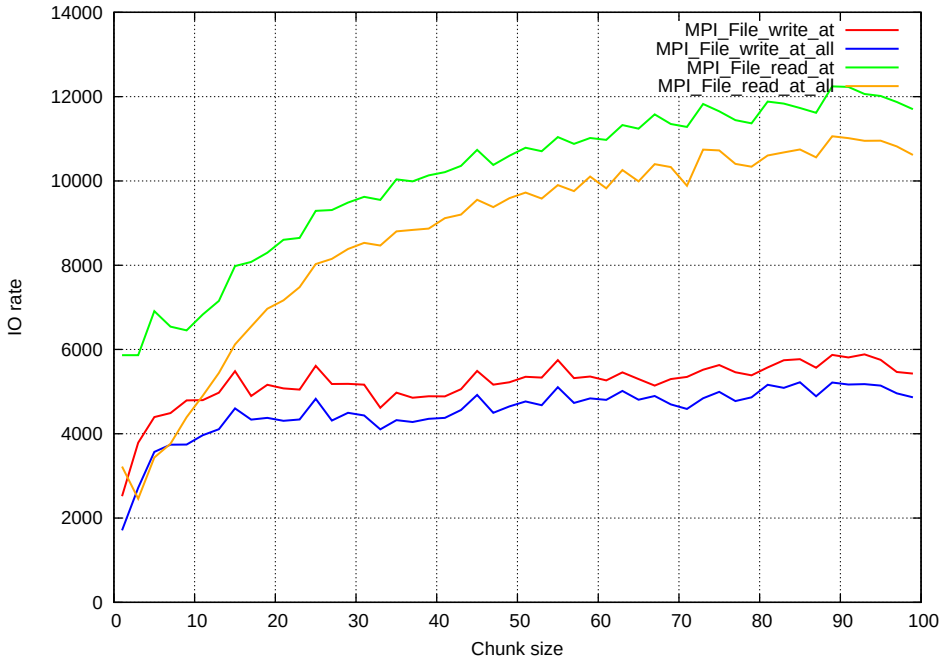


Рис. 3. Результат тестирования. Ось X – размер блока(МБ). Ось Y – эффективность функции(МБ/с)

На графике 3 показаны результаты тестирования. На этом графике видны следующие особенности:

1. Скорость записи при размере блока, ниже 5 МБ оказалась существенно ниже, чем скорость записи при размере блока $\geq$ 5 МБ, что связано с меньшим количеством накладных расходов при меньшем количестве вызовов функций.

2. Аналогичный вывод можно сделать и для чтения, только здесь хороший размер блока составляет около 25 МБ.

3. Коллективные операции проигрывают индивидуальным, из-за того, что для индивидуальных операций не требуется дополнительная синхронизация.

4. На графиках MPI_File_write_at, MPI_File_write_at_all прослеживаются как пики, так и падения производительности. Например это заметно при размере блока около 25 МБ, 45 МБ, 55 МБ, 97 МБ, а также видно несколько более мелких пиков. Аналогичное явление можно заметить и для графиков MPI_File_read_at, MPI_File_read_at_all. Причины такой регулярности пиков пока неясны, и требуют дополнительного исследованияю

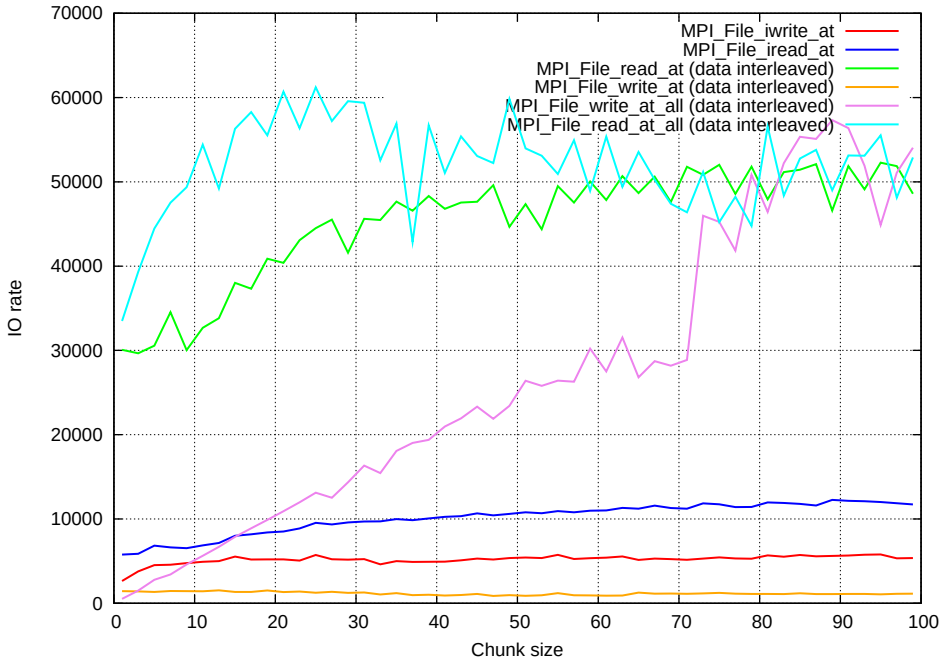


Рис. 4. Результат тестирования. Ось X – размер блока(МБ). Ось Y – эффективность функции(МБ/с)

На графике 4 приведены данные для тех же функций при третьем расположении данных (в легенде дополнительно добавлено «data interleaved», а также для наглядности приведены графики с диаграммы 3). Здесь можно выделить следующие тенденции:

1. Скорость ввода-вывода существенно выше при коллективных операциях, поскольку MPI удается оптимизировать все вызовы функций в один запрос к жесткому диску. Особенно хорошо это видно на графиках записи(оранжевый и фиолетовый). Это менее заметно на графиках чтения, поскольку при чтении из файлового хранилища происходит буферизация.

2. Скорость ввода-вывода при «перемешанном» расположении данных и достаточном размере блока значительно выше скорости ввода-вывода в случае, когда данные расположены последовательно.

3. При размере блока порядка 75 МБ происходит насыщение сети, поэтому скорость записи становится равна скорости чтения, то есть сеть, по которой данные доставляются к файловому хранилищу становится узким местом.

На графике 5 приведены результаты экспериментов, по которым была построена поверхность в трехмерном пространстве, изображающая зависимость скорости записи

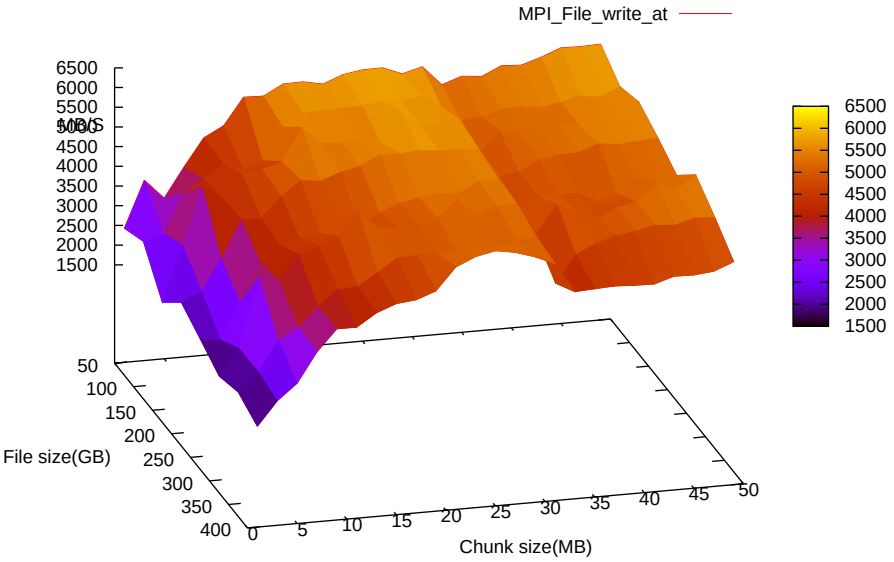


Рис. 5. Зависимость эффективности ввода-вывода от размера файла и размера блока.

(MPI_File_write_at) от размера файла и размера блока. Как видно из этого графика зависимость скорости записи от размера файла в целом несущественна. То есть характер зависимости скорости ввода-вывода и от размера блока схож при всех протестированных размерах файла. Однако, из этого графика видно, что пики в производительности могут все же иметь некоторую зависимость от размера файла.

Что касается функций, работающих с разделяемым указателем (MPI_File_read_shared, MPI_File_write_shared их их аналоги), то они показали себя плохо. Однако, это может быть связано с проблемами в настройке mvapich на кластере, поскольку во время запуска тестов наблюдалась ошибка «**io File or directory doesn't exist». Это сообщение не связано с недостатками или ошибками в разрабатываемой системе тестов, потому что аналогичное сообщение наблюдалось и при запуске систем тестов, реализованных другими разработчиками.

Разница в эффективности между неблокирующими и блокирующими вариантами функций не была обнаружена, равно как и разница между функциями, работающими с точными смещениями(explicit offsets) и их аналогами, но работающими с индивидуальными файловыми указателями (individual file pointers), такими как MPI_File_write и MPI_File_write_at, MPI_File_write_all и MPI_File_write_at_all, MPI_File_read и MPI_File_read_at и т.д. Данный факт связан с тем, что в реализации MPI-IO от mvapich происходит обращение к одним и тем же функциям более низкого уровня.

В заключение хотелось бы сказать несколько слов о масштабируемости ввода-вывода. Система тестов была запущена с аналогичными параметрами на 16 процессах, и на графике 6 приведены его результаты, из которых видно, что ввод-вывод хорошо масштабируется с 16 на 128 процессов.

6. Выводы

В ходе выполнения исследования производительности кластера СКИФ-МГУ «Чебышев» были выяснены следующие вещи:

- Эффективность ввода-вывода серьезно зависит от размера блока, записываемого или

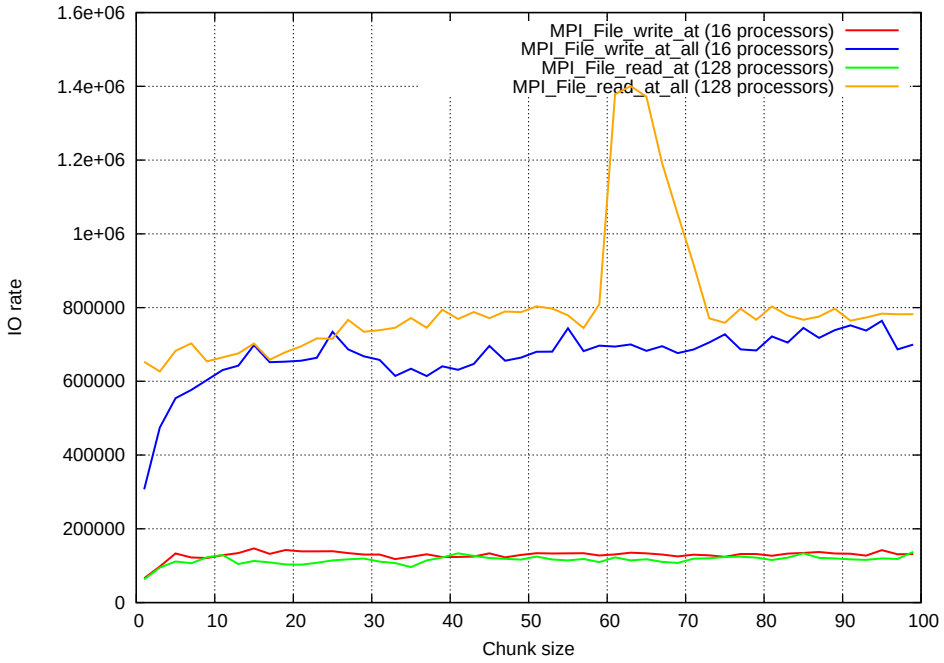


Рис. 6. Результаты тестирования. Ось X – размер блока. Ось Y – эффективность функции(МБ/С)

считываемого за один вызов функции. Запись/чтение крупными блоками более эффективна, поэтому по возможности стоит накапливать данные в памяти перед записью на диск.

- Расположение данных существенно влияет на производительность ввода-вывода, что стоит учитывать при написании приложений
- В случае записи необходимо более аккуратно подходить к выбору функции записи, поскольку разница между коллективными и индивидуальными операциями являются достаточно существенной. Что касается разницы между индивидуальными и коллективными операциями чтения, то она оказалась гораздо менее выраженной, что связано с буферизацией на файловом хранилище.
- Зависимость скорости записи от размера файла отсутствует(по крайней мере, для файлов порядка сотен ГБ).
- Ввод-вывод является хорошо масштабируемым(по крайней мере, при смене числа процессов с 16 на 128)

В дальнейшем авторы планируют расширить систему тестов новыми тестами, которые будут направлены на исследование взаимного влияния операций чтения и записи. планируется также доработать инструмент визуализации данных при помощи gnuplot или libgnuplot, чтобы наглядно представлять результаты тестирования (в текущей реализации данные необходимо предварительно преобразовывать к определенному формату). Планируется сравнить производительность различных реализаций MPI, а также производительность других вычислительных систем, к примеру IBM BlueGene/P и «Ломоносов» на операциях ввода-вывода. Планируется автоматически определять «устойчивые» пики и провалы в производительности, а также попробовать динамически менять величину шага в размере блока около таких точек.

Литература

1. Jianwei Li, Wei-keng Liao, Alok Choudhary «Parallel netCDF: A High-Performance Scientific I/O Interface» // SC '03 Proceedings of the 2003 ACM/IEEE conference on Supercomputing, 2003 P. 39, http://www.mcs.anl.gov/ robl/pnetcdf/docs/pnetcdf-sc2003.pdf

2. P. Balaji, W. Feng, J. Archuleta, H. Lin, R. Kettimuthu, R. Thakur, X. Ma «Semantics-based Distributed I/O for mpiBLAST» // PPoPP '08 Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming 2008, P. 293–294,

3. MPI: A Message-Passing Interface Standard , Version 2.2 2009 http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

4. W. Gropp, E. Lusk «Reproducible Measurement of MPI Performance Characteristics. In J. Dongarra et al. (eds.)» // Recent Advances in Parallel Virtual Machine and Message Passing Interface, proceedings of the 6th European PVM/MPI Users' Group Meeting, EuroPVM/MPI'99, Barcelona, Spain Sept. 26-29, 1999, LNCS 1697, P. 11–18.

5. J. Borrill, L. Oliker, J. Shalf, H. Shan, A. Uselton «HPC Global File System Performance Analysis Using A Scientific-Application Derived Benchmark» http://crd-legacy.lbl.gov/ oliker/papers/PC09_MADbench.pdf

6. R. Latham, R. Ross «PVFS, ROMIO, and the noncontig Benchmark» www.mcs.anl.gov/romio/noncontig-perf.pdf

7. Parallel I/O Benchmarking Consortium http://www.mcs.anl.gov/research/projects/pio-benchmark/

8. The Los Alamos National Lab MPI-IO Test http://public.lanl.gov/jnunez/benchmarks/mpiiotest.htm

Параллельный глобальный поиск оптимальных условий протекания реакции карбоалюминирования олефинов*

В.В. Рябов[1], М.В. Тихонова[2]

ННГУ им. Лобачевского[1], Институт нефтехимии и катализа РАН[2]

Данная статья продолжает цикл работ по исследованию реакции каталитического карбоалюминирования олефинов с использованием параллельного индексного метода глобальной оптимизации. В предыдущих работах идентифицированы кинетические и энергетические параметры реакции путем минимизации разности выходных данных математической модели и данных, полученных из химических экспериментов. В рамках нового исследования решается задача максимизации нескольких конечных продуктов, концентрации которых зависят от начальных данных и условий протекания реакции.

1. Введение

Все химико-технологические процессы протекают при определенных значениях технологических параметров, изменение которых может привести не только к снижению количества и качества выпускаемой продукции, но и к тяжелым авариям, взрывам и пожарам на производстве. Такие технологические параметры, как температура, давление, концентрация реагирующих веществ, влияют на равновесное состояние системы, в которой протекают обратимые химические реакции. Поэтому технологам важно знать оптимальные условия проведения химико-технологического процесса с целью повышения производительности аппарата.

2. Постановка задачи поиска оптимальных условий

Оценка эффективности проведения сложных химических реакций связана с выделением определенного свойства реакции (например, максимум выхода продукта, длина периода индукции и др.). Для этого требуется многократное решение вычислительно трудоемких прямых задач при различных параметрах, характеризующих условия проведения реакций и исходные концентрации реагирующих веществ. Таким образом, возникает многопараметрическая задача глобальной оптимизации, требующая значительного объема вычислений, в связи с чем, актуальным является использованием эффективных численных методов, использующих параллельные вычисления.

2.1 Прямая кинетическая задача

Прямая кинетическая задача для изотермической нестационарной модели без изменения объема в закрытой системе на основе закона действующих масс представляет собой задачу Коши для системы обыкновенных нелинейных дифференциальных уравнений с полиномиальными правыми частями степени не выше 3:

$$\frac{dx_i}{dt} = Q_i,\ i = 1..M;\quad Q_i = \sum_{j=1}^{N} S_{ij} w_j; \tag{1}$$

$$w_j = k_j \prod_{i=1}^{M} (x_i)^{|\alpha_{ij}|} - k_{-j} \prod_{i=1}^{M} (x_i)^{\beta_{ij}}, \tag{2}$$

* Работа выполнена при поддержке Совета по грантам Президента Российской Федерации (грант № НШ-1960.2012.9).

с начальными условиями: t=0, $x_i(0)=x_i^0$,где x_i – концентрации веществ (мольные доли), участвующих в реакции; M – количество веществ; N – количество стадий; S_{ij} – стехиометрическая матрица; w_j – скорость j-ой стадии (*1/ч*); k_j, k_{-j} – приведенные константы скорости прямой и обратной реакции (*1/ч*) соответственно; α_{ij} – отрицательные элементы S_{ij}, β_{ij} – положительные элементы S_{ij}.

Зависимость константы скорости химической реакции k от температуры T определяется уравнением Аррениуса:

$$k = Ae^{-\frac{E}{RT}}, \tag{3}$$

где k – приведенная константа скорости элементарной стадии, *1/ч*; E – энергия активации, *Дж/моль*, R – универсальная газовая постоянная, *Дж/(моль·К),* A – частота столкновений реагирующих молекул, T – температура, *К*.

В нешироких интервалах умеренных температур, в которых обычно производятся кинетические измерения, энергия активации и частота столкновений реагирующих молекул не зависит от температуры.

2.2 Задача максимизации выхода продукта

Для оценки эффективности проведения реакции используют такие показатели, как скорость реакции (интенсивность ее протекания), степень превращения (конверсия, прореагировавшая доля исходного вещества), выход продукта и селективность (мера того, насколько полно реакция осуществляется в направлении получения целевого продукта) [1]. Для расчета таких показателей, необходимо иметь достаточно точное распределение концентраций и скоростей стадий реакций во времени.

Одним из рациональных принципов управления химическими процессами является максимизация выхода продукта в зависимости от условий проведения реакции и концентраций исходных веществ.

Поскольку в реальных промышленных установках исходные реагенты берутся в значительно большем количестве, чем при лабораторных исследованиях, то в математическом описании задачи поиска оптимальных условий [9] предлагается вместо непосредственных концентраций исходных веществ брать их отношение к базовому катализатору:

$$\begin{gathered}\varphi(y) = \Phi(y) = \max_t\left(x_{прод}(x_{кат}, y)\right) \to \max, \\ y = \left(C_{исх}, T\right),\end{gathered} \tag{4}$$

где Φ – максимум концентрации исследуемого продукта (*моль/л*); T – температура, при которой протекает реакция (*К*); t - время проведения реакции (*ч*); $x_{кат}$ – базовая концентрация катализатора (*моль/л*); $x_{прод}$ – концентрация исследуемого продукта химической реакции (*моль/л*), $C_{исх}$ – вектор отношений концентраций исходных веществ химической реакции к концентрации катализатора.

В зависимости от числа целевых продуктов для каждой химической реакции может возникнуть несколько задач оптимизации.

3. Реакция карбоалюминирования олефинов

Одной из ключевых реакций металлокомплексного катализа является реакция карбоалюминирования олефинов, которая получила применение в лабораторной практике как эффективный способ построения новых *Me-C (металл-углеродных)* и *C-C (углерод-углеродных)* связей. Использование циркония *Zr* в качестве катализатора этой реакции позволяет в мягких условиях с высокой селективностью функционализировать непредельные соединения. До настоящего времени не было проведено систематического исследования влияния структуры катализатора и условий проведения процесса на селективность реакции.

Для исследования Me-C типа связи в ИНК РАН г. Уфы было предложено описание двух механизмов реакции карбоалюминирования [2], катализируемых циркониевыми катализаторами L_2ZrCl_2 (рис. 1).

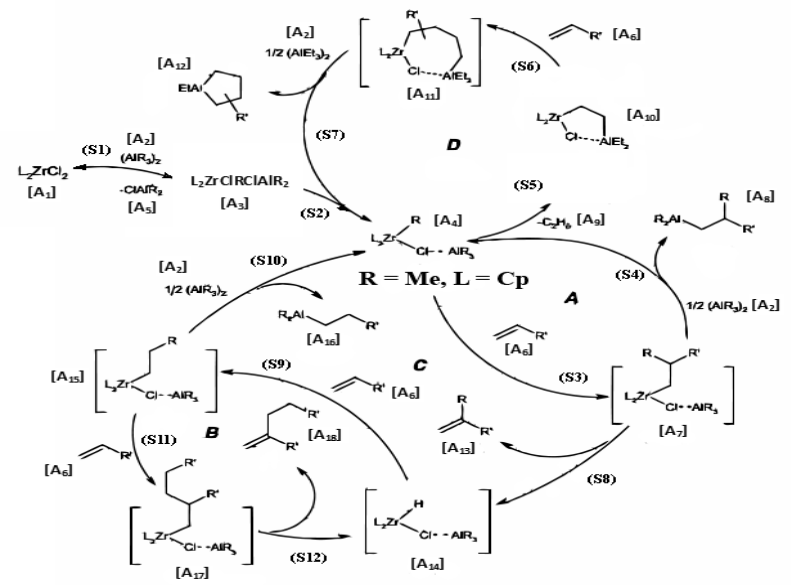


Рис. 1. Реакция карбоалюминирования олефинов, катализируемая Cp_2ZrCl_2

В качестве лиганда L используется Ср (таблица 1, схема 1) и $CpMe_5$ (таблица 1, схема 2), где $Cp=C_5H_5$. В качестве A_i выступают вещества, представленные в таблице 2. Исходными веществами реакции являются A_2, A_6 и катализатор A_1, продуктами – A_8, A_{18}, A_{16}. Остальные вещества являются промежуточными, образующимися во время протекания реакции.

Таблица 1. Реакция карбоалюминирования, катализируемая $(CpMe_5)_2ZrCl_2$

	Схема 1 в присутствии катализатора Cp_2ZrCl_2	**Схема 2 в присутствии катализатора $(CpMe_5)_2ZrCl_2$**
1)	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\leftrightarrow}} A_3$	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\leftrightarrow}} A_3$
2)	$A_2 + A_3 \underset{k_{-2}}{\overset{k_2}{\leftrightarrow}} A_4 + A_5$	$A_3 + A_6 \overset{k_2}{\rightarrow} A_{19}$
3)	$A_4 + A_6 \overset{k_3}{\rightarrow} A_7$	$A_2 + A_{19} \overset{k_3}{\rightarrow} A_3 + A_8$
4)	$A_2 + A_7 \overset{k_4}{\rightarrow} A_4 + A_8$	$A_{19} \overset{k_4}{\rightarrow} A_1 + A_8$
5)	$A_7 \overset{k_5}{\rightarrow} A_{13} + A_{14}$	$A_{19} \overset{k_5}{\rightarrow} A_{13} + A_{20}$
6)	$A_6 + A_{14} \overset{k_6}{\rightarrow} A_{15}$	$A_6 + A_{20} \overset{k_6}{\rightarrow} A_{21}$
7)	$A_6 + A_{15} \overset{k_7}{\rightarrow} A_{14} + A_{18}$	$A_6 + A_{21} \overset{k_7}{\rightarrow} A_{18} + A_{20}$
8)	$A_2 + A_{15} \overset{k_8}{\rightarrow} A_{14} + A_{16}$	$A_{21} \overset{k_8}{\rightarrow} A_1 + A_{16}$
9)		$A_2 + A_{21} \overset{k_9}{\rightarrow} A_3 + A_{16}$

Таблица 2. Список веществ, участвующих в реакции карбоалюминирования

$A_1 = L_2ZrCl_2$, $L = CpMe_5$ или Cp	$A_{13} = H_2CCMeR+$
$A_2 = AlMe_3$	$A_{16} = Me_2AlCH_2CH_2R+$
$A_3 = L_2ZrClMeClAlMe_2$	$A_{18} = H_2CCRCH_2CH_2R+$

$A_6 = CH_2CHR$	$A_{19} = L_2ZrC\ H_2CHMeRClClAlMe_2$
$A_8 = CH_2CHMeRAlMe_{2+}$	$A_{20} = L_2ZrHClClAlMe_2$
	$A_{21} = L_2ZrC\ H_2C\ H_2RClClAlMe_2$

В предыдущих работах [7, 8] в результате решения обратных кинетических задач (рис. 2) по восстановлению на основе экспериментального материала видов кинетических моделей реакции были определены кинетические и энергетические параметры для обоих механизмов (таблицы 3 и 4).

Таблица 3. Кинетические параметры реакции, катализируемой Cp_2ZrCl_2

№ стадии	**1 →**	**2 →**	**3 →**	**4 →**	**5 →**	**6 →**	**7 →**	**8 →**	**1 ←**	**2 ←**
***E*, ккал/моль**	13.06	8.97	8.27	12.33	6.18	2.86	2.38	5.42	7.21	3.94
***A*, 1/ч**	$1.72\cdot10^8$	$2.33\cdot10^9$	$3.57\cdot10^9$	$4.69\cdot10^{19}$	$2.20\cdot10^{11}$	$1.11\cdot10^8$	$3.57\cdot10^9$	$7.59\cdot10^7$	$9.94\cdot10^6$	$2.63\cdot10^{10}$

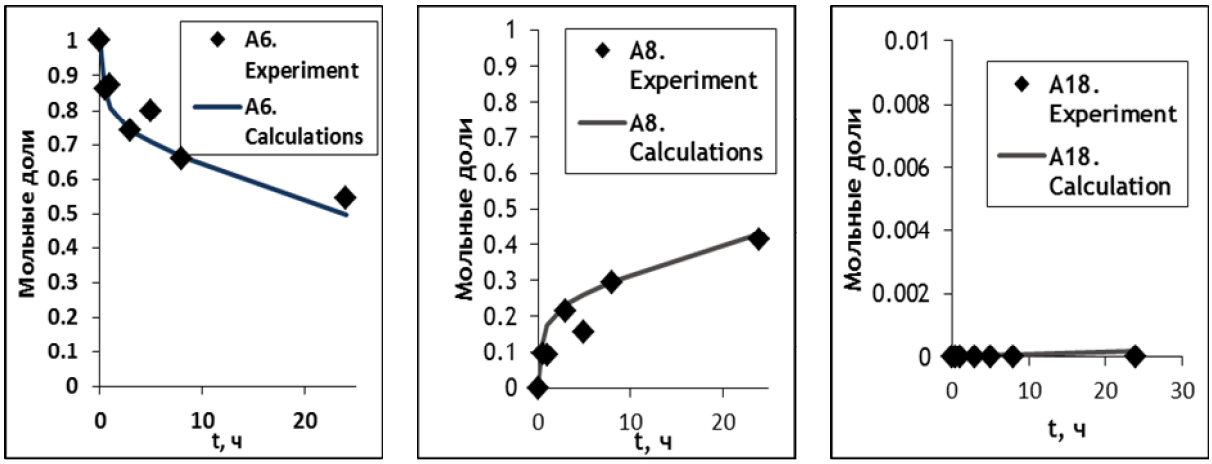


Рис. 2. Сопоставление расчетных и экспериментальных данных реакции карбоалюминирования олефинов, катализируемой $(CpMe_5)_2ZrCl_2$ при 30°С

Таблица 4. Кинетические параметры реакции, катализируемой $(CpMe_5)_2ZrCl$

№ стадии	**1 →**	**2 →**	**3 →**	**4 →**	**5 →**	**6 →**	**7 →**	**8 →**	**9 →**	**1 ←**
***E*, ккал/моль**	5.36	9.94	4.05	22.14	4.28	11.88	10.98	2	9.6	4.66
***A*, 1/ч**	$3.78\cdot10^8$	$1.14\cdot10^9$	$7.99\cdot10^7$	$1.75\cdot10^5$	$5.27\cdot10^6$	$7.99\cdot10^7$	$1.49\cdot10^4$	$2.30\cdot10^9$	$7.99\cdot10^7$	$3.60\cdot10^7$

Таким образом, зная механизм, кинетические и энергетические параметры химической реакции, можно оценить эффективность ее проведения в направлении какого-либо продукта путем рассмотрения различных условий ее протекания:

$$\begin{aligned} \varphi_8 &= \max_t\left(x_8(x_1, y)\right) \to \max, \\ \varphi_{16} &= \max_t\left(x_{16}(x_1, y)\right) \to \max, \\ \varphi_{18} &= \max_t\left(x_{18}(x_1, y)\right) \to \max, \\ y &= \left(c_2, c_6, t, T\right). \end{aligned} \tag{5}$$

4. Эффективные алгоритмы многоэкстремальной оптимизации

Алгоритмы, развиваемые Нижегородской научной школой многоэкстремальной оптимизации, предполагают следующую постановку задачи:

$$\varphi^* = \varphi(y^*) = \min\{\varphi(y):\ y \in D\},$$
$$D = \{y \in R^N:\ a_i \le y_i \le b_i,\ 1 \le i \le N\}, \quad (6)$$

где целевая функция $\varphi(y)$ удовлетворяет условию Липшица с соответствующей константой L, а именно

$$|\varphi(y_1) - \varphi(y_2)| \le L \|y_1 - y_2\|,\ y_1, y_2 \in D.$$

Используя кривые типа развертки Пеано $y(x)$, однозначно отображающие отрезок $[0,1]$ на N-мерный гиперкуб P

$$P = \{y \in R^N:\ -2^{-1} \le y_i \le 2^{-1},\ 1 \le i \le N\} = \{y(x):\ 0 \le x \le 1\},$$

исходную задачу можно редуцировать к следующей одномерной задаче:

$$\varphi(y_D(x^*)) = \min\{\varphi(y_D(x)):\ x \in [0,1]\}.$$

Рассматриваемая схема редукции размерности сопоставляет многомерной задаче с липшицевой минимизируемой функцией одномерную задачу, в которой целевая функция удовлетворяет равномерному условию Гельдера (см. [7]), т.е.

$$|\varphi(y_D(x')) - \varphi(y_D(x''))| \le K|x' - x''|^{1/N},\ x', x'' \in [0,1],$$

где N есть размерность исходной многомерной задачи, а коэффициент K связан с константой Липшица L исходной задачи соотношением $K \le 4L\sqrt{N}$.

Различные варианты индексного алгоритма для решения одномерных задач и соответствующая теория сходимости представлены в работах [3], [5].

Параллельная версия индексного метода основана на построении множественных отображений Пеано, получаемых путем сдвига или вращения гиперкубов друг относительно друга (сдвиговые и вращаемые развертки), что позволяет заодно и улучшать сходимость алгоритма за счет более точной адаптивной оценки неизвестной константы Гельдера K в процессе вычислений. Целым рядом преимуществ обладает схема построения вращаемых разверток, предложенная в работе [6] и позволяющая использовать до $N(N-1)+1$ вычислительных ядер.

5. Поиск оптимальных условий проведения химического процесса

Для оценки оптимальных условий проведения реакции потребовалось решать задачи глобального поиска (5) для каждой из двух предложенных схем реакции и для каждого продукта. В результате решения шести задач определен оптимальный температурный режим реакции и начальные концентрации, при которых выход исследуемых продуктов реакции был максимален (таблица 5).

Реакция каталитического карбоалюминирования олефинов протекает в нешироком интервале температур от 15 до 35 °С. В качестве базовой концентрации катализатора для всех расчетов бралась концентрация, при которой проходило лабораторное исследование – 0.2 ммоль при объеме реакционной смеси 1.9 мл.

Таблица 4. Оптимальные условия проведения реакции карбоалюминирования олефинов

	Схема 1 в присутствии катализатора Cp_2ZrCl_2			**Схема 2 в присутствии катализатора $(CpMe_5)_2ZrCl_2$**		
	Продукты реакции			**Продукты реакции**		
Условия проведения	A_8	A_{16}	A_{18}	A_8	A_{16}	A_{18}
Температура, °С	35	35	35	35	35	35
Концентрация Базового катализатора x_1, моль/л	0.105	0.105	0.105	0.105	0.105	0.105

Отношение x_2/x_1	100 : 1	5.85 : 1	0.1 : 1	100 : 1	6.1 : 1	24.76 : 1
Отношение x_6/x_1	87.7 : 1	4.7 : 1	100 : 1	100 : 1	21.6 : 1	100 : 1
Максимум выхода продукта, *моль/л*	**0.2475**	**$8\cdot10^{-5}$**	**0.4945**	**0.3746**	**0.1197**	**0.00135**

Из полученных результатов видно, что на выход каждого продукта сильное влияние оказывает выбор катализатора. Если для максимизации веществ A_8, A_{16} эффективнее использовать катализатор $(CpMe_5)_2ZrCl_2$, то для получения максимума вещества A_{18} – катализатор Cp_2ZrCl_2.

Таким образом, многокритериальную задачу имеет смысл рассматривать только для максимизации веществ A_8, A_{16} с использованием катализатора $(CpMe_5)_2ZrCl_2$. Для решения такой двухкритериальной задачи можно, например, применять метод Стронгина-Маркина, который сводит двухкритериальную задачу размерности N к однокритериальной задаче размерности 2N (независимо от числа критериев) и позволяет получить приближенное покрытие множества Парето-оптимальных точек. Однако такой подход требует более длительных вычислений и может быть применен в дальнейших исследованиях.

Стоит отметить, что последовательное решение задачи максимизации (на примере вещества A_8) с относительной точностью $\varepsilon=0.0001$ занимает более 7 суток 6 часов на процессоре Intel Core i7 (2.8 ГГц). Размерность задачи оптимизации (N=3) позволяет индексному методу использовать до 6 вычислительных ядер, что позволяет получить результат за 30 часов (с коэффициентом ускорения 5.8).

6. Заключение

В результате данного исследования проведена первичная оценка оптимальных условий протекания реакции каталитического карбоалюминирования олефинов при различной структуре катализатора. При этом увеличение температуры способствует увеличению выхода всех продуктов, а использование соответствующего катализатора может быть рекомендовано химикам-технологам для максимизации конкретного продукта.

Алгоритмы нижегородской научной школы глобальной оптимизации подходят для широкого класса задача. Индексный метод глобальной оптимизации (и его модификации) продолжает с успехом применяться: на этот раз для решения задачи увеличения выхода продуктов в зависимости от условий проведения сложных химических реакций. Таким образом, эффективные методы глобальной оптимизации позволяют решать всю цепочку исследовательских задач, начиная с идентификации математической модели реакции и заканчивая максимизацией выхода продукта, выбором наиболее эффективного катализатора и других технологических показателей.

Литература

1. Царева З.М., Орлова Б.И. Теоретические основы химической технологии. – К: Вища шк. Головное издательство, 1986. - С. 35.
2. Parfenova L.V., Gabdrakhmanov V.Z., Khalilov L.M., Dzhemilev U.M. On study of chemoselectivity of reaction of trialkylalanes with alkenes, catalyzed with Zr π-complexes // J. Organomet. Chem. V. 694. № 23. 2009. P. 3725-3731.
3. Стронгин Р.Г. Поиск глобального оптимума. М.: Знание, 1990.
4. Стронгин Р.Г. Параллельная многоэкстремальная оптимизация с использованием множества разверток // Ж. вычисл. матем. и матем. физ. Т.31. №8. 1991. С. 1173-1185.
5. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht, 2000.

6. Баркалов К.А., Рябов В.В., Сидоров С.В. Параллельные вычисления в задачах многоэкстремальной оптимизации. // Ж. Вестник ННГУ, 2009. №6. С. 171-177.
7. Губайдуллин И.М., Рябов В.В., Тихонова М.В. Применение индексного метода глобальной оптимизации при решении обратных задач химической кинетики. // Ж. Вычислительные методы и программирование, 2011. Т.12. С. 137-145.
8. Тихонова М.В., Рябов В.В. Решение агрегированных обратных задач химической кинетики параллельным индексным методом глобальной оптимизации. // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (19-24 сентября 2011 г., г. Новороссийск). М.: Изд-во МГУ, 2011. С. 572-579.
9. Рябов В.В., Тихонова М.В., Губайдуллин И.М. Методы глобальной оптимизации и исследование эффективности химических реакций карбоалюминирования олефинов. // Материалы XI Всероссийской конференции “Высокопроизводительные параллельные вычисления на кластерных системах”, Нижний Новгород. 2011. С. 278-281.

Комплекс параллельных программ для моделирования упругопластических волн в структурно неоднородных средах*

В.М. Садовский, О.В. Садовская, М.П. Варыгина

Институт вычислительного моделирования СО РАН

Разработан комплекс параллельных прикладных программ, представляющий собой набор инструментальных средств, предназначенных для решения задач динамики деформируемых сред с определяющими уравнениями достаточно общего вида на основе явных разностных схем сквозного счета. Созданы компьютерные приложения 2Dyn_Granular, 2Dyn_Cosserat и 3Dyn_Granular, 3Dyn_Cosserat для численного исследования процессов распространения упругопластических волн в структурно неоднородных средах с учетом и без учета моментных свойств материала на многопроцессорных вычислительных системах кластерного типа.

1. Введение

Классические модели механики деформируемого твердого тела – теории упругости, пластичности и ползучести – не учитывают различное сопротивление материалов растяжению и сжатию, а также их структурную неоднородность. Симметричными по отношению к растягивающим и сжимающим деформациям можно с некоторой погрешностью считать металлы и их сплавы, но этим свойством отнюдь не обладают грунты, горные породы, углеграфиты, полимеры, пористые материалы и т.п. Например, идеальные среды типа сухого песка, частицы которых свободно контактируют между собой, при сжатии ведут себя как упругие или упругопластические тела, в зависимости от уровня напряжений, и не сопротивляются растяжению. В связных средах (грунтах, горных породах) допустимые растягивающие напряжения существенно меньше сжимающих и не превышают критического значения, обусловленного сцеплением частиц. В последнее время в технических приложениях при конструировании демпфирующих элементов применяются новые искусственные материалы – пенометаллы, пористость которых достигает 75 % (пенистый алюминий, пористая медь и т.п.). При растягивающих или сжимающих нагрузках до момента схлопывания пор такие материалы достаточно податливы и их деформация сопровождается значительной пластической диссипацией энергии, а при дальнейшем сжатии прочность резко возрастает.

Структуру (структурную неоднородность), связанную с атомным и кристаллическим строением вещества, имеют, очевидно, все материалы. Это так называемая наноразмерная – мелкомасштабная структура. Многие материалы обладают также структурой более крупного масштаба. В частности, горные породы имеют естественную кусковатость и, таким образом, неоднородны на макроуровне. Костные ткани, искусственные материалы на основе пены, нефтенасыщенные горные породы и прочие материалы неоднородны на микро- и мезоуровнях.

При численном анализе деформаций структурированных материалов на основе методов конечных элементов или конечных разностей приходится использовать достаточно мелкие расчетные сетки, размер ячеек которых существенно меньше характерного масштаба неоднородности материала. При решении динамических задач в пространственной постановке оказываются эффективными параллельные алгоритмы, поскольку они позволяют распределять вычислительную нагрузку между большим числом узлов кластера. Использование распределенных вычислений дает возможность измельчать расчетные сетки, повышая тем самым точность численного решения.

