Российская академия наук
Суперкомпьютерный консорциум университетов России

Электронное издание

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ (ПаВТ'2011)

Труды международной научной конференции

г. Москва, 28 марта – 1 апреля 2011 г.

Челябинск,
Издательский центр ЮУрГУ
2011

УДК 004.75
П 18

Параллельные вычислительные технологии (ПаВТ'2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2011. – 730 с. – URL: http://omega.sp.susu.ac.ru/books/conference/PaVT2011

ISBN 978-5-696-04090-5

Данный сборник содержит статьи, включенные в программу Международной научной конференции «Параллельные вычислительные технологии». Конференция проводится с 28 марта по 1 апреля 2011 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

Эксклюзивный партнер конференции:
Группа компаний Т-Платформы

ISBN 978-5-696-04090-5

ОРГАНИЗАТОРЫ

Учредителями конференции являются Российская академия наук и Суперкомпьютерный консорциум университетов России

СПОНСОРЫ

Платиновые спонсоры:
Корпорация Intel
Компания РСК СКИФ
Корпорация IBM

Золотые спонсоры:
Корпорация AMD
Компания CADFEM
Компания ТЕСИС

Серебряные спонсоры:
Компания DEPO Computers
Корпорация NVIDIA

ИНФОРМАЦИОННАЯ ПОДДЕРЖКА

Информационно-аналитический центр Parallel.ru
Газета «Поиск»
Информационно-аналитический журнал «Rational Enterprise Management»
Международный информационно-аналитический журнал «CAD/CAM/CAE Observer»
Журнал «Суперкомпьютеры»

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

Сопредседатели программного комитета:
Абрамов С.М., чл.-корр. РАН, ИПС РАН, г. Переславль-Залесский
Соколинский Л.Б., д.ф.-м.н., НИУ ЮУрГУ, г. Челябинск
Четверушкин Б.Н., чл.-корр. РАН, ИПМ РАН, г. Москва

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
Бердышев В.И., чл.-корр. РАН, ИММ УрО РАН, г. Екатеринбург
Болдырев Ю.Я., д.т.н., НИУ СПбГПУ, г. Санкт-Петербург
Бухановский А.В., д.т.н, НИУ СПбГУ ИТМО, г. Санкт-Петербург
Газизов Р.К., д.ф.-м.н., УГАТУ, г. Уфа
Гергель В.П., д.т.н., НИУ ННГУ, г. Нижний Новгород
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
Мейер Х. (H. Meuer, ISC General Chair), Германия
Модорский В.Я., д.т.н., НИУ ПГПУ, г. Пермь
Немухин А.В., д.х.н., МГУ, г. Москва
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., НИВЦ МГУ, г. Москва
Сулимов В.Б., д.ф.-м.н., НИВЦ МГУ, г. Москва
Турлапов В.Е., д.т.н, НИУ ННГУ, г. Нижний Новгород
Шабров Н.Н., д.т.н, НИУ СПбГПУ, г. Санкт-Петербург
Якобовский М.В., д.ф.-м.н., ИММ РАН, г. Москва

ОРГАНИЗАЦИОННЫЙ КОМИТЕТ

Председатель организационного комитета:
Садовничий В.А., ректор МГУ

Сопредседатели организационного комитета:
Моисеев Е.И., декан факультета ВМК МГУ
Тихонравов А.В., директор НИВЦ МГУ

Члены организационного комитета:
Аксенова Е.В., нач. отдела ЛСМ ЮУрГУ
Антонов А.С., с.н.с. НИВЦ МГУ
Березин Б.И., зам. декана факультета ВМК МГУ
Брызгалов П.А., н.с. НИВЦ МГУ
Воеводин Вад.В., н.с. НИВЦ МГУ
Воеводин Вл.В., зам. директора НИВЦ МГУ
Гуляев А.В., зам. декана факультета ВМК МГУ
Королев Л.Н., зав. кафедры автоматизации систем вычислительных комплексов факультета ВМК МГУ
Пан К.С., программист кафедры системного программирования ЮУрГУ
Попова Н.Н., доцент кафедры автоматизации систем вычислительных комплексов факультета ВМК МГУ
Репина К.В., программист кафедры системного программирования ЮУрГУ
Руденко Н.М., зам. директора НИВЦ МГУ
Соболев С.И., н.с. НИВЦ МГУ
Соколинский Л.Б., декан факультета Вычислительной математики и информатики ЮУрГУ
Цымблер М.Л., доцент кафедры системного программирования ЮУрГУ

Увеличение вычислительной мощности распределенных систем с помощью грид-систем из персональных компьютеров*

R. Lovas[1], А.П. Афанасьев[2], В.В. Волошинов[2],
М.А. Посыпкин[2], О.В. Сухорослов[2], Н.П. Храпов[2]

[1]MTA SZTAKI,
[2]ИСА РАН

Рассматриваются технологии грид-систем из персональных компьютеров, получившие широкое распространение в последнее время. Эти технологии позволяют задействовать свободные от полезной работы ресурсы рабочих станций, настольных ПК, ноутбуков для проведения ресурсоемких расчетов. Приводится обзор программного обеспечения, используемого для создания гридов из персональных компьютеров. Рассматриваются средства разработки приложений для таких систем. Также в статье уделяется внимание вопросам интеграции различных типов грид-систем.

1. Введение

Современная наука и производство немыслимы без масштабных расчетов, требующих колоссальной вычислительной мощности, которую можно обеспечить только в рамках распределенных и параллельных систем. В настоящее время существует несколько подходов к организации распределенных вычислений. Наибольшую производительность обеспечивают специализированные суперкомпьютеры большой мощности, которых насчитывается несколько сотен по всему миру[1]. Они могут решать широкий спектр вычислительных задач, но требуют больших расходов на создание и эксплуатацию.

Несколько меньшей производительностью обладают так называемые сервисные гриды, которые соединяют кластеры, установленные в различных организациях. Это решение дешевле по отношению к суперкомпьютерам, но также требует выделенных ресурсов и значительных усилий, связанных с эксплуатацией. Инфраструктура сервисных гридов состоит из набора сервисов, обеспечивающих доступ к брокерам ресурсов, информационным службам, хранилищам данных, вычислительным ресурсам. Пользователи сервисных гридов имеют соответствующие права доступа к предоставляемым сервисам. Контроль доступа к ресурсам осуществляется посредством сертификатов безопасности. Хорошо известны следующие технологии создания сервисных гридов, как Globus [2], LCG-2/gLite (EGEE) [3], ARC [4] и Unicore [5].

На нижнем уровне рассматриваемой иерархии находятся грид-системы персональных компьютеров (ГПК). ГПК основаны на наблюдении, что большую часть времени персональные компьютеры либо простаивают, либо загружены лишь на некоторую небольшую долю своей мощности. ГПК обеспечивают возможность объединения свободных вычислительных мощностей домашних компьютеров и персональных компьютеров учреждений в единую распределенную систему для решения сложных вычислительных задач. В отличие от сервисных гридов, грид-системы из персональных компьютеров легко установить и поддерживать. Как правило, требуется одна рабочая станция, на которой запускается серверная часть инфраструктуры. Пользователи со всего мира имеют возможность подключать свои персональные компьютеры к этому ресурсу, предоставляя тем самым свободные ресурсы своих компьютеров для работы приложений, размещенных на центральном сервере. Грид-системы из персональных компьютеров являются наиболее дешевым решением, обеспечивающим большую вычислительную мощность. ГПК обладают огромным потенциалом роста – в настоящее время в мире насчитывается более одного миллиарда персональных компьютеров и их число стремительно увеличивается. К сожалению, далеко не все распределенные приложения могут эффективно выполняться на подобных системах из-за серьезных ограничений, накладываемых возможностями по передаче

* Работа выполнена при поддержке Седьмой Рамочной программы Европейского Союза (FP7/2007-2013), грант № 261561 (DEGISCO).

данных и высокой вероятностью отказа узлов, участвующих в вычислениях. Вместе с тем, достаточно широкий класс практических задач укладывается в модель управляющий-рабочие, которая является основной моделью приложения в ГПК. К этому классу относятся многие переборные и комбинаторные задачи, моделирование методом Монте-Карло, задачи идентификации и многие другие. Для таких задач использование ГПК оправдано и позволяет разгрузить суперкомпьютеры и сервисные гриды. Резюмируя можно сказать, что грид-системы персональных компьютеров являются дешевой альтернативой суперкомпьютерам и сервисным гридам и для ряда задач могут их успешно заменять.

В настоящее время широко известны следующие системы ГПК: Condor[6], BOINC [7], XtremWeb [8], OurGrid[9]. Эти системы могут быть использованы для создания распределенных систем масштаба лаборатории, предприятия, города или всего мира. Несмотря на различия между сервисными гридами и грид-системами персональных компьютеров между ними много общего. В частности, модель управляющий-рабочие используется при разработке приложений для систем обоих типов. Поэтому, важнейшей задачей является обеспечение интероперабельности между сервисными гридами и ГПК. Различают два вида интероперабельности: на программном и на системном уровне. Интероперабельность на программном уровне означает наличие универсального интерфейса для подключения вычислительного ресурса, позволяющего объединять разнородные ресурсы в единую распределенную систему. На этом принципе основаны системы SAGA[10], P-GRADE[11], MathCloud[12], BNB-Grid[13]. Интероперабельность на системном уровне предполагает наличие прозрачных для приложения механизмов, обеспечивающих «бесшовную» интеграцию вычислительных ресурсов. Примером такого подхода является механизм мостов[14], применяемый для интеграции сервисных гридов и ГПК.

В данной работе рассматриваются основные технологии, применяемые для организации грид-систем персональных компьютеров (раздел 2), методы и технологии разработки приложений для таких систем (раздел 3), средства обеспечения интероперабельности с сервисными гридами (раздел 4). В заключении (раздел 5) приводятся примеры использования данной технологии для решения научных задач и рассматриваются организационные аспекты создания и развития ГПК.

2. Технологии гридов из персональных компьютеров

2.1 Condor

Система Condor [6] разрабатывается с 1988 года в рамках одноименного исследовательского проекта в University of Wisconsin-Madison (США). По предоставляемой пользователю функциональности Condor близок к традиционным системам управления пакетной обработкой, используемых на вычислительных кластерах. В то же время, оригинальная архитектура Condor поддерживает режимы вычислений, недоступные в традиционных системах. Режим вычислений высокой пропускной способности (High-Throughput Computing) позволяет надежно поддерживать высокие объемы вычислительных ресурсов в течение длительных периодов времени путем эффективного использования всех доступных в сети вычислительных ресурсов. Режим оппортунистических вычислений (opportunistic computing) позволяет использовать ресурсы в любой момент времени, когда они доступны, не требуя постоянного монопольного доступа.