*Работа выполнена при финансовой поддержке РФФИ (код проекта 11-01-00053).

2. Математические модели

Исходная математическая модель для описания процесса деформирования упругих тел задается системой уравнений

$$A\,\frac{\partial U}{\partial t} = \sum_{i=1}^{n} B^i\,\frac{\partial U}{\partial x_i} + Q\,U + G, \tag{1}$$

где U – m-мерная искомая вектор–функция, A – симметричная положительно определенная матрица коэффициентов при производных по времени, B^i – симметричные матрицы коэффициентов при производных по пространственным переменным, Q – антисимметричная матрица, G – заданный вектор, n – размерность задачи (2 или 3). Размерность m системы (1), а также конкретный вид матриц–коэффициентов определяется используемой математической моделью.

При учете пластической деформации материала система уравнений (1) заменяется вариационным неравенством

$$(\widetilde{U} - U)\Big(A\,\frac{\partial U}{\partial t} - \sum_{i=1}^{n} B^i\,\frac{\partial U}{\partial x_i} - Q\,U - G\Big) \geq 0, \qquad \widetilde{U},\ U \in F, \tag{2}$$

где F – заданное выпуклое множество, с помощью которого накладываются ограничения на возможные состояния среды, $\widetilde{U}$ – произвольный элемент F.

В задачах механики сыпучих сред с пластическими свойствами возникает более общее вариационное неравенство

$$(\widetilde{V} - V)\Big(A\,\frac{\partial U}{\partial t} - \sum_{i=1}^{n} B^i\,\frac{\partial V}{\partial x_i} - Q\,V - G\Big) \geq 0, \qquad \widetilde{V},\ V \in F, \tag{3}$$

в котором вектор–функции V и U связаны между собой уравнениями

$$V = \lambda\,U + (1-\lambda)\,U^{\pi}, \qquad U = \frac{1}{\lambda}\,V - \frac{1-\lambda}{\lambda}\,V^{\pi}. \tag{4}$$

Здесь $\lambda \in (0,1]$ – параметр регуляризации модели, характеризующий отношение модулей упругости при растяжении и сжатии, U^{π} – проекция вектора–решения на заданный выпуклый конус K, с помощью которого описывается различное сопротивление материала растяжению и сжатию.

Входящее в (2) и (3) множество допустимых вариаций F определяется условием пластичности Мизеса:

$$F = \Big\{\sigma \;\Big|\; \tau(\sigma) \leq \tau_s\Big\},$$

где σ – тензор напряжений, $\tau(\sigma)$ – интенсивность касательных напряжений, τ_s – предел текучести частиц. В качестве выпуклого конуса напряжений K, допускаемых критерием прочности, используется круговой конус Мизеса–Шлейхера:

$$K = \Big\{\sigma \;\Big|\; \tau(\sigma) \leq æ\,p(\sigma)\Big\},$$

где $p(\sigma)$ – гидростатическое давление, æ – параметр внутреннего трения.

В плоских задачах динамики упругопластических сред вектор U включает в себя 6 неизвестных функций – 2 компоненты вектора скорости и 4 компоненты симметричного тензора напряжений:

$$U = \Big(v_1, v_2, \sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}\Big).$$

В пространственных задачах 9 неизвестных функций,

$$U = \Big(v_1, v_2, v_3, \sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{23}, \sigma_{13}, \sigma_{12}\Big).$$

При учете вращательных степеней свободы частиц микроструктуры материала в модели моментного континуума Коссера [1] этот вектор, наряду с компонентами вектора скорости v и тензора напряжений σ, содержит также компоненты вектора угловой скорости ω и несимметричного тензора моментных напряжений m. В двумерных задачах вектор U состоит из 12 неизвестных функций:

$$U = \Big(v_1, v_2, \sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}, \sigma_{21}, \omega_3, m_{23}, m_{32}, m_{31}, m_{13}\Big),$$

а в трехмерных задачах – из 24 неизвестных функций:

$$U = \Big(v_1, v_2, v_3, \sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{23}, \sigma_{32}, \sigma_{31}, \sigma_{13}, \sigma_{12}, \sigma_{21},$$
$$\omega_1, \omega_2, \omega_3, m_{11}, m_{22}, m_{33}, m_{23}, m_{32}, m_{31}, m_{13}, m_{12}, m_{21}\Big).$$

При учете сыпучести упругопластических материалов искомый вектор V включает в себя отличные от нуля компоненты вектора скорости частиц среды v и тензора действительных напряжений σ, а в вектор U вместо σ входит тензор условных напряжений s, определяемый по закону Гука. В наиболее общей модели сыпучей среды, по-разному сопротивляющейся растяжению и сжатию, в которой учитывается вращение частиц, вектор U выглядит так:

$$U = \Big(v_1, v_2, v_3, s_{11}, s_{22}, s_{33}, s_{23}, s_{32}, s_{31}, s_{13}, s_{12}, s_{21},$$
$$\omega_1, \omega_2, \omega_3, n_{11}, n_{22}, n_{33}, n_{23}, n_{32}, n_{31}, n_{13}, n_{12}, n_{21}\Big),$$

а вектор V получается из U заменой компонент тензоров s и n на σ и m, соответственно.

3. Параллельный вычислительный алгоритм

Для численного решения задач о распространении волн напряжений и деформаций в средах со сложными реологическими свойствами в рамках рассматриваемых математических моделей разработан параллельный вычислительный алгоритм [2].

К решению системы уравнений (1) применяется метод расщепления по пространственным переменным. Вариационные неравенства (2) и (3) решаются методом расщепления по физическим процессам, который на каждом шаге по времени приводит к системе (1).

Технология распараллеливания вычислительного алгоритма основывается на методе двуциклического расщепления по пространственным переменным [3]. В трехмерном случае на временно́м интервале $(t_0, t_0 + \Delta t)$ метод расщепления включает в себя 7 этапов: этап решения одномерной задачи в направлении x_1 на интервале $(t_0, t_0 + \Delta t/2)$, аналогичные этапы в направлении x_2 и в направлении x_3, этап решения системы линейных обыкновенных дифференциальных уравнений с матрицей Q, этап повторного пересчета задачи в направлении x_3 на интервале $(t_0 + \Delta t/2, t_0 + \Delta t)$ и этапы повторного пересчета в направлениях x_2 и x_1. Применение процедуры расщепления к системе уравнений (1) приводит к следующим одномерным системам:

$$A\,\frac{\partial U^1}{\partial t} = B^1\,\frac{\partial U^1}{\partial x_1} + G^1, \quad U^1(t_0, x) = U(t_0, x),$$
$$A\,\frac{\partial U^2}{\partial t} = B^2\,\frac{\partial U^2}{\partial x_2} + G^2, \quad U^2(t_0, x) = U^1(t_0 + \Delta t/2, x),$$
$$A\,\frac{\partial U^3}{\partial t} = B^3\,\frac{\partial U^3}{\partial x_3} + G^3, \quad U^3(t_0, x) = U^2(t_0 + \Delta t/2, x),$$
$$A\,\frac{\partial U^4}{\partial t} = Q\,U^4, \qquad\qquad U^4(t_0, x) = U^3(t_0 + \Delta t/2, x),$$
$$A\,\frac{\partial U^5}{\partial t} = B^3\,\frac{\partial U^5}{\partial x_3} + G^3, \quad U^5(t_0 + \Delta t/2, x) = U^4(t_0 + \Delta t, x),$$

$$A\,\frac{\partial U^6}{\partial t} = B^2\,\frac{\partial U^6}{\partial x_2} + G^2, \quad U^6(t_0+\Delta t/2, x) = U^5(t_0+\Delta t, x),$$
$$A\,\frac{\partial U^7}{\partial t} = B^1\,\frac{\partial U^7}{\partial x_1} + G^1, \quad U^7(t_0+\Delta t/2, x) = U^6(t_0+\Delta t, x).$$

Искомое значение $U(t_0+\Delta t, x)$ равно $U^7(t_0+\Delta t, x)$. При расчете двумерной (плоской или осесимметричной) задачи в методе расщепления отсутствуют третий и пятый этапы, относящиеся к направлению x_3. В плоском случае $Q=0$ и, следовательно, отсутствует также четвертый этап.

Известно, что рассматриваемый метод расщепления является методом второго порядка точности, если на его этапах используются схемы второго порядка, и что он обеспечивает устойчивость численного решения в пространственном случае при выполнении условия устойчивости для одномерных систем. Чтобы сохранить второй порядок точности, на четвертом этапе при решении системы обыкновенных дифференциальных уравнений в линейных задачах, когда векторы V и U равны друг другу, применяется неявная бездиссипативная разностная схема Кранка–Николсон:

$$A\,\frac{U^{k+1}-U^k}{\Delta t} = Q\,\frac{U^{k+1}+U^k}{2}$$

(k — номер шага по времени). Для нелинейной системы уравнений на четвертом этапе используется более общая схема и решение строится по методу последовательных приближений [2].

Оставшиеся шесть одномерных систем уравнений на этапах расщепления решаются с помощью явной монотонной ENO–схемы типа “предиктор–корректор” с кусочно-линейными распределениями скоростей и напряжений по ячейкам, построенной по принципам сеточно-характеристических методов [4]. Для наиболее общей модели, включающей в себя векторы U и V и описываемой неравенством (3), в случае постоянных матриц–коэффициентов на шаге “корректор” используются соотношения

$$U^{j+1/2} = U_{j+1/2} + \frac{\Delta t}{2}\,A^{-1}\Big(B^i\,\frac{V_{j+1}-V_j}{\Delta x_i} + G^i\Big).$$

Здесь индекс $j+1/2$ относится к центру ячейки пространственной разностной сетки, верхний индекс соответствует текущему временно́му слою, нижний – предыдущему слою. Вектор $V^{j+1/2}$ вычисляется через $U^{j+1/2}$ по формуле (4). Если матрицы переменные, то в качестве разностной производной по x_i берутся соответствующие слагаемые консервативной аппроксимации. Для замыкания схемы необходимо доопределить ее соотношениями шага “предиктор”. Пусть Y_l и c_l – полная система левых собственных векторов и собственных чисел матрицы B^iA^{-1}: $Y_l\,B^i = c_l\,Y_l\,A$, $Y_l\,A\,Y_h = \delta_{lh}$. Умножая уравнения системы слева на вектор Y_l, перейдем к системе дифференциальных уравнений, которые для модели упругой среды представляют собой уравнения на характеристиках:

$$Y_l\,A\,\frac{\partial U}{\partial t} = c_l\,Y_l\,A\,\frac{\partial V}{\partial x_i} + Y_l\,G^i.$$

После аппроксимации получим

$$(I_l^{j+1/2})^{\pm} = I_{l\,j+1/2} \pm \alpha_{l\,j+1/2}\,\frac{\Delta x_i}{2} + \big(c_l\,\beta_l + Y_l\,G^i\big)_{j+1/2}\,\frac{\Delta t}{4},$$

где $\alpha_{l\,j+1/2}$ и $\beta_{l\,j+1/2}$ – производные от коэффициентов разложения U и V по базису Y_l: $I_l = (Y_l\,A)_{j+1/2}\,U$ и $J_l = (Y_l\,A)_{j+1/2}\,V$, вычисленные с помощью итерационной процедуры предельной реконструкции. Индексами “–” и “+” отмечены значения этих коэффициентов на левой и правой границах одной и той же ячейки. Процедура предельной реконструкции позволяет повысить точность численного решения и состоит в построении монотонных

кусочно-линейных сплайнов, приближающих I_l и J_l с минимальными разрывами на границах соседних ячеек сетки. Во внутренних узлах расчетной области на шаге "предиктор" величины U_j находятся по формуле осреднения $U_j = (U_j^+ + U_j^-)/2$ через значения $U_j^\pm$, относящиеся к разным сторонам границы между ячейками и удовлетворяющие системе нелинейных алгебраических уравнений:

$$(Y_l A)_{j+1/2}\, U_j^+ = I^-_{l\,j+1/2} \quad \text{для } c_l \geq 0,$$
$$(Y_l A)_{j-1/2}\, U_j^- = I^+_{l\,j-1/2} \quad \text{для } c_l \leq 0,$$
$$D_l V_j^+ = D_l V_j^-.$$

В этой системе уравнения с матрицами D_l представляют собой условия непрерывности вектора скорости и вектора напряжений при переходе через границу, а число уравнений с учетом этих условий равно числу неизвестных величин. Условия непрерывности реализуются численно методом последовательных приближений. Значения V_j пересчитываются через U_j в соответствии с (4).

Для учета пластических свойств материалов вариационные неравенства (2) и (3) после аппроксимации производной по времени конечной разностью на интервале $(t_0, t_0 + \Delta t)$ в каждой ячейке пространственной сетки приводятся к виду:

$$(\widetilde{V} - V)\, A\, (U - \bar{U}) \geq 0, \qquad \widetilde{V},\, V \in F,$$

где $\bar{U}$ – решение упругой задачи в данный фиксированный момент времени t_0 (для неравенства (2) $V = U$). С учетом (4) получается неравенство

$$(\widetilde{V} - V)\, A\, \big(V - (1-\lambda)\, V^\pi - \lambda\, \bar{U}\big) \geq 0.$$

Отсюда, по определению проекции,

$$V = \Big((1-\lambda)\, V^\pi + \lambda\, \bar{U}\Big)^\Pi.$$

Отображение, заданное правой частью, является сжимающим. Процедура корректировки решения для учета пластических свойств состоит, таким образом, в определении неподвижной точки сжимающего отображения, и реализуется методом последовательных приближений. В случае упругопластической среды Прандтля–Рейсса данная процедура не требует итераций и совпадает с известной процедурой корректировки напряжений Уилкинса.

Распараллеливание вычислений производится на этапе расщепления системы уравнений по пространственным переменным. Для численной реализации алгоритма используется библиотека передачи сообщений MPI [5,6], язык программирования Fortran. Обмен данными между процессорами осуществляется на уровне коэффициентов разложения решения по базису из левых собственных векторов на этапе предельной реконструкции. Значения элементов массивов α_l и β_l в ячейках сетки, граничащих с линией раздела, в параллельной программе пересчитываются с использованием законтурных ячеек, обмен данными через которые производится с помощью функции MPI_Sendrecv.

Вычислительный алгоритм реализован в виде комплекса параллельных прикладных программ для решения плоских и пространственных задач динамики упругопластических сред с микроструктурой на многопроцессорных вычислительных системах. Программный комплекс позволяет проводить расчеты распространения волн, вызванных внешними механическими воздействиями, в массиве среды, составленном из произвольного числа разнородных блоков с криволинейными границами. Встраивание математической модели в программный комплекс осуществляется посредством программных модулей, реализующих определяющие уравнения нестационарной модели, начальные данные и граничные условия

задачи. Комплекс состоит из программы–препроцессора, основной программы расчета полей скоростей и напряжений, подпрограмм реализации граничных условий и условий склейки решений на несогласованных сетках соседних блоков, и программы–постпроцессора.

Универсальность программ достигается за счет произвола в выборе числа неизвестных функций, описывающих изменяющиеся со временем искомые физические поля, а также в возможности задания любого количества слоев, полос в слое и блоков в полосе, числа узлов сетки в каждом из блоков и количества исполняющих процессоров кластера. Комплекс оснащен программами автоматического построения криволинейных расчетных сеток с помощью кубических сплайнов в плоских и пространственных областях блочной структуры по заданным координатам вершин блоков и касательным векторам к поверхностям раздела в вершинах, со статическим распределением области решения задачи между рабочими процессорами по принципу равномерной загрузки. Склейка решений на межблочных границах выполняется специальной процедурой, в которой решение на измельченной сетке, получаемой пересечением граничных ячеек соседних блоков, определяется с помощью уравнений на характеристиках, а затем переносится на исходные сетки методом осреднения. Процессоры обмениваются данными посредством блокирующих функций MPI_Send и MPI_Recv.

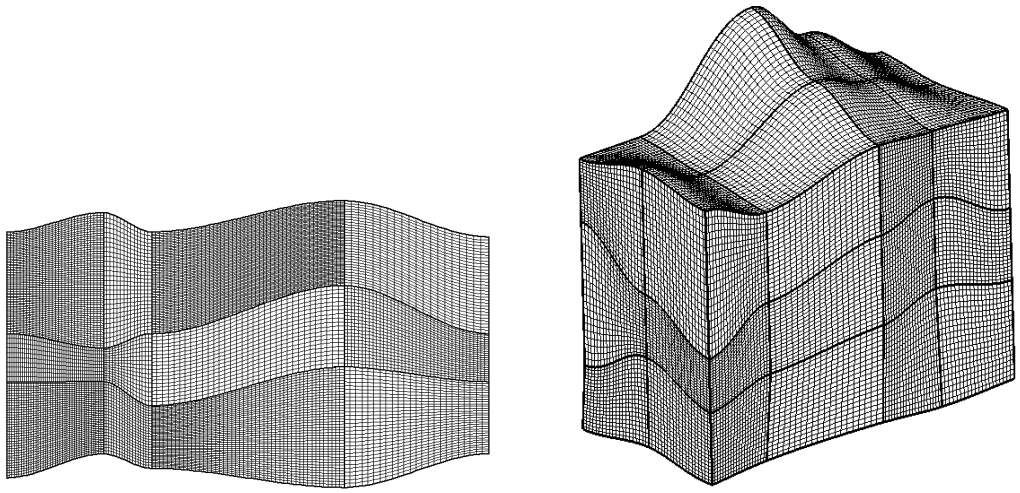
Необходимые для расчета исходные данные задачи представляются в виде текстовых файлов. Один из таких файлов содержит механические параметры материалов в блоках, другой – условия нагружения, в третьем хранится информация о блочной структуре массива – количество слоев по переменной x_1, количество полос в слое по переменной x_2 и количество блоков в полосе по переменной x_3, координаты вершин блоков, а также идентификационные номера материалов и пространственные размерности сеток. Каждый из узлов кластера при выполнении программы–препроцессора упаковывает исходные данные в двоичный файл прямого доступа – файл вещественных чисел, в который поблочно записываются параметры материала, часть сетки, приходящуюся на этот узел, и начальные значения решения, и файл целых чисел, содержащий соответствующие им адреса (указатели) – порядковые номера первых элементов. Вещественные файлы такой же структуры в дальнейшем создаются основной программой для организации контрольных точек при счете и для последующего анализа полученных результатов. Суммарный размер таких файлов может значительно превышать объем оперативной памяти отдельного процессора. Каждый процессор при старте основной программы считывает целочисленный файл и соответствующий ему вещественный файл. Далее целочисленный массив редуцируется – параметры сетки и указатели принимают в нем индивидуальные значения для данного узла.

Балансировка вычислительной нагрузки достигается за счет равномерного распределения сеточной области между узлами кластера. Если размерность сетки какого-либо блока больше средней размерности в расчете на один узел кластера, то этот блок обслуживается несколькими узлами, и наоборот, один и тот же узел обслуживает несколько блоков, если их суммарная размерность не превосходит средней. Из соображений минимизации количества пересылок используются 1D, 2D или 3D–разбиения расчетной области.

Основной программой на каждом узле кластера выполняются в принципе одни и те же вычисления, которые сводятся к взаимно согласованной поэтапной реализации метода расщепления по пространственным переменным (на каждом шаге по времени). Исключение составляют процессоры, производящие склейку решений на внутренних границах раздела. Условия склейки реализуются по следующей схеме. Процессоры, обслуживающие приграничные блоки (блоки, расположенные по обе стороны от границы раздела), передают необходимую информацию одному из таких процессоров, который в дальнейшем производит расчет всей границы в целом и рассылает результаты в обратном направлении. Если разделяющая блоки граница лежит внутри области, обслуживаемой одним процессором, то склейка выполняется автономно. Обмен данными между процессорами происходит при реализации шага “предиктор” ENO–схемы на этапах расщепления задачи по пространственным переменным.

Программа–постпроцессор производит сжатие файлов, содержащих результаты счета в контрольных точках, поскольку размер таких файлов может быть очень большим и для их транспортировки по глобальной сети потребуются значительные ресурсы. Графический вывод результатов осуществляется с помощью специальных программ, предназначенных для обычного персонального компьютера. Разработана также программа для представления результатов расчетов волновых задач, полученных на кластерах, в формате SEG-Y Международного геофизического общества с целью последующей обработки данных в системе SeisView.

Комплекс программ для численного решения задач динамики структурно неоднородных деформируемых сред ориентирован на применение при изучении процессов распространения сейсмических волн в блочных массивах с криволинейными поверхностями раздела. Блочная структура массива считается регулярной в том смысле, что входящие в него отдельные блоки из однородных материалов могут быть пронумерованы тремя индексами вдоль осей декартовой системы координат. Такая нумерация возможна, только если внутренние границы раздела согласованы между собой. При наличии несогласованных границ необходимо, продолжив эти границы, произвести фиктивное регулярное разбиение массива, в котором будет участвовать большое количество блоков из одного и того же материала. Примеры регулярного разбиения на блоки, ограниченные криволинейными поверхностями, приведены на рис. 1. В двумерном случае (рис. 1 а) массив среды включает в себя 12 блоков, в трехмерном случае (рис. 1 б) он состоит из 24 блоков. Если материал в соседних блоках один и тот же, а на границе раздела заданы условия непрерывности решения, то эта граница является фиктивной. Возможно задание различных материалов во всех блоках.



а б

Рис. 1. Разностные сетки в блочных массивах с криволинейными поверхностями раздела

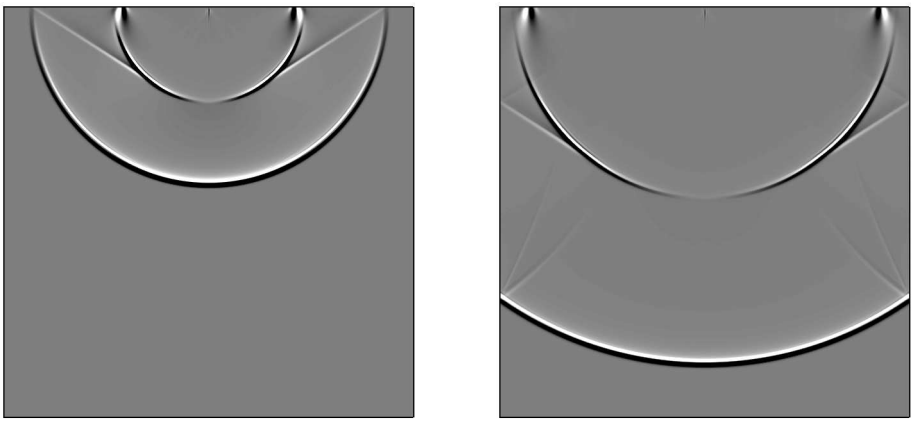
Криволинейные разностные сетки в пространственных блоках строятся с помощью алгебраического подхода, который заключается в вычислении взаимно-однозначных отображений $x = x(\xi)$ единичного куба с равномерной сеткой в пространстве параметров ξ_1, ξ_2, ξ_3 на физическую область. Выбор данного подхода связан с относительной простотой реализации алгоритмов склейки решений на поверхностях раздела. Функция, осуществляющая отображение, ищется в виде многомерного кубического сплайна $x = \sum_{i_1,\, i_2,\, i_3=0}^{3} C_{i_1 i_2 i_3} \xi_1^{i_1} \xi_2^{i_2} \xi_3^{i_3}$, векторные коэффициенты которого находятся в явном виде из условий сопряжения в вершинах блоков. В двумерных (плоских, осесимметричных) задачах отображение не зависит от одного из параметров.

4. Комплексы программ

4.1. Комплекс параллельных программ для решения двумерных упругопластических задач динамики сыпучих сред (2Dyn_Granular)

Разработан комплекс прикладных программ 2Dyn_Granular, предназначенный для численного решения динамических задач в плоской постановке на основе универсальной математической модели, описывающей малые деформации упругих, пластических и сыпучих сред. Допускается задание блочной расчетной области, составленной из разнородных материалов с произвольным числом слоев и блоков. В случае упругого материала модель сводится к гиперболической по Фридрихсу системе уравнений (1), записанной в терминах скоростей и напряжений в симметрической форме. В случае упругопластического материала модель представляет собой специальную формулировку теории течения Прандтля–Рейсса в виде вариационного неравенства (2) с односторонними ограничениями на напряжения. Обобщение модели для описания деформации сыпучей среды (3), (4) получено на основе реологического подхода, учитывающего различное сопротивление материала растяжению и сжатию. Начальные данные краевой задачи формулируются в терминах смещений, скоростей и напряжений. На межблочных границах ставятся условия непрерывности векторов скорости и напряжения. На внешних границах расчетной области могут быть заданы граничные условия в скоростях, в напряжениях, смешанные граничные условия или условия симметрии, обеспечивающие математическую корректность задачи.

В качестве иллюстрации работоспособности программного комплекса на рис. 2 приведены результаты численного решения задачи Лэмба о действии сосредоточенной импульсной нагрузки на границе упругого блока для двух моментов времени (а и б), выполненные на сетке из 1000×1000 узлов на 15 процессорах кластера МВС–100k Межведомственного суперкомпьютерного центра РАН.



а б

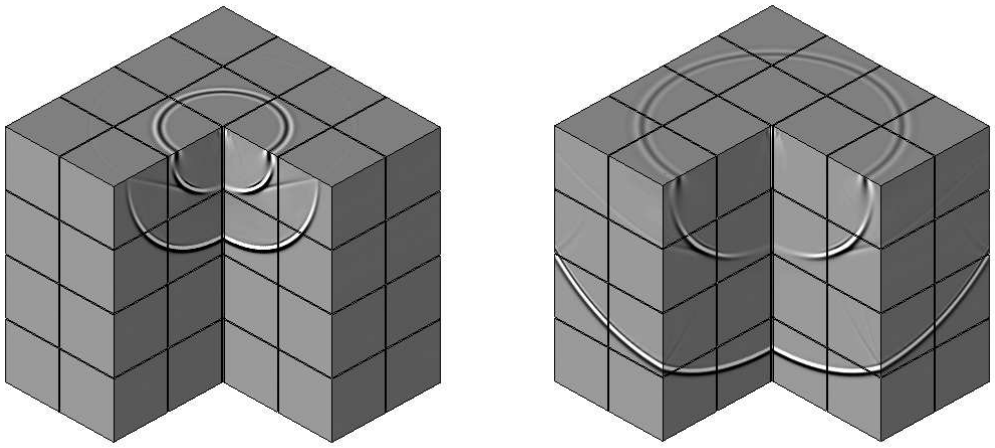
Рис. 2. Результаты расчета двумерной задачи Лэмба (линии уровня напряжения σ_{11})

На рис. 2 с помощью линий уровня нормального напряжения выделены фронты продольной, поперечной, конической и поверхностной волн, движущихся от точки приложения нагрузки в середине верхней границы вовнутрь блока. На боковых сторонах блока поставлены неотражающие граничные условия в некотором упрощенном варианте, которые дают, судя по рисунку, незначительное отражение при падении волны под углом. Сравнение с точным решением показало соответствие результатов в пределах погрешности используемой разностной схемы. Решение задачи Лэмба автомодельно, не зависит от масштаба по пространственным переменным и времени, поэтому маркировка осей не приведена.

4.2. Комплекс параллельных программ для решения трехмерных упругопластических задач динамики сыпучих сред (3Dyn_Granular)

Разработан комплекс программ 3Dyn_Granular, предназначенный для численного решения динамических задач в пространственной постановке на основе универсальной математической модели, описывающей малые деформации упругих, пластических и сыпучих сред. По аналогии с двумерным вариантом при подготовке расчета допускается задание пространственной блочной области, составленной из разнородных материалов с произвольным числом слоев, полос в слое и блоков в полосе с согласованными между собой криволинейными границами раздела. В случае изначально несогласованных границ необходимо применить процедуру фиктивного разбиения на блоки, задав в блоках, полученных в результате такого разбиения, одинаковые параметры материала. Используется трехмерный вариант представленной в предыдущем пункте модели. Постановка начальных данных и граничных условий осуществляется подобно двумерному варианту задачи.

На рис. 3 а, б приведены результаты численного решения задачи Лэмба для упругого блока (в различные моменты времени), выполненные на сетке из $400 \times 400 \times 400$ узлов на 64 процессорах кластера МВС–100k МСЦ РАН. Поверхности уровня нормального напряжения изображают фронты продольной, поперечной, конической и поверхностной пространственных волн.



а б

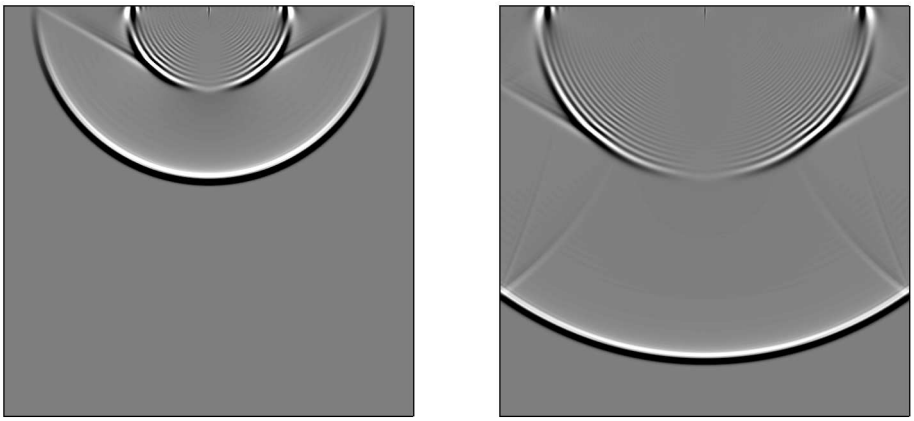
Рис. 3. Результаты расчета трехмерной задачи Лэмба (линии уровня напряжения σ_{11})

4.3. Комплекс параллельных программ для расчета двумерных динамических задач моментной теории упругости (2Dyn_Cosserat)

Разработан комплекс программ 2Dyn_Cosserat, предназначенный для численного решения плоских динамических задач моментной теории упругости Коссера, учитывающей в рамках теории малых деформаций вращательные степени свободы частиц микроструктуры материала. Допускается задание блочной расчетной области, составленной из разнородных материалов с произвольным числом слоев и блоков с согласованными криволинейными границами раздела. Модель формулируется в виде гиперболической по Фридрихсу системы уравнений (1), записанной в терминах вектора скорости поступательного движения и скорости вращательного движения, а также тензоров напряжений и моментных напряжений. Начальные данные краевой задачи формулируются в терминах смещений, угла поворота, скоростей поступательного и вращательного движения, напряжений и моментных

напряжений. На межблочных границах ставятся условия непрерывности вектора скорости и угловой скорости, векторов напряжения и моментного напряжения. На внешних границах могут быть заданы граничные условия в скоростях поступательного и вращательного движений, в напряжениях и моментных напряжениях, а также смешанные граничные условия или условия симметрии.

Для иллюстрации работоспособности программного комплекса на рис. 4 приведены результаты расчетов задачи Лэмба о действии сосредоточенной импульсной нагрузки на границе упругого блока для двух моментов времени (а и б), выполненные на сетке из 1000×1000 узлов на 10 процессорах кластера.



а б

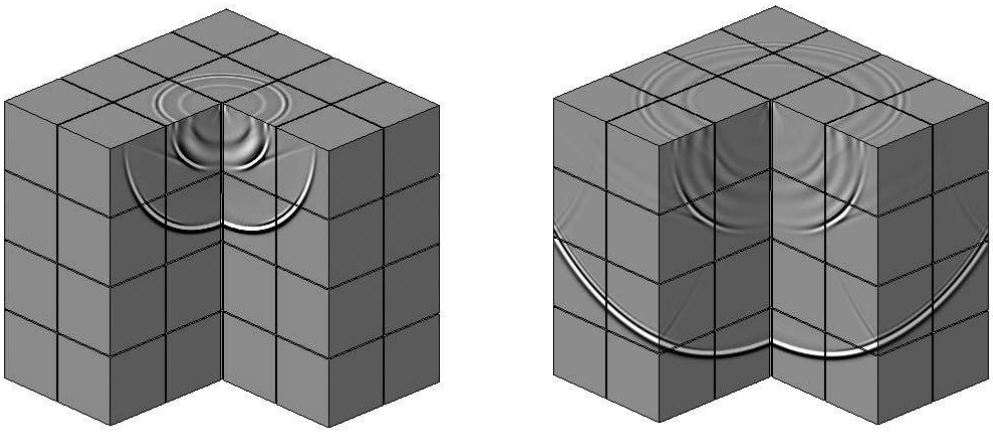
Рис. 4. Результаты численного решения двумерной задачи Лэмба для моментной среды (линии уровня напряжения σ_{11})

На рис. 4 с помощью линий уровня нормального напряжения выделены фронты продольной, поперечной, конической и поверхностной волн, движущихся от точки приложения нагрузки в середине верхней границы вовнутрь блока. Видны также осцилляции за фронтом поперечной волны, которые зависят от характерного размера частиц микроструктуры. На боковых сторонах блока поставлены неотражающие граничные условия в упрощенном варианте.

4.4. Комплекс параллельных программ для расчета трехмерных динамических задач моментной теории упругости (3Dyn_Cosserat)

Разработан комплекс программ 3Dyn_Cosserat, предназначенный для численного решения пространственных динамических задач моментной теории упругости Коссера. Допускается задание блочной расчетной области, составленной из разнородных материалов с произвольным числом слоев, полос в слое и блоков в полосе с согласованными между собой криволинейными границами раздела. Уравнения модели записываются в виде гиперболической по Фридрихсу системы уравнений (1) в терминах векторов скоростей поступательного и вращательного движения, а также тензоров напряжений и моментных напряжений. Начальные данные краевой задачи формулируются в терминах смещений, углов поворота, скоростей поступательного и вращательного движения, напряжений и моментных напряжений. На межблочных границах ставятся условия непрерывности векторов скорости и угловой скорости, векторов напряжения и моментного напряжения. На внешних границах могут быть заданы граничные условия в скоростях поступательного и вращательного движений, в напряжениях и моментных напряжениях, смешанные граничные условия или условия симметрии.

На рис. 5 приведены численные результаты для трехмерной задачи Лэмба, выполненные на сетке из $400 \times 400 \times 400$ узлов на 64 процессорах кластера МВС–100k МСЦ РАН. Поверхности уровня нормального напряжения изображают фронты продольной, поперечной, конической и поверхностной пространственных волн.



а б

Рис. 5. Результаты численного решения трехмерной задачи Лэмба для моментной среды (линии уровня напряжения σ_{11})

Существенное отличие результатов расчетов по моментной теории (рис. 4 и 5) от безмоментной теории (рис. 2 и 3) состоит в том, что в моментной среде за фронтом поперечной волны появляется дополнительная система высокочастотных волн, обусловленных вращательным движением частиц.

5. Заключение

Разработанные алгоритмы и комплексы программ для численного решения задач динамики структурно неоднородных деформируемых сред ориентированы на применение при изучении процессов распространения сейсмических волн в блочных массивах с криволинейными поверхностями раздела.

Литература

1. Cosserat E., Cosserat F. Theorie des Corps Deformables // Chwolson's Traité Physique: 2nd ed. Paris, 1909. P. 953–1173.
2. Садовская О.В., Садовский В.М. Математическое моделирование в задачах механики сыпучих сред. М.: Физматлит, 2008. 368 с.
3. Марчук Г.И. Методы расщепления. М.: Наука, 1988. 263 с.
4. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. М.: Физматлит, 2001. 607 с.
5. Корнеев В.Д. Параллельное программирование в MPI. Новосибирск: Изд-во ИВМиМГ СО РАН, 2002. 215 с.
6. Антонов А.С. Параллельное программирование с использованием технологии MPI. М.: Изд-во МГУ, 2004. 71 с.

Методы и языковые средства описания взаимосвязанных задач в распределенных пакетах прикладных программ

И.А. Сидоров, Е.И. Поздняк

Учреждение Российской академии наук
Институт динамики систем и теории управления Сибирского отделения РАН

В работе рассматриваются различные подходы к описанию схем решения задач, допускающих декомпозицию составных частей общей задачи на отдельные блоки и последующее решение набора полученных подзадач в распределенной вычислительной среде. Обсуждаются преимущества и недостатки каждого подхода, приводятся примеры их реализации в инструментальном комплексе DISCOMP.

1. Введение

Интенсивное развитие сетевых технологий и аппаратных средств, наблюдаемое в последние годы, позволило многократно повысить производительность вычислительных систем и обеспечило возможность организации эффективных параллельных вычислений. Качественные требования к реализации процесса параллельной обработки данных в прикладных программах (например, необходимость обеспечения эффективности, масштабируемости, переносимости и т.д.) породили большое многообразие систем для организации параллельных вычислений (см., например, работы В.Э. Малышкина, В.С. Бурцева, В.А. Васенина, В.В. Воеводина, А.Б. Жижченко, В.Н. Коваленко, В.В. Корнеева, Д.А. Корягина, А.О. Лациса, В.В. Топоркова и др.). Такие системы, как правило, требуют от специалиста-предметника достаточно высокого уровня программистской квалификации и навыков разработки параллельных программ (в их числе проблемы выявления внутреннего параллелизма алгоритма, необходимость следования тем или иным технологиям и моделям параллельного программирования, учета архитектуры и особенностей используемой вычислительной системы).

Однако существует широкий класс ресурсоемких задач, для решения которых не требуется существенной модификации реализующих алгоритмов и их адаптации к применению на высокопроизводительных вычислительных системах. Такие задачи могут характеризоваться необходимостью проведения многовариантных расчетов над полем независимых между собой входных данных различными программами для их обработки. Одним из инструментов организации такого рода вычислений являются пакеты прикладных программ (ППП), в составе которых выделяют три компонента: функциональное наполнение, высокоуровневые языковые средства описания исследуемой предметной области и системное программное обеспечение (ПО) для организации процесса решения исследовательской задачи. Сочетание в ППП разнообразных сложных моделей, алгоритмов и методик их исследования базируется на использовании принципа модульной организации функционального наполнения пакета. Использование принципа модульности позволяет заменить написание программы (в традиционном понимании) конструированием ее схемы на основе готовых программных блоков крупного размера.

Создание инструментальных средств, обеспечивающих разработку и применение ППП для параллельных и распределенных вычислительных систем (РВС), является одним из наиболее перспективных и актуальных направлений развития пакетной проблематики. Большинство инструментальных систем (например, OLYMPUS, ПРИЗ, САФРА, СПОРА и др.) разрабатывались, в основном, для создания традиционных ППП. Применение таких инструментариев для построения ППП, функционирующих в распределенных средах, является затруднительным. Известные системы организации распределенных вычислений (например, кластерная система Condor, программный комплекс BOINC, инструментарий X-COM и др.) позволяют осуществить в РВС решение не связанных между собой задач, допускающих распараллеливание по данным, но не обладают необходимыми возможностями для организации многофункциональ-

ных пакетов, поддерживающих выполнение взаимосвязанных прикладных программ.

В данной работе рассматривается инструментальный комплекс (ИК) DISCOMP [1], ориентированный на автоматизацию разработки и применения распределенных пакетов прикладных программ (РППП) в разнородных РВС. РППП ориентированы на класс задач, характеризующихся следующими свойствами:

- решение задачи требует проведения расчетов на ЭВМ с использованием больших объемов вычислительных ресурсов (процессорного времени, оперативной памяти, дискового пространства и др.);
- возможна декомпозиция общей сложной задачи на более простые (с вычислительной точки зрения) подзадачи;
- процесс решения общей задачи подразумевает распределенное решение набора ее взаимосвязанных подзадач;
- не предполагается интенсивного взаимодействия между параллельными процессами решения подзадач;
- задача допускает декомпозицию данных на блоки и независимую параллельную обработку этих блоков.