Помимо управления кластерами из выделенных узлов Condor может эффективно использовать ресурсы простаивающих персональных компьютеров, объединяя все доступные ресурсы организации в единый «виртуальный» кластер или пул. Например, машина может быть автоматически добавлена в пул в те моменты, когда отсутствует активность клавиатуры или мыши. В случае, если машина вновь задействована ее владельцем, Condor выводит машину из пула, осуществляя миграцию выполнявшихся на ней заданий на другие доступные узлы. В отличие от традиционных систем управления кластером, Condor не требует наличия разделяемой между узлами файловой системы. Система поддерживает передачу файлов задания между узлами и прозрачное перенаправление потоков ввода/вывода задания на машину, запустившую данное задание.

Пул под управлением Condor состоит из выделенного узла, играющего роль центрального менеджера, и произвольного количества других машин. Любая машина в пуле может играть как

роль узла, запускающего задания, так и роль узла, выполняющего задания. На концептуальном уровне пул представляет собой совокупность ресурсов (машин) и запросов (заданий). Центральный менеджер пула играет роль посредника, сопоставляя поступившие запросы с доступными ресурсами. Каждая машина в пуле периодически отправляет свое состояние центральному менеджеру, который использует данную информацию при выборе узлов для выполнения заданий.

Для описания заданий и ресурсов в Condor используется единый механизм ClassAd, обладающий высокой гибкостью и выразительностью. В описании задания могут быть указаны как обязательные требования, так и дополнительные предпочтения по выбору машин для выполнения задания. Аналогично, в описании ресурса (машины) могу быть указаны требования и предпочтения относительно заданий, которые могут быть запущены на данной машине. Важно отметить, что данные условия могут определяться индивидуально владельцем каждой машины. Гибкость и выразительность языка ClassAd позволяет описать практически любые политики использования ресурсов. Использование единого механизма описания для заданий и ресурсов обеспечивает эффективный выбор ресурсов для выполнения заданий с учетом взаимных требований.

Несмотря на то, что основной областью применения Condor является интеграция ресурсов внутри организации, благодаря реализованному механизму «flocking» система также поддерживает объединение ресурсов нескольких независимых пулов, принадлежащих различным организациям.

2.2 BOINC

BOINC (Berkeley Open Infrastructure for Network Computing) [7] представляет собой платформу с открытым кодом для организации проектов добровольных вычислений. Разработка системы ведется в U.C. Berkeley Spaces Sciences Laboratory (США) исследовательской группой, которая также разрабатывала проект SETI@home. Работа над BOINC была начата в 2002 году с целью создания универсальной программной платформы для проектов подобного рода, которая бы упростила процесс развертывания необходимой инфраструктуры и разработки приложений. Первый проект добровольных вычислений на основе BOINC был запущен в 2004 году. В настоящее насчитывается около 60 публичных проектов на основе BOINC, делая платформу стандартом де-факто в данной области. Система также часто используется для организации внутренних гридов из персональных компьютеров.

Программное обеспечение BOINC состоит из двух основных частей: серверной, которая обеспечивает функционирование проекта, и клиентской, размещаемой на машинах добровольцев. Каждый проект на основе BOINC создается и функционирует независимо от других проектов, поддерживая собственный центральный сервер и веб-сайт. Для участия в проекте добровольцы устанавливают на своих компьютерах универсальный клиент BOINC, распространяемый с сайта платформы и доступный для всех основных операционных систем.

Проект на основе BOINC идентифицируется с помощью уникального адреса (URL), являющегося одновременно главной страницей веб-сайта проекта и точкой входа для клиентов. В рамках проекта могут выполняться несколько приложений, состав которых может со временем изменяться. В рамках BOINC предусмотрена поддержка широкого класса вычислительных приложений, которые могут быть сформулированы в виде совокупности из большого количества независимых заданий. Платформа обеспечивает надежное и эффективное выполнение приложений в динамичной распределенной среде, реализуя механизмы планирования заданий, передачи данных и обработки отказов. Существующие приложения на таких языках, как C, C++ и FORTRAN, могут быть использованы в рамках BOINC без или с минимальной модификацией их кода.

После установки клиента BOINC, добровольцы могут подключить клиент к одному или нескольким проектам. При этом пользователь может указать то, каким образом в процентном отношении ресурсы его компьютера будут распределены между данными проектами. При подключении к проекту пользователь фактически дает разрешение на выполнение на своей машине любых исполняемых файлов, загруженных с сервера проекта. Несмотря на то, что BOINC обеспечивает определенную изоляцию выполняемого на клиентской стороне кода (sandboxing),

в общем случае пользователь самостоятельно должен убедиться в подлинности, защищенности и научной значимости проекта.

Для учета индивидуального вклада каждого из добровольцев в проект в BOINC реализован механизм учета кредитов, которые вычисляются пропорционально процессорному времени, использованному для выполнения заданий проекта. В BOINC также предусмотрена возможность экспорта информации о кредитах пользователя на уровне отдельного проекта. Это, а также поддержка глобальной идентификации пользователя по его адресу электронной почты, позволяет агрегировать и делать доступной сводную статистику кредитов пользователя по всем проектам.

Механизм менеджеров учетных записей (account manager) позволяет централизованно управлять подключением клиентов к тем или иным проектам. В данном случае, вместо прямого подключения к проекту, клиент подключается к веб-сервису менеджера учетных записей. Во время работы клиент периодически связывается с менеджерам, получая от него список проектов, к которым необходимо подключиться. Данный механизм может быть использован для делегации пользователем решений по выбору поддерживаемых проектов некоторому доверенному комитету.

2.3 XtremWeb и XWHEP

XtremWeb [8] представляет собой ПО с открытым кодом для создания грид-систем из персональных компьютеров, разработанное в рамках исследовательского проекта по глобальным вычислениям в INRIA/IN2P3 (Франция). В настоящее время дальнейшее развитие системы под именем XWHEP (XtremWeb-HEP) ведется в LAL (Laboratoire de l'Accélérateur Linéaire), IN2P3-CNRS (Франция). Система реализована на языке Java и доступа на условиях лицензии GPLv2.

Архитектура XWHEP состоит из трех основных типов компонентов: серверы, рабочие и клиенты. Серверы обеспечивают функционирование центральных сервисов системы, таких как планировщицах и агрегатор результатов заданий. Рабочие (workers) устанавливаются владельцами ресурсов на своих компьютерах для предоставления данных ресурсов в рамках системы. Клиенты устанавливаются пользователями ресурсов на своих компьютерах для взаимодействия с системой и использования агрегированных ею ресурсов. Клиент позволяет пользователям управлять системой, размещать приложения, запускать их в виде заданий и загружать полученные результаты. Запущенные клиентом задания регистрируются на сервере и назначаются рабочим. Аналогично системе Condor, в рамках XWHEP любой рабочий может одновременно играть роль клиента и наоборот.

XWHEP значительно расширяет изначальные возможности XtremWeb в плане обеспечения безопасности. Так, в системе реализованы механизмы прав доступа как на уровне пользователей, так и групп пользователей. Данные механизмы однородным образом могут быть применены для ограничения доступа пользователей к ресурсам, приложениям и данным.

2.4 OurGrid

Система OurGird [9], разрабатываемая в Federal University of Campina Grande (Бразилия), реализует оригинальную концепцию кооперативных грид-систем на основе модели peer-to-peer. В рамках данной концепции простаивающие вычислительные ресурсы участников системы объединяются и разделяются между ними таким образом, что размер доступных участнику ресурсов пропорционален количеству ресурсов, предоставленных им сообществу. Система реализована на языке Java и доступна в исходных кодах на условиях лицензии GPL.

OurGird реализует масштабируемую и защищенную распределенную платформу для выполнения приложений класса Bag-of-Tasks (BoT), к которому относится широкий спектр прикладных задач. Данные параллельные приложения, состоящие из множества независимых заданий, идеально подходят для запуска в грид-системах из персональных компьютеров.

Архитектура OurGrid основана на симметричной peer-to-peer модели, где каждый грид-сайт представлен в системе равноправным узлом (peer). Ключевой проблемой, связанной с применением подобной модели, является возможность появления в системе узлов-паразитов (freeriders),

которые потребляют ресурсы других узлов, никогда не давая взамен своих ресурсов сообществу. Для решения данной проблемы в OurGrid используется оригинальный подход «Сеть услуг» (Network-of-Favors), мотивирующий кооперацию между узлами. Под услугой подразумевается выделение ресурса узлу, который запросил данный ресурс. Ценность (количественная мера) услуги определяется ценностью вычислений, выполненных по запросу узла. Каждый узел локально хранит суммарное количество предоставленных и полученных им услуг. При выделении своих ресурсов узлы предпочитают те узлы, которым они «должны» больше всего услуг.

С 2004 года разработчики OurGrid поддерживают одноименную открытую кооперативную грид-систему, участники которой обмениваются свободными вычислительными ресурсами. К данной системе может присоединиться любой желающий, загрузив и установив на своих ресурсах ПО OurGrid. При этом, в отличие от традиционных грид-систем, не требуется никаких контактов и согласований.

3. Разработка приложений

Несмотря на широту класса приложений, пригодных для запуска в гридах из персональных компьютеров, данный класс предъявляет определенные требования к приложениям. Перечислим основные из них:

- Наличие большого количества подзадач (заданий), укладывающихся в модель «управляющий-рабочие»;
- Типичный размер заданий соответствует нескольким часам вычислений на типичном персональном компьютере и не более десятков мегабайт входных и выходных данных;
- Отсутствие необходимости в общем доступе к данным и взаимодействии между рабочими узлами, выполняющими задания. Синхронизация между рабочими узлами возможна только через центральный сервер.

Приложения, запускаемые в гридах из персональных компьютеров, как привило, состоят из двух распределенных частей: серверной (управляющий) и клиентской (рабочий). Серверная часть приложения отвечает за формирование заданий, проверку результатов заданий и последующую их обработку. Клиентская часть приложения выполняется на рабочих узлах и отвечает за выполнение заданий. Данная часть приложения должна поддерживать работу в фоновом режиме, без интерактивного ввода или графического интерфейса. Гетерогенность рабочих узлов обуславливает необходимость компиляции клиентской части приложения для всех используемых вычислительных платформ.

Технологии гридов из персональных компьютеров предоставляют функционирующее между двумя указанными частями приложения промежуточное ПО, которое прозрачным образом осуществляет распределение клиентского кода и заданий между доступными машинами, мониторинг выполнения заданий, обработку отказов, сбор результатов заданий и передачу их серверной части приложения. Для разработки, развертывания и управления приложениям в каждой из технологий, как правило, предусмотрены универсальные интерфейсы командной строки и/или прикладного программирования.

Например, BOINC предоставляет подобного рода интерфейсы как для серверной (генерация заданий, валидация результатов и их обработка), так и для клиентской части (управление выполнением приложения, сохранение промежуточного состояние и учет используемых ресурсов) приложения. Данные интерфейсы подразумевают модификацию исходного кода приложения. Поскольку существует много «унаследованных» приложений, код которых недоступен или требует значительных усилий по модификации, в BOINC предусмотрен готовый конфигурируемый адаптер (wrapper), позволяющий запускать на клиентской стороне любое приложение без модификации его исходного кода.

Технологии XtremWeb/XWHEP и OurGrid поддерживают запуск произвольных исполняемых файлов, что упрощает перенос приложений.

В случае с XtremWeb/XWHEP разработчик приложения в первую очередь должен зарегистрировать приложение, загрузив исполняемый файл на центральный сервер с помощью клиента. Поддерживается регистрация сразу нескольких версий исполняемого файла для разных платформ. Любой пользователь может зарегистрировать приложение, при этом права доступа

приложения будут определятся правами доступа пользователя. Для регистрации публично доступного приложения требуются права администратора. После того, как приложение зарегистрировано, пользователь может запускать задания, указывая идентификатор приложения, аргументы командной строки и ссылки на входные данные. Задания могут быть объединены в группы, что упрощает реализацию приложений с большим количеством заданий.

В случае с OurGrid приложение описывается с помощью дескриптора задачи (job description file). Задача (job) состоит из нескольких заданий (tasks). Каждое задание формируется из трех подзаданий: начального (init), удаленного (remote) и финального (final), которые выполняются последовательно. В качестве подзаданий указываются внешние команды, вызываемые OurGrid. Таким образом, допускается использование произвольных исполняемых файлов и скриптов. Начальное и финальное подзадания выполняются локально, на машине с которой производится запуск задачи. Начальное подзадание предназначено для подготовки окружения для выполнения задания, например — передачи входных данных задания на машину, которая выбрана для его выполнения. Финальное подзадание обычно используется для загрузки результатов задания на машину пользователя. Удаленное подзадание запускается на машине в гриде и выполняет основные вычисления, связанные с заданием. Помимо описания подзаданий, в описание задания также входят требования к ресурсам. OurGrid позволяет описывать подзадания, абстрагируясь от специфики конкретных машин, на которых будут запущены данные подзадания, например - организации файловой системы. Для этого применяются абстракции, представляющие хранилище данных на удаленной машине в гриде, и набор команд для отправки и получения файлов из хранилища.

Система Condor поддерживает запуск как немодифицированных исполняемых файлов, так и приложений, собранных с использованием библиотеки Condor. Во втором случае система берет на себя обязанности по автоматическому сохранению контрольных точек приложения (checkpointing) и перенаправлению системных вызовов с исполняемой на запускаемую машину. Задание описывается с помощью текстового дескриптора (submit description file), содержащего информацию об исполняемом файле, аргументах запуска, входных и выходных файлах, требованиях к ресурсам и т. п. Дескриптор заданий поддерживает возможность описания задания-кластера, состоящего из набора параметризованных подзаданий.

4. Технологии интеграции сервисных гридов и гридов из персональных компьютеров

Механизм мостов, предоженный в работе [14], позволяет осуществлять интеграцию сервисных гридов и ГПК на системном уровне, т.е. прозрачным для пользователя образом. На данный момент этот подход реализован для связи грид-инфраструктуры EGEE/EGI с несколькими ГПК (Рис. 1). Суть подхода состоит в специальном программном компоненте 3G-Bridge (a Generic Grid to Grid bridge) который, опираясь на абстрактное понятие задания, может быть использован для интеграции двух грид-систем. По выполняемым функциям интегрирующее программное обеспечение можно подразделить на два типа:

- Мост EGEE ⇒ DG, обеспечивающий запуск заданий сервисного грида в инфраструктуре ГПК.
- Мост DG ⇒ EGEE, позволящий, наоборот, запускать задания ГПК в инфраструктуре EGEE.

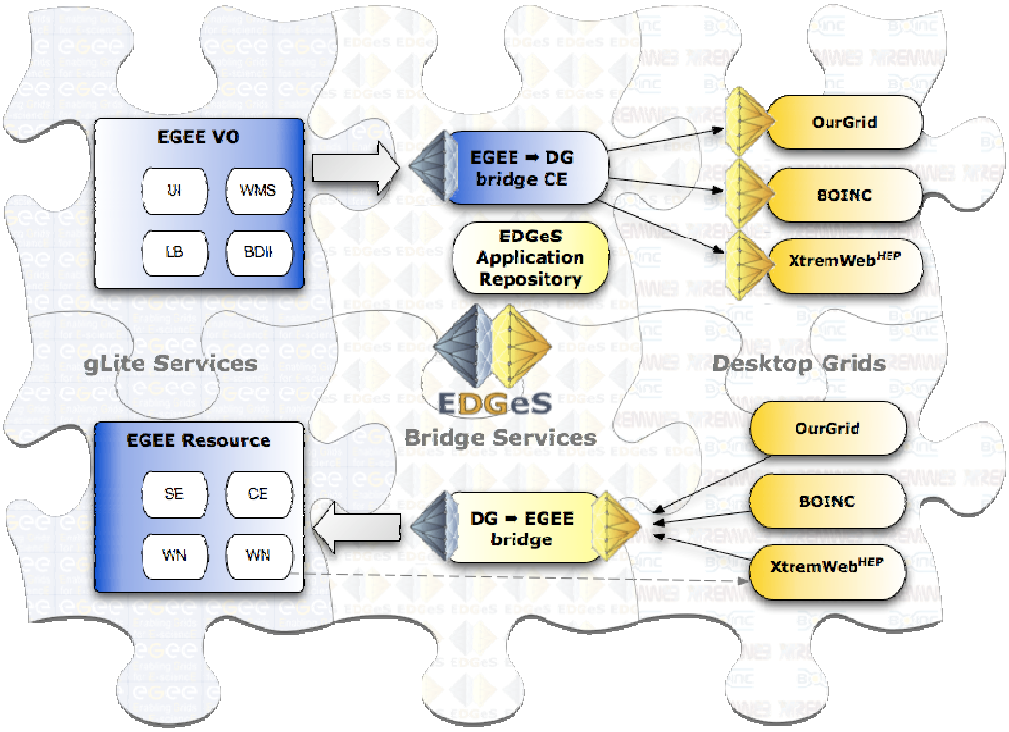


Рис. 1. Основные элементы инфраструктуры, обеспечивающие взаимодействие разнородных грид-систем.

5.1 Мост EGEE ⇒ DG

Данное соединение функционирует как Computing Element (CE) сервисного грида, где задания вместо вычислительных узлов направляются в инфраструктуру грида из персональных компьютеров (BOINC, XWHEP, OurGrid). Взаимодействие различных типов грид-систем обеспечивается тремя основными программными компонентами:

1. Функционирующий на стороне gLite модифицированный Computing Element, который отправляет принятые из инфраструктуры сервисного грида задания на удалённый мост. Данный CE поставляется в качестве модуля YAIM, и может быть установлен и настроен вместе с другими компонентами gLite.
2. На стороне сервера ГПК функционирует специальный адаптер, отвечающий за получение заданий, их преобразование для новой инфраструктуры, и выполнение.
3. Репозиторий приложений (Application Repository — AP), содержащий информацию о всех приложениях, проходящих через данный мост.

5.2 Мост DG ⇒ EGEE

Поскольку принцип работы и основное программное обеспечение зависит от типа подключаемой инфраструктуры ГПК, для каждой из них создана отдельная реализация моста:

1. **Мост BOINC ⇒ EGEE,** который функционирует как клиент BOINC, отправляя скачанные задания в виртуальную организацию EGEE. В инфраструктуре сервисного грида задание запускается специальной программой (jobwrapper), которая которая запускает приложение BOINC, и эмулирует для него окружение клиента BOINC.
2. **Мост XWHEP ⇒ EGEE,** подключающий рабочие узлы EGEE к гриду XWHEP путём запуска рабочих компонентов инфраструктуры в виде заданий EGEE.
3. **Мост OurGrid ⇒ EGEE,** который непосредственно запускает задания на вычислительных узлах виртуальной организации EGEE.

Для организации взаимодействия с инфраструктурой EGEE в двух последних реализациях используется библиотека jLite[15].

В рамках EGEE создана виртуальная организация desktopgrid.vo.edges-grid.eu, предназначенная для выполнения заданий ГПК. Гриды из персональных компьютеров(включая три основных варианта: BOINC, XWHEP, OurGrid), подключённые к EDGeS, и поддерживающие запуск EGEE приложений, позволяют добровольцам внести свой собственный вклад в проект EGEE и инфраструктуру сервисного грида EGEE. Инфраструктура EDGeS, содержащая более 100 тыс. процессорных ядер создавалась на основе нескольких типов ГПК и европейского грида EGEE/EGI. Использование гибридной вычислительной инфраструктуры в научных целях даёт следующие преимущества:

1. Операторы гридов из персональных компьютеров и сервисных гридов могут объединить свои системы посредством инфраструктуры EDGeS. Таким образом можно увеличить вычислительный ресурс, доступный для пользователей.
2. Менеджеры виртуальных организаций (ВО) EGEE или другого сервисного грида могут подключиться к виртуальной организации EDGeS, и таким образом добавить вычислительный ресурс в инфраструктуру, и запускать в ней свои приложения.
3. Потребители ресурсов, подключённые к EDGeS имеют в распоряжении больший вычислительный ресурс, чем на локальных гридах. Если используется приложение, портированное на несколько типов грид-систем, то соответствующие задания будут автоматически запускаться в нужных частях инфраструктуры.
4. Любому желающему предоставить свой вычислительный ресурс для нужд науки достаточно подключиться к одной из поддерживаемых EDGeS грид-систем, и таким образом стать участником этой вычислительной инфраструктуры.

5. Заключение

В настоящее время грид-системы персональных компьютеров стали серьезной альтернативой традиционным высокопроизводительным вычислительным системам и получили признание международного научного сообщества. Созданы инфраструктуры ГПК, в рамках которых ведутся расчеты различных научных приложений. Примерами таких инфраструктур являются локальная инфраструктура университета Вестминстера[16], открытый проект EDGeS@home[17], российский проект центра Грид-технологий и распределенных вычислений[18].