Ниже приведены характерные особенности среды функционирования, языковых средств, функционального наполнения и системной части РППП.

Средой функционирования РППП является РВС с разнородными вычислительными узлами, организованными на основе различных программно-аппаратных платформ и управляемыми разными операционными системами (ОС). Зачастую разнородные РВС имеют низкую степень отказоустойчивости.

Системная часть включает ряд распределенных подсистем и предоставляет средства для организации вычислений на основе удаленного запуска модулей и распределенного обмена данными. Все компоненты системной части являются платформо-независимыми и могут функционировать под управлением различных ОС (например, MS Windows, Linux, Mac OS X и др.).

Функциональное наполнение РППП составляют модули, представленные в виде исполняемых в пакетном режиме программ, реализованных на различных языках программирования (например, C, Fortran, Pascal). Модули размещаются в разных узлах РВС, в каждом узле РВС может быть установлено несколько модулей. Допустимо включение в состав функционального наполнения унаследованного ПО, а также нетиражируемых программных комплексов, жестко привязанных к узлам РВС. Обмен данными между модулями осуществляется через файлы.

Языковые средства РППП предоставляют возможности для описания концептуальной модели предметной области, ее объектов и основных свойств, задания начальных данных и пр. Главным назначением языковых средств РППП является предоставление разработчику пакета различных способов для формирования предметно-ориентированных параллельных схем решения задач в разнородных РВС. В общем случае, благодаря гибкости и универсальности языковых средств РППП, становится возможным расширить класс допустимых задач, более эффективно использовать ресурсы РВС и повысить надежность вычислений.

Данная работа посвящена подробному обсуждению различных подходов к описанию параллельных схем решения задач в ИК DISCOMP.

2. Модель распределенных вычислений

Формализованное описание предметной области РППП относится к классу вычислительных моделей [2] и представляет собой совокупность значимых параметров предметной области, а также модулей РППП, реализующих вычислительные операции с этими параметрами. В простейшем случае описание предметной области [3] РППП можно определить в виде структуры $S =< Z, T, M, Y >$, где Z – множество параметров, T – множество допустимых типов параметров, M – множество модулей, Y – множество вычислительных единиц РВС, которые способны выполнить тот или иной модуль из M. Связи между элементами множеств *Z, T, M* и *Y* заданы отношениями $ZT \subset Z \times T$, $IN \subset Z \times M$, $OUT \subset Z \times M$ и $MY \subset M \times Y$ (в общем случае типа «многие-ко-многим»). Для каждого объекта структуры *S* (параметра, типа параметра, модуля и вычислительной единицы) определен набор его атрибутов.

Множество *типов параметров T* включает: тип *file*, используемый для описания параметров неопределенной структуры (блоков произвольного текста большого размера); тип *filelist*, предназначенный для поддержки распараллеливания по данным (параллельный список параметров типа *file*). К основным атрибутам *параметра* относятся: имя параметра; тип параметра; ограничения на параметр (максимальное количество элементов, лимит предоставляемого дискового пространства). К основным атрибутам *модуля* относятся: имя модуля; тип; список входных параметров; список выходных параметров; команда запуска модуля (способ запуска модуля, аргументы командной строки); требования к среде выполнения (архитектура процессора, объем оперативной памяти, тип операционной системы, зависимые системные библиотеки); ограничения при выполнении (максимальное время выполнения, объем предоставляемого дискового пространства и оперативной памяти). К основным атрибутам *вычислительной единицы* относятся: имя; список установленных модулей; характеристики операционной системы, процессора, памяти, дискового пространства, и др.

Содержательно модуль $m_i \in M$ реализует возможность вычисления множества параметров $OUT^i \subset Z$ по множеству параметров $IN^i \subset Z$, а множества IN^i и OUT^i называются соответственно множествами входных и выходных параметров для модуля m_i. Поэлементная обработка параметра z_j типа *filelist* модулем m_i выполняется следующим образом: *k*-й элемент параметра z_{jk} обрабатывается *k*-м экземпляром модуля m_{ik}.

Постановка задачи для структуры *S* задается в процедурном виде и в общем случае формулируется следующим образом: «зная *S* выполнить *Q*», где $Q \subset M$ представляет собой частично упорядоченную последовательность модулей из *M*, которые необходимо выполнить для решения задачи.

Схемой решения задачи (СРЗ) в ИК DISCOMP называется модель крупноблочной программы, отражающей информационно-логическую структуру вычислений в терминах предметной области. Таким образом, вычислительный процесс (процесс интерпретации схемы решения задачи) предполагает выполнения ряда зависимых между собой подзадач (модулей).

Далее в работе представляются различные подходы к формированию СРЗ в порядке их реализации в ИК DISCOMP.

3. Ярусно-параллельная форма

Ярусно-параллельная форма является одним из самых естественных и простых подходов к описанию распараллеливания некоторой комплексной задачи [4]. Схему решения задачи, задаваемую в ярусно-параллельной форме, графически можно представить в виде направленного ациклического графа, включающего два непересекающихся множества вершин (рис. 1): множества вершин-параметров и вершин-модулей. Вершины-параметры, как и вершины-модули, являются попарно несмежными. На рис. 1 модули представлены овалами, параметры – кружками.

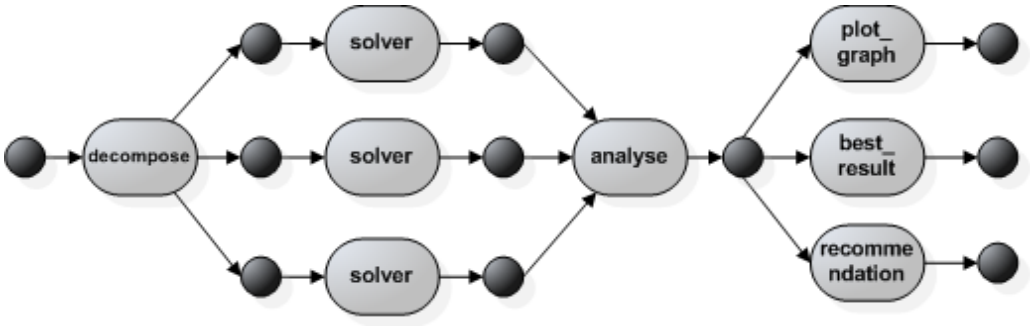


Рис 1. Ярусно-параллельная форма СРЗ

В терминах представленной выше модели при установлении частичного порядка множество *Q* разбивается на *k* непустых подмножеств. Упорядочение подмножеств осуществляется в зависимости от того, какие модули должны быть выполнены раньше. В рамках каждого *k*-го подмножества входящие в него модули могут выполняться независимо друг от друга в любой

последовательности или параллельно.

Описание частично упорядоченной последовательности в ИК DISCOMP осуществляется на специализированном языке, разработанном на основе расширяемого метаязыка разметки XML. Спецификация постановки задачи включает список ярусов (*stage*), на каждом из которых размещаются определенные модули (*module*). Модуль, предназначенный для поэлементной обработки параметра-списка, задается в СРЗ как модуль-список (*listmodule*).

Ниже приводится пример СРЗ, соответствующей приведенному на рис. 1 графу. На первом ярусе должен быть выполнен модуль *decompose*, формирующий элементы параметра типа *filelist*. На втором ярусе производится поэлементная (параллельная) обработка полученного параметра-списка экземплярами модуля *solver*. На третьем ярусе запускается модуль *analyse*, который анализирует полученные на предыдущем ярусе значения параметра-списка. И на четвертом ярусе одновременно (параллельно) могут быть запущены модули *plot_graph*, *best_result* и *recommendation*.

```
<scheme>
      <stage>
            <module name='decompose'/>
      </stage>
      <stage>
            <listmodule name='solver'/>
      </stage>
      <stage>
            <module name='analyse'/>
      </stage>
      <stage>
            <module name='plot_graph'/>
            <module name='best_result'/>
            <module name='recommendation'/>
      </stage>
</scheme>
```

Интерпретация СРЗ, заданной в таком виде, выполняется в соответствии с принципами ветвления и слияния FORK/JOIN [5]. Переход FORK моделирует «разветвление» – создание из одной ветви выполнения двух или более параллельных ветвей. Это, как правило, реализуется путем создания дополнительных ветвей вдобавок к существующим. Переход JOIN, в свою очередь, осуществляет «слияние» нескольких ветвей по завершению их работы (уничтожение созданных параллельных ветвей за ненадобностью).

3.1. Событийно-управляемые конструкции

Задание схемы решения задачи в виде типовой ярусно-параллельной формы накладывает ряд ограничений значительно сужающих допустимый класс решаемых задач. К числу наиболее существенных недостатков следует отнести отсутствие средств для описания логических операторов, необходимых для отсечения ненужных веток в процессе вычислений.

С целью повышения степени интеллектуализации средств управления вычислительным процессом в спецификацию схемы решения задачи заложена возможность включения управляющих конструкций для анализа текущего состояния вычислительного процесса и принятия решений о дальнейшем ходе вычислений. Применение таких конструкций основывается на системе событий, происходящих при интерпретации СРЗ. К событиям относятся:

- запуск модуля (или яруса);
- завершение работы модуля (или яруса);
- некорректное завершение модуля (или яруса);
- вынужденная остановка модуля (или яруса);
- периодическая проверка результатов вычислений для модуля (или яруса), выполняемая с определенным интервалом.

Для каждого структурного элемента СРЗ (модуля или яруса) при необходимости указывается тип обрабатываемого события и непосредственно сам обработчик, реализованный в виде

функции, содержащей блок операторов на высокоуровневом языке программирования *JavaScript*. Для взаимодействия с интерпретатором СРЗ используется специализированный программный интерфейс *DiscompAPI*, средства которого позволяют получать/изменять значения параметров, останавливать/запускать требуемый модуль (все модули на ярусе), и в некоторых случаях передавать управление на необходимый ярус.

Ниже приведен пример описания условия остановки вычислений при обработке параметра типа *filelist*. После завершения каждого экземпляра модуля *solver* запускается обработчик *checkListResult*, который выполняет анализ выходного элемента списка. Если значение выходного элемента содержит подстроку *«RESULT FOUND»*, то производится остановка всех модулей на текущем ярусе и процесс вычислений завершается. Применение данной конструкции позволяет избежать избыточных вычислений при решении комбинаторных задач большой размерности, основным назначением которых является поиск результата, удовлетворяющего определенным условиям.

```
<scheme>
      <stage>
            <listmodule name='solver'
                        onFinish='checkListResult($element_num)'/>
      </stage>
</scheme>
<control><![CDATA[
      function checkListResult (element_num) {
            // получить значение выходного элемента параметра-
            // списка result с порядковым номером element_num
            var res = DiscompAPI.getLPV('result', element_num);
            // если значение элемента содержит подстроку
            // "RESULT FOUND", то остановить вычисления на текущем
            // ярусе и перейти к следующему
            if ( res.match(/RESULT FOUND/) ) {
                  DiscompAPI.stopStageModules ();
            }
      }
]]></control>
```

Следующий пример иллюстрирует способ обработки события *onFinish* при параллельном запуске модулей, реализующих различные алгоритмы решения одной и той же задачи и идентичных по формату входных и выходных параметров. Применение подобной конструкции целесообразно, если заведомо неизвестно, какой из алгоритмов окажется наиболее эффективным для обработки заданных входных значений параметров.

```
<stage>
      <module name='m1' onFinish='DiscompAPI.stopStageModules();'/>
      <module name='m2' onFinish='DiscompAPI.stopStageModules();'/>
      <module name='m3' onFinish='DiscompAPI.stopStageModules();'/>
</stage>
```

Описанные в данном разделе языковые средства ИК DISCOMP успешно применены при решении задач моделирования логистических складских систем, исследования биоресурсов озера Байкал, построения множеств достижимости управляемых летательных объектов, при решении систем булевых уравнений. В качестве отдельного примера стоит выделить РППП D-SAT [6], реализующий технологию крупноблочного распараллеливания SAT-задач.

Однако, несмотря на все описанные усовершенствования, приведенные языковые средства не позволяют описывать сложные недетерминированные схемы, в которых порядок выполнения модулей в цепочке может задаваться неоднозначно. Построение таких схем требует наличия в составе инструментария более мощных языковых средств, предоставляющих возможности включения в алгоритм логических операторов, условных конструкций, операторов цикла, позволяющих динамически выстраивать схему вычислений на основе анализа текущих данных.

4. Компонентно-ориентированный подход

В данном разделе рассматривается подход к описанию схем решения задач на основе компонентно-ориентированной парадигмы, в основе которой лежит объектно-ориентированный подход с определенными ограничениями (в частности, отсутствие механизмов наследования). СРЗ в данном случае описывает взаимодействие независимых объектов через строго определенный набор интерфейсов. Каждый объект обладает определенными свойствами и поведением. Свойства – это элементы внутренней структуры объекта. Поведение – это совокупность действий, выполняемых в качестве реакции на принимаемые объектом сообщения. Реакция на сообщение реализуется в виде метода и представляет собой программу на объектно-ориентированном языке, составленную из посылок сообщений другим объектам, в том числе создаваемым временно в рамках одного метода.

Для работы с основными объектами вычислительной модели был разработан специализированный интерпретируемый язык *DCScript*, являющийся надмножеством объектно-ориентированного скриптового языка программирования *JavaScript*. При описании схемы решения задачи на языке *DCSript* разработчику пакета доступны для использования все возможности объектно-ориентированного языка *JavaScript*, основные объекты вычислительной модели (параметры и модули), а также ряд специализированных средств для работы с ресурсами РВС. Остановимся более подробно на описании двух базовых объектов вычислительной модели: параметров и модулей.

Основными методами параметра являются: получение значения параметра; установка значения параметра; добавление обработчика события. К числу событий относятся: параметр вычислен, изменены размерности для параметра с типом *filelist*.

Основными методами модуля являются: проверка готовности к запуску (все ли входные параметры определены); запуск модуля; остановка модуля; получение информации о состоянии модуля (в очереди, выполняется, завершен, остановлен или аварийно завершен); добавление обработчика события. К числу событий относятся: готовность к запуску, запущен, завершен, остановлен, завершен с ошибкой. Кроме того, в виду асинхронного режима работы интерпретатора, объект «модуль» включает методы синхронизации вычислений (к примеру, ожидание завершения модуля). Более подробно и полно все основные свойства и методы объектов языка *DCScript* рассматриваются в работе [7].

Ниже представлены примеры использования описанных объектов в СРЗ, задаваемой на входном языке *DCScript*.

```
//Для инициализации объектов используются конструкторы:
//DiscompModule, DiscompParameter.
module = new DiscompModule("module_name");

//Доступ к свойствам и методам объекта осуществляется в
//соответствии с  синтаксисом языка Javascript.
value = parameter.getValue();

//Добавление обработчика события для объекта
module.addEventListener("onStop", "moduleStopped()");

//Изменение хода вычислений с помощью условного блока:
if (parameter.getValue() == "1") {
      module1.start();
} else {
      module2.start();
}

//Синхронизация (барьер):
module1.start();
module2.start();
module1.waitForFinished();
module2.waitForFinished();
module3.start();
```

В качестве отдельного ниже приводится реализация метода простых итераций на языке *DCScript*. В данном примере отсутствуют параллельные ветви вычислений в теле цикла в виду особенностей алгоритма. Основное внимание сосредоточено на описании возможностей операторов языка.

```
// переменные программы
var e = 0.0001;  //точность
var t = 1;       //текущая погрешность
var n = 0;       //количество проведенных итераций
var max_n = 100; //максимально допустимое число итераций
var x0 = 0;      //начальное значение x
var param_x = new DiscompParameter("x");//объект для работы с пар.
//модуль, вычисляющий значение функции
var module_func = new DiscompModule ("func");
//выполнять цикл до тех пор, пока не будет достигнута
//заданная точность или превышено число итераций
while ( t > e || n++ > max_n ) {
      //запоминаем начальное значение x
      x0 = param_x.getValue();
      //запускаем модуль и дожидаемся его завершения
      module_func.start();
      module_func.waitForWinished();
      //вычисляем погрешность текущего и предыдущего значения
      t = Math.abs (param_x.getValue() - x0);
};
//печатаем найденный корень
DiscompAPI.logMessage ("Найден корень: " + param_x.getValue() );
//рисуем график
var module_graph = new DiscompModule ("graph");
module_graph.start();
//параллельно запускаем модуль анализа
var module_analyse = new DiscompModule ("analyse");
module_analyse.start();
//ожидание завершения модулей
module_analyse.waitForFinished();
module_graph.waitForFinished();
```

Рассмотренные языковые средства предоставляют возможности для описания логически-сложных схем решения задач с использованием механизмов динамического управления вычислительным процессом. Данные возможности позволили расширить класс задач, решаемых с применением инструментального комплекса DISCOMP. Например, стало возможным описывать алгоритмы итерационных процессов, задач сдваивания, сложных многометодных задач, а также определенных задач, в которых область данных не допускает разбиения на независимые блоки. Приведенные в данном разделе языковые средства ИК DISCOMP успешно применялись при решении задач поиска простых чисел в РВС и при реализации РППП CARMA[8].

Однако, как показывает практика, использование таких средств является для разработчиков пакета весьма затруднительным и требует навыков объектно-ориентированного программирования. Кроме того описание схемы решения задачи на языке *DCScript* существенно усложняет структуру распределенной программы, делает практически невозможным ее графическое представление, усложняет проверку целостности программы и значительно затрудняет ее модификацию.

5. Сети Петри

В предыдущих разделах были показаны два различных подхода к описанию связанных распределенных вычислительных процессов. Подход, основанный на ярусно-параллельной форме, не позволяет описывать логически-сложные схемы вычислений. Компонентно-ориентированный подход имеет значительно более широкие возможности, однако в виду изна-

чальной ориентированности скриптового языка на последовательные вычисления существенно усложняется структура распределенной программы. В настоящее время для ИК DISCOMP разрабатывается новый подход к описанию связанных вычислений в РВС, основанный на аппарате сетей Петри [9].

Сети Петри – инструмент моделирования динамических дискретных систем, являющийся одним из наиболее адекватных способов описания асинхронного выполнения параллельных процессов, в том числе в распределенных системах [4]. Сети Петри работают в терминах условий и событий, где первым сопоставлены позиции (особые узлы – емкости для хранения фишек), а последним – переходы (особые узлы-действия, связанные ориентированными дугами с позициями и перемещающие фишки из входных позиций в выходные).

В нашем случае позициям соответствуют параметры, а переходам – модули предметной области. Наличие фишки в позиции отражает состояние соответствующего параметра (определен или не определен). Условием срабатывания перехода (выполнения модуля) является наличие во всех входных позициях определенного количества фишек, кратного количеству входных дуг, соединенных с данным переходом. При срабатывании перехода из каждой входной позиции изымается, а в каждую выходную позицию добавляется некоторое количество фишек, равное кратности соответствующих дуг, что порождает новую маркировку сети. Если одновременно соблюдаются условия для срабатывания нескольких переходов, то выполниться может любой из них или любая их комбинация. Последнее в свою очередь определяет асинхронную природу сетей Петри.

На рис. 2 проиллюстрирован пример реализации конструкции FORK/JOIN на основе графового представления сети Петри. При срабатывании перехода *Fork* создаются четыре дополнительные ветви, и выполняется перемещение фишки из позиции *p1* в позиции *p21*, *p22*, *p23*, *p24*. После чего соблюдаются условия для срабатывания переходов *Solver*. На рисунке изображена маркировка сети для случая, когда первый, второй и четвертый переходы *Solver* завершили свою работу, а третий переход еще выполняется. Переход *Join* осуществляет барьерную синхронизацию и последующее слияние всех ветвей вычислений в одну.

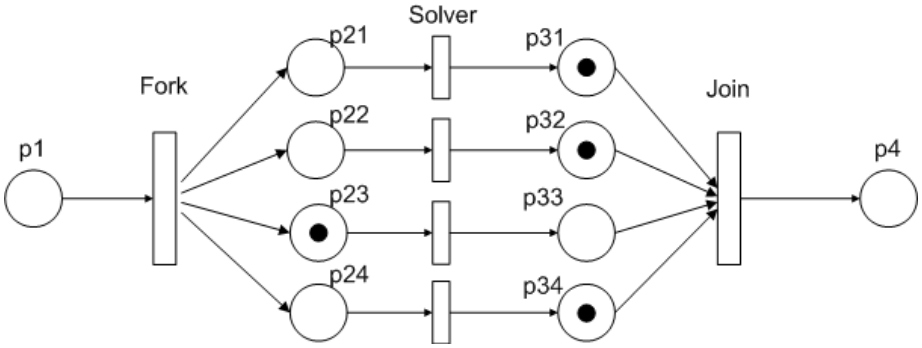


Рис. 2. Пример промежуточной маркировки сети Петри для конструкции FORK/JOIN

Однако стоит отметить, что применение сетей Петри в чистом виде является весьма затруднительным для описания вычислительных процессов и зачастую используются различные расширения в виде раскрашенных и иерархических сетей Петри. Первые позволяют более конкретно специфицировать условия срабатывания переходов, а вторые осуществлять иерархическую композицию или декомпозицию объектов сети. Было доказано, что сети Петри эквивалентны машине Тьюринга и составляют универсальную алгоритмическую систему [10]. Это дает право утверждать, что с использованием этого класса сетей Петри можно описывать достаточно сложные алгоритмы.

На рис. 3 изображена реализация метода простых итераций с использованием аппарата раскрашенных сетей Петри. Позиции n_{max}, n и e являются контролирующими и используются для срабатывания перехода *Check*. Двунаправленные дуги идущие от позиций к переходу *Check*, отражают сохранение фишек в этих позициях. Переход *Check* выполняет проверку количества допустимых итераций и сравнивает модуль разницы параметров x и x_0. В случае если превышено число допустимых итераций или найден корень, удовлетворяющий заданной погрешности, то осуществляется переход по дуге *true*. В противном случае осуществляется переход по дуге *false* и выполняется очередная итерация цикла.

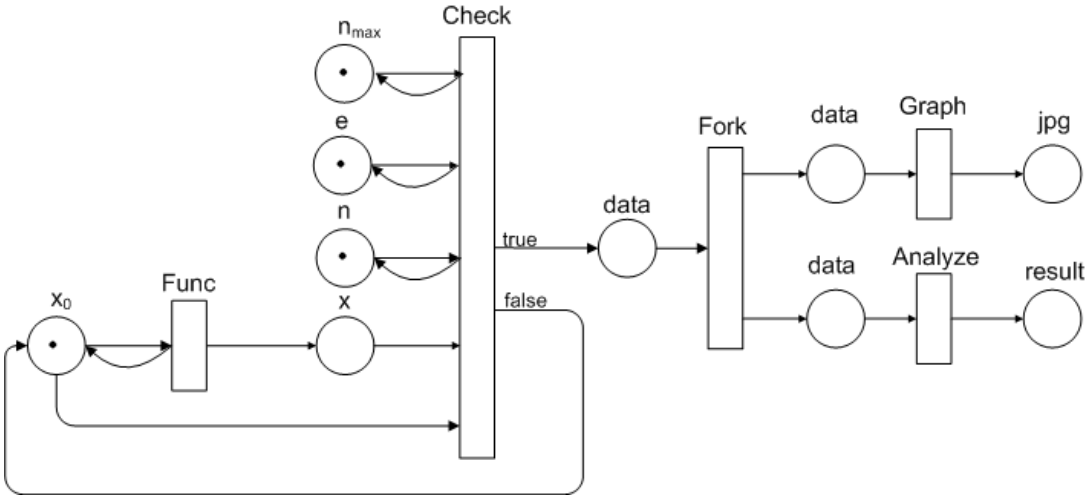


Рис. 3. Пример реализации метода простой итерации с применением раскрашенных сетей Петри

Для описания сети Петри в ИК DISCOMP используется модификация стандарта Petri Net Markup Language (PNML). PNML-описание передается в качестве входного параметра интерпретатору СРЗ, который в совокупности с описанием предметной области проверяет такие свойства как достижимость заданной разметки, живость переходов, ограниченность, безопасность. Для визуализации сети Петри, описанной с помощью PNML, планируется разработать графический редактор, который будет предоставлять возможности редактирования, моделирования и анализа цветных сетей Петри. Кроме того в составе данного графического редактора планируется реализовать средства для интерактивного управления ходом вычислительного процесса.

С использованием описанного в данном разделе подхода планируется реализовать РППП, предназначенный для решения задач бионформатики. В этой предметной области накоплено большое количество разрозненных программных продуктов, которые используются предметными специалистами на различных этапах вычислительного эксперимента. Проектируемые схемы решения задач для таких пакетов требуют развитых языковых средств в составе инструментального комплекса. Аппарат сетей Петри с нашей точки зрения является наиболее приемлемым подходом для описания схем такой сложности.

6. Вычислительный эксперимент

Одним из примеров применения рассмотренных в работе языковых средств является РППП CARMA, предназначенный для таксономической и функциональной классификации коротких метагеномных последовательностей. Пакет позволяет задавать управляющие параметры, влияющие на время вычисления, точность предсказания происхождения того или иного фрагмента последовательности и пр. Полное описание всех реализованных схем решения задач для данного пакета рассматривается в работах [7, 8].

В таблице 1 представлены результаты вычислительного эксперимента для двух наборов, состоящих из 40 и 80 файлов по 100 последовательностей. Вычисления проводились по последовательной схеме на 1 вычислительном ядре, по параллельной синхронной схеме (40 ядер), реализованной на основе параллельно-ярусной формы, и по асинхронной параллельной схеме (40 ядер), реализованной с использованием компонентно-ориентированного языка *DCScript*.

Таблица 1. Результаты вычислительного эксперимента

Кол-во последовательностей в файле x кол-во файлов	**Последовательная схема (1 выч. ядро)** (ЧЧ:ММ:СС)	**Синхронная параллельная схема (40 выч. ядер)** (ЧЧ:ММ:СС)	**Асинхронная параллельная схема (40 выч. ядер)** (ЧЧ:ММ:СС)
100 x 40	53:30:43	4:55:01	3:27:14
100 x 80	97:15:13	5:32:47	4:40:39

7. Заключение

На сегодняшний день с использованием подходов к описанию СРЗ в виде ярусно-параллельной формы и в виде программы на компонентно-ориентированном языке реализовано семь РППП для решения задач из различных предметных областей. Разрабатываемый подход на основе аппарата сетей Петри позволит существенно расширить класс задач, решаемых с использованием ИК DISCOMP, повысить надежность и эффективность вычислений.

Все представленные в данной работе подходы к организации вычислений в РППП допускают естественное развитие и обобщение базовых методов параллельных и распределенных вычислений для решения новых классов фундаментальных и прикладных исследовательских задач.

Литература:

1. Сидоров И.А., Опарин Г.А., Феоктистов А.Г. Разработка и применение распределенных пакетов прикладных программ // Программные продукты и системы, 2010. – № 2. – С. 108-111.
2. Тыугу Э.Х. Концептуальное программирование. – М.: Наука, 1984. – 256 с.
3. Опарин Г.А., Новопашин А.П. Булево моделирование планирования действий в распределенных вычислительных системах // Теория и системы управления, 2004. – № 5. – С.105-108.
4. Федотов И. Е. Некоторые приемы параллельного программирования: Учебное пособие. – М.: Изд-во МГИРЭА(ТУ), 2008. – 188 с.
5. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. – Новосибирск, 2006. – 296 с.
6. Заикин О.С., Семенов А.А., Сидоров И.А., Феоктистов А.Г. Параллельная технология решения SAT-задач с применением пакета прикладных программ D-SAT // Вестник ТГУ. – 2007. – № 23. – С. 83-95.
7. Сидоров И.А. Сластной К.А. Разработка объектно-ориентированных языковых средств распределенного программирования // Винеровские чтения: Материалы IV Всерос. конф. – Иркутск: Изд-во ИрГТУ, 2011. – С. 35-42.
8. Поздняк Е.И., Сидоров И.А., Галачьянц Ю.П. Генерализация алгоритма таксономического классификатора CARMA // Вестник ИрГТУ. – 2011. – № 9. – С. 11-15.
9. Котов В.Е. Сети Петри. – М.: Наука, 1984. – 160 с.
10. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации. – СПб.: Наука, 2000. – 780 с.

Решение нестационарных задач двухфазной гравитирующей среды с применением суперкомпьютеров: проблемы и результаты *

О.П. Стояновская, В.Н. Снытников

Учреждение РАН Институт катализа им. Г.К.Борескова СО РАН

Проведено численное моделирования образования планетезималей и планет - газовых гигантов в околозвездных дисках. Воспроизведен процесс образования коллапсирующих сгустков в гравитирующей среде диска. Гибридная модель эволюции неустойчивого околозвездного диска состоит из уравнений газовой динамики, уравнения Власова для бесстолкновительной компоненты, уравнений для самосогласованного гравитационного поля. Разработанный код позволил рассчитать нелинейные режимы развития неустойчивостей массивного диска. Дальнейшее развитие кода, направленное на расчеты локальных гравитационных коллапсов, связывается с необходимостью значительного увеличения численного разрешения.

1. Введение

Механизм образования в околозвездных дисках планетезималей и планет — газовых гигантов является актуальной проблемой астрофизики.

Гравитационная неустойчивость — механизм, в рамках которого ищется объяснение наиболее загадочным аспектам эволюции планетных систем: формированию газовых гигантов и укрупнению метровых тел до километровых планетезималей. Такой механизм подразумевает образование крупных тел в результате коллапсирования газопылевых сгустков, формирующихся в диске за счет развития гравитационной неустойчивости. Эти коллапсы описываются сингулярными решениями в задачах динамики двухфазной гравитирующей среды.

К настоящему моменту описано несколько режимов формирования сгущений в массивных дисках [1–3]. Однако не получены ответы на ключевые вопросы: Какие условия во вращающейся среде из газа и первичных тел приводят к формированию коллапсирующих сгустков? Возникают ли в околозвездном диске такие условия? Какова будет функция распределения по массам тел, сформированных в коллапсирующих сгустках, в околозвездном диске?

В работе представлены результаты численного моделирования образования планетезималей и планет — газовых гигантов. Воспроизводится процесс образования коллапсирующих сгустков в гравитирующей среде диска (Рис.1). Эта среда состоит из газа и многочисленных метровых тел, которые двигаются с редкими столкновениями между собой на временах одного оборота вокруг протозвезды. Свыше 90 % массы диска принадлежит газовой компоненте.

*Работа выполнена при поддержке Программ Президиума РАН №28 (координаторы ак. Э.М.Галимов и ак. А.Ю.Розанов) и №21 (координатор ак. А.А.Боярчук), Интеграционного проекта СО РАН №130 (координатор ак. Б.Г.Михайленко).

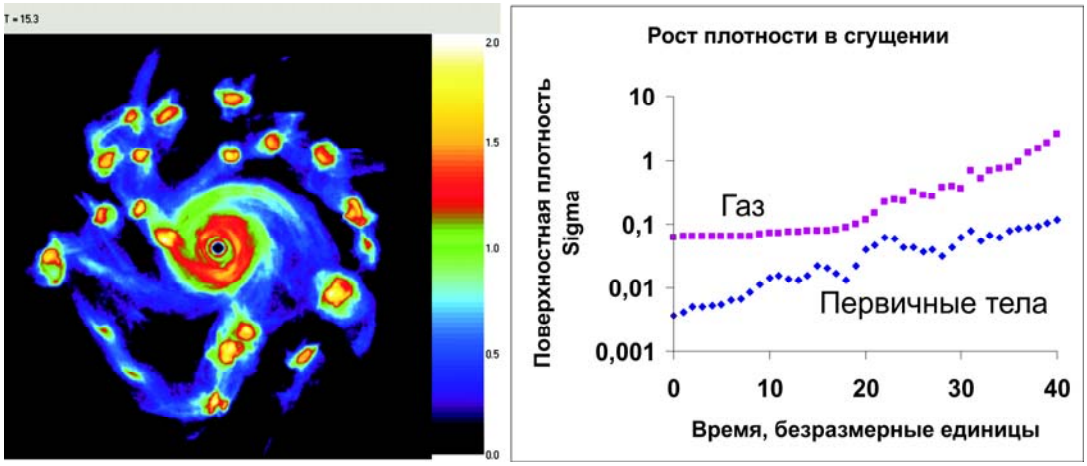


Рис. 1. Логарифм поверхностной плотности субдиска первичных тел в момент времени $T = 15.3$, отвечающий одному обороту периферии диска. Рост максимальной поверхностной плотности газа и субдиска первичных тел в сгущении.

2. Математическая модель субдиска околозвездного диска на этапе образования сгущений

2.1. Основные уравнения

Ввиду того, что толщина диска первичных тел существенно меньше его радиального размера, считается, что «твердая» компонента движется только в экваториальной плоскости системы. При этом в уравнениях газовой динамики используются поверхностные величины:

$$\sigma_{par,gas} = \int_{-\infty}^{+\infty} \rho_{par,gas} dz; \quad p^* = \int_{-\infty}^{+\infty} p dz.$$

$$\frac{\partial \sigma}{\partial t} + div(\sigma \vec{v}) = 0, \quad \sigma \frac{\partial \vec{v}}{\partial t} + \sigma(\vec{v}, \nabla)\vec{v} = -\nabla p^* - \sigma \nabla \Phi,$$

$$\frac{\partial S^*}{\partial t} + (\vec{v}, \nabla) S^* = 0, \quad p^* = T^* \sigma.$$

Здесь $\vec{v}$ — скорость газа, p^* — поверхностное давление газа, γ^* — эффективный показатель политропы для квазитрехмерного случая, связаный с показателем политропы γ соотношением $\gamma^* = 3 - \frac{2}{\gamma}$. $T^* = \frac{p^*}{\sigma}$, $S^* = \ln \frac{T^*}{\sigma^{\gamma^*-1}}$ — производные величины, аналогичные температуре и энтропии газа. Φ — гравитационный потенциал, в котором происходит движение.

Динамику субдиска первичных тел описывает уравнение Власова в пренебрежении столкновениями тел на временах нескольких оборотов:

$$\frac{\partial f}{\partial t} + \vec{u} \frac{\partial f}{\partial \vec{r}} + \vec{a} \frac{\partial f}{\partial \vec{u}} = 0,$$

где $\vec{a} = -\nabla \Phi$, $\vec{a}$ — ускорение частиц во внешнем и самосогласованном поле, $\vec{u}$ — скорость частиц, $f = f(t, \vec{r}, \vec{u})$ — функция распределения частиц по скоростям, связанная с поверхностной плотностью частиц соотношением $\sigma_{par} = \int f d\vec{u} dz$.

Φ — гравитационный потенциал, который представляет собой сумму потенциала неподвижного центрального тела и потенциала диска, $\Phi = \Phi_1 + \Phi_2$, $\Phi_1 = -\frac{M_c}{r}$, M_c — масса центрального тела. Φ_2 — потенциал самосогласованного гравитационного поля, который

определяется как решение смешанной задачи для уравнения Лапласа

$$\Delta\Phi_2 = 0, \quad \Phi_2 \longrightarrow_{r\to\infty} 0, \quad \frac{\partial\Phi_2}{\partial z}|_{z=0} = 2\pi(\sigma_{par} + \sigma_{gas}).$$

Уравнения записаны в безразмерных переменных. Базовыми размерными величинами являются G — гравитационная постоянная, $R_0 = 10AE = 1.5 \cdot 10^{12}$ м, $M_\odot = 2 \cdot 10^{30}$ кг — характерный размер и масса системы.

2.2. Начальные условия

В начальный момент времени задаются поверхностные температура и плотность диска. Плотность газа и субдиска первичных тел взята в виде диска Маклорена массы $M_{par,gas}$ и радиуса R: $\sigma_{par,gas}(r) = \frac{3M_{par,gas}}{2\pi R^2}\sqrt{1-(\frac{r}{R})^2}$. Температура газа в начальный момент времени определяется как $T^*(r) \sim \sigma(r)$ по заданной T_0 — температуре в центре диска.

Начальные скорости тел задаются в виде суммы регулярной и хаотической составляющих $\overrightarrow{u} = \overrightarrow{u'} + \overrightarrow{u''}$, где $\overrightarrow{u'}$ — регулярная, $\overrightarrow{u''}$ — хаотическая скорость. Скорость газа и регулярная скорость частиц определяются из условия равенства центробежной и центростремительной гравитационной сил: $\frac{v_\phi^2}{r} = \frac{1}{\sigma}\frac{\partial p^*}{\partial r} + \frac{\partial\Phi}{\partial r}$, $\frac{u_\phi'^2}{r} = \frac{\partial\Phi}{\partial r}$, $v_r = 0$, $u_r' = 0$. Хаотическая скорость частиц $\overrightarrow{u''}$ задается по гауссову закону с нулевым математическим ожиданием и заданной дисперсией v_d.

3. Численные методы и код Sombrero

Разработанный численный алгоритм решения системы уравнений общего вида основан на методе дробных шагов с расщеплением по физическим процессам. На каждом временном шаге решается уравнение Власова, система уравнений газовой динамики и смешанная задача для уравнения Лапласа.

Расчетная область представляет собой цилиндр, в нижнем сечении которого расположены модельные частицы. Радиус расчетной области в два раза превосходит начальный радиуса диска.

3.1. Решение уравнения Власова

Решение уравнения Власова осуществляется методом частиц в ячейках PIC. Для параллельной реализации метода применяется лагранжева декомпозиция области, поскольку модельные частицы движутся относительно независимо друг от друга, и их движение определяется только гравитационным потенциалом. Реализация требует пересылки плотности (двумерного массива), при этом процессоры взаимодействуют по схеме все-со-всеми. Методы распараллеливания процедур решения уравнения Власова и уравнения Лапласа, которые были реализованы в Sombrero, подробно описаны в работе [4].

3.2. Решение уравнения Лапласа

Для решения уравнения Лапласа используется комбинированный метод с итерациями, где в качестве начального приближения берется значение с предыдущего временного шага. В методе использовано быстрое преобразование Фурье по угловой координате вместе с процедурой блочной последовательной верхней релаксации. Параллельная реализация метода осуществляется через распределение по процессорам гармоник потенциала, полученных в результате дискретного преобразования Фурье. Затем необходима сборка всех гармоник потенциала на каждом процессоре. Это требует пересылки значений потенциала в экваториальной плоскости (двумерный массив), при котором взаимодействие процессоров

происходит по схеме все-со-всеми. Этот метод позволяет распараллелить вычисления на количество процессоров, представляющих собой степень 2.

3.3. Решение уравнений газовой динамики

Система уравнений газовой динамики решается методом SPH, который представляет собой ядерный свободно-лагранжев метод [5]. Сплошная среда заменяется дискретной системой плотно расположенных в пространстве частиц, являющихся носителями основных характеристик среды $m, \mathbf{v}, E$ *etc*. Ключевой особенностью SPH-метода по сравнению с другими методами частиц является способ вычисления пространственных производных без использования сетки. С помощью сглаживающей функции (ядра) строится гладкий интерполянт характеристики среды на основе значений величин, дискретно определяемых в частицах. Операция дифференцирования применяется к интерполянту. Таким образом, динамика частиц определяется только информацией о положении частиц в системе. Расчетные формулы метода SPH, реализованные в Sombrero [6], получаются из записанных в лагранжевом виде уравнений газовой динамики. В качестве ядра W мы использовали кубический сплайн для двумерного пространства. Поверхностная плотность газа, где расположена частица с номером i, вычисляется как интегральный (суммарный) интерполянт $\sigma_i = \sum_{j=1}^{N} m_j W_{ij}$, N — количество модельных SPH-частиц. Уравнение движения аппроксимируется таким образом, чтобы обеспечить сохранение линейного и углового моментов. Для предотвращения нефизического перемешивания со взаимным проникновением SPH-частиц друг сквозь друга в уравнение движения добавляется стандартная искусственная вязкость.