Грид-системам персональных компьютеров посвящено несколько проектов европейской рамочной программы (FP7). Проект DEGISCO[19], направлен на поддержку и развитие международной кооперации в области создания и расширения ГПК, всемерную популяризацию этой технологии, поддержку разработчиков распределенных приложений. Участниками проекта DEGISCO и родственного проекта EDGI [20] основана Международная федерация грид-систем из персональных компьютеров (IDGF)[21]. Международная федерация грид-систем из персональных компьютеров является организацией, объединяющей людей из различных компаний, университетов и исследовательских институтов, заинтересованных в использовании вычислительной мощности такого типа и желающих обменяться опытом друг с другом. Своим участникам федерация предоставляет несколько услуг: личные встречи на семинарах, доступ к документации, форум, веб-портал и консультации экспертов.

Учитывая темпы роста числа персональных компьютеров в мире и развития Интернет можно с уверенностью сказать, что грид-системы персональных компьютеров будут активно развиваться и в ближайшем будущем интерес к этой технологии будет возрастать. Технологии интеграции различных типов грид-систем будут способствовать популяризации ГПК в научной среде.

Литература

1. Top 500 Supercomputers. URL: http://www.top500.org (дата обращения: 13.02.2011).

2. I. Foster, C. Kesselman: The Globus Project: A Status Report, Proc. IPPS/SPDP ’98 Heterogeneous Computing Workshop, pp. 4-18, 1998.
3. The gLite webpage, http://glite.web.cern.ch/glite.
4. M.Ellert et al.: Advanced Resource Connector middleware for lightweight computational Grids, Future Generation Computer Systems 23 219-240, 2007.
5. A. Streit, D. Erwin, Th. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder L. Grandinetti (Edt.): UNICORE - From Project Results to Production Grids, Grid Computing: The New Frontiers of High Performance Processing, Advances in Parallel Computing 14, Elsevier, 2005, pages 357-376
6. Michael Litzkow, Miron Livny, and Matt Mutka, Condor-A Hunter of Idle Workstations. In Proc. The 8th International Conference of Distributed Computing Systems, San Jose, California, June, 1988, pp.204-111.
7. D. P. Anderson. Boinc: A system for public-resource computing and storage. In R. Buyya, editor, Fifth IEEE/ACM International Workshop on Grid Computing, pages 4-10, 2004.
8. F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Neri and O. Lodygensky: Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models,Security, Tests and Convergence with Grid FGCS Future Generation Computer Science, 2004.
9. Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro B. Costa, Alisson Andrade, Reynaldo Novaes and Miranda Mowbray. Labs of the World, Unite!!! J. Grid Computing 4(3), 2006, pp. 225-246.
10. Shantenu Jha, Hartmut Kaiser, André Merzky, Ole Weidner: Grid Interoperability at the Application Level Using SAGA. eScience 2007: 584-591
11. P. Kacsuk and G. Sipos: Multi-Grid, Multi-User Workflows in the P-GRADE Portal. Journal of Grid Computing, Vol. 3, No. 3-4, Springer Publishers, pp. 221-238, 2005
12. Астафьев А.С., Афанасьев А.П., Лазарев И.В., Сухорослов О.В., Тарасов А.С. Научная сервис-ориентированная среда на основе технологий Web и распределенных вычислений. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). – М.: Изд-во МГУ, 2009. – 524 с. (с. 463-467)
13. Посыпкин М.А. Решение задач глобальной оптимизации в среде распределенных вычислений // Программные продукты и системы. № 1. 2010. С. 23-29.
14. E. Urbach, P. Kacsuk, Z. Farkas, G. Fedak, G. Kecskeméti, O.Lodygensky, A. Cs. Marosi, Z. Balaton, Zoltán; G. Caillat, G. Gombás, A.Kornafeld, J. Kovács, H. He, R. Lovas: EDGeS: Bridging EGEE to BOINC and Xtrem Web, Journal of Grid Computing, 2009, Vol 7, No. 3, pages 335 -354
15. O.V. Sukhoroslov. jLite: A Lightweight Java API for gLite. // Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 3rd Intern. Conf. (Dubna, June 30 - July 4, 2008). - Dubna: JINR, 2008. - 401 p. , pp. 201-204
16. The University of Westminster Local DesktopGrid. URL: http://wgrass.wmin.ac.uk/index.php/Desktop_Grid:Westminster_Local_DG (дата обращения: 13.02.2011).
17. EDGeS@Home Desktop Grid. URL: http://home.edges-grid.eu/home/ (дата обращения: 13.02.2011)
18. Проекты Центра грид-технологий и распределенных вычислений. URL: http://boinc.isa.ru/dcsdg/ (дата обращения: 13.02.2011)
19. The DEGISCO project. URL: http://degisco.eu (дата обращения: 13.02.2011)
20. The EDGI project. URL: http://edgi-project.eu (дата обращения: 13.02.2011)
21. Международная федерация грид-систем персональных компьютеров. URL: http://desktopgridfederation.org (дата обращения: 13.02.2011)

Использование расширяемых языков для программирования графических процессоров

А.В. Адинец[1,2]

НИВЦ МГУ имени М.В. Ломоносова[1]
ОИЯИ, г. Дубна[2]

В настоящее время наблюдается активное использование графических процессорных устройств (ГПУ) для высокопроизводительных вычислений. В связи с этим важным становится создание высокоуровневых систем программирования ГПУ. В данной статье рассматривается подход к созданию таких систем на основе расширяемых языков программирования, синтаксис и семантика которых могут быть расширены для облегчения написания программ.
В статье представлена система программирования NUDA, созданная на основе расширяемого языка Nemerle. Основной целью системы является дать программисту контроль над процессом переноса программы, переложив при этом механическую работу на компилятор. Макросы для переноса данных и исполняемого кода значительно облегчают перенос программ на ГПУ. Целевым языком для генерации кода для ГПУ является OpenCL. Аннотации преобразования циклов позволяют повышать производительность программ на конкретных архитектурах ГПУ без ущерба для размера исходного кода и его читаемости. Система протестирована на различных архитектурах ГПУ на ряде задач. В ряде случаев удалось добиться повышения производительности в несколько раз по сравнению с исходным вариантом.

1. Введение

Графические процессорные устройства (ГПУ) в настоящее время активно используются для решения вычислительных задач. Использование ГПУ позволяет получить ускорение на 1–2 порядка по сравнению с традиционными вычислительными системами, и достигнуть при этом значительно больших показателей производительности на один процессор, энергоэффективности и производительности в расчёте на стоимость. Активно создаются гетерогенные кластеры на основе ГПУ. Например, в ноябрьском списке “Топ-500” за 2010 год [1] 3 из 5 самых мощных вычислительных систем построены на основе ГПУ NVidia, а всего в списке 11 систем на основе ГПУ.

В связи с этим актуальной является проблема создания высокоуровневых средств программирования для систем на основе ГПУ. В настоящее время для решения вычислительных задач на ГПУ преимущественно используются низкоуровневые средства разработки: CUDA [2], Brook+ и OpenCL [3]. Несмотря на то, что все они являются расширениями языка C, использование низкоуровневых инструментов для программирования ГПУ сопряжено с трудностями. Создаваемые с их помощью программы являются громоздкими и зачастую трудными для чтения. Из перечисленных выше инструментов только OpenCL позволяет создавать переносимые программы; однако и на этом языке оптимизированные версии программ для разных ГПУ будут сильно отличаться.

В последнее время разработка высокоуровневых средств программирования ГПУ ведётся достаточно активно. Наиболее популярным подходом является добавление в программу на языке Fortran или C директив размещения данных и компиляции блоков кода для исполнения на графическом процессоре. По этому пути пошли создатели компилятора PGI [4] и CAPS HMPP [5]. Система директив PGI является более краткой и высокоуровневой; в CAPS HMPP директивы длиннее, однако они позволяют осуществлять более тонкий контроль за использованием ускорителя. Однако ни та, ни другая система директив в настоящее время не является стандартной, и понимается только компиляторами данных произво-

дителей. Если производительность сгенерированных программ оказалась низкой, или если программа задействует нестандартный паттерн работы с ГПУ, единственной возможностью программиста является переход на низкоуровневое средство. Возможности добавить свою директиву преобразования или воспользоваться лишь частью функциональности системы у программиста нет. Наконец, далеко не всегда удобно создавать программу для ГПУ на последовательном языке; часто удобнее сразу записывать программу в параллельном виде.

В данной статье рассматривается подход с использованием парадигмы *расширяемого программирования* [6] с целью создания программ для систем на основе ГПУ. Расширяемый язык программирования — этой такой язык программирования, синтаксис и семантика которого могут быть расширены или изменены для облегчения написания программы. Вклад данной статьи можно кратко обозначить следующим образом:

- Предложена идея использования расширяемых языков для создания систем программирования высокопроизводительных вычислений
- Идея продемонстрирована на примере системы расширений NUDA (Nemerle Unified Device Architecture), предназначенной для программирования ГПУ на языке Nemerle
- Система апробирована на целом ряде модельных задач, и во многих из них позволила достигнуть ускорения в несколько раз по сравнению с исходным вариантом без увеличения размера исходного кода

Статья организована следующим образом. В разделе 2 даётся краткий обзор расширяемых языков и их особенностей на примере языка Nemerle. В разделе 3 описывается предложенная система расширений NUDA. В разделе 4 приводятся результаты использования системы NUDA для решения задач на ГПУ. В разделе 5 обсуждаются полученные результаты и направления дальнейшей работы. Наконец, в разделе 6 подводится итог результатов работы.

2. Расширяемые языки программирования

Расширяемый язык — это такой язык программирования, синтаксис и семантика которого могут быть изменены и расширены для облегчения написания программы; при этом должны соблюдаться ряд условий. Во-первых, расширение должно выполняться без изменения компилятора языка, что предполагает наличие в компиляторе механизма плагинов. Во-вторых, расширения должны восприниматься одинаково всеми компиляторами, и как следствие, должен существовать интерфейс компилятора, которыми могут пользоваться расширения. Наконец, средства поддержки расширений должны быть предусмотрены в синтаксисе языка. В настоящее время существует небольшое количество расширяемых языков: LISP с его диалектами [7], Seed7, Nemerle [8], xoc [9]. В последнем случае описывается механизм расширений языка C, при этом сами расширения пишутся на языке Zeta.

По сравнению с традиционными языками, расширяемые языки обладают рядом преимуществ. Строить системы программирования на основе расширяемых языков проще, и они по определению являются открытыми. В рамках расширяемого языка можно строить многоуровневые модели программирования. Например, сначала реализуется просто обёртка над низкоуровневым средством, затем — удобная система аннотаций для программирования ускорителя, и наконец — система автоматического распараллеливания программ. Программист получает, с одной стороны, больший контроль над используемыми преобразованиями программ, а с другой — возможность программировать на низком уровне в случае необходимости. Вследствие открытости системы программист позволяет возможность использовать преобразования, полезные только для специфических задач, и которые поэтому не имеет смысл реализовывать в самом компиляторе. Открытость и расширяемость системы позволит задействовать ресурсы сообщества разработчиков и осуществлять таким

образом последовательное наращивание функциональности. Наличие фиксированного ядра означает, что для программирования для новых моделей или архитектур требуется изучение сравнительно небольшого объёма расширений. Таким образом, облегчается изучение новых архитектур и моделей. Наконец, код уже созданных приложений может быть обработан при помощи новых расширений, что облегчит его перенос на уже существующие архитектуры.

Для работ, описываемых в данной статье был выбран язык Nemerle, по двум причинам: во-первых, он является относительно стабильным и развитым, и во-вторых, он больше остальных похож на традиционные языки программирования, используемые для высокопроизводительных вычислений.В своей основе Nemerle является .NET-языком, похожим на C#; основным синтаксическим отличием является объявление типа переменной после её имени, через двоеточие. Тип локальных переменных выводится автоматически на основании её инициализатора и выражений, в которых она используется; поэтому явно тип объявляется только для параметров функций. Основным средством расширения языка Nemerle является *макрос* — аналог функции, которая исполняется на этапе компиляции, принимает на вход фрагменты кода (и, возможно, константы) и возвращает новый фрагмент кода. После исполнения макроса возвращаемый фрагмент кода подставляется на место вызова макроса. Для создания фрагментов кода используется механизм *квазицитирования* — фрагмент кода как объект представляется его записью на языке Nemerle, заключённой между скобками `<[` и `]>`. Внутри квазицитирования можно использовать выражение `$expr /*$*/` для подстановки результата вычисления выражения `expr` в фрагмент кода. Если результатом выражения является список фрагментов кода, то используется оператор `..$expr /*$*/` Аналогичные конструкции внутри сопоставителя в операторе **match** позволяют выполнять разбор кода. С помощью объявления **syntax** можно задать для вызова макроса более удобный синтаксис. Внутри макроса можно обращаться к стандартному интерфейсу компилятора, например, для генерации ошибок или получения типа выражения. Пример макроса Nemerle для развёртки цикла приведён на рис. 1. Более подробная информация о возможностях языка приведена в `man-nemerle`.

```
macro inlineMacro(aloop) syntax("inline", aloop) {
  match(aloop) {
    | <[ for(mutable $(i:name) = $(a:int); $(i1:name) < $(b:int);
      $(i2:name)++) $body ]> =>
      if(i.Equals(i1) && i.Equals(i2)) {
        mutable res1 = [] : list[PExpr];
        for(mutable v1 = a; v1 < b; v1++)
          res1 ::= <[ $(i:name) = $(v1:int); $body ]>;
        res1 = res1.Reverse();
        res1 ::= <[ mutable $(i:name) = 0; ]>;
        <[ { ..$res1 } ]> /*$*/
      } else {
        Error(aloop.Location, "variables don\'t match"); <[ ]>
      }
    | _ => Error(aloop.Location, "loop expected");
  }
}
```

Рис. 1. Пример макроса на языке Nemerle

Несмотря на свою простоту, механизм макросов в языке Nemerle является достаточно мощным. Например, операторы условного перехода и циклов реализованы при помощи макросов через оператор сопоставления и вложенные функции, или замыкания. Стандартная библиотека макросов Nemerle содержит реализацию асинхронных методов и блоков, регулярных выражений и контрактов на параметры и возвращаемые значения функций. Макросы могут применяться не только к фрагментам кода, но также и к классам и их членам.

3. Система расширений NUDA

NUDA [10] — это система расширений языка Nemerle, предназначенная для программирования ГПУ. Входящие в NUDA расширения можно условно разделить на следующие группы:

- Макросы, функции и типы данных, реализующие вложение подмножества OpenCL в язык Nemerle
- Макросы и аннотации, предназначенные для переноса на ГПУ кода и данных
- Аннотации, предназначенные для преобразования циклов и повышения производительности программ
- Прочие макросы и аннотации

Помимо расширений, NUDA включает набор обычных библиотечных функций и классов, осуществляющих интерфейс с системой времени исполнения OpenCL, а также вспомогательные операции по работе с кодом на языке Nemerle.

Работа система расширений NUDA организована следующим образом. На этапе компиляции срабатывают макросы NUDA, выполняются преобразования кода, а также генерация кода для ГПУ. Если компиляция проходит без ошибок, сгенерированный код на OpenCL сохраняется в создаваемую .NET-сборку в виде атрибутов. Во время исполнения этот код компилируется при помощи системы времени выполнения OpenCL, и исполняется на ГПУ.

Цикл nfor является макросом, позволяющим компактно записывать тесно вложенные гнёзда циклов. Общий вид полной формы цикла **nfor** вместе с кодом, в который он преобразуется по умолчанию, изображён на рис. 2. Существует также *сокращённая форма*, изображённая там же. Шаг по умолчанию равен 1; если же вместо диапазона указано только одно скалярное значение, n, то параметр цикла изменяется от 0 до $n-1$ включительно.

```
/* nfor macro ... */
nfor((i1, ..., iN) in (a1 <> b1 :/ c1, ..., aN <> bN :/ cN))
  S;
/* ... is expanded into */
for(mutable i1 = a1; i1 <= b1; i1 += c1)
  ...
  for(mutable iN = aN; iN <= bN; iN += cN)
    S;

/* short form is also possible */
nfor((i, j, k) in (n, n, n))
  a[i, j, k] += b[i, j, k];
```

Рис. 2. Общая (и сокращённая) форма цикла **nfor** и результат его трансляции

Для работы с памятью ГПУ используются специальные типы данных. Для работы с глобальной памятью служит тип `nuarrayXd[T]` (X — размерность массива), или *NUDA-массивы*, для изображений `nuimageXd` для локальной памяти `nulocalarrayXd` и для константной памяти `nuconstarrayXd`. Последние два могут использоваться только в программах, исполняемых на ГПУ. Эти типы данных эмулируют функциональность .NET-массивов и реализуют доступ к элементам по индексам, а также получение информации о ранге и размере по каждому из измерений.

Выделение памяти под массивы для ГПУ реализуется путём применения макросов к операции выделения массива в Nemerle. Массивы в локальной памяти могут выделяться только в коде на стороне ГПУ при помощи макроса **nulocal**; размер массива по каждому из измерений должен быть константой. NUDA-массивы и изображения могут выделяться только на стороне хоста. Для этого служат макросы **nunew** и **nuimg** соответственно,

генерирующие обращение к системе времени выполнения OpenCL для выделения памяти устройства. С каждым потоком хоста ассоциирован свой номер устройства (по умолчанию 0), на котором и выделяется память под массив. Управление глобальной памятью и памятью изображений осуществляется при помощи сборки мусора. Массивы константной памяти существуют только на стороне ГПУ; на стороне хоста они представляются как обычные NUDA-массивы.

NUDA-массивы и изображения содержат методы для копирования данных между ними и .NET-массивами соответствующего типа и ранга. Кроме того, в NUDA-массивах и изображениях предусмотрена возможность доступа к данным на стороне хоста и автоматической синхронизации между ними. Синхронизация осуществляется лениво: на сторону ГПУ массив копируется при вызове ядра, если он мог быть изменён на хосте. Обратно массив копируется при обращении к его элементу на стороне хоста. Память под такие массивы на стороне ГПУ выделяется всегда, а на стороне хоста — только когда там производится обращение к данным.

Точкой входа в исполнение программы на ГПУ является *NUDA-ядро*. Это статический метод класса, помеченный макросом **nukernel**. Ядро должно возвращать тип **void**, а типами его параметров должны быть простые типы данных, изображения или NUDA-массивы. Для метода-ядра генерируется, во-первых, *метод-заглушка* для его вызова на хосте, и во-вторых, код ядра OpenCL. Метод-заглушка выполняет установку параметров ядра, синхронизацию массивов и запуск ядра. В параметры заглушки, помимо параметров ядра, входят также номер устройства, размер сетки потоков и размер блока потоков. Заглушка возвращает управление только после завершения исполнения ядра на ГПУ. Вызов NUDA-ядра осуществляется только при помощи макроса **nucall**, который дополнительно указывает номер устройства и параметры сетки потоков. Пример объявления ядра и его вызова приведён на рис. 3. По сути, пара макросов **nucall** и **nukernel** аналогична нотации `<<<...>>>` и `__global__`-функциям в CUDA. Если на стороне ГПУ требуется вызвать пользовательскую функцию, к ней применяется макрос **nucode**; при этом сохраняется возможность вызова её на стороне хоста.

```
/* kernel declaration */
nukernel arrayBy2(b : nuarray2d[float], a : nuarray2d[float]) : void {
  def i = globalId(0); def j = globalId(1);
  b[i, j] = 2.0f * a[i, j];
}
/* ... */
def a = nunew array(n, n) : array[2, float];
def b = nunew array(n, n) : array[2, float];
/* ... here init array a ... */
/* kernel invocation */
nucall(0, [n, n], [1, 128]) arrayBy2(b, a);
```

Рис. 3. Пример объявления ядра в NUDA и его вызова

Чаще всего в качестве ядра для ГПУ используется тело цикла, поэтому хотелось бы иметь макрос для переноса цикла на ГПУ. В Nemerle это можно легко реализовать; такой макрос реализован в NUDA и называется **nuwork**. Он применяется к циклам **nfor** и группам из нескольких таких циклов. В качестве обязательных параметров он принимает размер блока потоков, по одному целочисленному выражению (не обязательно константному) по каждому из измерений. Макрос **nuwork** на основании текущего контекста и анализа кода тела цикла определяет набор переменных, которые используются в теле цикла, но определены вне него. Из них составляется список параметров ядра. Тело ядра формуриется из тела цикла, вычисления индексов цикла через глобальный номер потока, а также условия, позволяющего корректно исполнять цикл даже в том случае, когда глобальный размер сетки не делится на размер группы потоков. На место цикла подставляется вызов макроса **nucall**, осуществляющий вызов сгенерированного ядра. Пример использования **nuwork** приведён

на рис. 4; он решает ту же задачу, что и пример на рис. 3. Размер программы меньше за счёт отсутствия необходимости обявлять параметры ядра. Кроме того, в данном примере можно легко заменить типы массивов на изображения; если ядро выделено явно, это делать сложнее.

```
/* array allocation and initialization */
nuwork(1, 128) nfor((i, j) in (n, n))
  b[i, j] = 2.0f * a[i, j];
```

Рис. 4. Использование макроса **nuwork** для переноса цикла на ГПУ

Код на языке OpenCL генерируется из кода Nemerle-методов, помеченными макросами **nukernel** и **nucode**. Перед собственно генерацией кода выполняется раскрытие макросов. Для этого используется специальная функция, не раскрывающая стандартные макросы.