Отличие в распараллеливании метода частиц в ячейках для решения уравнения Власова и метода SPH для решения уравнений газовой динамики состоит в том, что динамика SPH-частицы определяется не только гравитационным потенциалом, но и градиентом давления, который зависит от параметров ее «соседей». Для распараллеливания SPH - кодов применяется как лагранжева, так и эйлерова декомпозия области, в зависимости от того, каким будет движение SPH - частиц и как организуется поиск «соседей» частицы. Так при эйлеровой декомпозиции области легко распараллелить поиск соседей, но трудно обеспечить равномерную загрузку процессоров и минимизировать межпроцессорные коммуникации, поскольку частицы должны быть перемещены с одного процессора на другой. При лагранжевой декомпозиции области, напротив, минимизируется количество межпроцессорных коммуникаций и естественным образом обеспечивается равномерность загрузки процессоров, однако трудности начинает представлять эффективная организация поиска соседей. Кроме этих двух подходов используется «operational-based» подход, который подразумевает хранение всех данных на всех процессорах и распараллеливание только наиболее трудоемких с вычислительной точки зрения процедур [5]. Преимуществами подхода являются простота программной реализации и естественное обеспечение равномерной загрузки процессоров. К недостаткам относятся высокие требования к памяти и эффективность только для небольшого количества процессоров.

В таблице 1 приведены данные о времени выполнения отдельных процедур SPH-подпрограммы для разного количества частиц. Для поиска соседей использовалась одна и та же неподвижная сетка с размером ячейки 0.005. Количество соседей каждой частицы поддерживалось на уровне 50. Видно, что наиболее трудоемкой является процедура вычисления сумм для определения правых частей в уравнениях движения SPH-частиц.

В SPH-подпрограмме кода Sombrero применялся «operational-based» подход с распараллеливанием процедуры вычисления сумм и пересылкой рассчитанных значений массивов. При вычислении ускорений каждый процессор с номером j определял значения массивов только в диапазоне $[NMIN_j, NMAX_j]$, а затем осуществлялась пересылка насчитанных значений массивов по всем остальным процессорам. В силу того, что

Таблица 1. Время счета(сек) на AMD Athlon 2.41 Ггц, 1.87 Гб отдельных процедур SPH-подпрограммы

Количество частиц	2500	10 000	40 000	160 000
Вычисление сумм	0.11	0.41	1.66	6.34
Сортировка	0.26	0.36	0.78	2.5
Всего	0.37	0.77	2.59	8.93

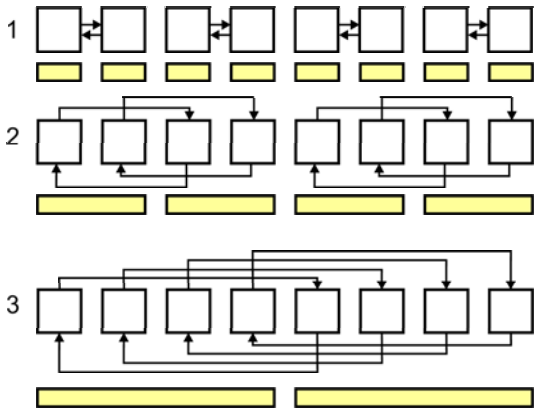


Рис. 2. Редукционная схема пересылки массивов

число используемых процессоров всегда равно 2^k, для пересылки массивов используется редукционная схема, которая приведена на Рис. 2. На первом шаге насчитанными данными обмениваются «соседние» процессоры 0 и 1, 2 и 3, 4 и 5, 6 и 7. В результате обмена в группе процессоров 0 и 1 оказываются одинаковые данные в ячейках массивов $[NMIN_0, NMAX_1]$, аналогично в группе 2 и 3 $[NMIN_2, NMAX_3]$ и так далее. На втором шаге происходит пересылка массивов увеличенной длины, при этом данными обмениваются процессоры 0 и 2, 1 и 3. Пересылка одинаковых значений выполняется параллельно, в результате обмена на всех процессорах 0, 1, 2 и 3 оказываются одинаковые данные в ячейках массивов $[NMIN_0, NMAX_3]$. Аналогичный обмен происходит между процессорами 4 и 6, 5 и 7. На третьем шаге обмен обновленными значениями массивов происходил между процессорами 0 и 4, 1 и 5, 2 и 6, 3 и 7, в результате которого все насчитанные данные оказались переданы на все процессоры. Эта схема позволяет сократить количество коммуникаций каждого процессора с $2^k - 1$ до k.

Параллельные алгоритмы были реализованы с использованием библиотеки MPI. Расчеты проводились на кластере Itanium2 Сибирского суперкомпьютерного центра и на кластере МВС-100 Московского суперкомпьютерного центра.

Максимальная эффективность распараллеливания составляет 0.8 и достигается на 2 процессорах, с увеличением числа процессоров эффективность снижается. Максимальное достигнутое ускорение составляет 3.5 и достигается на 32 процессорах.

3.4. Тестирование кода

В вычислительных экспериментах для контроля правильности решений проверяется выполнение законов сохранения основных физических величин: массы, импульса, полной энергии, момента импульса, а также сохранение центра масс системы. Применимость

реализованных методов для решения интересующего нас класса задач исследовалась в том числе при моделировании динамики осесимметричных и радиально-азимутальных возмущений, распространяющихся в двухфазной среде гравитирующего диска. Путем сравнения результатов вычислительных экспериментов, проведенных с использованием SPH и FLIC методов, показана способность метода SPH воспроизводить нелинейные волны в среде газа и бесстолкновительных тел при возникновении в системе сдвиговых и встречных течений [6].

4. Результаты моделирования

Разработанная версия программы позволила получить нелинейные режимы развития неустойчивостей массивного диска. Нами воспроизведено такое развитие «двухфазной» гравитационной неустойчивости, в результате которой появляются сингулярные решения. Эти решения связаны с коллапсированием сгустков из газа и тел (см. Рис.1). В расчете, представленном на Рис.1, мы воспроизводили динамику диска массы $M = 0.51 M_\odot$ и радиуса $R = 2R_0 = 20AE$, вращающегося вокруг центрального тела массы $M_c = 1 M_\odot$. В начальный момент времени задавалась дисперсия первичных тел по скоростям $v_d = 0.1$, масса частиц составляла $M_{par} = 0.01 M_\odot$, температура газа в центре для эксперимента 1 составляла $T_0 = 0.004$. В этом расчете эффективный показатель адиабаты $\gamma^* = \frac{5}{3}$.

В вычислительных экспериментах мы показали, что самогравитирующие сгущения в двухфазном массивном диске могут быть сформированы в результате развития «двухфазной» неустойчивости, когда на динамику массивного газа оказывает свое влияние маломассивный субдиск первичных тел при их коллективном движении без столкновений друг с другом. На Рис.3 приведены результаты расчетов динамики диска с одними и теми же параметрами массивной компоненты диска - газа, но с изменением параметра субдиска первичных тел v_d. Видно, что при значении $v_d = 0.01$ в диске наблюдается формирование отдельных сгустков, тогда как при $v_d = 0.2$ появляются только спиральные рукава, не разваливающиеся на сгустки.

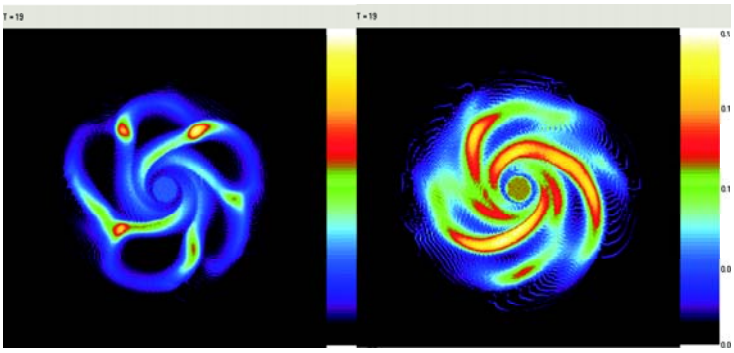


Рис. 3. Поверхностная плотность газа в момент времени $T = 22$. $T = 15$ соответствует полному обороту периферийной части диска. Слева - расчет динамики диска с заданным значением $v_d = 0.01$, справа - с заданным значением $v_d = 0.2$. Расчет на сетке 100*128*100, 5 000 000 PIC частиц, 40 000 SPH частиц, τ=0.001

Из приведенного вывода следует, что возможность формирования сгущений в условиях диска определяется характеристиками как газа, так и твердой фазы. Влияние темпа охлаждения газа и перераспределения его плотности на формирование структур в околозвездных дисках исследуется в ряде работ [1, 3]. Наши результаты показывают, что формирование сгустков может определяться также темпом концентрации крупных (более 1 м) первичных тел в субдиске и темпом снижения дисперсии их скоростей (охлаждения первичных тел).

Для исследования спектра масс крупных тел, получающихся в сгустках, необходимо более чем на порядок увеличить численное разрешение модели. На Рис.4 показано более детальное воспроизведение отдельных стадий развития неустойчивости при увеличении количества модельных частиц и ячеек сетки. Однако дальнейшее увеличение числа модельных частиц и ячеек сетки без изменения алгоритма сталкивается с комплексом ограничений на действующих суперкомпьютерах. Поэтому получение новых физических результатов требует развития кода. Предполагается применение техники многомасштабного моделирования, увеличение количества модельных частиц, а также введение особого типа частиц «sink-particle» при расчете локальных коллапсов в отдельных сгустках вещества.

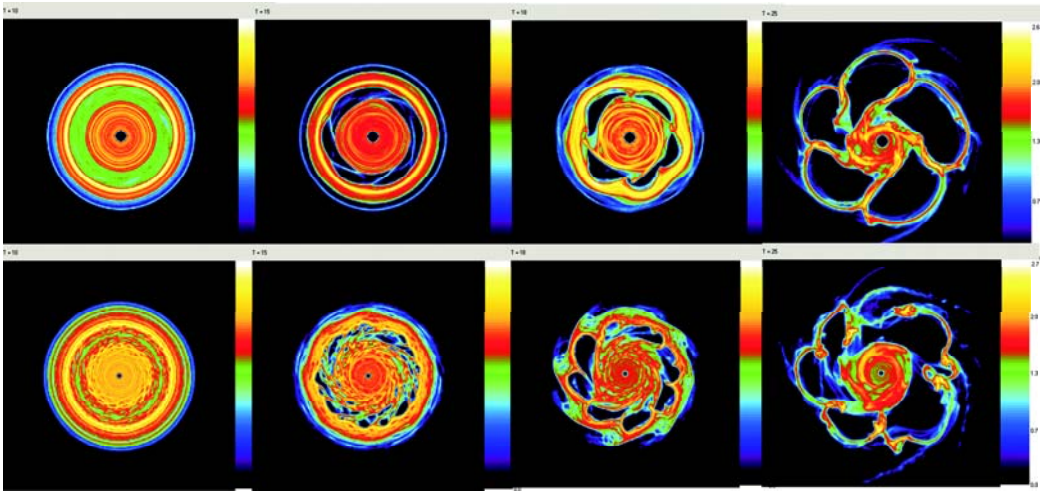


Рис. 4. Логарифм поверхностной плотности субдиска первичных тел в моменты времени $T = 10; 15; 18; 25$. $T = 15$ соответствует полному обороту периферийной части диска. Верхний ряд - расчет на сетке 100*128*100, 5 000 000 PIC частиц, 40 000 SPH частиц, τ=0.001, нижний ряд - расчет на сетке 200*256*200, 10 000 000 PIC частиц, 160 000 SPH частиц, τ=0.0005

Литература

1. Boss A.P. Possible Rapid Gas Giant Planet Formation In The Solar Nebula And Other Protoplanetary Disks // The Astrophys. Journal. 2000. Vol. 536. PL.101–104.
2. Meru F., Bate M.R. Exploring the conditions required to form giant planets via gravitational instability in massive protoplanetary discs // Mon. Not. R. Astron. Soc. 2010. Vol.406, Is.4 P.1060-1072.
3. Rice, W.K.M, Lodato, G., Pringle J.E., Armitage P.J., Bonnell I.A. Planetesimal formation via fragmentation in self-gravitating protoplanetary discs // Mon.Not.R.Astron.Soc. 2006. Vol.372. P.9-13.
4. Вшивков В.А., Кукшева Э.А., Никитин С.А., Снытников А.В., Снытников В.Н. О параллельной реализации численной модели физики гравитирующих систем // Автометрия. 2003. Т.39,№3. С.115-123.
5. Liu G.R., Liu M.B. Smoothed Particle Hydrodynamics, a meshfree particle method. World Scientific Publishing Co.Pte.Ltd., 2007.
6. Стояновская О.П., Снытников В.Н. Особенности SPH-метода решения газодинамических уравнений для моделирования нелинейных волн в двухфазной гравитирующей среде // Математическое моделирование. 2010. Т.22, № 5. С.29-44.

Разработка параллельных алгоритмов для решения задач каротажа на графических процессорах

И.В. Суродина, И.Б. Лабутин

Институт вычислительной математики и математической геофизики,
Институт нефтегазовой геологии и геофизики имени Трофимука

Для численного решения задач каротажа на графических процессорах (GPU) используется метод сопряженных градиентов (CG). В работе предлагается простой способ построения предобуславливателей, аппроксимирующих обратную матрицу. Эффективность данного метода продемонстрирована на задаче бокового каротажного зондирования (решение двумерного уравнения Пуассона). Предложенный метод легко обобщается на случай комплексных систем (задача высокочастотного электромагнитного каротажа) и может быть использован не только в методе CG. Данный подход позволяет улучшить и некоторые предобусловливатели, что продемонстрировано на примере SSOR-AI.

1. Введение.

Численное решение дифференциальных уравнений в частных производных приводит к системам линейных алгебраических уравнений (СЛАУ) высокого порядка. Задача решения таких систем находит широкое применение во многих инженерных и научных исследованиях в самых разных областях. Вследствие этого становятся актуальными быстрые программные реализации алгоритмов, решающие данную задачу. Этой тематике посвящено огромное количество работ. В последние годы очень активно часть вычислений переносят на графические процессоры (GPU), такие как NVIDIA Ge Force, Tesla и другие. Все современные суперкомпьютерные центры развивают гибридные кластеры, в которых для ускорения вычислений применяются и графические карты. Многие алгоритмы, успешно применяемые в ряде последовательно решаемых задачах, становятся неэффективными при решении на GPU. Разработка параллельных алгоритмов для решения многомерных задач остается очень актуальной и необходимой задачей.

В статье рассматривается численное решение уравнения Пуассона. Уравнение Пуассона возникает во многих приложениях: в гидродинамике, в электростатике, в магнитостатике, в геоэлектрике. Численное решение данного уравнения методом конечных разностей или методом конечных элементов приводит к системе линейных алгебраических уравнений с разреженной матрицей большой размерности, поэтому предпочтительнее использовать итерационные методы решения вместо прямых (метод Гаусса или факторизиция Холецкого).

Метод сопряженных градиентов (CG) является одним из лучших хорошо известных методов для решения систем с симметричными, положительно определенными матрицами. Известны его обобщения на случай комплексной области [1].

Использование предобуславливания [2,3] существенно повышает эффективность алгоритма. Предобусловленный метод сопряженных градиентов (PCG) доказал свою эффективность и работоспособность в широкой области приложений. Предобуславливание состоит в замене исходной системы уравнений на эффективно решаемую систему, имеющую то же самое решение.

Предобусловленный метод сопряженных градиентов успешно применяется для плохо обусловленных задач. Такие задачи возникают при электромагнитном каротаже, в частности при моделировании показаний зондов бокового каротажного зондирования (БКЗ). Плохая обусловленность матриц связана во-первых, с сильно неравномерной сеткой, необходимой для адекватного описания прибора (сгущающейся около источников и приемников и имеющей геометрических шаг к границам области) и имеющей сгущения около радиальных границ; во-вторых, с применением биополимерных буровых растворов, имеющих низкое значение сопротивления (0.02 -0.05 Ом м), появился большой контраст между удельным электрическим сопротивлением бурового раствора и пород-коллекторов. Все эти факторы приводит к плохо обу-

словленной задаче с сильно изменяющимися коэффициентами уравнений.

Наша цель – развить для такого рода задач PCG алгоритм на GPU архитектуре. Стандартные техники предобуславливания – LU разложение,неполная факторизация или симметричная последовательная верхняя релаксация (SSOR) остаются весьма трудоемкой для распараллеливания, хотя известны некоторые подходы к этому [4].

Простой предобуславливатель Якоби хорошо распараллеливается, но незначительно влияет на эффективность метода. Применение предобуславливателя Якоби на GPU рассматривается в [5]. Широко известны также полиномиальные предобуславливатели, многоуровневые (многосеточные) и их различные модификации [4,7] и предобуславливатели, связанные с нахождением приближенной обратной матрицы (AINV) [8,9]. Аппроксимация обратной матрицы очень привлекательна и имеет большое будущее для графических процессоров. Применение предобуславливающей матрицы в PCG алгоритме в данном случае сводится к матрично-векторным операциям, достаточно хорошо распараллеливаемым. Но техника построения приближенной обратной матрицы достаточно сложна, трудна в применении и нет никаких гарантий, что построенная матрица будет также симметричной и положительно определенной. В [6] предложен эвристический подход к построению неполного предобуславливателя Пуассона для решения уравнений Пуассона на нескольких GPU, но для нашей задачи этот подход не работает. В [10] рассматривается SSOR-AI предобуславливатель, который уже можно успешно применить к решению каротажных задач.

Мы предлагаем простой и хорошо распараллеливаемый способ получения предобуславливателей, аппроксимирующий обратную матрицу. Предлагаемый метод позволяет получить не только серию новых предобуславливателей, но также улучшить и существующие, в частности SSOR-AI.

На примере двумерной задачи БКЗ (решение уравнения Пуассона) показана высокая эффективность данного подхода для расчетов на GPU.

2. Метод сопряженных градиентов

В задачах, связанных со скважинной геоэлектрикой, удобно использовать цилиндрическую систему координат (r,φ,z). Для простоты рассмотрим изотропную среду с распределением проводимости $\sigma(r,z)$. С целью выделения в явном виде особенности решения задачи, связанной с источником первичного поля, искомый потенциал электрического поля U представим в виде суммы аномального потенциала U^a и первичного потенциала U^0, связанного с источником поля, расположенным в однородной среде с проводимостью σ_0:

$$U = U^0 + U^a.$$

Записывая уравнения непрерывности плотности токов, соответствующих полному и первичному потенциалам и вычитая из первого второе, приходим к следующему уравнению [11]:

$$div(\sigma \nabla U^a) = -div((\sigma - \sigma_0)\nabla U^0). \qquad (1)$$

В цилиндрической системе координат уравнение (1) примет вид

$$\frac{1}{r}\frac{\partial}{\partial r}\left(\sigma r \frac{\partial U^a}{\partial r}\right) + \frac{\partial}{\partial z}\left(\sigma \frac{\partial U^a}{\partial z}\right) = \frac{1}{r}\frac{\partial}{\partial r}\left((\sigma_0 - \sigma) r \frac{\partial U^0}{\partial r}\right) + \frac{\partial}{\partial z}\left((\sigma_0 - \sigma)\frac{\partial U^0}{\partial z}\right). \qquad (2)$$

В зависимости от вида источника U^0 может быть следующим:

1. Точечный источник, находящийся в однородной среде на вертикальной оси в точке z=0

$$U^0 = \frac{I}{4\pi\sigma_0 R},$$

I - сила тока, $R = \sqrt{r^2 + z^2}$.

2. Кольцевой источник постоянного тока с центром в начале координат расположен на диэлектрической трубе радиуса d в однородной среде

$$U^0(r,z,d)=\frac{I}{2\pi^2\sigma_0 R}\int_0^\infty K_0(mr)I_0(md)\cos(mr)dm\,,\quad r\ge d\,,$$

K_0, I_0 - модифицированные функции Бесселя 0-го порядка.

При удалении от источника потенциал затухает как $1/R$, поэтому для функции U^a вдали от источников $U^a\Big|_{r=R}=0$, $U^a|_{z=\pm Z}=0$, $\partial U/\partial r|_{r=0}=0$ или $\partial U/\partial r|_{r=d}=0$.

Дискретизация уравнения (2) конечно-разностным методом и последующая его симметризация приводит к линейной системе

$$Ax=b, \qquad (3)$$

где A действительная, симметричная положительно определенная матрица.
Существует много методов решения подобных систем. Это прямые методы и итерационные. Поскольку матрица A разреженная и имеет большую размерность
выбор делают обычно в пользу итерационных методов.

Метод сопряженных градиентов является одним из лучших итерационных методов для решения разреженных положительно определенных линейных систем линейных алгебраических уравнений. Метод достаточно гибок, прост для применения и сходится теоретически за конечное число шагов. Для его реализации необходимы только матрично-векторные операции.

Алгоритм метода сопряженных градиентов (CG):

$k=0$: Инициализация x_0, $p_0=r_0=b-Ax_0$

$k\ge 0$: Пока $\|r_k\|/\|r_0\|>\varepsilon$

1. $q_k=Ap_k$, $\alpha_k=\dfrac{r_k^T r_k}{p_k^T q_k}$

2. $x_{k+1}=x_k+\alpha_k p_,\quad r_{k+1}=r_k-\alpha_k q_k$

3. $\beta_k=\dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$, $p_{k+1}=r_{k+1}+\beta_k p_k$

На каждой итерации необходимо одно умножение матрицы на вектор и два скалярных произведения векторов. Все операции можно выполнить с использованием стандартных библиотек, например BLAS.

Пусть $(x,y)_A=x^TAy$ и соответствующая A норма $\|x\|_A=\sqrt{x^TAx}$.

Известна оценка скорости сходимости метода CG [15]. Если x^* - точное решение (3), то для последовательности решений $\{x_k\}$ имеем

$$\|x^*-x_k\|_A\le 2\|x^*-x_0\|_A\left(\frac{\sqrt{\mu-1}}{\sqrt{\mu+1}}\right)^k,\quad \text{где } \mu=\lambda_{\max}/\lambda_{\min},\ \lambda_{\max} \text{ и } \lambda_{\min} \text{ - максимальное и мини-}$$

мальное собственные числа матрицы A соответственно. Очевидно, что наилучшая сходимость будет достигаться при cond(A)= $\mu\approx 1$. На практике же $\mu\gg 1$. В рассматриваемом случае из-за сильно неравномерной сетки и большого контраста сред μ может достигать 10^{16}. Эффективность метода CG существенно повышается переобуславливанием системы. Вместо уравнения (3) решается уравнение

$$M^{-1}Ax=M^{-1}b \qquad (4)$$

или

$$AM^{-1}y = b, \quad x = M^{-1}y \quad , \tag{5}$$

где M также симметрична и положительно определена. Уравнение (4) соответствует левому предобуславливанию, уравнение (5) - правому. Матрица M выбирается так, что $\det A \neq 0$, M^{-1} легко вычислима и $M \approx A$. При этом $cond(M^{-1}A) \square\ cond(A)$ или $cond(AM^{-1}) \square\ cond(A)$.

Алгоритм предобусловленного метода сопряженных градиентов (PCG):

$k = 0$: Инициализация x_0, $r_0 = b - Ax_0, Mz_0 = r_0, p_0 = r_0$

$k \geq 0$: Пока $\|r_k\| / \|r_0\| > \varepsilon$

1. $q_k = Ap_k, \quad \alpha_k = \dfrac{z_k^{\ T} z_k}{p_k^{\ T} q_k}$
2. $x_{k+1} = x_k + \alpha_k p_{,} \quad r_{k+1} = r_k - \alpha_k q_k$
3. $Mz_{k+1} = r_{k+1}$
4. $\beta_k = \dfrac{z_{k+1}^{\ T} r_{k+1}}{z_k^{\ T} r_k}, \quad p_{k+1} = r_{k+1} + \beta_k p_k$

В этом алгоритме появляется дополнительный шаг 3, вычисление z_{k+1}.

От его решения зависит эффективность параллельного алгоритма, так как остальные операции –матрично-векторные, которые хорошо распараллеливаются.

Левый предобуславливающий CG алгоритм с M - скалярным произведением математически эквивалентен правому предобуславливающему CG алгоритму с M^{-1} - скалярным произведением.

Для левого предобуславливания (4) обычно используется неполная LU факторизация.

Шаг 3 сводится к последовательному решению двух треугольных систем, что не подходит для GPU –реализации. Можно найти способ, не требующий решения системы с матрицей M, а именно вычислить M^{-1} как аппроксимацию матрицы, обратной к A [8,9,10]. К сожалению, эти подходы сложны в реализации. Аппроксимацию обратной матрицы можно получить достаточно легко и просто, используя алгоритм Хоттенлинга.

3. Алгоритм Хоттенлинга.

Пусть D_0 - некоторое начальное приближение обратной матрицы. Составим произведение исходной матрицы A и D_0. Отклонение этого произведения от единичной матрицы указывает степень неточности полученных результатов:

$$R_0 = E - AD_0 \tag{6}$$

Если $\|R_0\| \leq q < 1$, (7)

где норма может быть взята любая, хорошо вычисляемая, то тогда можно составить итерационный процесс уточнения элементов обратной матрицы A^{-1} со сколь угодно большой точностью. Впервые процесс был предложен Хоттеллингом [12], изложен в [13].

Образуем последовательность матриц

$$\begin{aligned} &D_1 = D_0(E + R_0), \ R_1 = E - AD_1 \\ &D_2 = D_1(E + R_1), \ R_2 = E - AD_2 \\ &\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ &D_m = D_{m-1}(E + R_{m-1}), \ R_m = E - AD_m \end{aligned} \tag{8}$$

Теорема 1. Если начальное приближение обратной матрицы D_0 выбрано так, что $\|R_0\| \le q < 1$, то погрешность $\|D_m - A^{-1}\| \le \|D_0\| \frac{q^{2^m}}{1-q}$.

Доказательство.

Вначале покажем, что матрица $R_m = R_0^{2^m}$. Действительно,

$R_m = E - AD_m = E - AD_{m-1}(E + R_{m-1}) =$

$E - (E - R_{m-1})(E + R_{m-1}) = R_{m-1}^2 = R_{m-2}^4 = \cdots = R_0^{2^m}$

Отсюда следует, что

$$D_m = A^{-1}(E - R_0^{2^m}). \qquad (9)$$

Формула (9) показывает, что D_m стремится к A^{-1}, причем сходимость процесса очень быстрая. Принимая во внимание, что $A^{-1} = D_0(AD_0)^{-1} = D_0(E - R_0)^{-1}$, дадим оценку погрешности:

$$\|D_m - A^{-1}\| = \|-A^{-1}R_0^{2^m}\| = \|-D_0(E-R_0)^{-1}R_0^{2^m}\| \le \|D_0\|\|(E-R_0)^{-1}\|\|R_0^{2^m}\| \le \|D_0\|\frac{q^{2^m}}{1-q} \quad .\square$$

Из этой оценки видно, что как только начальное приближение выбрано так, что выполняется (7), то число верных десятичных знаков возрастает в геометрической прогрессии.

Теорема 2. Если $A = A^T$, $D_0 = D_0^T$, то $D_m = D_m^T$.

Доказательство.

Заметим, что

$$D_m = D_{m-1} = (E + R_{m-1}) = D_{m-1} + D_{m-1}(E - AD_{m-1}) = 2D_{m-1} - D_{m-1}AD_{m-1}$$

Тогда

$$D_m^T = 2D_{m-1}^T - D_{m-1}^T A^T D_{m-1}^T = 2D_{m-1} - D_{m-1}AD_{m-1} = D_m \,.\square$$

Сохранение симметрии оказывается очень хорошим и полезным в дальнейшем свойством этого алгоритма.

4. Построение предобуславливателей.

Применим этот алгоритм для получения предобуславливателей. В качестве D_0 возьмем предобуславливатель Якоби, т.е. $D_0 = diag\{a_{11}^{-1}, a_{22}^{-1}, \ldots, a_{nn}^{-1}\}$. Если $\|R_0\| \le q < 1$, то можем строить процесс. На первом шаге получим $D_1 = D_0 + D_0(E - AD_0)$. D_1 будет иметь такую же структуру, как и исходная матрица. В нашем случае двумерного уравнения Пуассона она пятидиагональная. Решение вспомогательной системы в методе PCG сведется к умножению вектора на матрицу.

Рассмотрим второй член в последовательности (8) приближающих обратных матриц

$$D_2 = D_1 + D_1(E - AD_1) = 2D_1 - D_1AD_1. \qquad (10)$$

Матрица D_2 является 25-диагональной. Однако ее можно выразить через более разреженные матрицы D_1 и R_0^2.

$$D_2 = D_1(E + R_0^2) \qquad (11)$$

Тогда для умножения матрицы D_2 на вектор потребуется одно умножение на $(E + R_0^2)$ - девятидиагональную и одно умножение на D_1 - пятидиагональную матрицы.

Далее, рассмотрим третий член в последовательности приближающих обратных матриц:

$$D_3 = D_2 + D_2(E - AD_2) = (2D_1 - D_1AD_1)(2E - A(2D_1 - D_1AD_1)) = \\ 2(2D_1 - D_1AD_1) - (2D_1 - D_1AD_1)A(2D_1 - D_1AD_1) \quad (12)$$

Этот подход потребует 7 умножений на пятидиагональные матрицы. Можно уменьшить количество умножений, используя выражение D_2 через R_0^2, аналогично предыдущему.

В итоге получим

$$D_3 = D_1(E + R_0^2)(2E - AD_1(E + R_0^2)) = 2D_1(E + R_0^2) - D_1(E + R_0^2)AD_1(E + R_0^2) .$$

(13)

Всего потребуется три умножения на пятидиагональные матрицы и два умножения на девятидиагональную.

Предложенный способ построения предобуславливателей может быть применен к матрице любой структуры, если выполнено условие (7). Сделав хотя бы первый шаг - D_1 - получаем предобуславливатель по структуре соответствующий исходной матрице.

Способность сохранять структуру исходной матрицы является достоинством предобуславливателя D_i, так как применение его в итерационном процессе сводится к операциям над матрицами схожей структуры. Такие операции могут быть эффективно реализованы на GPU.

Выбор подходящего предобуславливателя D_i является по сути компромиссом между точностью приближения обратной матрицы и производительностью (количеством операций над матрицей в итерационном процессе).

5. Предобуславливатель SSOR-AI

В работе [10] R.Helfenstein and J.Koko предложили предобуславливатель SSOR-AI, основанный на аппроксимации обратной матрицы из метода симметричной верхней релаксации (SSOR). Далее изложим принцип его построения.

Пусть $A = L + D + L^T$, D - матрица диагональных элементов A, L -нижняя треугольная часть A. SSOR предобуславливатель определяется как $M = KK^T$, где

$$K = \frac{1}{\sqrt{2-\omega}}(\overline{D} + L)\overline{D}^{-1/2},\ 0 < 2 < \omega \text{ и } \overline{D} = (1/\omega)D .$$

Матрица K может быть представлена следующим образом:

$$K = \frac{1}{\sqrt{2-\omega}}\overline{D}(I + \overline{D}^{-1}L)\overline{D}^{-1/2} \text{ и тогда } K^{-1} = \sqrt{2-\omega}\,\overline{D^{1/2}}(I + \overline{D}^{-1}L)^{-1}\overline{D}^{-1} .$$

Далее, предполагая что спектрадьный радиус $\rho(\overline{D}^{-1}L) < 1$, аппроксимируем K^{-1} рядом Неймана:

$$K^{-1} \approx \sqrt{2-\omega}\,\overline{D}^{-1/2}[I - \overline{D}^{-1}L + (\overline{D}^{-1}L)^2 - (\overline{D}^{-1}L)^3 + \ldots]\overline{D}^{-1}$$

Возьмем только первый член ряда Неймана

$$\overline{K} := \sqrt{2-\omega}\,\overline{D}^{1/2}(I - \overline{D}^{-1}L)\overline{D}^{-1} = \sqrt{2-\omega}\,\overline{D}^{-1/2}(I - L\overline{D}^{-1})$$

SSOR-AI предобуславливатель определим как $\overline{M} = \overline{K}^T\overline{K}$.

Мы предлагаем в качестве D_0 взять $\overline{M}$ в алгоритме Хоттенлинга. Это будет эквивалентно использованию второго, третьего и т.д. членов ряда Неймана.

Какой предобуславливатель использовать в качестве стартового - диагональный Якоби или SSOR-AI – все зависит от конкретной задачи. В SSOR-AI есть возможность управлять релаксационным параметром ω, что может оказаться более успешным в некоторых случаях.

6. Реализация на GPU.

CUDA (Compute Unified Device Architecture) – это программно-аппаратная архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную

производительность благодаря использованию GPU (графических процессоров). Программа в модели CUDA состоит из хост-программы исполняемой на центральном процессоре и ядра (kernel-program) исполняемого параллельно на GPU. Хост программа подготавливает данные и копирует их на GPU. Ядро обрабатывает данные, используя потенциально большое число параллельных потоков. Потоки ядра сгруппированы в массив блоков. Потоки внутри блока имеют доступ к разделяемой локальной памяти и поддерживают барьерную синхронизацию. Потоки из разных блоков не могут быть синхронизованы. Современные NVIDIA GPU состоят из набора мультипроцессоров с разделяемой памятью. Каждый мультипроцессор состоит из 8 скалярных процессоров и 16 kB высокоскоростной памяти. В Cuda Programming Guide[14] описаны приемы достижения максимальной производительности на GPU.

В алгоритме PCG наиболее трудоемкой операцией является умножение матрицы на вектор. В то время как для остальных операций линейной алгебры, необходимых для реализации метода сопряженных градиентов, использовались функции библиотеки cublas, поставляемой вместе с CUDA SDK.

Для хранения матрицы был выбран диагональный формат. Поскольку матрица симметрична, то храним только 3 из 5 диагоналей матрицы в одномерных массивах в глобальной памяти. Для оптимазации работы с памятью видеокарты использовались текстуры .

7. Численные эксперименты.

Решалась задача для пятидиагональной матрицы размером 17139*17139 с числом обусловленности

$$cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{1,44*10^{4}}{1,68*10^{-6}} = 8,57*10^{9}$$

Время решения последовательного варианта с предобуславливателем SSOR составило ~1.5 с на компьютере Intel(R) Core (TM) 2 Quad CPU (при относительной норме невязки 10^{-9}).

В примерах мы ограничились построением D_3, так как дальнейшее построение потребует гораздо большего числа умножений матрицы на вектор.

Таблица 1.

	Iter	Time(sec)
D_0 (Jacobi)	2241	0.65
D_1	1427	0.32
$D_2(D_1)$	925	0.22
$D_2(R_0^2)$	926	0.2
$D_3(D_1)$	714	0.17
$D_3(R_0^2)$	716	0.14

В таблице приведены количество итераций и время решения системы уравнений на GPU NVIDIA GeForce GTX 480 с различными предобуславливателями, вычисленными согласно формулам (8)-(13).

Таблица 2.

	Iter	
SSOR-AI	1522	
D1(SSOR-AI)	1384	
D2(SSOR-AI)	656	

В таблице приведены количество итераций для последовательности предобуславливателей, по-

лученных методом Хоттелинга с начальной матрицей SSOR-AI.

8. Заключение

Предложенный метод построения приближенной обратной матрицы оказался достаточно простым и эффективным для параллельных вычислений. Метод применим для любых матриц - как вещественных так и комплексных и будет достаточно полезным для многих научных и инженерных расчетов.

Данный подход является достаточно гибким, сохраняет симметрию (если матрица и первое приближение симметричны) и разреженность, позволяет строить различные предобусловливатели с учетом конкретной структуры матрицы.

Литература

1. R.W.Freund. Conjugate Gradient type methods for linear systems with complex symmetric coefficient matrices. Numer.Math., 57:285 312, March 1990
2. R.Barrett, M.Berry, T.Chan, J.Demmel, J.Donato, J.Dongarra, V.Eijkhout, R.Rozo, C.Romine, H. van der Vorst. Templates for the solution of linear system: building blocks for iterative methods. www.netlib.org/templates/templates.pdf
3. J. Dongarra, I.Duff, D.,Sorensen, H.van der Vorst. Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, PA, 1998.
4. R.Li, Y.Saad GPU-Accelerated Preconditioned Iterative Linear Solvers Technical Report umsi-2010-112, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2010.
5. S. Georgescu, H. Okuda, Conjugate gradients on graphic hardware: Performance & Feasibility, 2007. 07-129
6. M. Ament, G. Knittel, D. Weiskopf, W. Strasser, A parallel preconditioned conjugate gradient solver for the poisson problem on a multi-gpu platform, PDP ’10: Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Networkbased Processing (Washington, DC, USA), IEEE Computer Society, 2010, pp. 583–592.
7. Z. Feng, Z. Zeng. Parallel Multigrid Preconditioning on Graphics Processing Units (GPUs) for Robust Power Grid Analysis. DAC 2010: 661-666.
8. Jun Zhang. A sparse approximate technique for parallel preconditioning general sparse matrices.
9. G.Meurant A multilevel AINV preconditioner. Numerical Algorithms v 29 n 1-3 (2002) pp 107-129.
10. R.Helfenstein, J.Koko Parallel preconditioned conjugate gradient algorithm on GPU.
11. Ю.А. Дашевский, И.В. Суродина, М.И. Эпов. Квази-трехмерное математическое моделирование диаграмм неосесимметричных зондов постоянного тока в анизотропной среде. СИБЖВМ 2002, т.V,№3(11),с.76-91
12. Hottelling H. Analysis of a complex of statistical variables into principal components, J.Educ.Psych.,1933, **24**, 417-441,498-520.
13. Д.К.Фаддеев, В.Н.Фаддеева. Вычислительные методы линейной алгебры. Физматгиз, Москва-Ленинград, 1963.
14. NVIDIA Corporation, NVIDIA CUDA Programming Guide, NVIDIA, 2011

Повышение эффективности реализации индексного метода решения задач глобальной оптимизации*

А.В. Сысоев, Т.А. Сысоева

Нижегородский госуниверситет им. Н.И. Лобачевского

Рассматривается индексный метод решения задач многомерной многоэкстремальной условной глобальной оптимизации. Предложена новая схема редукции размерности на основе построения семейства множественных отображений с использованием кривых Пеано, позволяющая существенно повысить число используемых процессоров при параллельной реализации индексного метода. Предложена и реализована модификация индесного метода, более точно оценивающая константы Липшица в процессе численного решения задач глобальной оптимизации. Реализована эффективная схема работы с оценками констант Липшица на основе приоритетных очередей.

1. Постановка задачи многомерной глобальной оптимизации

Рассмотрим многомерную задачу глобальной оптимизации:

$$\varphi(y^*) = \min\{\varphi(y): y \in D, g_j(y) \le 0, 1 \le j \le m\},$$
$$D = \{y \in R^N: a_i \le y_i \le b_i, 1 \le i \le N\}.$$

Пусть $y(x)$, $x \in [0,1]$ есть развертка Пеано, однозначно отображающая отрезок $[0,1]$ на единичный N-мерный гиперкуб P, т.е.

$$P = \{y \in R^N: -2^{-1} \le y_i \le 2^{-1}, 1 \le i \le N\} = \{y(x): 0 \le x \le 1\}. \quad (1)$$

Используя отображение $y(x)$, многомерная задача может быть сведена к одномерной

$$\varphi(y(x^*)) = \min\{\varphi(y(x)): x \in [0,1], g_j(y(x)) \le 0, 1 \le j \le m\}. \quad (2)$$

Введя области

$$q_1 = [0,1], q_{j+1} = \{x \in q_j: g_j(y(x)) \le 0\}, 1 \le j \le m,$$

и положив

$$1 \le M = \max\{v(y(x)): x \in [0,1]\} \le m + 1,$$

получим следующее представление задачи (1)-(2):

$$g_M^* = g_M(x_M^*) = \min\{g_M(x): x \in Q_M\} = \min\{g_M(y(x)): x \in q_m\}.$$

При этом испытание в любой точке $x^k \in [0,1]$ позволяет определить пару

$$z^k = g_v\left(y(x^k)\right), v = v\left(y(x^k)\right).$$

Если $\varphi(y)$ удовлетворяет условию Липшица, то $\varphi(y(x))$ удовлетворяет равномерному условию Гельдера

$$|\varphi(y(x_1)) - \varphi(y(x_2))| \le 4L\sqrt{N}(|x_1 - x_2|)^{1/N}. \quad (3)$$

Рассмотренная схема сведения многомерной многоэкстремальной задачи условной оптимизации к эквивалентной ей одномерной задаче позволяет применить для ее решения индексный метод[1,2].

* Работа выполнена в лаборатории «Информационные технологии» ВМК ННГУ при поддержке проекта «Подготовка и переподготовка профильных специалистов на базе центров образования и разработок в сфере информационных технологий», госконтракт № 07.Р20.11.0030, а такжеСовета по грантам Президента Российской Федерации (грант № НШ-64729.2010.9) и РФФИ (гранты №№ 11-01-00682-а, 11-07-97017-р_поволжье_а)

2. Индексный метод решения многомерных задач

Первое испытание осуществляется в произвольной внутренней точке $x^1 \in (a,b)$. Выбор точки x_{k+1}, $k \geq 1$, любого последующего испытания определяется следующими правилами.

Правило1. Перенумеровать точки $x^1,\ldots, x^k$ предшествующих испытаний нижними индексами в порядке увеличения значений координаты, т.е.

$$0 = x_0 < x_1 < \ldots < x_k < x_{k+1} = 1 \quad (4)$$

и сопоставить им значения $z_i = g_v(x_i)$, $v = v(x_i)$, $1 \leq i \leq k$, вычисленные в этих точках; точки $x_0 = 0$ и $x_{k+1} = 1$ введены дополнительно (значения z_0 и z_{k+1} не определены) для удобства последующих обозначений.

Правило 2 . Провести классификацию номеров i , $1 \leq i \leq k$, точек из ряда (4) по числу ограничений задачи, выполняющихся в этих точках, путем построения множеств

$$I_v = \{i: 1 \leq i \leq k, v = v(x_i)\}, \qquad 1 \leq i \leq m+1.$$

содержащихномеравсехточекx_i, $1 \leq i \leq k$, имеющихиндексы, равныеодномуитомужезначениюv. Граничныеточки$x_0 = a$ и$x_{k+1} = b$ интерпретируются как имеющие нулевые индексы, иимсопоставляетсядополнительноемножество$I_0 = \{0, k+1\}$.

Правило3. Вычислить текущие нижние границы

$$\mu_v = max\left\{\frac{|z_i - z_j|}{(x_i - x_j)^{1/N}},\ i,j \in I_v, i > j\right\} \quad (5)$$

для неизвестных констант Липшица L_vфункций g_v, $1 \leq v \leq m+1$. Если множество I_vсодержит менее двух элементов или если μ_vиз (5) оказывается равным нулю, то принять $\mu_v = 1$.

Правило 4 . Для всех непустых множеств I_v, $1 \leq v \leq m+1$, определить величины

$$z_v^* = \begin{cases} min\{z_i: i \in I_v\}, v = v^* \\ 0,\ v < v^*. \end{cases} \quad (6)$$

Т.е. $z_v^* = 0$, если существуют точки x_i, $1 \leq i \leq k$, имеющие индекс, больший v.

Правило 5 . Для каждого интервала (x_{i-1}, x_i) , $1 \leq i \leq k+1$, вычислить *характеристику* $R(i)$, где

$$R(i) = \Delta_i + \frac{(z_i - z_{i-1})^2}{r_v^2 \mu_v^2 \Delta_i} - 2\frac{(z_i + z_{i-1} - 2z_v^*)}{r_v \mu_v}, v = v(x_{i-1}) = v(x_i), \quad (7)$$

$$R(i) = 2\Delta_i - 4\frac{(z_i - z_v^*)}{r_v \mu_v}, v(x_{i-1}) < v(x_i) = v, \quad (8)$$

$$R(i) = 2\Delta_i - 4\frac{(z_{i-1} - z_v^*)}{r_v \mu_v},\ v = v(x_{i-1}) > v(x_i), \quad (9)$$

$$\Delta_i = (x_i - x_{i-1})^{\frac{1}{N}}. \quad (10)$$

Величины $r_v > 1$, $1 \leq v \leq m+1$, являются параметрами алгоритма. Подходящий выбор значений r_vпозволяет использовать произведение r_v и μ_vкак оценку константы Липшица L_v, $1 \leq v \leq m+1$.

Правило 6. Определить интервал (x_{t-1}, x_t), которому соответствует максимальная характеристика

$$R(t) = max\{R(i): 1 \leq i \leq k+1\}. \quad (11)$$

Правило7. Провести очередное испытание в серединной точке интервала (x_{t-1}, x_t), если индексы его концевых точек не совпадают, т.е.

$$x^{k+1} = \frac{x_t + x_{t-1}}{2}, v(x_{t-1}) \neq v(x_t). \quad (12)$$

В противном случае провести испытание в точке

$$x^{k+1} = \frac{x_t + x_{t-1}}{2} - sign(z_t - z_{t-1})\frac{1}{2r_v}\left[\frac{|z_t - z_{t-1}|}{\mu_v}\right]^N, v = v(x_{t-1}) = v(x_t). \quad (13)$$

Описанные правила можно дополнить условием остановки, прекращающим испытания, если

$$x_t - x_{t-1} \leq \varepsilon, \quad (14)$$

где t из (3.4.8) и $\varepsilon > 0$ есть заданная точность.

3. Улучшение работы с оценками констант Липшица в программной реализации индексного метода

В работе [3] высказано предположение относительно реализации индексного метода для многомерных задач, что оценки констант Липшица μ_v являются неубывающими. Это утверждение верно только для одномерного случая.

В многомерном случае оценки констант Липшица могут убывать, поскольку в процессе редукции размерности условие Липшица трансформируется в условие Гельдера, в котором модуль разности $|x_1 - x_2|$входит в степени $1/N$. Нетрудно показать, что при делении интервала (x_i, x_j) новой точкой испытания x^*оценки констант Липшица для двух новых интервалов (x_i, x^*) и (x^*, x_j) могут привести к уменьшению общего максимума по всем интервалам, а, значит, и уменьшить оценку μ_v.

Была выполнена реализация индексного метода, учитывающая возможность того, что оценки констант Липшица μ_v могут уменьшаться в процессе выполнения очередной итерации. В первоначальной версии это потребовало введения схемы пересчета оценок по всем интервала. Можно предположить, что более точное оценивание констант Липшица даст улучшение в характере сходимости, то есть позволит либо уменьшить число итераций метода поиска, либо находить лучшие оценки глобального оптимума.

Проведеныэксперименты на 100 функциях Гришагина для сравненияреализации, построенной на предположении из работы [3], и новой, в которой учтено возможное уменьшение оценок констант Липшица. Результаты сравнения представлены в табл. 1.

Таблица 1.

	Количество итераций		
Значение функционала	Меньше	Такое же	Больше
Лучше	10	-	8
Такое же	42	31	6
Хуже	-	-	3

Как видим, на 18 функциях (первая строка) получено лучшее значение критерия, еще на 42 функциях найдена та же оценка глобального оптимума, что и в исходной версии, но за меньшее число итераций и на 31 функции результат не изменился.

4. Использование очередей для оценок констант Липшица

В индексном методе оценки констант Липшица строятся для каждого функционала задачи. Необходимость пересчета оценок ведет к замедлению выполнения каждой итерации, а значит, и всего индексного метода. Для ускорения работы была применена следующая идея:лучшие оценки констант Липшица запоминаются в очередях. Число очередей равно числу функционалов. При выполнении текущей итерации из соответствующей очереди удаляется оценка для интервала, в котором поставлена точка текущего испытания, вычисляются необходимые новые оценки и добавляются в очереди, если это возможно (оценку можно добавить в очередь, если она больше наименьшей из оценок, уже имеющихся в очереди). Таким образом, полный пересчет оценки константы Липшица по всем интервалам требуется только в случае, когда очередь соответствующего функционала опустела.

Выполнена реализация индексного метода с очередями для оценок констант Липшица. В табл. 2 представлены результаты сравнения времени работы метода с очередями и без очередей на некоторых функциях Гришагина(время в секундах).

Таблица 2.

№ функции	Реализация с очередями для констант Липшица	Реализация без очередей для констант Липшица
0	0,05853	0,1939
66	25,607	32,284

69	20,6606	26,0449
73	23,3715	29,0687
98	0,0716	0,1418

Таким образом, в среднем время работы метода уменьшилось на 30%.

5. Повышение эффективности схемы редукции размерности

Отображения, называемые *кривыми* (*развертками*) *Пеано*, сопоставляют любой липшицевой в гиперкубе P функции$\varphi(y)$одномерную функцию $\varphi(y(x))$, удовлетворяющую на отрезке [0, 1] равномерному условию Гельдера с показателем 1/N, см. (3).

Алгоритм вычисления кривой Пеано с любой заданной точностью подробно описан, например, в [2]. Требуемая точность указывается целым числом M (*номер разбиения*), которое определяет допустимую максимальную погрешность оценки каждой координаты кривой $y(x)$ для любого заданного значения аргумента x, равную $2^{-(M+1)}$. Оценки точек кривой сопоставляют равномерной с шагом 2^{-NM} сетке на отрезке [0, 1] равномерную с шагом 2^{-M} сетку в гиперкубе P.

Редукция многомерных задач к одномерным с помощью разверток сохраняет непрерывность и равномерную ограниченность разностей при ограниченной вариации аргумента, однако теряет часть информации о близости точек в многомерном пространстве, поскольку точка x на отрезке [0, 1] имеет два соседа, тогда как соответствующая ей точка$y(x) \in P$имеет соседей по 2^N направлениям. Возможный способ учета этой информации состоит в следующем [4].

Вводится гиперкуб

$$P_0 = \{v \in R^N : -2^{-1} \le v_i \le 3 \cdot 2^{-1}, 1 \le i \le N\} \quad (15)$$

с длиной ребра, равной 2, и семейство гиперкубов

$$P_l = \{v \in R^N : -2^{-1} \le v_i + 2^{-l} \le 3 \cdot 2^{-1}, 1 \le i \le N\}, 1 \le l \le L, \quad (16)$$

где гиперкуб P_{l+1} получается путем сдвига гиперкуба P_l вдоль главной диагонали на шаг -2^{-l} по каждой координате, и для каждого гиперкуба P_l, $0 \le l \le L$ вводится своя развертка $y_l(x)$ типа кривой Пеано, отображающая отрезок [0, 1] на этот гиперкуб. Приближенное построение развертки $y_l(x)$ для точности, соответствующей разбиению с номером m+ 1 порождает в гиперкубе P_l равномерную сетку с шагом 2^{-m} по каждой координате. При этом на величину L накладывается ограничение $L<m$.

Поскольку гиперкуб P из (1) является общей частью всех гиперкубов семейства (15), (16), т.е.

$$P \cap P_0 \cap P_1 \cap \ldots \cap P_L = P \quad (17)$$

то имеют место включения

$$P \subset \{y_l(x) : x \in [0,1]\}, 0 \le l \le L. \quad (18)$$

Наконец, вводя ограничение

$$p(v) = max\{|v_i| - 2^{-1}, 1 \le i \le N\} \le 0, \quad (19)$$

можно задать гиперкуб P как множество вида

$$P = \{y_l(x), x \in [0,1] : p(y_l(x)) \le 0\}, 0 \le l \le L. \quad (20)$$

Введенные гиперкубы позволяют более полно отражать информацию о близости точек в многомерном пространстве. Если точки $v^1 = y_l(x_l^1) \in P, v^2 = y_l(x_l^2) \in P$ близки в гиперкубе P, но соответствующие им прообразы x_l^1, x_l^2 не являются близкими на отрезке [0, 1], то найдется такое соответствие $y_t(x), 0 \le t \le L$, что прообразы x_l^1, x_l^2, где $y^1 = y_t(x_t^1), y^2 = y_t(x_t^2)$, будут близкими на отрезке [0, 1] (см. [5]). Рассмотренная схема приведения исходной многомерной задачи к семейству одномерных называется *множественной разверткой*.

Множественная развертка позволяет естественным образом построить параллельную модификацию индексного метода. Действительно, достаточно раздать каждому процессу по одной развертке и обмениваться точками испытаний, чтобы последовательность точек на каждом процессе росла практически в p раз быстрее (где p – число процессов), чем в случае использования одного процесса.

Однако параллелизм на основе множественной развертки ограничен условием, которое накладывает приближенное построение каждой отдельной развертки, – L+1<=m. Это означает,

что при точности построения развертки $m = 10$, возможно использование только десяти разверток, а, значит, только десяти процессоров для параллельного поиска глобального оптимума. Это ограничение существенно сужает возможность применения параллельных вычислений. Для его преодоления можно использовать *модификацию исходной схемы*, позволяющую строить *семейство множественных разверток*.

Рассмотрим первоначально двумерный случай. На рис. 1 представлены гиперкубы P_0-P_3 исходной множественной развертки для $L = 3$.

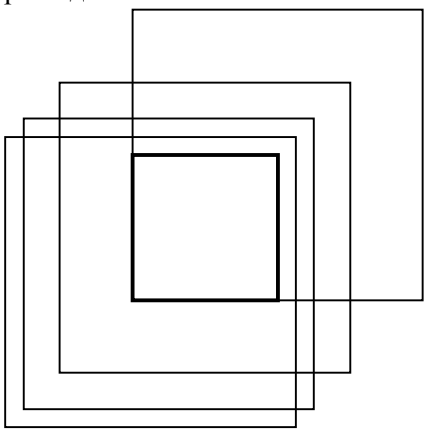


Рис. 1. Гиперкубы исходной множественной развертки ($N = 2$, $L = 3$)

При этом левый нижний угол гиперкуба P_0 совпадает с левым нижним углом гиперкуба P. Построим еще три множественных развертки, в каждой из которых положение базового гиперкуба P_0 выбирается так, чтобы у него с гиперкубом P совпадал один из трех оставшихся углов.

На рис.2 представлена множественная развертка, в которой у базового гиперкуба P_0 и гиперкуба P совпадают правые нижние углы. Нетрудно показать, что все четыре множественных развертки будут обладать одной и только одной общей разверткой, задаваемой в каждой из них гиперкубом

$$P_1 = \{v \in R^N : -1 \le v_i \le 1, 1 \le i \le N\} \quad (21)$$

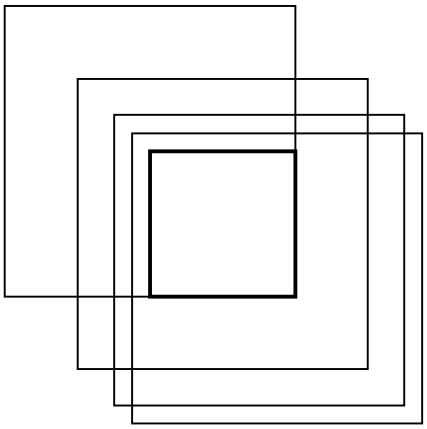


Рис. 2. Гиперкубы модифицированной множественной развертки

Таким образом, общее число разверток построенного семейства будет равно $4 \cdot 4 – 4 + 1 = 9$ для рассмотренного примера.

Дадим общее описание правила построения каждой множественной развертки для произвольного N.

Введем двоичную нумерацию множественных разверток, состоящую из N разрядов:

$$\alpha = (\alpha_0, \dots, \alpha_{N-1}),\ \alpha_i = 0, 1 \quad (22)$$

Пусть исходная множественная развертка имеет двоичный номер, в котором все разряды равны нулю.

Для построения каждой множественной развертки достаточно задать описание базового гиперкуба P_0 и правила сдвига гиперкубов.

Базовый гиперкуб P_0 для множественной развертки с двоичным номером α будет задаваться как

$$P_0 = \{v \in R^N : -2^{-1} - \alpha_i \le v_i \le 3 \cdot 2^{-1} - \alpha_i, 1 \le i \le N\} \quad (23)$$

То есть по тем разрядам, в которых в двоичном номере множественной развертки стоит единица, базовый гиперкуб будет иметь по соответствующим переменным границы $-3 \cdot 2^{-1} .. 2^{-1}$, а по разрядам равным нулю границы по переменным будут $-2^{-1} .. 3 \cdot 2^{-1}$.

Смещение же остальных гиперкубов $P_1, \ldots, P_L$в множественной развертке с номером α будет описываться вектором

$$\beta = (-2^{-l} \cdot (-1)^{\alpha_0}, \ldots, -2^{-l} \cdot (-1)^{\alpha_{N-1}}) \quad (24)$$

То есть по переменным, которым в двоичном номере множественной развертки соответствует единица, сдвиг будет выполняться на 2^{-l}, а по остальным на -2^{-l}.

Модифицированная множественная развертка значительно повышает потенциал использования параллелизма в процессе поиска глобально оптимума. Так уже для трехмерных задач при точности построения приближения развертки $m = 10$, общее число доступных разверток в семействе множественных разверток составит $m \cdot 2^3 - 2^3 + 1 = 73$. В общем же случае число разверток может быть получено согласно следующему правилу.

Утверждение 1. Для задач размерности N при точности приближенного построения кривых Пеано m максимальное число разверток в семействе множественных разверток равно

$$(m - 1) \cdot 2^N + 1. \quad (25)$$

6. Программная реализация модифицированной множественной развертки

Введем «сквозную» нумерацию гиперкубов во всех семействах модифицированной множественной развертки от 0 до $Lmax = (m - 1) \cdot 2^N$. По номеру гиперкуба k можно восстановить номер семейства и номер развертки внутри семейства.

Опишем общую схему использования разверток в процессе выполнения итерации индексного метода.

Пусть точка очередного испытания принадлежит интервалу [k, k+1), который соответствует гиперкубу P_k. По номеру k определяем номер семейства разверток, номер гиперкуба в нем и, следовательно, положение этого гиперкуба в пространстве. С помощью развертки необходимо отобразить точку x в данный гиперкуб P_k.

Для полученной точки проверяется ограничение из (19). Если оно выполняется, то необходимо вычислить значения функционалов, предварительно отобразив точку из гиперкуба P_k в область поиска D.

Проведя вычисления функционалов, необходимо построить все прообразы точки испытания по всем используемым разверткам. Происходит обратный процесс. Сначала точка «переводится» из области поиска D в очередной гиперкуб P_l.

Затем точка из гиперкуба P_l отображается в единичный гиперкуб P.

Далее полученную точку с помощью развертки отображаем на отрезок [l, l+1).

Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. М.: Наука, 1978.
2. Стронгин Р.Г. Поиск глобального оптимума. М.: Знание, 1990.
3. Баркалов К.А. Разработка и исследование методов ускорения сходимости алгоритмов глобальной условной оптимизации // Дисс. на соискание ученой степени канд. физ.-мат. наук. 2006.
4. Strongin R.G., SergeyevYa.D. Global optimization: fractal approach and non-redundant parallelism, Journal of Global Optimization, 2003, 27(1), 25-50.
5. Гергель В.П. Решение одного класса многомерных многоэкстремальных многокритериальных задач со сложными ограничениями // Дисс. на соискание ученой степени канд. техн. наук. 1984.

Параллельный алгоритм моделирования роста дендритных кристаллических структур

Д.И. Халирахманов, С.А. Маякова

Уфимский государственный авиационный технический университет

В данной работе рассмотрена имитационная модель образования дендритов при кристаллизации биологической жидкости. Разработан параллельный алгоритм динамической визуализации и численного расчета процесса кристаллизации для вычислительных систем с общей памятью. Проведены расчеты роста кристаллов при различных начальных условиях.

1. Введение

На сегодняшний день разработка и совершенствование методов исследования биологических жидкостей представляют большой интерес при диагностике и моделировании различных патологических состояний. Данные современных исследований [1 – 6] свидетельствуют о том, что первичные изменения, связанные с действием патогенного фактора на организм возникают, прежде всего, на молекулярном уровне.

В биологических жидкостях происходят постоянные изменения молекулярного состава и характера взаимодействия различных компонентов. Такие изменения являются наиболее информативными при исследовании гомеостаза молекулярного уровня и могут служить основой для диагностики ранних стадий различных заболеваний.

Одним из направлений в разработке новых, доступных для практической лаборатории диагностических методов является исследование структур биологических жидкостей, которые образуются при переходе их в твердое состояние в процессе кристаллизации. Преимуществами таких методов являются простота, информативность и массовость.

В настоящее время используются различные подходы к изучению структур твердой фазы биожидкостей с целью получения диагностической информации, однако основная часть полученных данных остается на уровне описания отдельных феноменов без выявления их ассоциативных связей с теми или иными патологическими отклонениями. Плохо изучена взаимосвязь процесса кристаллизации биологических жидкостей с данными их химического состава и физическими показателями. В большинстве работ, посвященных вопросам роста кристаллов, рассматриваются процессы роста из переохлажденной жидкости, что малоприменимо к жидкостям биологическим.

Математическое моделирование дает возможность построения моделей подобных процессов и структур, а также значительно упрощает процедуру самой диагностики. Проблемой при подобном изучении может стать значительная вычислительная ресурсоемкость, что можно обойти, использую параллелизм при моделировании.

Основной целью данного исследования является разработка параллельных алгоритмов модели роста дендритных структур в биологических жидкостях и тестирование эффективности этих алгоритмов на вычислительных системах с общей памятью.

2. Теоретическая часть

2.1. Математические модели физических процессов, проходящих при кристаллизации

Данная работа основана на развитии модели, описанной в статье Баранова В.Г. и Храмова А.Г. «Моделирование процесса роста дендритных кристаллических структур» [1]. В ней рассматривается кристаллизация небольшого количества раствора заданного вещества с примесями, зажатого между двумя стеклами, что исключает испарение капли. При росте кристалла в таких условиях главную роль играет диффузия. Однако скорость диффузии – величина очень

незначительная. Следовательно, у удаленных молекул мало возможности попасть к поверхности кристалла. При создавшихся условиях кристалл может расти только таким образом, чтобы при минимальной затрате строительного материала продвинуться максимально дальше в ту сторону, где этот материал имеется. Кристалл начинает расти тонкими отростками - дендритами. На практике для диагностики используется схожий метод клиновидной дегидратации – быстрого высушивания образца в течение 2 – 3 часов.

Разобьем область, в которой рассматривается образование кристалла, сеткой на элементарные квадратные ячейки. Одну ячейку этой сетки мы будем считать минимальной неделимой единицей пространства – элементарным объемом. Каждая элементарная ячейка характеризуется своим состоянием: раствором или кристаллом, находящимся в этой ячейке; концентрацией вещества в ячейке; концентрацией примеси в ячейке.

В растворе происходит диффузия вещества и примеси. В каждой точке границы между раствором и кристаллом в каждый момент времени есть вероятности кристаллизации элементарной ячейки около границы роста кристалла за определенный промежуток времени. При реализации этого случайного события происходит перенос вещества и примеси помимо диффузии: недостающая примесь поступает в ячейку из окружающего раствора, избыточное вещество распределяется по соседним ячейкам. Стоит также отметить, что из каждой ячейки количество забираемой примеси пропорционально ее изначальной концентрации. Примесь в уже кристаллизованных ячейках в вышеописанных процессах не участвует.

2.2. Диффузия вещества и примеси

Процессы диффузии вещества и примеси описываются следующим дифференциальным уравнением:

$$\frac{\partial C}{\partial t} = D\left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right), \tag{2.1}$$

где C – концентрация, D – коэффициент диффузии.

Считаются известными начальные условия (распределение концентрации вещества и примеси в начальный момент времени) и граничные условия (через границу раствора и кристалла диффузия не переносит вещество и примесь).

Для решения дифференциальных уравнений была выбрана простейшая пятиточечная явная разностная схема, имеющая следующий вид:

$$C_{ij}^{k+1} = C_{ij}^{k} + D\frac{h_t}{h_x^2}\left(C_{i+1,j}^{k} + C_{i-1,j}^{k} - 4C_{ij}^{k} + C_{i,j+1}^{k} + C_{i,j-1}^{k}\right),\ C_{ij}^{0} = \varphi_{ij}, \Delta C_{ij}^{k} = 0, (i,j) \in \Gamma\ . \tag{2.2}$$

Эта разностная схема имеет первый порядок аппроксимации по пространству h_x и времени h_t и устойчива при выполнении условия $\frac{h_t}{h_x^2} \le \frac{1}{4D}$.

2.3. Вероятности кристаллизации

Кристаллизация элементарных ячеек – это случайные события, происходящие с некоторой вероятностью. Рост грани кристалла рассматривается как пуассоновский поток событий. Событием является кристаллизация элементарной ячейки. Вероятности кристаллизации p элементарной ячейки в зависимости от шага по времени Δt равна:

$$p = 1 - \exp\left(-V\frac{\Delta t}{\Delta x}\right), \tag{2.3}$$

где V – средняя скорость роста грани кристалла; Δt, Δx – соответственно шаги дискретизации по времени и пространству. За один шаг по времени кристаллизуется одна ячейка, выбранная случайно среди ячеек с одинаковой вероятностью кристаллизации, равной максимальной.

Скорость роста грани кристалла зависит, в свою очередь, от концентрации примеси. Концентрация примеси в растворе $C_в$ может меняться от нуля до плотности этого вещества в твердой фазе $\rho_в$. Очевидно, что скорость роста грани равна нулю при нулевой концентрации и стре-

мится к бесконечности при приближении концентрации к максимальной $\rho_в$. Концентрация насыщенного раствора соли C_0 и базовая скорость роста кристалла V_0 являются известными величинами. Окончательные выражения для скорости роста имеют вид:

$$V = V_0 \frac{C_в}{C_0}\left(\frac{\rho_в - C_0}{\rho_в - C_в}\right). \qquad (2.4)$$

2.4. Перенос вещества и примеси при кристаллизации

В момент осуществления события кристаллизации концентрация примеси в ячейке ниже плотности кристалла, и в ней имеется некоторое количество вещества. Таким образом, при изменении статуса ячейки необходимо корректировать в ней концентрации вещества и примеси. При этом в кристаллизующейся ячейке необходимо сохранять общее количество жидкости. Для этого при кристаллизации одной ячейки приходится корректировать состояния соседних ячеек – лишнее вещество вытесняется в них, а недостающая примесь забирается из них.

Пусть кристаллизуется ячейка с концентрацией примеси C. После кристаллизации ее концентрация должна стать равной плотности вещества кристалла ρ. То есть в эту ячейку должна дополнительна поступить примесь в количестве $C_н = \rho - C$ из соседних ячеек. Если примеси в них не хватает, то нужное ее количество добирается из следующей группы соседей. В группу относятся все ячейки из раствора, находящиеся на одинаковом расстоянии от кристаллизуемой ячейки. Количество доступных «донорских» групп ограничено из соображений сохранения физичности процесса. Если же примеси не хватает во всех доступных группах, то кристаллизации не происходит и эта ячейка на следующем временном шаге как потенциальный претендент не рассматривается. При отсутствии изменения количества кристаллизованных ячеек на протяжении заданного числа шагов по времени процесс роста кристалла считается прекращенным.

3. Реализационная часть

3.1. Этапы алгоритма программы

На рис. 1 схематично представлен алгоритм, реализованный в программе. Проанализировав данную схему, можно сделать вывод, что не все ее модули имеет смысл подвергать параллелизации (в частности модуль отрисовки кристаллизованной на данном шаге ячейки и потенциальных претендентов на кристаллизацию или модуль вывода полученных результатов). Модули цикла по времени и модуль переноса вещества также не предрасположены к параллелизму изначально, так как они требуют для своего вычисления значений предыдущих операций, вычисляемых в них же. Цикл эффективно распараллеливается, если отсутствуют перекрестные зависимости между его итерациями. Таким образом, выделена та часть алгоритма, расчет которой возможно ускорить. Далее рассмотрим подробнее все оставшиеся модули, которые будут впоследствии распараллелены (выделены на рисунке заливкой).

Рис. 1. Схема модулей алгоритма программы

В модуле генерации матриц заполняются два массива: массив концентраций примеси в ячейках случайным образом из заданного диапазона и массив состояний ячеек нулями, а также задаются несколько кристаллизованных ячеек в центре для начала роста кристалла. В модуле уравнения диффузии пересчитываются значения в еще не кристаллизованных ячейках согласно разностной схеме и вычисляется максимальная вероятность кристаллизации среди ячеек, у которых есть уже кристаллизованные соседи, что также реализовано с использованием двойного цикла. В эту же часть алгоритма включена реализация условия непротекания на границе рассматриваемой области. В модуле определения кристаллизуемой ячейки снова происходит обход всех ячеек и выявление тех из них, вероятность кристаллизации которых равна максимальной (если таковых несколько, то выби-

рается одна из них случайным образом). На всех этих этапах выполнения программы подавляющее большинство расчетов происходит внутри циклов.

На этапе просмотра соседей происходит определение количества примеси во всех доступных группах соседей-доноров и их общее количество. Уже кристаллизованные ячейки при этом не учитываются. Здесь же принимается решение о возможности кристаллизации на основе наличия или отсутствия необходимого объема примеси, а также происходит планирование перераспределения вещества и примеси.

3.2. Тестирование последовательного алгоритма моделирования

На основе рассмотренного выше алгоритма была написана последовательная программа, способная динамически визуализировать процесс кристаллизации раствора с заданными начальными условиями. Вместе с этим программа выводит рядом с изображением дополнительные данные о самом процессе, такие как фрактальная клеточная размерность, средняя концентрация примеси в некристаллизованных ячейках, время роста кристалла, время работы программы и количество шагов роста. Также предусмотрена опция роста до определенного заданного момента времени и алгоритм отображения не только кристаллизованных ячеек, но и потенциальных претендентов на кристаллизацию.

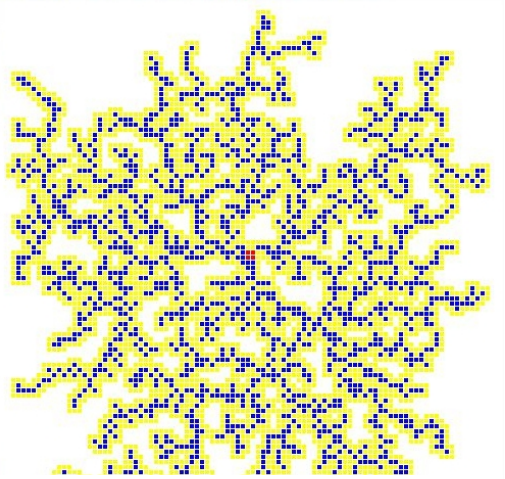


Рис. 2. Пример визуализации растущего кристалла

Проведены эксперименты с разными начальными параметрами, полученные изображения и посчитанная клеточная размерность кристалла говорят об адекватности построенной модели. В результате получены изображения кристалла с изрезанной границей и довольно сложной структурой. Исследована зависимость фрактальной размерности от таких параметров, как коэффициент диффузии, равновесная концентрация, равновесная скорость и максимальная концентрация примеси в ячейке.

Пример работы программы при размере решетки 100×100 ячеек с концентрацией примеси от 0 до 4.7%, коэффициентом диффузии $1.5*10^{-5}$ см^2/сек, базовой скоростью роста 0.01 мм/сек и глубиной забора вещества при кристаллизации в 3 группы ближайших ячеек представлен на рис. 2.

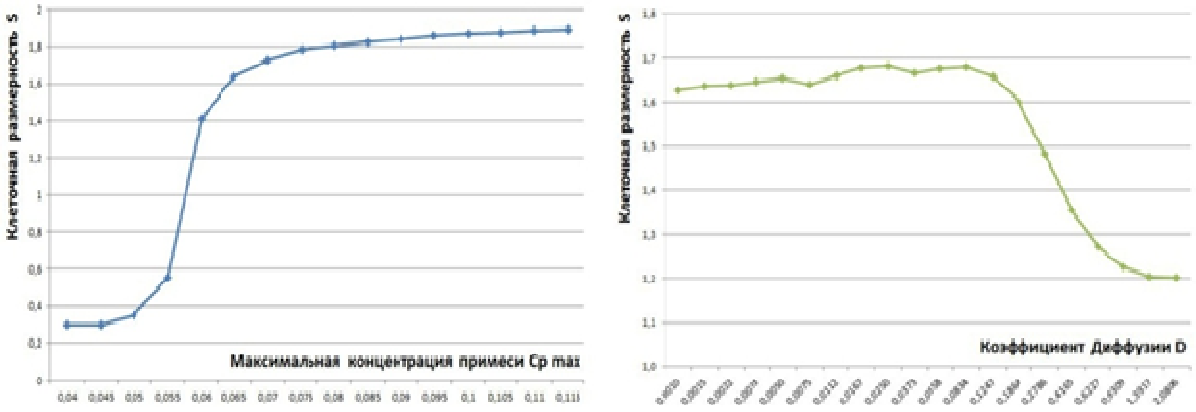


Рис. 3. Графики зависимости клеточной фрактальной размерности (слева – от максимальной концентрации примеси, справа – от коэффициента диффузии)

Данный кристалл имеет клеточную фрактальную размерность ≈ 1.62 и среднюю удельную концентрацию примеси в некристаллизованных ячейках 1.8%. При выборе диапазона начальных концентраций в ячейках от 0 до 3.5% кристаллизация при таких же условиях вообще не наблюдается.

Клеточная фрактальная размерность поначалу сильно зависит от максимальной концентрации примеси в растворе, но при достижении концентрации ≈ 7% рост клеточной размерности замедляется и асимптотически стремится к 2 (рисунок 3, слева). Также можно отметить, что клеточная размерность слабо зависит от базовой скорости кристаллизации V_0 и имеет обратную зависимость от концентрации насыщения C_0. При вариации коэффициента диффузии наблюдается эффект смены преобладания химических процессов к диффузионным, как основных кристаллообразующих сил (рис. 3, справа).

Очевидно, что для получения достоверных результатов требуется брать гораздо более мелкое разбиение сетки. Соответственно растет количество требуемых для проведения эксперимента вычислений. При тестовом запуске последовательной программы на одном ядре процессора расчет сетки 1000×1000 элементов занял 16.7 минут, при расчете сетки размером 5000×5000 время возросло до 509.5 минут. Подобная ресурсоемкость обуславливает необходимость адаптации алгоритма для параллельных вычислений.

3.3. Создание параллельного алгоритма

3.3.1. *Идеология распараллеливания*

Предварительно было произведено профилирование последовательной программы, на основе которого проводился дальнейший анализ и оптимизация алгоритма программы. При использовании явной разностной схемы, которая занимает значительную часть расчетного времени, а также реализованного алгоритма переноса вещества, модель программирования на общей памяти наиболее предпочтительна. Поэтому в ходе разработки параллельной программы было решено использовать библиотеки OpenMP, где все потоки могут иметь доступ к общим массивам данных, таким образом отсутствуют затраты на пересылку.

3.3.2. *Приемы распараллеливания*

Во всех вышеописанных модулях применено распараллеливание циклов путем распределения итераций между потоками с использованием управляемого планирования, так как при статическом планировании нет никакого способа, позволяющего сбалансировать нагрузку на разные потоки. При динамическом и управляемом планировании нагрузка распределяется автоматически, но, как правило, при управляемом планировании код выполняется быстрее, чем при динамическом, вследствие меньших издержек на само планирование. Наибольшая эффективность применения именно такого планирования проверена множественными тестовыми запусками программы при разных размерностях расчетной сетки. Использование подобного приема может обеспечить довольно хорошую масштабируемость алгоритма, особенно при условии основной вычислительной нагрузки внутри циклов.

Участки алгоритма, не входящие в циклы и не зависящие друг от друга, были выделены в области параллельных структурных блоков. При выполнении программы каждой секции в параллельном регионе ставится в соответствие один поток из группы потоков, и все секции выполняются одновременно.

Исключениями из параллельных областей распараллеленных модулей остались участки кода отвечающие за выделение памяти (в модуле генерации матриц), задание начальных кристаллизованных ячеек (также в модуле генерации матриц), случайный выбор одной из уже найденных ячеек с вероятностью кристаллизации, равной максимальной (в модуле кристаллизуемой ячейки). Однако стоит отметить, что время их выполнения в программе много меньше времени выполнения модулей, содержащих эти участки.

Таким образом, были ускорены все расчетные части алгоритма.

4. Экспериментальная часть

В ходе оптимизации последовательной программы для ее дальнейшего распараллеливания было ощутимо снижено как количество операций в отдельных частях алгоритма, так и количе-

ство обращений к памяти. Это позволило не только ускорить программу более чем в два раза, но и упростить задачу выделения участков потенциального параллелизма.

Все запуски последовательной и параллельной версий программы производились на рабочей станции с четырехядерным процессором Intel Xeon (2.00 GHZ), 4 Mb Cache, 4 Gb RAM и операционной системой Windows 7 x64.

Таблица 1. Результаты запуска распараллеленной программы на разном числе ядер процессора

p	Время, с	Ускорение	Эффективность	Процессорное время	Время простоя
1	595,2	1,00	1,00	589,4	5,8
2	435,6	1,37	0,68	805,4	65,8
3	365,7	1,63	0,54	959,3	137,8
4	310,2	1,92	0,48	1130,9	109,9

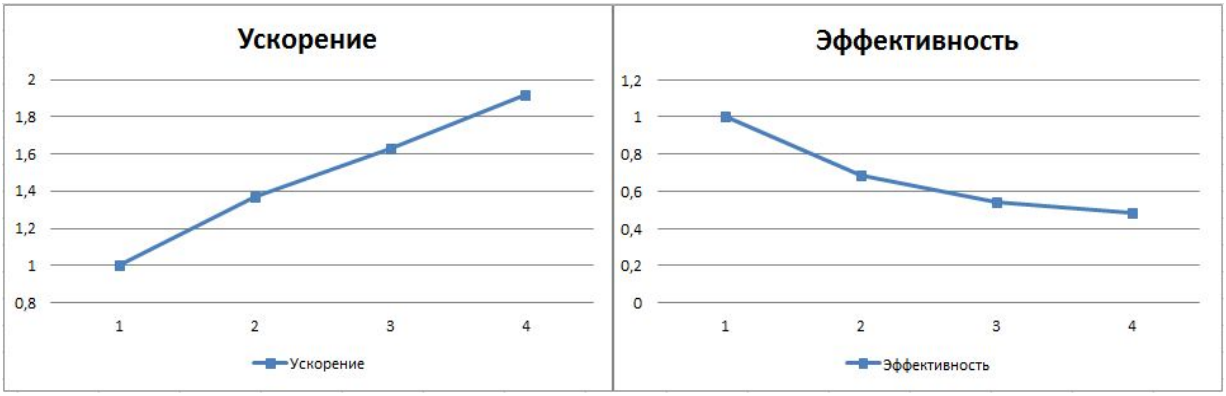


Рис. 4. Оценка распараллеливания алгоритма (слева – полученное ускорение, справа - эффективность)

Результаты тестовых запусков параллельной программы при размерности решетки 1500×1500 ячеек с максимальной концентрацией примеси 5.5%, коэффициентом диффузии $1.5*10^{-5}$ см^2/сек, базовой скоростью роста 0.01 мм/сек и глубиной забора вещества при кристаллизации в 3 группы ближайших ячеек представлены в табл. 1 и на рис. 4.