В сгенерированном OpenCL-коде сначала идут объявления типов, затем — прототипы функций, и наконец, код функций. При этом базовые типы языка Nemerle транслируются в базовые типы OpenCL. Указательные типы OpenCL представлены в Nemerle специальными типами `ptr[T]`, `globalptr[T]` и `localptr[T]`. Типы-массивы транслируются в структуры, содержащие размер массива и указатель на его данные. Типы-кортежи транслируются в структуры, содержащие поля кортежей. Типы-структуры транслируются в структуры OpenCL. Примеры сгенерированных для массивов и кортежей типов указаны на рисунке 5.

```
/* 2d floating-poing array */
typedef struct {
  global float* d;
  int l[2];
} array2d_g_float_t;
```

```
/* type type int * float
  * double */
typedef struct {
  int s0;
  float s1;
  double s2;
} tuple__int__float__double;
```

Рис. 5. Определения, сгенерированные в NUDA для типов-массивов и типов-кортежей

Прототипы сгенерированных функций в OpenCL получаются из прототипов функций в Nemerle заменой Nemerle-типов на соответствующие OpenCL-типы. При передаче в функции-ядра массивы разделяются на параметры, соответствующие указателям на данные и размерам массивов, которые передаются отдельно. В прологе кода ядра они собираются в структуры, описанные выше. Кроме того, в нестатические функции-члены структур добавляется первый параметр **this**, а возвращаемым типом конструкторов структур становится тип самих структур.

В подмножестве кода Nemerle, транслируемом в OpenCL, запрещено использование блочных операторов на уровне выражений. Таким образом, структура транслируемого кода Nemerle почти полностью соответствует структуре кода на C. Трансляция выполняется как процесс рекурсивной генерации строк кода на OpenCL по дереву кода на Nemerle. Кортежные объявления переменных транслируются в объявление нескольких переменных. Операции с массивами транслируются по-разному в зависимости от класса памяти массива. Наконец, вызовы функций также транслируются по-разному для обращений к статическим и нестатическим функциям, а также стандартным функциям OpenCL. Для ядра, изображенного на рис. 3, будет сгенерирован код ядра OpenCL, изображенный на рис. 6.

Иногда к выражению, чаще всего к циклу, требуется применить несколько преобразований подряд, при этом часть результата предыдущего преобразования подаётся на вход следующему. Просто макросы здесь не подходят, поскольку преобразование требуется применять не ко всему результату предыдущего преобразования, а к его “главной” части. Ко-

```
kernel arrayBy2(global float* b_d, int b_l0, int b_l1,
  global float* a_d, int a_l0, int a_l1) {
  array2d_g_float_t a; a.d = a_d; a.l[0] = a_l0; a.l[1] = a_l1;
  array2d_g_float_t b; b.d = b_d; b.l[0] = b_l0; b.l[1] = b_l1;
  int i = get_global_id(0);
  int j = get_global_id(1);
  b.d[i * b.l[0] + j] = 2.0f * a[i * a.l[0] + j];
}
```

Рис. 6. Пример ядра OpenCL, сгенерированного для ядра NUDA

нечно, можно главную часть вычленять в самом коде макроса — но как определить, что является главным? Если в результате преобразования одного цикла получается несколько новых — к какому из них применить очередное преобразование?

В этом случае используется *механизм аннотаций*. Каждая аннотация, помимо параметров и преобразуемого выражения, также принимает цепочку аннотаций для дальнейшего применения. Эта цепочка аннотаций применяется к "главной" части результата применения текущей аннотации. Для применения цепочки аннотаций к выражению служит макрос **annot**, пример использования которого показан на рис. 7. Сначала в гнезде будет изменён порядок циклов (**permut**), затем применена развёртка (**unroll**) и наконец, будет выполнен тайлинг цикла (**tilem**). Такой механизм позволяет выражать сложные преобразования через последовательность аннотаций, производящих относительно простые преобразования.

```
annot(tilem(8, 8), dmine(2, 2), permut(2, 1))
nfor((i, j) in (m, n)) c[j, i] = a[i] * b[j];
```

Рис. 7. Пример применения к циклу нескольких аннотаций

Аннотация **dmine** выполняет *глубокую развёртку* цикла **nfor**. Параметрами **dmine** являются размеры блока развёртки для каждого из измерений цикла, и они должны быть константами. В результате преобразования создаётся необходимое число копий тела цикла, которые затем перемешиваются: сначала выполняется первый оператор каждой из копий, затем второй и т.д. Если внутри тела цикла встречается другой цикл, параметры которого не зависят от номера итерации внешнего цикла, то содержимое его тела также перемешивается. Таким образом, глубокая развёртка позволяет увеличить количество однородных команд внутри наиболее вложенных циклов. Это приводит к улучшению шаблонов доступа в ОЗУ, и позволяет задействовать векторные команды для векторных архитектур. В экспериментах использование **dmine** в ряде случае позволило повысить производительность в несколько раз. Однако глубокая развёртка имеет и обратную сторону: слишком большой размер блока приводит к неэффективному использованию регистров. Пример использования аннотации **dmine** приведён на рис. 8.

Прочие аннотации Приведём описание ещё некоторых аннотаций, реализованных в NUDA:

- **inline** — выполняет полную развёртку цикла с постоянным числом итераций
- **nudevs** — исполняет каждую итерацию цикла на своём NUDA-устройстве в отдельном потоке; удобно при программировании нескольких ГПУ
- **peel** — позволяет отделить несколько итераций в начале и конце цикла от основного блока тела цикла
- **permut** — меняет порядок измерений цикла
- **tile**, **tilem** — тайлинг цикла; размер блока не обязан быть константой времени компиляции

```
/* dmine-annotated loop ... */
dmine(2) nfor(i in n) {
  mutable r = 0.0f;
  def aa = a[i];
  nfor(j in n) {def bb = b[j]; r += f(aa, bb);}
  c[i] = r;
}

/* is transformed into ... */
/* ... main loop ... */
nfor(i1 in n :/ 2) {
  mutable r0 = 0.0f; mutable r1 = 0.0f;
  def aa0 = a[i1]; def aa1 = a[i1 + 1];
  nfor(j in n) {
    def bb0 = b[0]; def bb1 = b[1];
    rr0 += f(aa0, bb0); rr1 += f(aa1, bb1);
  }
  c[i1] = r0; c[i1 + 1] = r1;
}
/* .. and tail */
nfor(i in n / 2 * 2 <> n - 1) {
  mutable r = 0.0f;
  def aa = a[i];
  nfor(j in n) {def bb = b[j]; r += f(aa, bb);}
  c[i] = r;
}
```

Рис. 8. Использование аннотации **dmine**

- **unroll** — простая развёртка циклов, без перемешивания итераций и преобразования вложенных циклов

С полным списком аннотаций в `nuda` можно ознакомиться в [10]

Параметры командной строки могут быть задействованы в NUDA-программах при помощи макроса **config**, который применяется к объявлению переменной или поля класса. Поддерживаются параметры строкового типа, а также простых типов; для последних используются стандартные функции .NET для преобразования строки в число. Значение параметра по умолчанию берётся из инициализатора в объявлении; в случае его отсутствия параметр должен быть указан в командной строке. Поддерживаются как длинные, так и короткие имена параметров. На рисунке 9 приведён пример аннотаций и командная строка, с которой может быть вызвано приложение. Для инициализация из переменных среды используется макрос **envfig**.

```
/* these variable declarations ... */
config("w") nwalkers = 4096;
config ("V") verbose = false;
config effNdevs = 1;

/* ... allow to invoke the program like this ... */
mono myprog.exe -w20480 --eff-ndevs 1 --verbose
```

Рис. 9. Инициализация переменных из параметров командной строки

4. Вычислительные эксперименты

Тестирование системы NUDA проводилось на ряде модельных задач, а также на задаче поиска коллизий для урезанной хэш-функции MD5 методом грубой силы. Тестовые модельные задачи и их характеристики приведены в таблице 1. Параметры систем, использовавшихся для тестирования, приведены в таблице 2. Тестирование проводилось в 2

этапа. На первом этапе выполнялась адаптация задачи для ГПУ с использованием макроса **nuwork**, а также подбирался оптимальный размер блока потоков. На втором этапе программа оптимизировалась при помощи аннотаций **dmine** и **inline**. При этом измерялось повышение производительности, а также определялся предполагаемый размер кода ядра при условии, что оно создавалось бы вручную. Производительность вычислялась исходя из времени исполнения ядра на ГПУ и не включает время копирования данных. Значения производительности и ускорения приведены в таблице 3. График роста размера кода для каждой из архитектур по сравнению с базовым вариантом приведён на рис. 10. Заметим, что размер кода указан после преобразований NUDA; размер исходного кода остаётся таким же. Полный код примеров, а также использованные аннотации можно найти в дистрибутиве NUDA [10].

Таблица 1. Задачи, используемые для тестирования NUDA

Имя	Название	Размер задачи	Размер кода
imgconv	Фильтрация изображений	4096 × 4096, фильтр 3 × 3	9 строк
nbody	Задача N тел	65536	27 строк
sgemm	Умножение матриц $A^T B$, 32 бит	2048 × 2048	10 строк
dgemm	Умножение матриц $A^T B$, 64 бит	2048 × 2048	10 строк

Таблица 2. Системы, используемые для тестирования NUDA-программ

Параметр	Система **tesla**	Система **fermi**	Система **ati**
ГПУ	NVidia Tesla C1060	NVidia Tesla C2050	AMD Radeon 5830
Пиковая производительность, 32 (64) бит	622 (90) ГФлоп/с	1030 (515) ГФлоп/с	1792 (358.4) ГФлоп/с
Реализация OpenCL	NVidia CUDA 3.1		AMD Stream SDK 2.2

Из всех задач, наименьший рост производительности достигается на задаче N тел, главным образом благодаря её высокой производрительности неоптимизированного варианта. Далее, производительность базовых вариантов всех задач на архитектуре NVidia Tesla C2050 выше — за счёт наличия аппаратной кэш-памяти. Тем не менее, и там фильтрацию изображений удаётся ускорить почти в 2 раза, а умножение матриц — в 4–6 раз. Максимальная достигаемая производительность на ГПУ AMD сравнительно невелика; очевидно, текущая реализация OpenCL не может организовать кэширование обращений в память по указателям. Тем не менее, и там удаётся получить ускорение в 1.5–4 раза, в зависимости от задачи. Ни в одной из реализаций не используется работа с локальной разделяемой памятью, т.к. соответствующие преобразования пока не реализованы в NUDA. Тем не менее, на задаче **dgemm** на ГПУ Tesla C2050 удаётся добиться эффективности решения задачи 40%. Это значение производительности находится на уровне NVidia CUBLAS 3.1, и всего лишь на 20 процентных пунктов хуже, чем самая эффективная на сегодняшний день версия, реализованная в NVidia CUBLAS 3.2.