5. Заключение

В ходе данной работы разработан универсальный алгоритм для решения задачи моделирования и изучения процесса роста дендритных кристаллических структур из растворов биологических жидкостей. На основе данного алгоритма спроектирована программная система с графическим пользовательским интерфейсом, который позволяет редактировать параметры алгоритма и входные данные, запускать этот алгоритм на исполнение, а также визуализировать процесс роста рассматриваемого кристалла. Исследована зависимость клеточной фрактальной размерности растущего кристалла от основных физических параметров раствора. Использование директив OpenMP позволило ощутимо сократить время выполнения программы на многопоточных системах с общей памятью по сравнению с последовательной версией, особенно при больших размерностях расчетной сетки. Представленные результаты вычислительных экспериментов показывают эффективность предложенного алгоритма

В будущем с помощью имитационного моделирования планируется исследовать влияние основных параметров, рассмотренных в работе, на форму кристаллов, а также предполагается установление взаимосвязи между доступными для измерения геометрическими параметрами изображений кристаллов и физическими параметрами раствора, а также расширить функциональность и информативность программы путем расчета дополнительных параметров изучаемого процесса.

Также в дальнейшем планируется использовать GPU с целью проверки эффективности решения подобного класса задач на гибридных системах и получения еще более производительного параллельного алгоритма.

Литература

1. Баранов В.Г., Храмов А.Г. Моделирование процесса роста дендритных кристаллических структур // Компьютерная оптика, Самара-Москва, 2001, № 21, с. 193-197.
2. Шабалин В. Н., Шатохина С. Н. Структурная форма информации в биологических жидкостях // Актуальные проблемы геронтологии.- М., 1999, с. 139-143.
3. Денисов А.Б. Слюнные железы. Слюна. М: РАМН 2003; 132.
4. Шабалин В.Н., Шатохина С.Н. Морфология биологических жидкостей человека. М 2001; с. 303.
5. Егорова Э.В., Шплкин Г. А., Толчипская А.И. и др. Кристаллографический анализ слезной жидкости пациентов с катарактой до и после оперативного вмешательства // Брошевские чтения: Тр. Всерос. конф.- Самара, 2002, с. 530-531.
6. Бузоверя М.Э., Сельченков В.Л., Сельченкова Н.И. и другие. Математический анализ структур твердой фазы биологических жидкостей // Альманах «Геронтология и гериатрия» Вып. 1. - М., 2001, с. 55-60.

Numerical modeling of chemical kinetics with using of supercomputers*

I.G. Chernykh[1], M.S. Antonova[2]

Institute of Computational Mathematics and Mathematical Geophysics SBRAS[1],
Novosibirsk State University[2]

Modeling of detailed chemical kinetics impossible without supercomputers due to the large (more than thousand) size of the chemical reaction system. Modern hybrid (CPU + GPU) supercomputers give possibility to research very large problems efficiently.

1. Introduction

Complex numerical modeling of fluid flows and chemical processes in chemical reactors plays important role in development of new and upgrading of existing chemical processes and chemical reactors [1]

2. Application of CHEMPAK GPU based solvers

There are many solvers for modeling of chemical kinetics tasks was written and a lot of articles with new efforts in supercomputer modeling in chemical engineering journals appear. New solver for CHEMPAK [2] software package was developed by authors. Table 1 shows the tests results of CHEMPAK GPU based solver.

Table 1. Comparison of Intel ODE solver and CHEMPAK GPU solver performance

Solver		Time, s	
		1000 chemical reactions with 500 species synthetic test	Propane conversion test
Intel_ODE [9]	dodesol	2.121614	0.2964019
	Rkm9st	0.9516060	19.12572
CHEMPAK GPU solver		0.15	0.0004

Literature

1. Snytnikov V.N., Mischenko T.I., Snytnikov Vl.N., Chernykh I.G. A reactor for the study of homogeneous processes using laser radiation energy // Chemical Engineering Journal. 2009. Vol. 150. P. 231-236.
2. Chernykh I.G., Stoyanovskaya O.P., Zasypkina O.A.. ChemPAK Software Package as an Environment for Kinetics Scheme Evaluation // Chemical Product and Process Modeling. 2008. Vol. 4 (4,3). P. 1-14.

* This work was partially supported by Federal Program Research and development on priority directions of scientific-technological complex of Russia in 2007-2013" of the Federal Agency for Science and Innovation, Russian Ministry Education and Science. Also this work was partially supported by Integration SB RAS project No. 130.

Метод глобального поиска в задаче оптимального управления со скалярным управляющим воздействием и его реализация на GPU*

А.С. Аникин, А.Ю. Горнов

Институт динамики систем и теории управления СО РАН

Рассматривается задача оптимального управления с системой, линейной по управлению и параллелепипедными ограничениями на управляющее воздействие:

$$\dot{x} = f_1(x,t) + f_2(x,t) \cdot u$$

Необходимо найти минимум терминального функционала $\varphi(x(t_1))$, где $[t_0, t_1]$- интервал определения времени процесса. В такой постановке, при сохранении регулярности условий оптимальности, управления, удовлетворяющие принципу максимума Понтрягина, имеют релейный характер. В докладе предлагается метод поиска глобального экстремума терминального целевого функционала, основанный на вычислении точек переключения оптимального управления.

Предложена параметризация скалярного управляющего воздействия, позволяющая конструировать допустимые управления с заранее заданным числом точек переключения. Первый параметр параметризации включает не только само значение первой точки переключения, но также и указывает на границу – нижнюю либо верхнюю, с которой начинается конструируемое управление. Данный параметр параметризации изменяется на интервале $[t_0, 2 \cdot t_1]$, остальные – на интервале $[t_0, t_1]$. Алгоритм решения задачи оптимального управления включает последовательность невыпуклых задач безусловной минимизации с возрастающим числом переменных, соответствующих числу искомых точек переключения. Оптимальные решения вспомогательных конечномерных задач составляют монотонную последовательность значений, сходящуюся к минимальному значению функционала в задачах, в которых число переключений оптимального управления конечно.

Для решения задачи безусловной минимизации на гиперкубе предлагается поисковый алгоритм, основанный на последовательном решении невыпуклых задач одномерного поиска по случайному направлению. Для каждого случайного направления вычисляется интервал изменения переменной, позволяющий гарантировать нахождение любого решения внутри допустимого параллелепипеда и формулируется задача одномерного поиска, в общем случае, естественно, невыпуклая. Поиск глобального минимума по направлению выполняется с помощью алгоритма, основанного на комбинации методов сплайн-поиска, предложенного в [1] и надежного, но медленного классического метода Стронгина [2]. Распараллеливание алгоритма, выполняемое с применением технологии Nvidia CUDA, производится путем формирования пакета запросов на вычисление одномерной функции, что требует многократного решения задачи Коши. Численные эксперименты, проведенные на GPU Nvidia поколений Tesla и Fermi подтверждают высокий потенциал параллелизма предложенного алгоритма.

Литература

1. Горнов А.Ю. Применение сплайн-аппроксимации для конструирования алгоритмов оптимизации с новыми вычислительными свойствами // Труды всеросс. конф. "Дискретная оптимизация и исследование операций". Владивосток, 2007. С. 99.

2. Стронгин Р.Г. Численные методы многоэкстремальной оптимизации // М., Наука, 1978. С. 238.

*Работа частично поддержана грантом РФФИ № 10-01-00595.

Применение многопроцессорных систем для решения задач гидродинамики водяных турбин

Д.В. Банников, С.Г. Черный, Д.В. Чирков

Институт Вычислительных технологий СО РАН

В [1] авторами предложены постановки и методы решения задач численного моделирования пространственных течений в турбомашинах. Наиболее общей является постановка прямой задачи гидродинамики турбин, в которой моделирование нестационарного течения проводится во всём проточном тракте. Нестационарная постановка позволяет моделировать весь диапазон режимов работы гидротурбин (ГТ), в том числе и режимы неполной загрузки, учитывать взаимодействие ротора и статора турбины, описывать пульсации сил и моментов на лопатках, моделировать прецессирующий вихревой жгут за ротором и т.д.

Для расчёта потока строится многосвязная блочно-структурированная сетка, покрывающая весь проточный тракт турбины. На каждой итерации расчёт проводится во всех блоках с последующим обменом данных между соседними блоками. Итерации повторяются до тех пор, пока не будет найдено решение на текущем временном слое, затем происходит переход на следующий шаг по времени. Нестационарная постановка требует значительных вычислительных ресурсов, а повышение точности расчетов за счёт увеличения количества ячеек сетки приводит к нехватке оперативной памяти персонального компьютера. Например, расчет периодически нестационарного течения с прецессирующим вихревым жгутом на одном периоде (около трех оборотов РК) с использованием сетки, содержащей 1 млн. узлов, требует двух дней работы процессора Core2Duo 2.6 ГГц.

В настоящей работе внимание уделяется вопросу сокращения времени счёта нестационарных задач при помощи кластерных ЭВМ. Распараллеливание счета осуществляется методом декомпозиции области, при котором проводится распределение блоков расчетной сетки на процессоры кластера. Коммуникации осуществляются с использованием стандарта MPI. Проведен сравнительный анализ различных способов разбиения расчетной области по процессорам. Приводятся результаты ускорения счета, полученные на различных кластерах, на примере расчета нестационарного течения в турбине ГЭС Платановрисси (Греция).

Работа выполнена при финансовой поддержке РФФИ (грант № 11-01-00475-а) и Междисциплинарного интеграционного проекта СО РАН № 26.

Литература

1. Черный С.Г., Чирков Д.В., Лапин В.Н., Скороспелов В.А., Шаров С.В. Численное моделирование течений в турбомашинах – Новосибирск: Наука. – 2006. – 202 с.

Система визуализации параметров работы больших вычислительных систем

П.А. Брызгалов, С.А. Жуматий, Д.А. Никитенко, А.В. Адинец

Научно-исследовательский вычислительный центр МГУ имени М.В. Ломоносова

В ходе работ по созданию комплексной системы исследования эффективности работы больших вычислительных систем и приложений в рамках российско-европейского проекта HOPSA (HOlistic Performance System Analysis) возникла необходимость в системе визуализации, способной дать пользователям и администраторам инструменты для быстрого и точного анализа. Из требований, предъявляемых к системе, отметим следующие: ориентация на большие объёмы данных, интерактивность, гибкость – возможность работать с данными различной природы, расширяемость, работа через веб-браузер, получение данных через сеть, возможность свободного распространения.

Были созданы два прототипа: для анализа потока задач на вычислительных системах и для анализа данных мониторинга параметров работы вычислительных систем в реальном времени.

В настоящее время прототипы используются на суперкомпьютерном комплексе Московского университета.

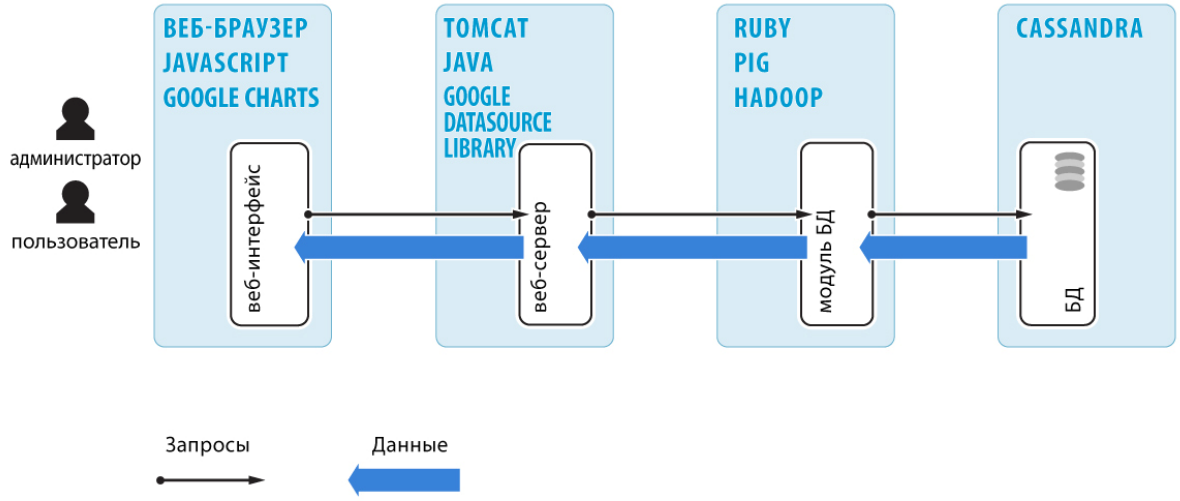


Рис. 1. Структура и технологии прототипа системы визуализации для анализа потока задач

Работа выполнена при поддержке государственного контракта №07.514.12.4001 и гранта РФФИ №10-07-00586-а.

Литература

1. А.В. Адинец, П.А. Брызгалов, В.В. Воеводин, С.А. Жуматий, Д.А. Никитенко Мониторинг, анализ и визуализация потока заданий на кластерной системе // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XI Всероссийской конференции - Нижний Новгород: Издательство Нижегородского госуниверситета, 2011. с. 10-14
2. А.В.Адинец, П.А. Брызгалов, Вад.В. Воеводин, С.А. Жуматий, Д.А. Никитенко Об одном подходе к мониторингу, анализу и визуализации потока заданий на кластерной системе // Вычислительные методы и программирование. Т. 12 (2011). стр. 90-93

Программный комплекс для анализа, отладки и оптимизации параллельных приложений

А.Ю. Власенко, Н.Н. Окулов, А.В. Демидов

ФГБОУ ВПО «Кемеровский государственный университет»

На сегодняшний день информационно-вычислительный портал (ИВП) КемГУ представляет собой программный комплекс для поддержки высокопроизводительных вычислений, предназначенный для решения задач в научной и образовательной сферах. Комплекс предоставляет пользователям разнообразные инструментальные средства для повышения удобства и эффективности процесса разработки параллельных программ.

Подсистема ИВП «Виртуальная лаборатория» выполняет следующие задачи: проведение серий вычислительных экспериментов с целью получения сведений для повышения эффективности параллельной реализации алгоритма, включая определение зависимости времени выполнения программы от задаваемых пользователем параметров; вычисление ускорения и эффективности исследуемого параллельного кода, а также прогноз времени, необходимого для выполнения расчета; организация виртуального лабораторного практикума студентов на высокопроизводительных вычислительных ресурсах.

Для определения степени влияния параметров на время выполнения используются методы корреляционного анализа. Для оценки предполагаемого времени выполнения программы с заданными значениями параметров применяется интерполяция и экстраполяция на основе данных, полученных при предыдущих запусках.

Сформированные системой сведения пользователь может применить для повышения эффективности и/или для предварительной оценки расчетного времени выполнения своей параллельной программы, а также может сравнить между собой данные времени выполнения программы на различных вычислительных ресурсах и выбрать оптимальную среду для запуска.

В ИВП также интегрирована система автоматического контроля корректности MPI-приложений, не требующая от пользователя никаких действий, кроме статического связывания своей программы с профилировочной библиотекой и стандартного запуска MPI-процессов [1]. На web-форме создания задания на компиляцию для проверки программы данной системой необходимо только поставить соответствующую «галочку», а при формировании задания на запуск – заполнить несколько параметров. После работы параллельной программы пользователю выдаются все найденные логические ошибки, специфические для MPI-приложений. К таким ошибкам относятся дедлоки, гонки данных, несоответствия и пр.

Разработанная система состоит из скрипта запуска, утилиты-препроцессора, профилировочной библиотеки, и программы сервера-анализатора. Схема работы системы заключается в следующем. На вычислительном кластере запускается скрипт, вызывающий препроцессор исходного кода MPI-программы; полученный файл передается на компиляцию с прилинковкой профилировочной библиотеки; на одном из вычислительных узлов кластера запускается сервер-анализатор, производящий анализ на наличие глобальных ошибок; при помощи mpiexec запускается программа пользователя. MPI-процессы порождают служебные потоки, которые производят поиск локальных ошибок. По окончании работы программы информация обо всех найденных ошибках сливается в один файл на сервере, который передается на web-форму пользователя.

Литература

1. Власенко, А.Ю. Система автоматического контроля корректности и виртуальная лаборатория как компоненты информационно-вычислительного портала [Текст] / А.Ю. Власенко, Н.Н.Окулов// Научно-технический вестник Поволжья. №6 – Казань, 2011г. –267с.

Об ускорении расчетов для задачи динамики газового пузырька с учетом направленной диффузии средствами Matlab*

Е.В. Волкова[1,2]

Центр «Микро- и наномасштабная динамика дисперсных систем», БашГУ, Уфа[1],
Институт механики УНЦ РАН, Уфа[2]

В работе численно исследуется влияние направленной диффузии на динамику одиночного сферически-симметричного газового пузырька при воздействии акустического поля в бесконечной малосжимаемой жидкости. Проведен анализ ускорения и эффективности параллельного алгоритма для рассмотренной задачи. Динамика пузырька описывается нелинейным дифференциальным уравнением Келлера-Миксиса [1], численное исследование которого проводилось по схеме Дормана-Принца. Уравнение конвекции-диффузии записано аналогично [2], решалось по схеме Кранка-Николсон. Для устранения вычислительных проблем, связанных с подвижной границей, по аналогии с [2] задача рассмотрена в Лагранжевых координатах. Необходимо проводить расчеты в широком диапазоне радиусов пузырька, что значительно увеличивает машинное время, если выполнять их последовательно. С помощью Matlab Parallel Computing Toolbox произведено распараллеливание кода программы.

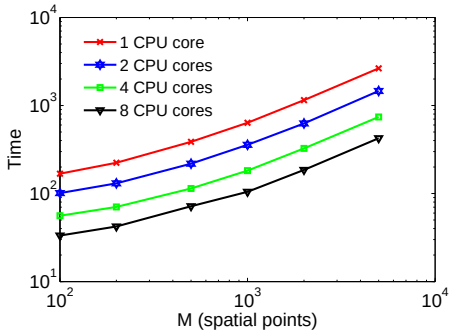


Рис. 1. Время работы программы в зависимости от количества точек по пространству

На рисунке 1 в логарифмическом масштабе показано сравнение расчетного времени на 8, 4, 2 ядрах CPU с расчетным временем последовательной работы программы в зависимости от количества точек по пространству. Коэффициенты ускорения $S_n = T_1/T_n$ и эффективности $E_n = S_n/n$ параллельного алгоритма, где T_n – время работы на n процессорах: $S_2 = 1.8370, E_2 = 0.9185; S_4 = 3.5328, E_4 = 0.8832; S_8 = 6.2028, E_8 = 0.7753$. Вычисления проводились на процессоре CPU Intel Xeon $5660, 2.8$ GHz. В планах создание алгоритма для выполнения расчетов на графической карте NVIDIA Tesla C2050.

Литература

1. I. Akhatov, N. Gumerov, et al. The Role of Surface Tension in Stable Single-Bubble Sonoluminescence. // Physical Review Letters. 1997. Vol. 78, No 2, pp 227-230.

2. Marios M. Fyrillas, Andrew J. Szeri. Dissolution or growth of soluble spherical bubble // J. Fluid Mech. 1994. vol. 277, P. 381-407.

*Автор выражает благодарность за помощь при постановке задачи и ценные замечания Ахатову И. Ш, Гумерову Н. А., Насибуллаевой Э. Ш. Работа выполнена при финансовой поддержке гранта Министерства образования и науки РФ (11.G34.31.0040).

Программный комплекс для моделирования динамики трехмерных астрофизических газовых объектов на суперЭВМ *

В.А. Вшивков[1], И.М. Куликов[1], Г.Г. Лазарева[1], А.В. Тутуков[2]

Учреждение Российской академии наук Институт вычислительной математики и математической геофизики Сибирского отделения РАН[1], Учреждение Российской академии наук Институт астрономии РАН[2]

Современная астрофизика – это эволюционная теория, которая изучает условия образования и устойчивости упорядоченных структур в первоначально бесструктурной материи – космическом хаосе [1]. Наблюдательное и теоретическое изучение астрофизических процессов – незаменимый метод исследования его свойств и эволюции. Математическое моделирование на суперЭВМ играет более чем важную роль в теоретическом исследовании астрофизических процессов. По результатам работы создан программный комплекс PEGAS (PErformance Gas Astrophysic Simulation) для решения задач гравитационной газовой динамики.

Результатом работы стало формирование интегральных критериев формирования коллапса, полученные с помощью вычислительного эксперимента на суперЭВМ, проверка законов сохранения при численном моделировании процесса коллапса астрофизических газовых объектов. По результатам работы был создан новый численный метод для решения задач коллапса астрофизических газовых объектов. Так было показано преимущество созданного авторского численного метода по сравнению с бессеточным методом, лагранжевым и эйлеровым сеточным методом. Создан набор тестовых решений, проведено численное сравнение процесса коллапса астрофизических объектов с другими авторами. По результатам работы по проекту создан программный комплекс PEGAS (PErformance Gas Astrophysic Simulation) для решения задач гравитационной газовой динамики [2–4].

Литература

1. Тутуков А.В., Лазарева Г.Г., Куликов И.М. Газодинамика центрального столкновения двух галактик: слияние, разрушение, пролет, образование новой галактики // Астрономический журнал. 2011. Т. 88, №9. С. 837-851.

2. Vshivkov V., Lazareva G., Snytnikov A., Kulikov I., Tutukov A. Hydrodynamical code for numerical simulation of the gas components of colliding galaxies // The Astrophysical Journal Supplement Series. 2011. V. 194, 47. 12 pp.

3. Vshivkov V., Lazareva G., Snytnikov A., Kulikov I. Supercomputer Simulation of an Astrophysical Object Collapse by the Fluids-in-Cell Method // PaCT-2009 proceedings. LNCS, Vol. 5698. 2009. P. 414-422

4. Вшивков В.А., Тутуков А. В., Лазарева Г.Г., Куликов И.М. Суперкомпьютерное моделирование столкновения галактик // Суперкомпьютерные технологии в науке, образовании и промышленности. 2010. №2. С. 88-92.

*НИР выполнена в рамках ФЦП "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы и "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы".

Сравнение эффективности распараллеливания термопрочностных задач в инженерных пакетах при моделировании процессов линейной сварки трением*

Р.К. Газизов, И.Ш. Насибуллаев, А.В. Юлдашев, К.Р. Юлмухаметов, А.М. Ямилева

Уфимский государственный авиационный технический университет

Сварка трением – это разновидность сварки давлением, при которой нагрев осуществляется трением, вызванным перемещением друг относительно друга соединяемых частей свариваемого изделия. В процессе линейной сварки трением (ЛСТ) тепло выделяется при возвратно-поступательном движении свариваемых частей, с частотой порядка 50 Гц и амплитудой до 3-х мм, сжимаемых для образования плотного контакта.

В данной работе рассматривается эффективность распараллеливания статической и динамической термопрочностной задачи при моделировании процессов ЛСТ в инженерных пакетах SIMULIA Abaqus 6.11 (далее Abaqus) и ANSYS Mechanical 13 (далее ANSYS) на суперкомпьютере УГАТУ.

В Таблице 1 показаны некоторые результаты расчета динамической термопрочностной задачи в MPI-версиях пакетов с использованием до 6 узлов кластера. В большинстве случаев в пакете Abaqus расчеты выполняются быстрее, чем в ANSYS, но используется больше оперативной памяти.

Таблица 1. Время расчета и объем используемой памяти для динамической термопрочностной задачи, N – число задействованных узлов, CPN – количество задействованных ядер на каждом узле

Характеристики задачи	Время (1 ядро), ч.		Мин. время, ч. (N x CPN)		Объем используемой памяти (1x1), МБ		Макс. объем на 1 ядро для мин. времени, МБ	
	Abaqus	ANSYS	Abaqus	ANSYS	Abaqus	ANSYS	Abaqus	ANSYS
Линейные элементы, 70 000 уравнений	3.65	7.55	0.41 (6x4)	0.95 (6x6)	1405	1432	471	597
Линейные элементы, 105 000 уравнений	9.38	13.86	0.86 (6x4)	1.79 (6x4)	2411	2505	784	657
Линейные элементы, 140 000 уравнений	18.92	19.65	1.60 (4x8)	2.06 (6x4)	3479	3621	1356	699
Квадратичные элементы, 200 000 уравнений	21.57	21.35	1.62 (4x8)	2.18 (6x6)	8277	5539	3196	732

Проведено исследование эффективности решения задачи в ANSYS Mechanical на гибридном вычислительном сервере, установленном при суперкомпьютере УГАТУ. Получено, что использование GPU NVIDIA Tesla M2050 совместно с CPU Intel Xeon 5670 позволяет сократить время расчета в SMP-версии в 1.5-2.5 раза по сравнению с расчетом только на CPU.

В Таблице 2 приведены некоторые результаты при расчете статической термопрочностной задачи (система 65 000 уравнений) в MPI-версиях пакетов Abaqus и ANSYS с использованием до 6 узлов кластера. В отличие от динамической задачи ANSYS здесь использует больше оперативной памяти, чем Abaqus, при параллельном расчете – почти в 2 раза.

Таблица 2. Время расчета и объем используемой памяти для статической термопрочностной задачи, N – число задействованных узлов, CPN – количество задействованных ядер на каждом узле

Пакет	Время (1 ядро), сек.	Мин. время, сек., (N x CPN)	Объем используемой памяти (1x1), МБ	Макс. объем на 1 ядро для мин. времени, МБ
Abaqus	2099	413 (5x4)	872	327
ANSYS	3542	386 (6x6)	964	607

* Работа выполнена в рамках проекта «Создание технологий и промышленного производства узлов и лопаток ГТД с облегченными высокопрочными конструкциями для авиационных двигателей новых поколений» (шифр 2010-218-01-133) в рамках реализации постановления № 218 Правительства РФ от 9.04.2010 г. «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства».

Распределенные символьные дробно-рациональные вычисления на процессорах x86 и x64

В.А. Голодов

Кафедра экономико-математических методов и статистики,
ФГБОУ ВПО ЮУрГУ (НИУ)

В рамках предыдущих исследований были созданы классы `overlong` и `rational`, реализованные в объектно-ориентированной парадигме на языке C++, а также библиотека классов `Exact Computational` [1]. Классы `overlong` и `rational` имеют MPI оболочки и, тем самым, позволяют производить безошибочные дробно-рациональные вычисления в параллельных средах. Для классов `overlong` и `rational` определены все операторы, операции и бинарные отношения, используемые для стандартных числовых типов данных.

На сегодняшний день возможность использования безошибочных вычислений представляет известная библиотека GMP (The GNU Multiple Precision Arithmetic Library) однако актуальная версия GMP 5.0.x **НЕ** предоставляет своим объектам возможность их использования в параллельных вычислениях (http://gmplib.org/gmp5.0.html).

Поскольку первые редакции описанных классов были выпущены в 1999 году, к настоящему времени выявился ряд недостатков относительно современного состояния вычислительной техники. Была проведена работа по актуализации кода, часть изменений, внесенных в базовый тип `overlong`, представлена ниже.

1. Основание системы счисления было заменено с 2^{16} на 2^{32}. Это существенно сокращает длину обрабатываемого массива цифр, а значит и время вычислений в целом.

2. Работа с памятью была организована более эффективно.

3. Были задействованы преимущества основания системы счисления вида 2^{32} за счет быстрых логических операций.

4. Были пересмотрены реализации операций. При делении и взятии остатка от деления удалось избавиться от порождения временных объектов.

5. Код класса был переписан в терминах обобщенного программирования. Это упростило операции со смешанными типами и сократило объемом сопровождаемого кода.

6. MPI оболочка типа `overlong` изменена в соответствии с новой внутренней структурой класса.

7. Все изменения обеспечивают обратную совместимость кода (необходимое требование к вносимым изменениям).

Для демонстрации эффективности усовершенствований был проведен вычислительный эксперимент по нахождению обратной матрицы Гильберта алгоритмом Жордана-Гаусса с применением исходных классов `overlong` и `rational` и их последних модификаций. Для матриц размера порядка 300 ускорение составило более 12 раз, с ростом размерности матрицы ускорение возрастает. Объем требуемой оперативной памяти для обработки матрицы порядка $n = 1000$ на одном вычислительном узле составил менее 1 Гб.

Дальнейшие исследования направлены на оптимизацию кода, а также на параллельную реализацию базовых арифметических операций. Привлекательным является использование для этого графических ускорителей.

Литература

1. Панюков А.В., Германенко М.И., Горбик В.В. Библиотека классов «Exact Computational» , // Программы для ЭВМ, базы данных, топологии интегральных микросхем: официальный бюллетень Рос. агентства по патентам и товарным знакам № 3. - 2009. - С. 251.

Параллельная реализация программы для вычисления объема молекул фуллеренов

И.М. Губайдуллин[1], Д.Ш. Сабиров[1], А.Д. Саитгалина[2]

Институт нефтехимии и катализа РАН[1],
Башкирский государственный университет[2]

Фуллерены представляют собой особый класс полых полиэдрических молекул. С момента открытия C_{60} и C_{70} сразу же стали рассматриваться разные варианты образования их эндоэдральных комплексов – топологических соединений, в которых атом или группа атомов расположены во внутренней полости фуллерена [1]. Интерес к таким комплексам обусловлен многими причинами. В частности, такие комплексы предполагается использовать в качестве блоков наноустройств и нанокапсул для хранения газов. В настоящее время получены эндофуллерены с различными инкапсулированными атомами (He, Ne, Ar и др.) и малыми молекулами (H_2, N_2, CO) [1], а также теоретически показана возможность образования таких комплексов с более сложными молекулами и коллективами молекул (например, $n H_2@C_{60}$, $C_6H_6@C_{60}$ и др.). Перечень атомов и молекул, вводимых внутрь каркасов фуллеренов и их производных будет расширяться, в связи с чем необходима предварительная оценка возможностей образования эндоэдральных комплексов, заключающаяся в сравнительном анализе геометрических параметров каркаса фуллерена и вводимых в его внутреннюю полость атомов и молекул. При этом ключевым параметром, характеризующим возможность инкапсулирования, является внутренний объем фуллерена, который может изменяться при переходе от одного фуллерена к другому, а также при функционализации их молекул.

На первом этапе этого исследования разработан алгоритм вычисления объема макромолекул, в основе которого лежит разбиение полиэдрической молекулы на симплексы (пирамиды), вычисление их объемов и последующее суммирование. Предложенный алгоритм допускает использование технологии параллельных вычислений, что позволяет сократить время при решении поставленной задачи. Программный комплекс «Volume», реализующий этот алгоритм, предполагается использовать для поиска фуллеренов и их производных с необходимыми значениями объема внутренней полости и оценки возможности их инкапсулирования.

Программа «Volume» была использована для изучения зависимости объема фуллеренов от числа атомов в молекуле, которая, как показали вычисления, не является линейной (рис. 1).

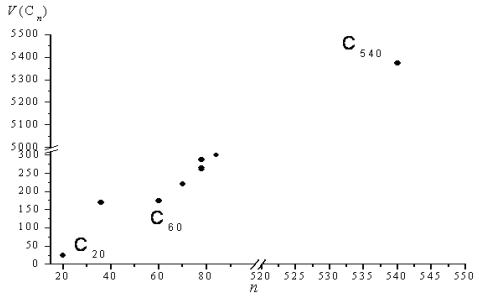


Рис. 1. Зависимость объема фуллеренов $V(C_n)$ от числа атомов в молекуле

Установлено (см. рис. 1), что изомеры фуллерена C_{78}, характеризующиеся одинаковым числом атомов в молекуле, но разными точеными группами симметрии, имеют неодинаковые объемы.

Литература

1. Y. Rubin. Ring Opening Reactions of Fullerenes: Designed Approaches to Endohedral Metal Complexes // Topics in Current Chemistry, 1999, V. 199, P. 67-91.

Использование технологии параллельных вычислений при разработке прибора автоматического контроля параметров детали сложной геометрической формы

М.Р. Еникеев[1], Л.С. Сайфуллина[1], И.М. Губайдуллин[2]

Башкирский Государственный Университет[1], Учреждение российской академии наук Институт нефтехимии и катализа РАН[2]

Разработан и реализован прибор для контроля геометрических параметров профиля детали, имеющей сложный геометрический профиль. Создано программное обеспечение для работы прибора. Проведен анализ эффективности работы параллельного варианта программы и определена погрешность метода расчета.

1. Введение

В настоящее время перед авиадвигателестроительными предприятиями в условиях жесткой конкурентной борьбы стоит задача создания конкурентоспособной, на мировом уровне, продукции. Одним из таких условий является повышение эффективности (КПД) двигателя. Важнейшим фактором повышения эффективности газотурбинного двигателя является совершенствование конструкций деталей, из которых он состоит.

В условиях серийного производства встает вопрос не только о качественном контроле геометрических параметров детали, но и о проведении этого контроля за минимальное время. Поэтому концепцией работы является разработка современного прибора, обеспечивающего бесконтактный контроль геометрических размеров детали, имеющей сложный геометрический профиль, в условиях серийного производства. Созданный прибор за счет применения цифровых методов получения и обработки информации [1], высокоточной оптики и применение параллельных вычислений для проведения расчетов, обеспечивает высокопроизводительный контроль параметров детали с заданной точностью.

При решении задачи контроля геометрических параметров возникает вопрос о сокращении времени расчетов. Распараллелить решение задачи можно на нескольких уровнях.

Программное обеспечение для прибора было создано на языке C++ с использованием библиотек Qt , OpenCV и OpenMP. Вычислительный эксперимент был проведен на вычислительном узле прибора (AMD Phenom X4 940 3.1 GHz, 3 GB Ram). Из результатов вычислительно эксперимента следует, что показатель эффективности распараллеливания не опускается ниже 0.83 (для 4 процессоров), что является довольно неплохим показателем. Обычно требуется обсчитать не одну деталь, а несколько десятков, в этом случае выигрыш во времени составляет, например, для 4 процессоров: (187-56)*50=6550 (сек.) ≈109 (мин.)

Осуществлен запуск в эксплуатацию прибора бесконтактного контроля геометрических параметров профиля сложной детали, выполнена сборка прибора и проведены пусконаладочные работы, предшествующие запуску прибора. Получено ускорение в работе прибора с использованием технологии параллельных вычислений. В дальнейшем планируется разработка более точных методов снятия изображения и реализация алгоритмов расчета геометрических параметров детали со сложным геометрическим контуром.

Литература

1. Шапиро Л., Стокман Дж. Компьютерное зрение // Москва: Бином, 2006 – С. 752.

Язык моделирования молекулярно-генетических систем SiBML*

Ф.В. Казанцев, В.В. Миронова, Е.С. Новоселова, Н.Л. Подколодный, В.А. Лихошвай

Институт Цитологии и Генетики СО РАН

Одним из центральных вопросов системной биологии является выяснение молекулярно-генетических механизмов функционирования живых систем. Для изучения их динамических характеристик все шире применяются методы математического и компьютерного моделирования, эффективное применение которых требует учитывать в моделях специфику строения молекулярно-генетических систем (МГС). К таким особенностям относятся: явления линейного характера кодирования информации, где значение имеет взаимное расположение генов, промоторов, терминальных и других генетических элементов на молекуле ДНК; топология молекулы ДНК в разные периоды клеточного цикла; существование обратимых состояний белковых молекул меняющих их свойства; явления полиаллельности генома; анизотропия пространственных компартментов живых систем разделяющая процессы биохимического синтеза, что добавляет регуляторные воздействия на систему процессами мембранного транспорта; эпигенетическая (вне генома) передача наследственной информации. Для адекватного моделирования МГС необходимо разрабатывать подходы, объединяющие эти явления в рамках одного стандарта.

В данной работе представлен разработанный в ИЦиГ СО РАН язык спецификации математических моделей молекулярно-генетических систем и процессов *SiBML* (Siberian Biology Modeling Language). Язык *SiBML* и его программное обеспечение изначально ориентированы на исследование приведенных выше особенностей МГС. Основной лексемой языка служит «объект», который представлен набором пар «*атрибут(значение)*». *Атрибут* может отражать информацию о названии, о принадлежности к классу моделируемых объектов (ген, белок, РНК и др.), о его локализации в структуре моделируемых компартментов (органеллы, клетки, принадлежность к ткани и др.) или любое свойство моделируемого объекта. Такой подход позволяет однозначно интерпретировать заложенную информацию об объекте и расширять существующую функциональность средств языка дополнительными инструментами анализа и репрезентации, как результатов, так и самих моделей. На первом этапе реконструкции математической модели в терминах языка *SiBML* проводится декомпозиция моделируемой системы до уровня элементарных подсистем, и описываются их математические модели (или берутся ранее разработанные). Затем, по заданному пользователем сценарию компиляции, начинается процесс объединения моделей элементарных подсистем или в функциональные блоки (молекулы плазмид, например), или по уровням иерархии (распределение подсистем по компартментам: органеллам, клеткам, тканям). Транслятор модели добавляет объектам ряд атрибутов (в зависимости от сценария), по которым однозначно интерпретируются: локализация объекта в модели, позиция и ориентация гена в плазмиде, порядок стадии в цепочке превращений и др. В результате работы транслятора получается и система уравнений, и модель в формате *SiBML*, которая может быть использована как элементарная подсистема при реконструкции новых моделей.

Данная технология была развернута на вычислительном кластере НКС-30Т ССКЦ СО РАН. С ее помощью проведено исследование модели распределения гормона ауксина (Auxin) в меристеме корня растения с учетом активного транспорта через его белки-транспортеры Pin1-Pin3. В модели рассматривался синтез, деградация и распределение веществ (Auxin, Pin1, Pin2, Pin3) по клеточному ансамблю (4x20 клеток). Итоговая модель содержала 240 динамических переменных и 38 параметров. Проведено несколько тысяч вычислительных экспериментов с варьированием параметров модели. Найдены значения параметров модели, которые выводят модель на стационарные решения, соответствующие данным эксперимента. На основе численного анализа модели предложены гипотезы о механизмах формирования в меристеме корня неоднородных распределений ауксина с несколькими максимумами и градиентами концентрации.

* Работа выполнена при частичной поддержке грантов РФФИ (№10-01-00717, №11-04-01254), МинОбрнауки (госконтракты № 07.514.11.4023, П857) и интеграц. проектами СО РАН (№ 130, № 107)

Параллельный метод Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой

Е.Г. Качко, К.А. Погребняк

Харьковский национальный университет радиоэлектроники

Известно, что стойкость значительной части криптографических примитивов, используемых в информационных системах, основывается на вычислительной сложности решения задачи дискретного логарифма в группе точек эллиптической кривой (ЭК), заданной над конечным полем. Методы дискретного логарифмирования в группе точек ЭК принято разделять на два типа. К первому типу относятся методы, которые применимы к ЭК с определенной структурой группы, а ко второму – методы логарифмирования для произвольной структуры группы. Следует отметить, что при правильном выборе ЭК атаки первого типа становятся нереализуемыми. Среди методов логарифмирования второго типа, наиболее эффективным считается вероятностный метод Полларда [1]. Фактически, метод Полларда включает в себя алгоритм построения псевдослучайной последовательности точек ЭК и алгоритм обнаружения коллизии. Повышение эффективности криптоанализа методом Полларда достигается за счет сокращения длины последовательности значений функции итерирования, улучшения алгоритма обнаружения коллизии и распараллеливания вычислений. Идея распараллеливания метода Полларда состоит в том, что итерирование точек возлагается на клиентские рабочие станции, а обнаружение коллизии – на сервер [2]. Предложенный параллельный метод Полларда предполагает наличие высокопроизводительных сетей и не учитывает использование многоядерной архитектуры клиентских рабочих станций и сервера. Таким образом, актуальной задачей является исследование известных модификаций метода Полларда и адаптации их к многоядерной архитектуре рабочих станций.

В работе предложен метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой для систем с общей памятью. На основе параллельного алгоритма Ооршота и Вейнера [2] для систем с распределенной памятью предложен комбинированный метод нахождения дискретного логарифма, который позволяет использовать преимущества как многопроцессорных, так и многоядерных систем. Проанализированы известные функции итерирования точек в алгоритме Полларда и построен обобщенный метод Полларда для произвольной функции итерирования и систем с общей памятью. Приведены аналитические выражения для предложенных параллельного метода Полларда и комбинированного метода Полларда для двуядерных процессоров. В работе получены эмпирические временные показатели для параллельного метода Полларда для систем с общей памятью для битовой длины порядка группы точек ЭК 21, 23 и 32 бита, которые согласуются с аналитическими выражениями. Следует отметить, что моделирование проводилось для двуядерных процессоров, что обусловлено наличием двух последовательностей в алгоритме обнаружения цикла. Предложенный подход к распараллеливанию алгоритма Полларда позволил снизить время вычислений на 30 – 50%. Исследованы временные показатели избыточных операций для параллельного алгоритма Полларда для систем с общей памятью.

В дальнейшем планируется провести моделирование для комбинированного метода и для различных итеративных функций, обобщить полученные результаты на случай произвольного числа ядер и на кривые с большей битовой длиной порядка подгруппы.

Литература

1. Pollard J.M. Monte Carlo methods for index calculus computation (mod p) // Mathematics of Computation. July 1978. Vol. 32, No. 143. P. 918-924.
2. P. Van Oorschot, Wiener M. Parallel collision search with cryptanalytic applications. // Journal of Cryptology. 1999. Vol. 12. P. 1–28.

Фрагментация полуинтерпретированных численных алгоритмов*

С.Е. Киреев

Институт вычислительной математики и математической геофизики СО РАН,
Новосибирский государственный университет

Технология фрагментированного программирования призвана автоматизировать решение задач параллельного программирования при реализации численных моделей. В ее основе лежит метод синтеза параллельных программ [1,2]. В рамках технологии численный алгоритм должен быть представлен в специальном виде как множество фрагментов данных и вычислений. Фрагментированный алгоритм содержит только минимальное управление, определяемое зависимостями по данным, и не содержит распределения ресурсов. Таким образом, он допускает множество способов исполнения, что обеспечивает его переносимость. Фрагментация – это технологический прием, позволяющий уменьшить число объектов алгоритма и тем самым упростить задачу построения эффективного распределения ресурсов и управления.

Настоящая работа посвящена методике разработки фрагментированных алгоритмов, которые могут быть эффективно исполнены на мультикомпьютере при наличии соответствующей исполнительной системы [3,4]. Были выделены следующие принципы построения фрагментированных алгоритмов:

- уменьшение числа фрагментов и связей,
- одинообразие фрагментов,
- масштабируемость алгоритма,
- настраиваемость алгоритма на конкретный вычислитель.

В качестве основного вопроса при построении фрагментированного алгоритма рассматривается вопрос о фрагментации регулярных структур, образуемых объектами исходного алгоритма. Вопрос о выборе структуры фрагментов решается на основе дополнительной информации, которая напрямую не следует из вида исходного алгоритма. Разрабатываемая методика продемонстрирована на примере фрагментации алгоритма умножения разреженной матрицы на вектор. Предложен подход к фрагментации алгоритма метода частиц-в-ячейках: рассмотрены требуемые регулярные структуры и их поддержка в исполнительной системе.

Дальнейшая работа по разработке методики фрагментации связана с рассмотрением конкретных регулярных структур для реализации распространенных численных методов и алгоритмов. Кроме того, формируется список требований к исполнительной системе, которым она должна удовлетворять для обеспечения должного качества исполнения фрагментированных программ.

Литература

1. В.А.Вальковский, В.Э.Малышкин. Синтез параллельных программ и систем на вычислительных моделях. – Наука, Новосибирск, 1988, 128 стр.
2. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. – In the Int. Journal on Future Generation Computer Systems. Vol. 17 (2001), No. 6, pp. 755-765.
3. Malyshkin V.E., Perepelkin V.A. Optimization of Parallel Execution of Numerical Programs in LuNA Fragmented Programming System // MTPP-2010 revised selected papers, Springer, LNCS 6083, 2010. pp. 1-10.
4. S.Kireev, V.Malyshkin Fragmentation of Numerical Algorithms for Parallel Subroutines Library // The Journal of Supercomputing. Vol. 57. Number 2. 2011. pp. 161-171.

* Проект поддержан грантом РФФИ № 10-07-00454а.

Фрагментированный алгоритм численного решения задачи Ламе попеременно-треугольным методом*

Е.О. Кондрашкин[1], О.Ю. Серегина[2], С.Б. Сорокин[1]

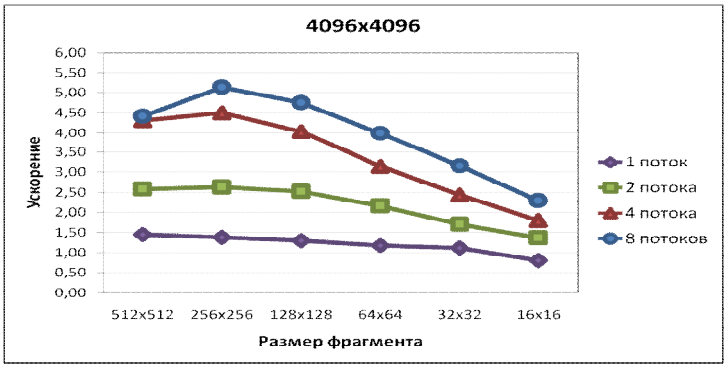
Институт вычислительной математики и математической геофизики[1],
Новосибирский государственный университет[2]

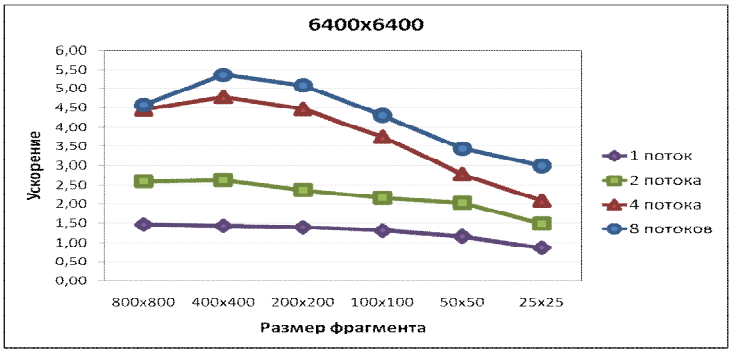
1. Введение

Алгоритм попеременно-треугольного метода – одного из эффективных итерационных методов решения систем линейных алгебраических уравнений считается сложным в распараллеливании. Сложность определяется зависимостью данных, характерной для алгоритмов обращения верхнетреугольных (нижнетреугольных) матриц. Целью данной работы является исследование возможности эффективной реализации фрагментированного алгоритма попеременно-треугольного метода для численного решения задачи Ламе на вычислительных системах с разделенной и общей памятью. Представлены результаты вычислительного эксперимента, иллюстрирующие ускорения фрагментированной программы относительно последовательной программы в зависимости от: числа потоков, узлов, размера и формы фрагментов.

2. Результаты численных экспериментов

На двух следующих рисунках для различного числа узлов расчетной сетки (указанного в верхней части) приведено ускорение=T1/Tp фрагментированной программы относительно последовательной в зависимости размера фрагментов.





* Работа выполнена при финансовой поддержке программы № 1.3 Фундаментальные исследования ОМН РАН «Современные вычислительные и информационные технологии решения больших задач», программы президиума РАН "Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация", ФЦП «Научные и научно-педагогические кадры инновационной России»

Автоматическое построение передаточных функций для систем визуализации распределенных данных на суперкомпьютерах*

О.В. Корж

Московский государственный университет имени М.В.Ломоносова

Объем данных, получаемых в процессе вычислительного эксперимента на системах петафлопсного уровня, значительно превосходит размеры информации, которая требуется для анализа результатов эксперимента [2]. Визуализация данных с автоматической и полуавтоматической селекцией данных позволяет структурировать данные и выделять объем информации, который будет визуализирован. Существуют различные подходы к децимации данных при визуализации [1]. Одним из таких подходов является автоматическое определение структуры и параметров передаточной функции для заданного входного набора данных.

Под передаточной функцией (transfer function) в данной работе подразумевается отображение значений данных, полученных в вычислительном эксперименте, на шкалу цвета и прозрачности для отображения этих значений при визуализации. Основная сложность определения таких функций при визуализации на суперкомпьютере – необходимо организовать предобработку большого объема данных, которые хранятся распределено.

Существует два основных подхода к автоматическому построению передаточной функции [3]. Первый подход: подбор функции по некоторой выборке начальных данных и распространение функции, полученной по этой выборке, на последующие данные. Второй подход: использование данных, которые поступают непрерывно в процессе эксперимента.

В данной работе рассмотрены подходы для выделения основных характеристик для таких функций для систем параллельной визуализации, построенных на базе библиотеки VTK. В модели рендеринга этой библиотеки используется построение двух передаточных функций: функция цвета и функция прозрачности определяются независимо друг от друга. В данной модели используется кусочно-линейная структура для представления таких функций.

Для расчета функций анализируется применение различных статистических распределений и аппроксимация таких функций с помощью различных статистических кривых. При этом применяется комбинация различных распределений. Такой подход позволяет выделить регионы интереса и определять для них более точные статистические функции.

Проведены эксперименты по применению таких функций при визуализации данных большого объема на системе Ломоносов.

Литература

1. Джосан О.В., Мурынин А.Б., Попова Н.Н. Метод визуализации многомерных динамических данных на многопроцессорных комплексах // Вестник компьютерных и информационных технологий. 2009. № 8. С. 8-12.

2. Peterka T., Goodell D., Ross R., Shen H.-W., Thakur R. A Configurable Algorithm for Parallel Image-Compositing Applications // Proceedings of the Conference on High Performance Computing, Networking, Storage, and Analysis, Portland, Oregon, USA, ACM, May 2009. Also Preprint ANL/MCS-P1624-0509, May 2009.

3. Джосан О.В., О сложности стратегий параллельного построения изображений для систем визуализации на суперкомпьютерах // Жур. Вестник ЮУрГУ. Компьютерные технологии, управление, радиоэлектроника, No 23 (240), вып. 14, 2011, изд. центр ЮУрГУ, сс.87-97. ISSN 2071-0216

* Работа выполнена при поддержке ФЦП “Научные и научно-педагогические кадры инновационной России” на 2009 - 2013 годы”, грантов РФФИ 11-07-00614-а, 11-07-00756-а.

Комплексная стохастическая модель процесса абразивной обработки, реализованная средствами параллельных алгоритмов

А.А. Кошин, Л.В. Шипулин

ФГБОУ ВПО «Южно-Уральский государственный университет» (НИУ)

В современном машиностроении возникает ряд задач, решение которых возможно лишь с использованием глубокого комплексного моделирования процесса. Такое моделирование позволило бы по заданным исходным данным – входным технологическим параметрам процесса определить важнейшие выходные параметры без проведения натурных экспериментов: шероховатости, эффективности съема припуска, качества поверхности детали и точности обработки.

При моделировании процесса учитываются его особенности, а именно: высокие скорости резания (до 80 м/с), теплонапряженность (точечная температура под зерном достигает температур плавления металла), стохастичность (десятки тысяч зерен расположены в круге случайным образом) и нелинейность процесса (прочностные свойства материала изменяются при нагреве). Поскольку имитационная модель является стохастической, то проводятся несколько расчетов-редукций для одних и тех же исходных данных. Работа имитационной модели при средних шагах сетки длится порядка 200-250 часов, поэтому принято решение применить параллельные вычислительные технологии.

Первым уровнем распараллеливания последовательного алгоритма является разбивка редукций (макроцикл) по процессорам кластера, реализацию которой можно представить следующим образом. На основе одних исходных данных одновременно на N процессорах – производится реализация N независимых редукций, после чего в процессоре N+1 производится статистическая обработка и структурирование выходных данных. Самой эффективной здесь является классическая параллельная схема, когда для каждой редукции исследуемой случайной функции выделяется отдельная вычислительная ветвь (количестве ветвей равно требуемому числу редукций) [1].

Вторым уровнем распараллеливания является распараллеливание микроциклов, вложенных в макроцикл. Анализ исходного кода и практика использования программного комплекса показали, что наибольшее время работы программного комплекса (до 99%) затрачивается на выполнение следующих операций:

1. Формирование режущей части абразивного инструмента. Сложность данного блока заключается в вероятностной реализации равномерного распределения абразивных зерен и формирования их размеров по закону нормального распределения. Анализ показал, что выполнение данного блока занимает от 5% до 20% от итогового времени расчета.

2. Расчет интенсивности тепловыделения каждого абразивного зерна, находящегося в зоне контакта с обрабатываемой поверхностью заготовки. Выполнение данного блока занимает от 5% до 25% от итогового времени расчета.

3. Расчет распределения поля температур по глубине поверхностного слоя обрабатываемой поверхности в заданные моменты времени. Выполнение данного блока занимает от 60% до 95% от итогового времени расчета.

В модели формирования шлифованной поверхности применен способ распараллеливания расчетов, исходные данные которых заранее не известны. Программа сама выполняет распределение расчетов по процессорам по определенным правилам.

Литература

1. Воеводин, В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. — СПб.: БХВ-Петербург, 2004. — 608 с.

Параллельный алгоритм для моделирования динамики мантийных течений*

Г.Г. Лазарева[1], В.Д. Корнеев[1], А.В. Бабичев[2]
ИВМиМГ СО РАН[1], ИГМ СО РАН[2]

Разработана параллельная версия программы для моделирования течений в мантии Земли. Нестационарная модель мантийных течений описывает сжимаемую среду с сильно изменяющимися реологическими и транспортными свойствами и основана на решении системы уравнений Навье-Стокса. Численная модель содержит как явные, так и неявные конечно-разностные схемы, реализованные векторными прогонками. Для численной реализации рассматриваемого алгоритма в качестве вычислительной системы с общим полем памяти используется вычислительный узел кластера ИВЦ НГУ. В работе проведен детальный анализ параллельного алгоритма, позволяющего получать близкое к линейному ускорение, не смотря на использование векторной прогонки.

Формирование крупных изверженных провинций на континентальной и океанической плитах связывают с нижнемантийными суперплюмами, поднимающимися с глубин границы мантии и ядра. Физические аспекты процесса всплывания плюма термической или термохимической природы достаточно хорошо изучены с помощью физического и математического моделирования. Эта концепция включает всплывание в локальной области легкого, высокотемпературного и маловязкого мантийного материала - плюма - на фоне крупномасштабных конвективных течений. Неясными остаются вопросы о конечном этапе эволюции диапиров: как высоко они могут подниматься и каково соотношение подъемной силы и вязкого сопротивления вещества при подъеме на верхние уровни литосферы. Диапировый механизм транспорта магмы в наиболее вязкой и холодной части мантийной литосферы требует подробного изучения. Моделирование мантийных течений имеет ряд существенных особенностей, поэтому не все хорошо зарекомендовавшие себя методы применимы к этому типу задач. В отличие от традиционного подхода, основанного на приближении Буссинеска, рассматриваемая модель основана на решении системы полных классических уравнений Навье-Стокса, описывающих динамику слабосжимаемой жидкости с переменными плотностью и вязкостью. Геодинамика рассматривает очень медленные течения, поэтому в теории ранее не использовалось число Маха, в отличие от сейсмологии и других разделов геофизики. Формально вычислив число Маха, можно обратиться к опыту вычислений в области существенно дозвуковых течений. Нелинейный характер уравнений данной модели приводит к необходимости использования методов решения, основанных на использовании вычислительной техники. Для численной реализации рассматриваемого алгоритма в качестве вычислительной системы с общим полем памяти используется вычислительный узел кластера ИВЦ НГУ (www.nusc.ru). Вычислительные узлы кластера состоят из двух 4-х ядерных процессоров Intel Xeon 5355, работающих на частоте 2.66 ГГц, и с 16-ю ГБайт общей оперативной памяти. Вычислительная система с общей памятью выбрана с учетом специфики численной модели, характеризующейся большим числом векторных прогонок и не требующей более десятка процессоров при распараллеливании. Показано, что на ускорение и эффективность параллельного алгоритма рассматриваемой задачи основное влияние оказывает размер сеточного пространства, точность расчета скоростей оказывает крайне слабое влияние. Детальный анализ параллельного алгоритма показал, что возможно получать для сеточных пространств среднего размера (5000x1000) близкое к линейному ускорение, не смотря на использование векторной прогонки. В ходе реализации параллельной версии программы смоделирован цикл, состоящий из зарождения диапира, подъема на предельную высоту и застывания при прекращении действия теплового источника или установления стационарного режима конвекции при постоянном действии источника тепла.

*НИР выполнена при финансовой поддержке интеграционного проекта СО РАН, в рамках реализации ФЦП "Научные и научно-педагогические кадры инновационной России" на 2009 - 2013 годы.

Решение уравнения импеданса растворения железа в кислом сульфатном электролите с использованием генетического алгоритма

М.А. Малеева[1], А.Р. Еникеев[2], И.М. Губайдуллин[2]

ФГБУН ИФХЭ РАН[1], Башкирский государственный университет[2], Институт нефтехимии и катализа РАН[3]

Получена передаточная дробно-рациональная функция третьего порядка адмиттанса анодного процесса с тремя типами интермедиатов. Определены кинетические константы элементарных стадий анодного процесса с помощью параллельного генетического алгоритма. Представлена оценка эффективности и распараллеливания алгоритма.

Процесс анодного растворения железа в водных электролитах является совокупностью последовательных и параллельных стадий. Существенную информацию о кинетике таких реакций можно получить с помощью метода импедансной спектроскопии. Однако интерпретация полученных результатов является сложной задачей, поскольку трудно получить решение конструируемых уравнений. Цель работы – определить кинетические константы элементарных стадий анодного процесса с помощью генетического алгоритма.

В работе [1] получены спектры импеданса растворения железного электрода в сульфатном (pH 1.3) электролите в области потенциалов -0.18÷-0.26 В. Спектры импеданса имеют индуктивный характер в низкочастотной области, причем число петель зависело от потенциала. На основании полученных спектров предложены электрические эквивалентые схемы [1], моделирующие импеданс железного электрода, и определены численные значения их элементов.

На основании схемы активного растворения железа в кислотах [2] и с помощью метода направленных графов получена передаточная дробно-рациональная функция [1] третьего порядка адмиттанса анодного процесса с тремя типами интермедиатов. Передаточная функция содержит константы скоростей и коэффициенты переноса элементарных стадий трех параллельных путей растворения металла. Найдено соответствие между ее параметрами и элементами электрической эквивалентной схемы. Полученные уравнения решали с помощью генетического алгоритма по схеме «мастер-рабочий» [3]. Использована схема ГА со стандартными генетическими операциями: скрещиванием, мутациями, кроссинговером мутацией, так же в наборах использовался элитизм(сохранение части лучших особей). Возможность распараллеливания алгоритма основа на том, что вычисление функции соответствия, включающие вычисления констант скорости и тафелевских коэфицентов, может быть вычислена независимо для особей в популяции на каждом шаге эволюционного процесса. Решение задачи строится по схеме «мастер-рабочий».

Вычислительные эксперименты проводились на кластере БашГУ (32 процессора AMD Opteron). Показатель эффективности распараллеливания выше 0.7, что говорит о хорошем параллелизме программы. В результате применение ПГА позволило рассчитать кинетические параметры изучаемых процессов.

Литература

1. Малеева М.А., Маршаков А.И., Рыбкина А.А., Елкин В.В. Физикохимия поверхности и защита материалов, 2008, 44, 587.
2. Keddam M., Mattos O.R., Takenouti H. J. Electrochem. Soc., 1981, 128, 257.
3. Л.А. Гладков, В.В. Курейчик Генетические алгоритмы: Учебное пособие. 2-е изд.. М: Физматлит, 2006. С.320.

Численное решение уравнения Больцмана на графических ускорителях

Е.А. Малков[1], М.С. Иванов[1], С. Полешкин[2]

ИТПМ СО РАН[1], НГУ[2]

В связи с развитием миниатюризации в различных областях техники становятся все более актуальными расчеты медленных течений разреженного газа. В связи со статистическими флуктуациями и медленной сходимостью методов прямого статистического моделирования, ПСМ-методов (DSMC-методов, Direct Simulation Monte Carlo), успешно применяемых для расчета сильно неравновесных течений, при расчетах медленных течений предпочтительными являются детерминированные конечно-разностные методы решения нелинейного уравнения Больцмана для одночастичной функции распределения $f = f(t, \mathbf{r}, \mathbf{v})$ в 6-мерном фазовом пространстве [1]:

$$\frac{\partial f}{\partial t} + \mathbf{v}\frac{\partial f}{\partial \mathbf{r}} = St(f, f). \tag{1}$$

В правой части уравнения стоит нелинейный интегральный оператор - интеграл столкновений, задаваемый интегрированием по области в восьмимерном пространстве $R^3 \times R^3 \times S^2$. В общем случае уравнение Больцмана представляет многомерную задачу с 7-ю независимыми переменными и его численное решение, с учетов необходимых вычислений интеграла столкновений, требует больших вычислительных затрат. Это обстоятельство объясняет сравнительно редкое использование методов численного решения уравнения Больцмана для расчетов течений разреженного газа по сравнению с ПСМ-методами. Однако, развитие технологии GPGPU должно привести к более широкому их применению. В связи с этим в последнее время к технологии CUDA проявляется большой интерес со стороны исследователей в области вычислительной аэродинамики [2, 3].

Естественным образом процесс переноса в фазовом пространстве расщепляется на два независимых физических процесса - свободно-молекулярный перенос и релаксацию, обусловленную столкновением молекул. В представляемой работе разработаны алгоритмы параллельных вычислений для решения бесстолкновительного уравнения Больцмана и вычисления интеграла столкновений, учитывающие особенности архитектуры CUDA, выполнена их программная реализация и проведены тестовые расчеты течения Куэтта на графических картах GeForce GTX 280 и NVIDIA Tesla M2090.

Литература

1. Коган М.Н. Динамика разреженного газа. Москва: Мир, 1978. 496 с.

2. Malkov E.A., Ivanov M.S. Parallelization of algorithms for solving the Boltzmann equation for GPU-based computations // 27th International Symposium on Rarefied Gas Dynamics July 10 – 15, 2010, Pacific Grove, California, USA. AIP Conf. Proc. 1333. 2010. P. 946–951.

3. Клосс Ю.Ю., Черемисин Ф.Г., Шувалов П.В. Решение уравнения Больцмана на графических процессорах // Вычислительные методы и программирование. 2010. Т. 11. С. 144–152.

Инструментальные средства поддержки языка Пифагор

И.В. Матковский

Сибирский Федеральный Университет

Пифагор представляет собой язык для написания функционально-потоковых параллельных программ [1, 2]. Отсутствие привязки к конкретной вычислительной системе и учета ресурсов позволяет создавать программы, описывающие максимальный параллелизм. С точки зрения системы программа на языке Пифагор может быть представлена в виде сочетания четырех слоев. Слой реверсивного информационного графа (РИГ) определяет зависимости по данным, из которых формируется исполняемая программа. В слое управляющего графа (УГ) описана последовательность передачи управляющих сигналов, определяющих готовность данных и необходимость запуска новых операций. Автоматный слой определяет текущее состояние вершин УГ; получающиеся по мере выполнения расчетов значения сохраняются в слое данных.

Система инструментальных средств поддержки состоит из транслятора, генератора УГ, интерпретатора и модульной библиотеки.

Транслятор принимает на вход программу на языке Пифагор и создает по ней РИГ. Сформированный РИГ полностью задает информационный слой программы и частично слой данных – заполняющийся за счет уже имеющихся константных выражений. Полученный РИГ транслятор может сохранить в одном из трех видов – текстовом, двоичном и графическом; хранятся промежуточные значения в модульной библиотеке. При необходимости сразу после формирования РИГ может быть проведена его оптимизация.

Генератор УГ принимает на вход двоичную форму РИГ и строит по ней УГ. Созданный УГ также сохраняется в модульной библиотеке в одном из трех видов – текстовом, двоичном и графическом.

Интерпретатору для выполнения функции нужны её РИГ и УГ и, при необходимости, двоичное представление входного аргумента функции. Поскольку формировать аргумент в двоичном виде напрямую неудобно, сохраненный в текстовом виде аргумент необходимо сначала обработать с помощью транслятора.

Промежуточные представления РИГ и УГ хранятся в модульной библиотеке. На данный момент библиотека представляет собой систему директорий; в дальнейшем планируются эксперименты над применением ряда популярных СУБД.

Описанная система позволяет писать и запускать программы на языке Пифагор. С помощью этой системы можно ставить эксперименты как над различными стратегиями управления вычислениями, так и над методами оптимизации функционально-потоковых параллельных программ. Инструментальные средства представляют собой набор независимых утилит; подобная модульность обеспечивает большую гибкость системы и упрощает отладку отдельных её элементов. В дальнейшем система может быть дополнена новыми инструментами – к примеру, средствами верификации и отладки.

Литература

1. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии : журнал. — 2005. — Т. 10. — № 1. — С. 71-89.

2. Легалов А.И, Непомнящий О.В., Матковский И. В., Фарков. М.А. Особенности преобразования и выполнения функционально-потоковых параллельных программ // Труды НПО 2011: материалы Ершовской конференции по информатике (Новосибирск, 27 июня – 1 июля 2011 г.). — Новосибирск: Институт систем информатики, 2011. — С. 146-153.

Программный комплекс клеточно-автоматного моделирования газопорошковых потоков*

Ю.Г. Медведев

Институт вычислительной математики и математической геофизики СО РАН

В работе рассмотрена двумерная модель газовой струи, несущей в себе нанокристаллические порошки FHP-GP [1]. Она является параллельной композицией [2] целочисленной модели FHP-MP [3], описывающей поведение газового компонента, и булевой модели FHP [4], описывающей поведение порошков.

Применять клеточные автоматы для моделирования газодинамических процессов стали не так давно. Клеточно-автоматные модели не доведены не только до практического применения, но также нет хотя бы удобного инструмента для их исследования. В настоящей работе представлен такой инструмент — программный комплекс, реализующий клеточно-автоматную модель FHP-GP и предназначенный как для последовательного, так и для параллельного исполнения.

Программный комплекс состоит из трех модулей: конструктора граничных условий, симулятора потока и модуля осреднения и визуализации и позволяет проводить вычислительные эксперименты с моделью FHP-GP, задавая различные параметры и начальные условия. Исходный текст программ написан на языке Си. Симулятор имеет параллельную реализацию с динамической балансировкой загрузки процессоров и может эффективно выполняться на тысячах ядер кластера [5]. Параллелизм реализован с помощью библиотеки MPI. Комплекс работает под операционными системами Windows и Linux.

Конструктор граничных условий позволяет задавать начальные и граничные условия в виде графических примитивов с атрибутами, соответствующими физическим и модельным величинам. Симулятор имеет набор предустанавливаемых параметров, позволяющих управлять ходом моделирования. При помощи модуля осреднения можно получить осредненные значения модельных величин в виде числовых массивов на любом этапе моделирования. Визуализатор позволяет строить графическое изображение поля давления газа, поля концентрации порошка, поля скорости потока, а также линии тока. Осреднение и визуализация результатов на протяжении процесса моделирования с определенным шагом по времени позволяют построить анимированные изображения протекания процесса.

В перспективе программный комплекс планируется расширить серией симуляторов других клеточно-автоматных моделей потока семейства FHP.

Литература

1. Медведев Ю.Г. Клеточно-автоматная модель формирования порошковой струи // Прикладная дискретная математика. 2009. №3. С. 50–58.
2. Bandman O.L. Cellular Automata composition techniques for spatial dynamics simulation // Bull. Nov. Comp. Center, Comp. Science, 27 (2008), P. 1–39.
3. Медведев Ю.Г. Многочастичная клеточно-автоматная модель потока жидкости FHP-MP // Вестник Томского государственного университета, серия «Управление, вычислительная техника и информатика» – №1(6) — 2009 С. 33–40.
4. Frish U., Crutchfield J.P., Hasslacher B., Lallemand p., Rivet L.-P. Lattice-Gas hydrodynamics in two and three dimensions // Complex Systems.– Vol.1.– 1987. – P.49–70.
5. Medvedev Yu. Dynamic Load Balancing for Lattice Gas Simulations on a Cluster // V. Malyshkin (Ed.): PaCT 2011, LNCS 6873, Springer-Verlag Berlin Heidelberg (2011), pp. 175–181.

* Работа выполнена при поддержке: гранта РФФИ №11-01-00567-а.

Рейтинг Топ50: тенденции*

Д.А. Никитенко, А.С. Антонов, С.И. Соболев

НИВЦ МГУ имени М.В.Ломоносова

Чтобы помочь правильно сориентироваться в мире высокопроизводительных вычислительных систем и иметь возможность оперативно отслеживать тенденции развития данной области, Межведомственный суперкомпьютерный центр РАН и Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова в мае 2004 года начали совместный проект по формированию списка 50 наиболее мощных компьютеров СНГ.

Рейтинг публикуется два раза в год: весной и осенью, т.е. со своеобразным квартальным сдвигом относительно объявления очередных редакций мирового рейтинга Top500. Это позволяет иметь четыре относительно равномерно распределенные по всему году контрольные точки, что особенно интересно и полезно в плане сравнения текущих тенденций в масштабах всего мира и характеров, темпов таких изменений в масштабе отечественной области HPC.

Так, анализ последних редакций списка Топ50 показывает: доминирование классических процессоров Intel, растущий интерес к GPU ускорителям, возобновление интереса к Ethernet как более доступному интерконнекту и т.д., чему и посвящен доклад.

Интернет-представительство проекта (*http://top50.supercomputers.ru)* изначально проектировалось с функционалом, аналогичным уже хорошо зарекомендовавшему себя на мировом уровне рейтингу Top500. По мере расширения функционала стали доступны изменения по разделам статистики по сравнению с предыдущей редакцией списка, появилась расширенная информация по системам с уникальной для держателей систем возможностью выделить самое значимое. К сожалению, далеко не всегда держатели систем находят время для подготовки наполнение для этого раздела. А его полнота, безусловно, интересна исключительно широкому кругу специалистов: студентам, исследователям, прикладным специалистам, разработчикам, т.е. — всем.

Хотелось бы подчеркнуть всеобщую заинтересованность не только в максимальной детализации сведений о действующих вычислительных системах, но и их актуальности. Не раз составители рейтинга сталкивались с ситуацией, когда система была расформирована или претерпела модернизацию, однако об этом можно было догадаться только по косвенным признакам. Поэтому составители рейтинга исключительно благодарны как представителям поставщиков, так и конечным держателям систем, которые сообщают открытые сведения о своих системах своевременно, информируют о любых неточностях, вносят новые предложения по расширению функционала, проявляют всяческую инициативу и не остаются безучастными к достоверности и полноте этого крайне важного для всех в отрасли ресурса.

Литература

1. Никитенко Д.А. Самые производительные и самые "зеленые" системы к концу 2010-го // Суперкомпьютеры. 2010, N4
2. Никитенко Д.А. Рейтинг Top50 как индикатор развития области HPC. // Материалы X международной конференции Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2010), г. Пермь
3. Никитенко Д.А. Top50 — рейтинг наиболее мощных суперкомпьютеров СНГ // Суперкомпьютеры. 2010, N2
4. Никитенко Д.А. Топ50. Рейтинг наиболее производительных вычислительных систем СНГ // Численные методы, параллельные вычисления и информационные технологии: Сборник научных трудов / Под ред. Вл.В.Воеводина и Е.Е.Тыртышникова. - М.: Издательство Московского Университета, 2008.

* Работа выполнена при частичной поддержке гранта РФФИ 10-07-00586-а.

Параллельный алгоритм разложения сейсмических данных по волновым пакетам: реализация и оптимизация для GPU

В.В. Никитин

Новосибирский государственный университет

Сейсморазведка является основным геофизическим методом, используемым при разведке залежей нефти и газа, а также других полезных ископаемых. Современные требования к точности разведки и рациональной разработке открытых месторождений приводит постоянному росту объемов получаемых сейсмических данных, которые требуют тщательной и, в то же время, оперативной обработки. Процесс предварительной обработки и сжатия сейсмических данных является весьма важным этапом работ, который требует применения высокопроизводительных вычислений. В контексте данного проекта рассматриваются процедуры: ***сжатие данных***, ***подавление шумов и нецелевых волн***, ***интерполяция и регуляризация данных***.

Для решения этих задач использовался (переопределенный) базис трехмерных волновых пакетов - локализованным плоским волнам [1], который является оптимальным для представления волновых полей. Тогда перечисленные задачи решаются простым применением прямого и обратного преобразования с сохранением только больших волновопакетных коэффициентов (в силу оптимальности базиса). Заметим, что кроме сейсмических приложений разложение по волновым пакетам может найти применение при выделении резких границ трехмерных изображений, например, в медицинской томографии.

Алгоритм прямого и обратного преобразования по трехмерным волновым пакетам был реализован на базе GPU фирмы NVidia при помощи технологии CUDA [2]. Проведено сравнение времени работы на разных вычислительных платформах - достигнуто ускорение до 45 раз на одной карте NVidia Tesla M2050 по сравнению с последовательным кодом. Рассмотрены два варианта распределения вычислений по процессорам, а затем при помощи закона Амдала проведён анализ масштабируемости для большого количества карт. В одном из вариантов с увеличением количества GPU наблюдается практически линейный рост производительности: в 7.7 раз для 8 карт.

Полученная программа тестировалась на синтетических сейсмических данных (процедуры сжатия, подавления шума и регуляризация трехмерных данных). В **таблице 1** представлены результаты тестирования программы для входных сейсмических данных размером 256^3.

Таблица 1. Время выполнения программы на различных платформах.

Платформа	Прямое пр-е (сек)	Обратное пр-е (сек)	Ускорение (прямое/обратное)
2x Tesla M2050	30,0	40,6	97/77
Tesla M2050	57,5	76,1	51/41
Tesla C2050	61,7	85,4	47/37
GeForce Quadro FX 4000	80,0	109,1	37/29
4x AMD Opteron 2218 (MPI)	1113,8	1140,1	3/3
Intel i7 (без оптимизаций)	2949,43	3137,04	1/1

Литература

1. Duchkov, A.A., Andersson, F.A., Hoop, M.V. Discrete almost-symmetric wave packets and multiscale geometrical representation of (seismic) waves // IEEE Transactions on Geoscience and Remote Sensing, Vol. 48, No. 9, 2010. - pp. 3408–3423.
2. V.V. Nikitin, A. A. Romanenko, A. A. Duchkov A.A., F. Andersson, 3D wave-packet decomposition implemented on GPUs //Expanded Abstracts, SEG Annual Meeting, 2011, pp. 3409-3413.

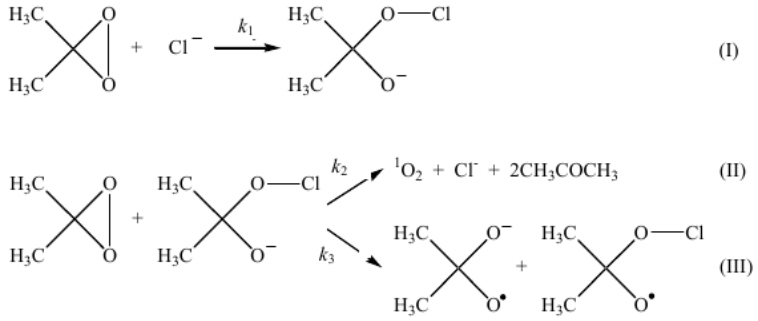
Моделирование генерации синглетного кислорода при разложении диметилдиоксирана с использованием технологии OpenMP

М.Ю. Овчинников[1], А.А. Юнусов[2], С.Л. Хурсан[1], И.М. Губайдуллин[3]

Институт органической химии УНЦ РАН[1], ГОУ ВПО БашГУ[2], Институт нефтехимии и катализа РАН[3]

Синглетный кислород $^1O_2$ на протяжении многих лет привлекает внимание исследователей ввиду его существенной роли в органическом синтезе, биологических процессах и хемилюминесцентных реакциях. Ранее была обнаружено, что распад диоксиранов, катализированный рядом анионов (Cl^-, Br^-, I^-, $t-BuO^-$,O_2^- и OH^-), сопровождается высокоэффективной генерацией $^1O_2$. Данный факт позволяет рассматривать систему диоксиран-нуклеофильный ион как перспективную в химических лазерах и органическом синтезе.

Была предложена схема взаимодействия ДМД(диметилдиоксирана) с хлорид-ионом, изображенная на рисунке.



Одной из особенностей данной системы является то, что во время проведения эксперимента единственной измеряемой величиной являлась интенсивность хемилюменисценции, определяемая по формуле

$$I_{cl} = \phi_{cl} w = \phi_{cl} k[Cl^-][DMD],$$

а не концентрации участвующих веществ.

Использование кинетического подхода в приближении метода квазистационарных концентраций позволило получить численное значение константы скорости k_1 и отношение констант k_2/k_3. Ввиду того, что значения констант k_2 и k_3 при данном подходе определить невозможно, было решено использовать подход обратных задач химической кинетики.

Обратная задача химической кинетики состоит в нахождении констант и активационных параметров по экспериментальным данным при известной системе дифференциальных уравнений, описывающих поведение системы. Она подразумевает решение большого количества прямых задач, заключающихся в решении системы обыкновенных дифференциальных уравнений. Ввиду этого, эта задача обладает большим потенциалом распараллеливания, к тому же, следующей задачу исследования является определение активационных параметров, для решения которой необходимо определить константы для определенного набора температур.

Применение подхода обратных задач и параллельных технологий позволило разделить константы, определить их с более высокой точностью. К тому же данный подход масштабируем на системы, описываемые более сложными системы дифференциальных уравнений, которые не имеют решения в аналитическом виде.

Вариационная организация технологии математического моделирования для решения природоохранных задач*

В.В. Пененко, Е.А. Цветова

Институт вычислительной математики математической геофизики СО РАН

Вариационный подход, зарекомендовавший себя как высокоэффективный инструмент построения численных моделей математической физики и алгоритмов для их реализации, предлагается использовать также для структурирования программного комплекса в целом, организации вычислительной технологии, включая разработку параллельных программ. Объектами применения вариационной технологии в настоящем докладе являются задачи природоохранного направления, имеющие междисциплинарный характер [1]. Они описываются сложными моделями, в которых участвуют большие массивы многомерных полей функций состояния и данных наблюдений различной природы. Поэтому создание экономичной технологии моделирования и системы обработки информации на каждом этапе вычислений представляет нетривиальную задачу.

Вариационный подход, предлагаемый авторами, как раз и предназначен для создания нового способа организации вычислений. Его преимущества обусловлены целостностью методики, позволяющей провести весь процесс математического моделирования от формулировки задачи до обработки результатов численных экспериментов с единых позиций. Инструментом, на основе которого производятся все построения, являются вариационные принципы. Они не только объединяют модели, различные по физическому содержанию и целевому назначению, но и обеспечивают согласованность всех этапов технологии математического моделирования и способов их практической реализации.

Для этих целей сформулирован вариационный принцип для оценки совокупности целевых функционалов прогнозирования, являющийся основой для построения универсальной технологии математического моделирования. С помощью вариационного принципа строятся численные схемы и алгоритмы для решения прямых и обратных задач по изучению изменений качества атмосферы и организации природоохранной деятельности в условиях изменяющего климата. При этом модели и алгоритмы организуются «сверху вниз», от более высоких системных уровней описания задач до более низких, и от общей структуры конкретных моделей до базовых элементарных алгоритмов их реализации.

Анализ проблемы в целом показывает, что можно организовать распараллеливание вычислений по нескольким системным уровням процесса моделирования. В рассматриваемом классе задач выделяются четыре основных системных уровня. Самый верхний из них организует набор сценариев, полностью повторяющих цикл вычислений прямых и обратных задач, но с разными параметрами и целевыми функционалами. Объединение здесь происходит лишь на этапе обработки конечных результатов. Самый нижний системный уровень реализует отдельные элементарные базовые схемы в рамках методов расщепления и декомпозиции. В рамках вариационного подхода выявляется множество различных вариантов сочетаний алгоритмов последовательного и параллельного действия, а также способов их оптимизации.