Эффективность NUDA также тестировалась на задаче поиска коллизий для урезанной хэш-функции MD5 методом грубой силы. Наиболее вычислительно ёмкой частью данной задачи является вычисление результатов последовательного применения хэш-функции к

Таблица 3. Производительность системы NUDA на различных тестовых задачах

Задача	Система	Базовая производительность, ГФлоп/с	Оптимизированная производительность, ГФлоп/с	Ускорение, раз
imgconv	**tesla**	8	40	×5
imgconv	**fermi**	97	187	×1.93
imgconv	**ati**	8.8	13.5	×1.53
nbody	**tesla**	129	154	×1.19
nbody	**fermi**	328	419	×1.28
nbody	**ati**	77	176	×2.28
sgemm	**tesla**	11.34	85.91	×7.57
sgemm	**fermi**	77	445	×5.78
sgemm	**ati**	54	234	×4.33
dgemm	**tesla**	17.84	43.5	×2.44
dgemm	**fermi**	52	216	×4.15
dgemm	**ati**	28	111	×3.96

исходному значению. Эта операция, в свою очередь, применяется к большому количеству значений параллельно, и для этого задействуются графические процессоры. Задача практически не взаимодействует с ОЗУ, все вычисления происходят на регистрах. Тем не менее, за счёт использования аннотаций NUDA, а именно глубокой развёртки цикла, отправляемого на ГПУ, как видно из рис. 11, на ГПУ AMD удалось повысить производительность почти в 4 раза. В терминах целочисленных операций, эффективность использования ресурсов ГПУ AMD составила 95%.

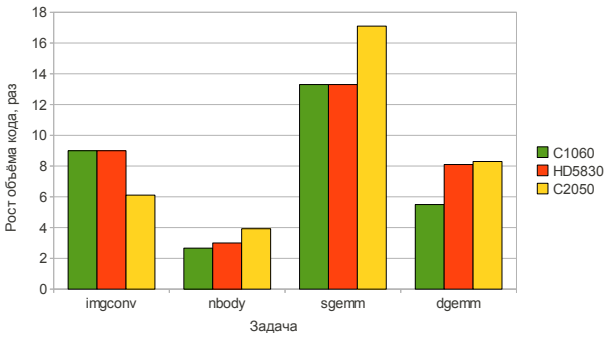


Рис. 10. Увеличение объёма кода при (предполагаемой) ручной оптимизации по сравнению с базовым вариантом

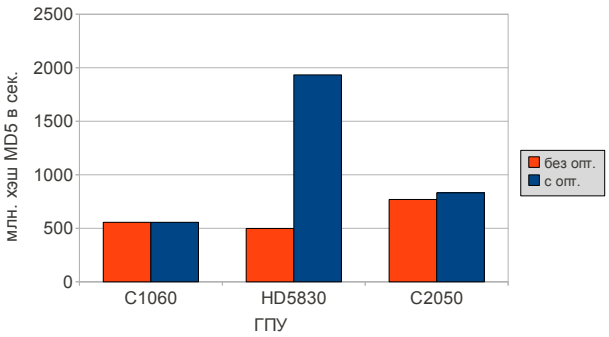


Рис. 11. Производительность вычисления хэш-функции MD5, реализованного с помощью системы NUDA, оптимизациями и без, на различных ГПУ

5. Обсуждение и дальнейшая работа

При помощи макросов в рамках системы NUDA для расширяемого языка Nemerle удалось реализовать функциональность программирования ускорителей и преобразования циклов при помощи аннотаций. Для ряда задач это позволило получить значительное увеличение производительности без увеличения размера исходного кода. Заметим, что изначально язык Nemerle не был особым образом предназначен для параллельного программирования — он просто является расширяемым языком. Тем не менее, в итоге удалось создать систему, по функционалу сопоставимую с коммерческими решениями, такими как PGI Accelerator или CAPS HMPP. Объём кода системы NUDA (без тестов) составляет всего 11 тыс. строк. Объём кода коммерческой системы, такой как CAPS HMPP, скорее всего, намного больше. Таким образом, продемонстрирована эффективность использования расширяемых языков для создания систем для прогарммирования высокопроизводительных архитектур, в частности ГПУ.

Тем не менее, функциональность системы NUDA требует дальнейшего расширения. Прежде всего, следует создать аннотации для автоматизации работы с другими уровнями иерархии памяти ГПУ, в особенности разделяемой общей памяти. Далее, актуальной задачей является автоматический подбор оптимизирующих преобразований — как на основе моделей, так и на основе эмпирического поиска — и это тоже одно из направлений дальнейших исследований. Также имеет смысл разрабатывать расширения, облегчающие программирование систем из нескольких ГПУ. Наконец, важным направлением будет создание расширений для решения на ГПУ нетривиальных задач — например, задач с рекурсивным параллелизмом, или задач, требующих использования сложных примитивов синхронизации.

Кроме того, Nemerle не является идеальным расширяемым языком для высокопроизводительных вычислений. Прежде всего, получение семантической информации внутри макросов затруднено. В частности, информация об объявленных переменных и типах выражений становится доступной только в процессе развёртки макросов. Если же требуется и получить информацию, и сохранить макросы в дереве исходного кода, требуется использовать обходные пути, которые не всегда работают. Во-вторых, все функции в Nemerle-программе транслируются независимо, и нет возможности получить код одной функции из другой функции. Как следствие, реализация преобразований типа встраивания функций или генерации специальных версий функций для определённых наборов параметров затруднены. В-третьих, для инициирования преобразований требуется применять макросы; возможность инициирования преобразований через определённые шаблоны кода отсутствует. Соответственно, нет возможности выполнить преобразования, добавляющие новую семантику без добавления синтаксиса — как добавление в язык массивного программирования. Наконец, в языке отсутствует гибкий механизм назначения атрибутов вершинам дерева кода и пере-

менным. Всё это говорит о том, что для высокопроизводительных вычислений потребуется разработать более гибкий расширяемый язык программирования. Но для этого сначала потребуется попробовать решить при помощи NUDA и Nemerle более широкий круг задач — чтобы определить требования, предъявляемые к новому языку.

6. Заключение

В статье рассказано о парадигме расширяемого программирования, её преимуществах и возможностях использования для программирования ГПУ. Была описана система NUDA, представляющая собой набор расширений языка Nemerle для программирования ГПУ и преобразования исходного кода программ. Разработанная система была апробирована на ряде задач, для которых позволила добиться увеличения производительности в несколько раз без увеличения объёма исходного кода. При этом, если бы соответствующий код писался вручную, его объём вырос бы в несколько раз, а читабельность программы значительно снизилась бы.

Проведённые исследования показывают перспективность использования расширяемых языков для программирования высокопроизводительных вычислений. Однако существующие в настоящее время языки не являются для этого достаточно гибкими. Для получения реальных преимуществ потребуется разрабатывать новые расширяемые языки, а для этого требуется исследовать возможность решения существующих задач при помощи языка Nemerle с целью выработки требования к новым инструментам. Что и является основным направлением дальнейшей работы.

Литература

1. TOP 500 List — November 2010. URL: http://top500.org/list/2010/11/100 (дата обращения 13.02.2011).
2. NVidia Corporation. NVidia CUDA C Programming Guide, Version 3.2. 183 p.
3. Khronos Group. The OpenCL Specification, version 1.1, document revision 33. 379 p. URL: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (дата обращения 13.02.2011).
4. The Portland Group. PGI Accelerator Programming Model for Fortran & C, version 1.3. 36 p. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf
5. Caps Enterprise. CAPS HMPP Workbench User Guide, version 2.3.1. 120 p.
6. Wilson G.V. Extensible Programming for the 21st Century // ACM Queue. January 2005. vol. 2. pp. 48–57.
7. Seibel P. Practical Common Lisp. ISBN 1590592395. Apress, 2005.
8. Nemerle Homepage. URL: http://nemerle.org/ (дата обращения 13.02.2011).
9. Cox R., Bergan T., Clements A.T., Kaashoek F., Kohler E. Xoc, an extension-oriented compiler for systems programming. // Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS XIII). pp. 244-254
10. Adinetz A.V., Shvets P., Sitchikhin V. NUDA Project web site. URL: http://nuda.sf.net/ (дата обращения 13.02.2011).

Параллельные алгоритмы решения СЛАУ с блочно-трехдиагональными матрицами на многопроцессорных вычислителях*

Е.Н. Акимова, Д.В. Белоусов

Институт математики и механики УрО РАН

Для решения СЛАУ с блочно-трехдиагональными матрицами предложены и численно реализованы на многопроцессорных вычислителях различного типа параллельный алгоритм матричной прогонки и параллельный метод сопряженных градиентов с предобуславливателем. Проведено сравнение времени счета параллельных алгоритмов на видеоускорителе NVIDIA GeForce GTX, 4-х ядерном процессоре Intel Core I5-750 и многопроцессорном вычислительном комплексе МВС-1000 при решении задачи о нахождении распределения потенциала на поверхности земли в проводящей среде.

1. Введение

Системы линейных алгебраических уравнений (СЛАУ) с блочно-трехдиагональными матрицами возникают при решении ряда задач математического моделирования, в частности, при решении задачи диффузии и задач электроразведки.

Важной задачей при моделировании структурных превращений в многокомпонентных сплавах является решение задачи диффузионного массопереноса, когда в каждый момент времени необходимо знать распределение концентраций диффундирующих компонентов. Диффузионный массоперенос описывается системой параболических дифференциальных уравнений в частных производных вида уравнений в частных производных вида

$$\frac{\partial C_i}{\partial \tau} = \frac{1}{r^q}\frac{\partial}{\partial r}\left(r^q \cdot \sum_{j=1}^{N-1} \tilde{D}_{ij} \frac{\partial C_j}{\partial r} \right), \tag{1}$$

где N – число компонентов сплава; r – пространственная координата; C_j – концентрация j -го компонента; $\tilde{D}_{ij}$ – парциальные коэффициенты взаимной диффузии; q –показатель степени, зависящий от симметрии задачи: 0 – для плоской, 1 – для цилиндрической, 2 – для сферической.