Таким образом, вариационный подход, используемый авторами, делает прозрачной всю систему взаимосвязей в технологии моделирования, что позволяет структурировать систему и унифицировать отдельные элементы, тем самым рационально организовать вычисления и информационные потоки.

Литература

1. V. Penenko, A.Baklanov, E. Tsvetova and A. Mahura. Direct and Inverse Problems in a Variational Concept of Environmental Modeling //Pure and Applied Geophysics. 2011 DOI: 10.1007/s00024-011-0380-5, pp. 1-19.

* Работа поддержана проектом РФФИ 11-01-00187-а, а также Программами фундаментальных исследований ОМН РАН №3 и Президиума РАН №4.

Алгоритмы распределения ресурсов в системе фрагментированного программирования LuNA

В.А. Перепелкин

Институт вычислительной математики и математической геофизики СО РАН

Программа в системе LuNA представляется как множество информационно-зависимых фрагментов вычислений, обрабатывающих множество фрагментов данных. Распределение фрагментов по узлам мультикомпьютера во времени определяет эффективность исполнения программы. В системе LuNA эта задача в значительной степени решается автоматически. В работе представлены и исследованы алгоритмы, конструирующие описанное распределение в системе LuNA.

В области высокопроизводительных вычислений задача распределения ресурсов мультикомпьютеров (например, кластеров) – памяти, процессорного времени и коммуникационной подсистемы – является сложной задачей системного параллельного программирования. Актуальна задача автоматизации (частичной или полной) распределения ресурсов в параллельных программах.

Технология фрагментированного программирования (ТФП) – это развивающаяся технология разработки параллельных программ, реализующих большие численные модели для суперкомпьютеров, одна из основных задач которой – обеспечение высокой эффективности реализаций прикладных алгоритмов (эффективность понимается в смысле равномерного и полного использования ресурсов мультикомпьютера с целью уменьшения общего времени работы программы).

Существенной особенностью ТФП является то, что параллельное программа явно разделена на две части, одна из которых – описание прикладного алгоритма, а вторая – задание способа его исполнения (т.е., распределения ресурсов во времени). Такое разделение позволяет манипулировать распределением ресурсов программы (и, соответственно, её эффективностью) не затрагивая собственно вычисления алгоритма.

В работе поставлена задача автоматического (или полуавтоматического) конструирования распределения ресурсов для заданных программы и класса мультикомпьютеров в системе фрагментированного программирования LuNA. Круг прикладных алгоритмов ограничивается регулярными вычислениями на сетках и сходных структурах данных со значительной долей массовых вычислений (однотипных независимых операций). Кроме того, используется подход с «подсказками» человека о том, какими свойствами должно обладать распределение ресурсов. С учетом этих подсказок и строится требуемое распределение.

В работе предложены и исследованы два алгоритма, использующие подсказки в виде «опорных точек» распределения и с разбиением программы на «секции». Проведено тестирование, которое показало эффективность автоматически сконструированных распределений ресурсов близкую к эффективности распределений, построенных вручную.

Литература

1. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 127 с.
2. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. – In the Int. Journal on Future Generation Computer Systems. Vol. 17 (2001), No. 6, pp. 755–765
3. Топорков В.В. Модели распределенных вычислений. – М.: ФИЗМАТЛИТ, 2004. – 320 с. – ISBN 5-9221-0495-0

О распараллеливании решения краевых задач на квазиструктурированных сетках.

Б.Д. Рыбдылов, В.М. Свешников

Институт вычислительной математики и математической геофизики СО РАН

Рассматриваются технологические аспекты решения краевых задач на предлагаемых квазиструктурированных сетках специального вида. Их особенностью является то, что и макросетка в расчетной области, и подсетки в подобластях являются структурированными и прямоугольными сетками, что обеспечивает создание экономичных структур данных и эффективное применение численных алгоритмов. В то же время, результирующая квазиструктурированная сетка является адаптивной к неоднородностям внутри области и к сложной конфигурации внешней границы. Решение ищется предлагаемым вариантом метода декомпозиции[1], который основан на отдельной аппроксимации краевой задачи на интерфейсе и в подобластях.

Адаптивные квазиструктурированные сетки, рассматриваемые в настоящей статье, имеют ряд преимуществ по сравнению со структурированными и неструктурированными сетками. По отношению к первым они дают возможность избавиться от лишнего числа узлов, которые зачастую вводятся лишь для поддержки структурированности, а по отношению ко вторым имеют гораздо более простую структуру данных небольшого объёма, что позволяет более эффективно строить численные алгоритмы.

Процесс распараллеливания алгоритмов решения краевых задач на квазиструктурированных сетках имеет некоторые особенности, связанные с равномерной загрузкой процессоров. Обычно используемое при распараллеливании отображение одна подобласть – один процессор в этом случае недопустимо вследствие того, что подсетки в подобластях могут быть несогласованными, то есть иметь существенно различное число узлов. Это приводит к разбалансировке загрузки процессоров. Для её выравнивания предлагается группировать подобласти в объединения, имеющие приблизительно одинаковое число узлов, что привносит некоторые изменения в технологию распараллеливания, так как в объединение могут входить как подобласти, не требующие межпроцессорных пересылок, так и подобласти, для которых они необходимы. В последнем случае обмены происходят с процессорами-соседями, номера которых должны быть предварительно определены и сохранены.

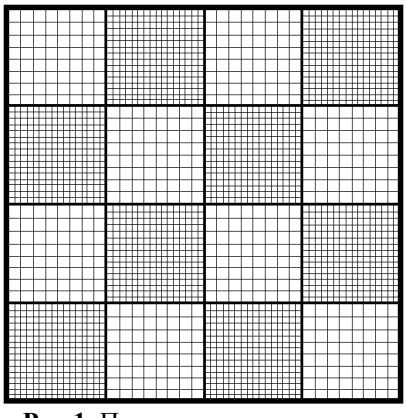


Рис.1. Пример квазиструктурированной несогласованной сетки

С целью исследования эффективности данного нового подхода был проведён цикл численных экспериментов на модельной краевой задаче для уравнения Лапласа с известным аналитическим решением. Расчеты проводились на кластере НКС-30Т Сибирского суперкомпьютерного центра СО РАН. Расчётная область покрывалась квазиструктурированной сеткой, представляющей собой объединение подсеток, которые строились в шахматном порядке, а именно соседние подобласти имели по каждому направлению число узлов, отличающееся в два раза, то есть были несогласованными. На рис.1 приведён пример такой сетки, включающей подсетки с числом узлов 8x8, 16x16. Подсетки группировались в объединения естественным образом: слева направо, снизу вверх. Все объединения имели одинаковое число узлов.

Проведенные расчеты показали высокую эффективность распараллеливания и позволили сделать рекомендации по выбору параметров декомпозиции при расчетах на сетках с большим числом узлов:

Литература

1. Свешников В.М. Построение прямых и итерационных методов декомпозиции // СибЖИМ. 2009. Т.12, № 3(39). С. 99 – 109.

Специализированные компактные высокопроизводительные вычислительные системы

А.С. Рыбкин, Т.А. Агапова, И.А. Крючков, Ю.В. Логвин, А.Г. Ломтев, Р.М. Шагалиев, В.В. Южаков

РФЯЦ-ВНИИЭФ, Саров

На текущий момент в мире уже созданы и эксплуатируются высокопроизводительные гибридные вычислительные системы с арифметическими ускорителями на базе графических процессоров. В РФЯЦ-ВНИИЭФ ведутся интенсивные исследования по применению арифметических ускорителей для научных расчетов. Работы выполняются с использованием:

- NVIDIA CUDA на графических процессорах G80, GT200, GF100/GF110, специализированных серверных модулях Tesla S1070/S2050 выполненных на основе процессоров T10 и T20;
- OpenCL на графических процессорах AMD RV870, Cypress, Cayman, а так же универсальных процессорах фирм Intel и AMD.

С 2009 года в РФЯЦ-ВНИИЭФ разработана серия компактных гибридных вычислительных систем, таких как ГВС-10 «Кубань», ГВС-14 и ГВС-18А. Потребляемая мощность не превышает 1,5 кВт, а уровень акустического шума менее 39 дБА. Теоретическая пиковая производительность, например, ГВС-18А более 3,5 ТФлоп/с. Для сравнения, универсальная компактная супер-ЭВМ АПК-1 [1] обладает производительностью 1,044 ТФлоп/с. Высокие технические характеристики и низкая стоимость выделяют их на фоне аналогичных разработок.

В мире постоянно растет количество прикладных программ, способных использовать в полной мере возможности специализированных компактных вычислительных машин с арифметическими ускорителями. Сотрудниками РФЯЦ-ВНИИЭФ в интересах атомной промышленности уже разработаны и адаптированы программные комплексы для моделирования большого класса физических процессов методами Монте-Карло [2] и молекулярной динамики.

Метод Монте-Карло используется для расчёта эффективного коэффициента размножения нейтронов активных зон ядерных реакторов, обоснования радиационной и ядерной безопасности транспортных упаковочных комплектов для перевозки и долговременного хранения отработанного ядерного топлива АЭС. Например, длительность численных исследований активной зоны реактора ВВЭР-1000 на ГВС-14 уменьшена по сравнению с двенадцатиядерным универсальным процессором AMD Opteron 6180 SE в 7 раз.

Для атомной промышленности и энергетики актуальны задачи влияния облучения на механические свойства металлов и их сплавов. На данном классе задач (молекулярная динамика) ускорение вычислений составляет более 10 раз.

Для моделирования задач физики взрыва и удара хорошо себя зарекомендовал программный комплекс MASTER Professional. Наиболее требовательными в вычислительным ресурсам являются задачи трехмерного моделирования. Получено ускорение вычислительного процесса до 20 раз по сравнению с универсальным процессором.

Совместно с пользовательскими программами специализированная компактная вычислительная система предоставляет высокопроизводительный аппаратно-программный комплекс для моделирования различных задач математики и физики.

Литература

1. Стрюков В.Н., Бартенев Ю.Г., Басалов В.Г., Варгин А.М., Вялухин В.М. и др. Универсальная компактная супер-ЭВМ // Супервычисления и Математическое Моделирование. Труды XII международного семинара / Под ред. Р.М. Шагалиева. – Саров: ФГУП-«РФЯЦ-ВНИИЭФ», 2011. – с. 329-331.
2. Рыбкин А.С., Залялов А.Н., Малькин А.Г., Огнев С.П., Рослов В.И. Программный комплекс на базе гибридных вычислительных систем для расчета критических параметров методом Монте-Карло // Супервычисления и Математическое Моделирование. Труды XII международного семинара / Под ред. Р.М. Шагалиева. – Саров: ФГУП-«РФЯЦ-ВНИИЭФ», 2011. – С. 310-315.

Решение уравнения Хохлова-Заболоцкой-Кузнецова на многопроцессорной вычислительной системе с распределенной памятью

О.А. Савицкий, Т.А. Чистякова, А.В. Шишеня

Технологический институт Южного федерального университета в г. Таганроге

Математическое моделирование является мощным инструментом изучения сложных нелинейных волновых процессов [1]. Модели звуковых пучков и их программные реализации находят практическое применение в гидроакустике, неразрушающем контроле, медицинской диагностике.

Для описания распространения звуковых пучков конечной амплитуды в нелинейно-диссипативной среде использовано уравнение Хохлова-Заболотской-Кузнецова:

$$\frac{\partial}{\partial\theta}\left(\frac{\partial v}{\partial z}-v\frac{\partial v}{\partial\theta}-\Gamma\frac{\partial^2 v}{\partial\theta^2}\right)=\frac{N}{4}\Delta_\perp v$$

с начальным условием: $v(0,\theta,r)=V(\theta,r)$

В результате применения метода расщепления по физическим процессам, исходное уравнение заменяется двумя дифференциально-разностными аналогами уравнения Бюргерса и параболического уравнения квазиоптики. Проведено исследование устойчивости и сходимости построенной дискретной математической модели к решению уравнения Хохлова – Заболотской – Кузнецова. Разностная схема устойчива и монотонна при следующих ограничениях на шаги по пространству[2]:

$$h_\theta \le \frac{2\Gamma}{|u|},\ h_z \le \frac{h_\theta^{\,2}}{2\Gamma(1-\sigma)},$$

где Г - диссипативный параметр, $|u|$ - характерный масштаб возмущения скорости среды, h_z, h_θ - шаги дискретизации по пространству и времени, σ- вес разностной схемы.

Погрешность аппроксимации математической модели равна $O(h_z + h_\theta^2)$. Разностная схема абсолютно устойчива при: $\sigma \ge 1/2$ и имеет второй порядок погрешности аппроксимации при $\sigma = 1/2$. Построенная дискретная модель соответствует ее непрерывному аналогу с точки зрения баланса энергии и является консервативной.

Параллельная реализация алгоритма выполнена методом геометрического параллелизма по двум направлениям, что требует использования $PN \cdot PM$ процессоров. Прямое БПФ реализовано на основе прореживания по частоте, а обратное – на основе прореживания по времени, в этом случае, при работе с данными между прямым и обратным преобразованием не требуется выполнения передач для переупорядочивания элементов, но следует учитывать порядок их следования. Отладка и выполнение программной реализации решения уравнения Хохлова-Заболоцкой-Кузнецова проводились на кластере ТТИ ЮФУ.

Разработанный на основе изложенного алгоритма программный пакет успешно применяется как для научных исследований, так и для инженерных расчетов, требующих использования подробных сеток. С помощью построенной модели выполнен численный эксперимент и доказана возможность существования фокусировки звуковых пучков [3].

Литература

1. Чистякова Т.А. Дискретная конечно-разностная модель распространения волновых пучков, описываемая квазилинейным уравнением параболического типа. Известия ЮФУ. Технические науки. – Таганрог: Изд-во Технологического Института ЮФУ, 2009. – С. 118-129.
2. Самарский А.А. Теория разностных схем. М., Наука, 1989.
3. Савицкий О.А. Пространственные нелинейные эффекты при взаимодействии волн с различными временными масштабами, Сборник трудов XXII сессии РАО, Т.1., с.204-208. М.: ГЕОС. 2010г.

Моделирование аппаратной архитектуры и коммуникационных сетей вычислительных кластеров с гибридными узлами для параллельных систем баз данных*

Ю.Н. Сафина, П.С. Костенецкий

Южно-Уральский государственный университет

На сегодняшний день наиболее распространены высокопроизводительные вычислительные комплексы с кластерной архитектурой: из 500 мощнейших суперкомпьютеров мира 410 построены на основе кластерной архитектуры. С 2010 года в перовой десятке рейтинга ТОП 500 появились гибридные вычислительные комплексы с узлами, включающими графические ускорители. Использование *GPU* позволяет добиться высокой производительности на задачах связанных с решением СЛАУ, в том числе на тесте *Linpack*. Существуют исследования, оценивающие производительность подобных систем под нагрузкой, сходной с нагрузкой, создающейся при выполнении запросов к параллельным системам баз данных [1].Во втором квартале 2010 г. корпорация Intel анонсировала ускорители с архитектурой *Intel Many Integrated Core* (*MIC*), позволяющей создавать платформы выполняющие триллионы операций в секунду, в тоже время, сохраняя преимущества стандартных процессоров Intel. Сортировка – это один из ключевых алгоритмов для многих операций обработки баз данных. В работе [2] представлено сравнение эффективности выполнения различных алгоритмов сортировки, показывающее значительную эффективность на ускорителях *MIC* по сравнению с *CPU* и*GPU*.Использование *MIC* позволяет ускорить в несколько раз производительность поиска в базах данных по сравнению с традиционными архитектурами и графическими процессорами.

Данное исследование посвящено моделированию и анализу иерархических конфигураций аппаратных архитектур параллельных систем баз данных, построенных на основе вычислительных узлов с ускорителями *MIC*.Исследования проводятся с использованием эмулятора многопроцессорных иерархических машин баз данных *DMS*, позволяющего моделировать различные аппаратные архитектуры мультипроцессоров баз данных [3].Исследуется эффективность различных иерархических многопроцессорных конфигураций, использующих *MIC*, и оценивается оправданность использования подобных систем для организации мультипроцессоров параллельных систем баз данных. Так же исследованиедолжно позволить определить оптимальнуютопологию коммуникационной сети, наиболее полно удовлетворяющую задачам обработки в параллельных системах баз данных ицелесообразность использования в подобных архитектурах типовых параллельных систем хранения данных. В 2012 году планируется собрать найденные оптимальные архитектуры из серверов с ускорителями *MIC*и выполнить тестирование полученных систем для подтверждения результатов моделирования.

Литература

1. Bakkum P., Skadron K. Accelerating SQL Database Operations on a GPU with CUDA // Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU'10), Pittsburgh, USA, March 14, 2010. ACM, 2010. P. 94-103.
2. Satish N., Kim C., et.al. Fast Sort on CPUs, GPUs and Intel MIC Architectures.Technical Report. IntelCorporation, 2010.
3. Костенецкий П.С. Моделирование параллельных систем баз данных для вычислительных кластеров // Научный сервис в сети Интернет: Труды Всероссийск. науч. конф. (21-26 сентября 2009 г., г. Новороссийск). М.: Изд-во МГУ. 2009. С. 300-304.

* Работа выполнена при финансовой поддержке Минобрнауки РФ (государственный контракт № 07.514.11.4036) и Российского фонда фундаментальных исследований (проект 12-07-00443-а).

Разработка библиотеки параллельного программирования для компьютеров с общей памятью для языка Common Lisp

А.П. Свиридов

Южно-Уральский государственный университет

Большинству современных компьютеров от персональных до супербольших характерен параллелизм на различных уровнях: одновременно функционирует множество процессоров, передаются данные по коммуникационной сети, работают устройства ввода/вывода, осуществляются другие действия. Любой параллелизм направлен на повышение эффективности использования компьютера. В настоящее время появляется все большее количество специализированных языков программирования для написания параллельных программ, однако наибольшей популярностью пользуются системы параллельного программирования, расширяющие возможности традиционных языков программирования в области создания параллельных программ. Яркими примерами таких систем являются OpenMP и MPI [1].

Целью представляемой работы является разработка библиотеки параллельного программирования для компьютеров с общей памятью для языка Common Lisp [1], которая должна предоставлять следующие инструменты:

– базовые конструкции параллельного программирования parallel и parallel-sections;

– параллельные аналоги для циклов dotimes, for и dolist;

– параллельные аналоги для семейства отображающих функционалов map, функции reduce, функции sort, семейства фильтрующих функций remove, семейства функций предикатов (or, and, some, any и др.);

– средства синхронизации потоков;

– пул потоков, минимизирующий количество операций создания потока;

– набор атомарных операций (инкремент, декремент, добавление и извлечение элемента из списка), ориентированных на конкретную платформу;

– набор низкоуровневых функций общего назначения.

Базовые конструкции parallel и parallel-sections должны иметь различные параметры конфигурирования, такие как количество потоков выполняющих параллельный регион, правила синхронизации между потоком, инициализировавшим параллельные вычисления, и побочными потоками, локальные переменные и др. Также данные конструкции должны поддерживать команды параллельного региона, такие как for-initial-thread, barrier и др.

Параллельные аналоги для циклов помимо возможностей parallel должны поддерживать дополнительные параметры конфигурирования, такие как стратегия распределения порций итераций между потоками в цикле, размер порций и др.

С целью повышения производительности, разрабатываемая библиотека должна быть реализована с учетом особенностей каждой отдельной реализации Common Lisp, а также с учетом особенностей различных операционных систем.

Литература

1. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. Спб.:БХВ-Петербург, 2002. С. 221-266.
2. Guy L. Steel. Common LISP. The Language. Second Edition. Digital Press; second edition (June 15, 1990). P. 1029.

Исследование эффективности хранения и обработки баз данных в графической памяти видеокарт с поддержкой CUDA*

А.И. Семенов, П.С. Костенецкий

Южно-Уральский государственный университет

Исследования, посвященные обработке баз данных в оперативной памяти, известны достаточно давно, нов связи с появлением технологии CUDA возникло новое направление исследований, посвященное обработке баз данных с использованием графических ускорителей [1]. В данном направлении известны работы, посвященные ускорению обработки запросов к базе данных [3] и оптимизации процесса интеллектуального анализа данных (data mining) [2]. Так как пропускная способность видеопамяти достигает 327 ГБ/с [5], что примерно в 3 раза превышает среднюю скорость работы с оперативной памятью [4], возникает вопрос эффективности не только обработки, но и размещения базы данных непосредственно в графической памяти. На сегодняшний день на рынке доступны гибридные вычислительные серверы с объемом видеопамяти, достигающим 16 Гб. Таких объемов памяти обычно достаточно для эффективного хранения и обработкибольшинства баз данных. Для увеличения объема хранимой в видеопамяти базы данных, отдельные серверы могут бытьобъединены в вычислительные кластеры. Например, объем графической памяти суперкомпьютера Tianhe-1A, находящегося на второй позиции ТОП 500, составляет 21.5 ТБ. Кроме того, при необходимости можно реализовать механизмы загрузки блоков данныхиз оперативной памятивычислительных узлов.

В даннойработе описывается незаконченное исследование, посвященноеоценкеэффективности хранения и обработки баз данных непосредственно в графической памяти видеокарт с поддержкой технологии программирования CUDA. На текущий момент авторами разработана система, моделирующая выполнение запроса JOINнепосредственно в памяти GPU ускорителя. Для хранения отношений базы данных, над которым выполняется запрос, используется глобальная памятьмодели программирования CUDA. Нитям необходимы очень частые обращения к их исполняемому программному коду и метаданным, поэтому для размещения этих данных используется константная память, имеющая значительно меньшую латентность, чем у глобальной памяти. Локальная память являетсяабстракцией модели программирования CUDA и означает блок памяти отдельной нити, расположенный в глобальной памяти. Локальная памятьиспользуется для хранения промежуточных данных во время выполнения запроса. Регистровая память используется для хранения локальных переменных нитей.Текстурная память на текущем этапе работы не используется.

Следующим этапом работы будет выполнение вычислительных экспериментов с целью исследования эффективности использования памяти GPU для хранения и обработки баз данных.

Литература

1. Bakkum P., Skadron K. Accelerating SQL Database Operations on a GPU with CUDA // Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU'10), Pittsburgh, USA, March 14, 2010. ACM, 2010. P. 94-103.
2. Fang W., Lau K.K., Lu M., et al. Parallel Data Mining on Graphics Processors // Technical Report HKUST-CS08-07, 2008. P. 1-10.
3. Govindaraju N., Lloyd B., Wang W., et al. Fast computation of database operations using graphics processors. In SIGGRAPH ’05: ACM SIGGRAPH 2005 Courses, New York, NY, USA, 2005. ACM. P. 206.
4. Measuring Memory Bandwidth (White Paper). Intel Corporation. URL: http://www.intel.com/performance/resources/briefs/memband.pdf (датаобращения: 02.12.2011). 2010.
5. Спецификации NvidiaGeForceGTX 590. URL: http://www.nvidia.ru/object/product-geforce-gtx-590-ru.html (дата обращения 01.12.2011). 2011.

* Работа выполнена при финансовой поддержке Минобрнауки РФ (государственный контракт № 07.514.11.4036) и Российского фонда фундаментальных исследований (проект 12-07-00443-а).

Математическая модель транспортных потоков на основе схемы с двумя масштабами времени

В.А. Соловьев, И.С. Сунгуров, Р.Т. Файзуллин

Омский Государственный Технический Университет

Существующая реализация программного комплекса позволяет моделировать движение транспортных средств для произвольных городов. На данный момент проведено успешное испытание программы на картах таких городов как Омск, Новосибирск, Томск и Москва.

Данный комплекс состоит из двух частей: собственно имитатора дорожного движения (C++) и графического интерфейса (Java), позволяющего в реальном времени отслеживать результаты моделирования. Имитатор представляет собой распределённое приложение, реализованное с использованием протокола обмена сообщениями MPI. Он состоит из одного центрального узла и некоторого количества вычислительных узлов, каждый из которых обрабатывает определённый участок города.

На первом этапе происходит распределение нагрузки по вычислительным узлам. Для этого центральный узел загружает карту из файла, а затем с помощью алгоритма кластеризации (немодифицированный алгоритм k-means) разбивает её на области и назначает каждую область вычислительному узлу. Закон движения, описывающий изменения положения и скорости транспортного средства на каждом шаге, подробно рассмотрен в [1].

При разработке параллельной версии программы моделирования транспортных потоков естественным образом возникает задача разбиения карты города на регионы. Основными критериями разбиения при этом являются: фиксированное количество регионов (по количеству вычислительных узлов); примерно одинаковая общая длина дорог в каждом регионе; наименьшее количество связей между регионами.

Апробация модели проводилась на компьютерах с двухядерными процессорами с частотой 2,8ГГц на каждое ядро. Получены результаты работы алгоритма кластеризации применительно к имеющимся картам городов, характеризующие количество связей между кластерами, количество дорог в каждом кластере и т.д. Кроме того определены следующие характеристики исследуемых карт городов: количество дорог, количество перекрёстков, общая длина дорог, средняя длина дороги, максимальная длина дороги. Вводится такое понятие как виртуальная секунда. Под виртуальной секундой понимается одна секунда в модели, а характеризуется она коэффициентом ускорения времени ε, показывающим во сколько раз время в модели течет быстрее, чем в действительности. Чем ε больше, тем лучше. Набрана статистика показывающая отношение времени в модели к времени в действительности при различной загрузке. Выявлено, что скорость обработки одного шага модели изменяется линейно в зависимости от количества процессоров (в определенных условиях).

Данная модель позволяет поставить задачу нахождения оптимальной работы светофоров для максимальной «прокачки» транспортных средств через регион [1], которая в некотором смысле является развитием задачи поиска так называемой «зеленой волны».

Литература

1. Соловьев, В. А. Математическое моделирование транспортных потоков на основе схемы с двумя масштабами времени / В.А. Соловьев, Р.Т. Файзуллин // Омский научный вестник. Сер. Приборы, машины и технологии. — 2011. — №3(103). — С. 37–40.

Применение распределённого по замкнутому тору алгоритма LU-разложения к решению упругопластической задачи*

А.В. Толмачев, А.В. Коновалов, А.С. Партин

Институт Машиноведения УрО РАН

Решение упругопластической задачи с большими пластическими деформациями выполняется методом конечных элементов пошагово. На каждом шаге нагрузки оно состоит из трёх этапов: 1) расчёт локальных матриц жёсткости и формирование матрицы жёсткости A и вектора правой части b системы линейных алгебраических уравнений (СЛАУ) относительно искомого вектора x обобщённой скорости в узлах конечно-элементной сетки, а именно

$$Ax = b; \qquad (1)$$

2) решение СЛАУ (1); 3) вычисление напряжённо-деформированного состояния конечных элементов в конце шага нагружения.

Использовали параллельный алгоритм решения СЛАУ, описанный в работе [1]. Он является модификацией параллельного алгоритма блочного LU-разложения для полной матрицы [2]. Суть модификации в том, что производится хранение только блоков с ненулевыми элементами.

В качестве тестовых задач выбраны задачи сжатия цилиндра и параллелепипеда. Вычисления проводились на кластере «Уран» Института математики и механики УрО РАН. При решении задач в двухмерной задаче взяли конечноэлементную сетку с количеством разбиений по каждой координате $d = 100$, 200 и 300; в трёхмерной задаче соотвественно $d = 5$,10,15,20,25.

В двухмерной задаче для конечноэлементной сетки с количеством разбиений $d = 300$ на этапе разложения матрицы наблюдается рост ускорения при использовании до 16 процессоров. Максимальное ускорение в этом случае достигает 2,5. Решение СЛАУ с разложенной матрицей показывает наилучшие результаты при использовании 8 процессоров. Ускорение при этом достигает 1,2. Из-за малости полуширины ленты матрицы A, затраты на межпроцессорное взаимодействие уменьшают выигрыш от распараллеливания. В трёхмерной задаче с сеткой $d = 25$ на этапе разложения матрицы при использовании 16 процессоров наблюдается ускорение в 6,5 раз по сравнению с использованием 1 процессора. При решении СЛАУ с разложенной матрицей ускорение составляет 1,7. Увеличение быстродействия по сравнению с двухмерной задачей объясняется большей полушириной ленты матрицы жёсткости. В этом случае доля времени, затрачиваемая на вычисления, больше доли времени, затрачиваемой на межпроцессорные коммуникации.

Литература

1. D. W. Walker, T. Aldcroft, A. Cisneros, G. C. Fox, and W. Furmanski, LU decomposition of banded matrices and the solution of linear systems on hypercubes. In Proceedings of the third conference on Hypercube concurrent computers and applications, Vol. 2. ACM, 1989, New
2. J. Choi, J. J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines. Scientific Programming, 1996, pp. 173-184.

* Работа выполнена в рамках программы Президиума РАН «Информационные, управляющие и интеллектуальные технологии и системы»

Архитектура суперкомпьютера для решения целочисленных задач с нерегулярной интенсивной работой с памятью

А.С. Фролов, Д.В. Мошкин, А.С. Семенов

ОАО «НИЦЭВТ»

С развитием глобальных информационных систем (социальные сети, ERM-системы, поисковые машины) задачи, связанные с обработкой и анализом данных большого объема (проблема "Больших Данных" или Big Data) становятся все более актуальными и требуют повышения эффективности их решения за счет применения новых программных и аппаратных технологий, претерпевающих в настоящее время фазу интенсивного роста и повышенного внимания со стороны сообщества высокопроизводительных систем. Одним из примеров задач из области Big Data являются графовые задачи. Существуют различные подходы к построению суперкомпьютеров, эффективно решающих графовые задачи [1].

В данной работе рассматривается архитектура гетерогенного суперкомпьютера, ориентированного на повышение продуктивности решения задач целочисленной обработки с интенсивной работой с памятью большого объема (более Пбайта). В основе предлагаемой архитектуры лежит вычислительный узел, состоящий из одного или нескольких коммерчески доступных суперскалярных многоядерных микропроцессоров и коммуникационного сопроцессора, встроенного в адаптер высокоскоростной сети.

Главными архитектурными особенностями сопроцессора являются: аппаратная мультитредовость, поддержка глобальной общей памяти, поддержка активных сообщений. Архитектура мультитредового коммуникационного сопроцессора с кодовым названием J7-C построена на основе архитектуры мультитредово-потокового процессора J7, разрабатывавшегося в ОАО «НИЦЭВТ» в рамках проекта «Ангара» в 2005-2010 г. [2]. Однако за счет того, что J7-C является сопроцессором, архитектура J7-C является значительно более простой по сравнению с J7. Также в J7-C добавлена поддержка обеспечения функций взаимодействия с центральным процессором вычислительного узла. В качестве интерконнекта будет использована разрабатываемая отечественная высокоскоростная коммуникационная сеть EC8430 с топологией 4D-тор [3]. При этом в 2012 будет разработан первый прототип СБИС маршрутизатора на технологии 65 нм.

Возможны различные варианты размещения мультитредового коммуникационного сопроцессора в адаптере коммуникационной сети. Первый вариант – размещение сопроцессора в виде отдельной микросхемы на плате адаптера, второй – размещение сопроцессора внутри СБИС маршрутизатора. Предполагается последовательная реализация обоих вариантов.

Главным отличием рассматриваемого подхода к построению гетерогенного суперкомпьютера для графовых задач от подходов, применявшихся в Cray XMT и НИЦЭВТ «Ангара» – программная модель, предусматривающая выполнение задачи на многоядерно-суперскалярной и мультитредровой частях системы. Для этого прорабатывается программная модель с рабочим названием GAPL (Graph Application Programing Language), построенная в виде расширения языка Си в стиле CUDA и OpenCL.

В настоящий момент завершается разработка принципов работы J7-C, планируется создание полнофункционального параллельного эмулятора и стека программного обеспечения (драйвер, библиотеки, компилятор Си и ассемблер).

Литература

1. А.Фролов, А.Семенов, А.Никитин, А.Мошкин, В. Кабыкин, Д.Мошкин Суперкомпьютеры для графовых задач. «Открытые системы», №7, 2011

2. Л.Эйсымонт, А.Слуцкин, Российский суперкомпьютер с глобально адресуемой памятью. «Открытые системы», №9, 2007

3. Д.Макагон, Е.Сыромятников Сети для суперкомпьютеров. «Открытые системы», №7, 2011

Макетирование и оценочное исследование производительности мультитредового сопроцессора J7-C

А.С. Фролов, Д.В. Мошкин, М.А. Белкин, К.А. Курочкин,
В.С. Бобков, И.И. Долженков, А.С. Садчиков

ОАО «Научно-исследовательский центр электронной вычислительной техники»

В последнее время все более актуальными становятся графовые задачи, связанные с обработкой и анализом данных большого объёма, вынуждающие применять новые программные и аппаратные решения, повышающие эффективность их счета [2]. В качестве одного из таких решений в настоящее время в ОАО «НИЦЭВТ» проводятся работы по созданию коммуникационного сопроцессора J7-C с поддержкой глобально адресуемой памяти на базе разрабатываемого с 2005 года микропроцессора J7 [1]. Главными архитектурными особенностями сопроцессора являются: аппаратная мультитредовость, поддержка глобальной общей памяти, поддержка активных сообщений, сопряженность с маршрутизатором коммуникационной сети «Ангара» с топологией 4D-тор [3].

Сопроцессор будет содержать кэш команд IC (64 КБ, 4-х ассоциативный, длина строки 64Б), 2 или 4 целочисленных конвейера (в каждом по 16 тредов), блок 64-х разрядного целочисленного умножения и деления MDU (один на каждую пару конвейеров), блок обращения в память LSU (один на каждую пару конвейеров, до 8-ми незавершённых команд от каждого треда), кэш команд и данных второго уровня L2 (4МБ, 4-х ассоциативный, длина строки 64Б), блок трансляции адреса MMU, блок сетевых обращений MSU, предоставляющий интерфейс к маршрутизатору RT, блок управления CU, выполняющий команды с хост-машины.

Для отработки микроархитектуры сопроцессора J7-C, запланирована серия макетов на ПЛИС с поэтапным добавлением блоков сопроцессора.

На данный момент реализован макет M2 на основе FPGA Xilinx XC5VFX100T, содержащий: IC (только физическая адресация), один целочисленный конвейер, LSU (до 2-х незавершённых команд от каждого треда, только физическая адресация), блок BUI, предоставляющий через PCIe доступ к памяти хост-машины. На данном макете были верифицированы в однотредовом режиме арифметические, логические и операции сдвига над 64-х разрядными целыми числами со знаком и без знака, скалярные чтение и запись в память.

На следующем этапе (декабрь 2011 — апрель 2012) планируется создать макет M3 на основе FPGA Xilinx XC6VLX240T в следующей конфигурации: IC (только физическая адресация), один целочисленный конвейер, MDU (деление только 32-х разрядное), LSU (только физическая адресация), L2 (128 Кб), 512 Мб памяти DDR3, CU (только загрузка и выгрузка данных из памяти макета). Целью данного этапа является отладка и верификация RTL модели одноконвейерного ядра сопроцессора J7-C и подсистемы памяти с физической адресацией.

Далее (май 2012 — октябрь 2012) планируется создать макет M4, в котором по сравнению с M3 будут добавлены: еще один целочисленный конвейер, MMU, MSU, RT. Целью этапа является верификация RTL модели двухконвейерного ядра J7-C, подсистемы памяти с глобальной адресацией и сетевого маршрутизатора. Далее на этом макете будет проводиться оценка производительности выбранных микроархитектурных решений.

Литература

1. Слуцкин А.И., Эйсымонт Л.К. Российский суперкомпьютер с глобально адресуемой памятью. URL: http://www.osp.ru/os/2007/09/4569294/ (дата обращения: 15.12.2011).
2. Семенов А., Фролов А., Никитин А., Кабыкин В. Суперкомпьютеры для графовых задач. URL: http://www.osp.ru/os/2011/07/13010498/ (дата обращения: 15.12.2011).
3. Макагон Д., Сыромятников Е. Сети для суперкомпьютеров. URL: http://www.osp.ru/os/2011/07/13010500/ (дата обращения: 15.12.2011).

Применение параллельных вычислительных технологий для решения сложных задач аэрогазодинамики в ВЦ ФАЛТ МФТИ

В.И. Шалаев[1], Д.В. Апраксин[1], А.В. Ваганов[1,4], И.В. Воронич[1], Л.Ф. Ивчик[2], В.Н. Коньшин[3], С.В. Михайлов[1,4], С.А. Рыжов[5], А.А. Савельев[1,4], М.А. Стародубцев[1,4], В.В. Ткаченко[1,3], И.А. Хохлов[1], Т.Д. Чан[1], В.Л. Юмашев[1,4]

Московский физико-технический институт (ГУ)[1],
ММКБ «Салют»[2], ФГУП «СКЦ Росатома»[3],
Центральный аэрогидродинамический институт[4], ОАО «Тесис»[5]

Высокая стоимость и ограниченность информации экспериментальных исследований о деталях течений делает актуальным разработку новых численных подходов к анализу аэро- и газодинамических течений и использование численного эксперимента для определения их характеристик. Сложность практически важных задач предполагает применение для решения многопроцессорных систем и параллельных алгоритмов. В докладе представлен обзор приложений такого подхода к расчету реальных течений в ВЦ ФАЛТ МФТИ.

Внутренние газогидродинамические течения, такие как течения в трактах авиационных двигателей и теплообменных установок, являются одними из самых сложных вычислительных объектов. Расчеты характеристик компрессора авиационного двигателя с учетом деформации лопаток под воздействием аэродинамических и центробежных нагрузок, с целью ее оптимизации, выполнены с использованием программных комплексов CFX и NASTRAN [1].

Анализ внешних аэродинамических течений связан с разработкой перспективных космических аппаратов, таких как бескрылый аппарат «Клипер» (РКК «Энергия») и крылатый аппарат «МВКА-ЦАГИ», перспективный гиперзвуковой ЛА и многоблочная ракетная система. Расчеты аэродинамических характеристик этих объектов вдоль траектории спуска выполнены с помощью программного комплекса «АРГОЛА-2» [2]. На отдельных участках траектории были проведены расчеты в рамках уравнений Навье-Стокса, Рейнольдса и Больцмана. Для верификация методов использовались экспериментальные данные [3].

Моделирование методом крупных вихрей турбулентных струйных течений является одним из интенсивно развивающихся направлений, некоторые результаты исследований представлены в докладе.

Литература

1. И.В. Воронич, Л.Ф. Ивчик, В.Н. Коньшин, В.В. Ткаченко, В.Л. Юмашев, В.И. Шалаев. Применение современных программных комплексов и многопроцессорных систем для решения прикладных задач а области авиадвигателестроения. // Доклады конференции «Новые наукоемкие технологии в авиационно-космическом комплексе и энергосбережении». Жуковский. 2003. С.28-1 – 28-6.
2. Дроздов С.М., Косых А.П., Нерсесов Г.Г., Шалаев В.И. Юмашев В.Л.Численное исследование аэродинамики перспективного возвращаемого воздушно-космического аппарата с органами управления//Авиакосмическая техника и технология. 2007. № 4.
3. А.В. Ваганов, С.М. Дроздов, Г.Н. Дудин, С.М. Задонский, В.И. Пляшечник, М.А. Стародубцев, С.В. Чернов, В.Л. Юмашев. Расчетно-экспериментальные исследования аэродинамики контрольной модели при больших сверхзвуковых скоростях // Труды МФТИ. 2008. № 4.

Индекс по фамилиям

Ю

Я

Содержание

Короткие статьи **336**

Всероссийская конференция молодых ученых «Параллельные и распределенные вычисления»

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ
ПаВТ'2012

Труды международной научной конференции
(Новосибирск, 26 – 30 марта 2012 г.)

Издательский центр Южно-Уральского государственного университета

Подписано в печать 27.02.2012. Формат 60×84 1/8.
Усл. печ. л. 89,98. Заказ 40.