При использовании абсолютно устойчивой неявной разностной схемы система дифференциальных уравнений (1) сводится к системе уравнений с блочно – трехдиагональной матрицей [1–2]. Для i -го компонента в k -ом узле пространственной сетки система уравнений имеет вид

$$\sum_{j=1}^{N-1} \left[a_{ik}^j C_j(k-1) + c_{ik}^j C_j(k) + b_{ik}^j C_j(k+1) \right] = d_k^i, \tag{2}$$

где $a_{ik}^j, b_{ik}^j, c_{ik}^j, d_k^i$ – коэффициенты, рассчитываемые в зависимости от модели, $C_j(k)$ – концентрация j - го компонента в k - м узле сетки.

Важнейшими задачами исследования неоднородности земной коры являются задачи электроразведки. Одним из самых известных методов электроразведки является метод вертикального электрического зондирования (ВЭЗ). На поверхности земли собирают электроразведочную установку, которая состоит из двух питающих и двух приемных электродов. К питающим

* Работа выполнена в рамках Программы фундаментальных исследований Президиума РАН № 14 при поддержке УрО РАН, проект 09-П-1-1003.

электродам A и B подключают источник тока, а на приемных электродах M и N измеряют напряженность электрического поля. По результатам выполненных измерений вычисляют кажущееся электрическое сопротивление $\rho_k = K \cdot \Delta V_{MN} / I_{AB}$ для неоднородной среды, где K – геометрический коэффициент, зависящий от расстояний между электродами A, B, M и N, ΔV_{MN} – разность потенциалов на приемных электродах M и N, I_{AB} – сила тока, протекающего в питающей линии.

Для модели среды с плоско-параллельными горизонтальными границами, когда слои земной коры залегают горизонтально и сопротивление среды зависит только от глубины $\rho = \rho(z)$, кажущееся сопротивление $\rho = \rho(r)$, где $r = AO$ – расстояние от центра приемной цепи до питающего электрода. Задача интерпретации результатов ВЭЗ заключается в определении $\rho(z)$, дающего «электрический разрез» среды по известным значениям $\rho(r)$ (ось z направлена вниз, ось r направлена вправо).

Задача ВЭЗ о нахождении потенциала $V_0(r, z = 0)$ на поверхности земли сводится к решению двумерного уравнения Лапласа в цилиндрической системе координат [3]

$$\Delta V \equiv \frac{\partial^2 V}{\partial r^2} + \frac{1}{r}\frac{\partial V}{\partial r} + \frac{\partial^2 V}{\partial z^2} = 0 \tag{3}$$

с граничными условиями непрерывности потенциала и непрерывности нормальной составляющей к границам плотности тока

$V_o\big|_{z=l} = V_1\big|_{z=l}$, $\frac{1}{\rho_0}\frac{\partial V_0}{\partial z}\big|_{z=l} = \frac{1}{\rho_1}\frac{\partial V_1}{\partial z}\big|_{z=l}$ – условия на горизонтальных прямых $z = l$, (4)

$\frac{\partial V_0}{\partial z}\big|_{z=0} = 0$ – сверху, $\frac{\partial V_0}{\partial r}\big|_{r=0} = 0$ – слева, $V_0 = 0$ – справа и снизу. Здесь $\rho = \frac{2\pi r^2}{I} \cdot \left|\frac{\partial V}{\partial r}\right|$.

После использования конечно-разностной аппроксимации краевая задача (3) – (4) сводится к решению СЛАУ с блочно-трехдиагональной матрицей.

Другой важной задачей электроразведки является задача бокового каротажного зондирования (БКЗ), предусматривающая использование приборов однотипных зондов разной длины при измерении потенциала электрического поля. В результате интерпретации данных каротажа получают значение удельного электрического сопротивления пласта, близкое к истинному.

В работе [4] показано, что после использования конечно-разностной аппроксимации задача БКЗ сводится к решению системы линейных алгебраических уравнений с блочно-трехдиагональной матрицей следующего вида

$$AV \equiv -\frac{1}{r}\left(\bar{r}\, a V_{\bar{r}}\right)_{\hat{r}} - \left(b V_{\bar{z}}\right)_{\hat{z}} = F. \tag{5}$$

Здесь A – матрица размерности $(N_r \times 2N_z) \times (N_r \times 2N_z)$, имеющая блоки размерности $N_r \times N_r$ (рис. 1); V и F – векторы размерности $N_r \times 2N_z$; N_r – число точек по r; $2 \times N_z$ – число точек по z; $\sigma = \frac{1}{\rho}$ – коэффициент электропроводности,

$$a_{ij} = \sigma\left(r_i - \frac{h_i^{(r)}}{2}, z_j + \frac{h_j^{(z)}}{2}\right), \quad b_{ij} = \sigma\left(r_i + \frac{h_i^{(r)}}{2}, z_j - \frac{h_j^{(z)}}{2}\right),$$

$$(V)_{\bar{r},ij} = \left(V_{ij} - V_{i-1,j}\right)/ h_i^{(r)}, \quad (V)_{\hat{r},ij} = \left(V_{i+1,j} - V_{i,j}\right)/ \hbar_i^{(r)},$$

$$(V)_{\bar{z},ij} = \left(V_{ij} - V_{i,j-1}\right)/ h_i^{(z)}, \quad (V)_{\hat{z},ij} = \left(V_{i,j+1} - V_{i,j}\right)/ \hbar_i^{(z)},$$

$$\hbar_i^{(r)} = \left(h_i^{(r)} + h_{i+1}^{(r)}\right)/2, \quad h_i^{(r)} = r_i - r_{i-1}, \quad i = 1,..., N_r,$$

$$\hbar_j^{(z)} = \left(h_j^{(z)} + h_{j+1}^{(z)}\right)/2, \quad h_j^{(z)} = z_j - z_{j-1}, \quad j = -N_z+1,..., N_z.$$

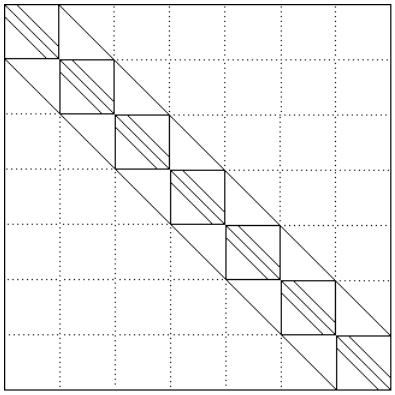


Рис. 1. Вид матрицы СЛАУ

Целью данной работы является разработка и реализация алгоритмов решения СЛАУ с блочно-трехдиагональными матрицами на многопроцессорных вычислителях различного типа: на видеоускорителе NVIDIA GeForce GTX, 4-х ядерном процессоре Intel Core I5-750 и многопроцессорном вычислительном комплексе МВС-1000 и сравнение времени счета параллельных алгоритмов при решении модельной задачи.

2. Параллельные методы решения задачи

Рассмотрим систему уравнений с блочно-трехдиагональными матрицами

$$\begin{cases} C_0\overline{Y}_0 - B_0\overline{Y}_1 = \overline{F}_0, & i = 0 \\ -A_i\overline{Y}_{i-1} + C_i\overline{Y}_i - B_i\overline{Y}_{i+1} = \overline{F}_i, & i = 1,..., N-1 \\ -A_N\overline{Y}_{N-1} + C_N\overline{Y}_N = \overline{F}_N, & i = N, \end{cases} \tag{6}$$

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.

Для решения СЛАУ (6) предлагается использовать параллельный прямой метод матричной прогонки [5] и параллельный итерационный метод сопряженных градиентов с предобуславливателем в случае решения СЛАУ с симметричной положительно-определенной матрицей.

Ранее параллельный метод матричной прогонки, реализованный на многопроцессорном вычислительном комплексе МВС-1000, эффективно использовался при решении модельной задачи диффузии о насыщении плоской металлической пластины тремя компонентами с различной диффузионной подвижностью, а также при моделировании эволюции выделений в двухфазном многокомпонентном сплаве [6–7].

2.1 Прямой метод решения задачи

Опишем параллельный алгоритм матричной прогонки для решения СЛАУ с блочно-трехдиагональными матрицами, к которым сводятся разностные задачи для эллиптических уравнений второго порядка с переменными коэффициентами в области P. Будем предполагать, что область P представляет собой прямоугольник.

При построении параллельного алгоритма исходную область P разобьем на L подобластей (рис. 2) вертикальными линиями так, что $N = L \times M$.

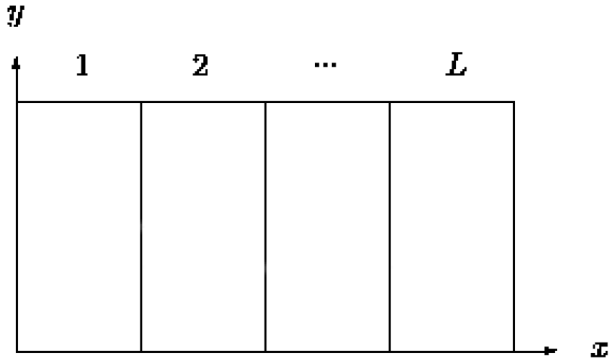


Рис. 2. Разбиение области на подобласти

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. Для этого в подобластях, определяемых интервалами $(K, K+M)$, рассматриваются задачи:

$$\begin{cases} -A_i\overline{U}^1_{i-1} + C_i\overline{U}^1_i - B_i\overline{U}^1_{i+1} = 0, \quad \overline{U}^1_K = \begin{pmatrix} 1 \\ 0 \\ .. \\ 0 \end{pmatrix}, \quad \overline{U}^1_{K+M} = \begin{pmatrix} 0 \\ .. \\ 0 \\ 0 \end{pmatrix}, \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ -A_i\overline{U}^n_{i-1} + C_i\overline{U}^n_i - B_i\overline{U}^n_{i+1} = 0, \quad \overline{U}^n_K = \begin{pmatrix} 0 \\ .. \\ 0 \\ 1 \end{pmatrix}, \quad \overline{U}^n_{K+M} = \begin{pmatrix} 0 \\ .. \\ 0 \\ 0 \end{pmatrix}. \end{cases} \tag{7}$$

$$\begin{cases} -A_i\overline{V}^1_{i-1} + C_i\overline{V}^1_i - B_i\overline{V}^1_{i+1} = 0, \quad \overline{V}^1_K = \begin{pmatrix} 0 \\ 0 \\ .. \\ 0 \end{pmatrix}, \quad \overline{V}^1_{K+M} = \begin{pmatrix} 1 \\ 0 \\ .. \\ 0 \end{pmatrix}, \\ \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ -A_i\overline{V}^n_{i-1} + C_i\overline{V}^n_i - B_i\overline{V}^n_{i+1} = 0, \quad \overline{V}^n_K = \begin{pmatrix} 0 \\ .. \\ 0 \\ 0 \end{pmatrix}, \quad \overline{V}^n_{K+M} = \begin{pmatrix} 0 \\ .. \\ 0 \\ 1 \end{pmatrix}. \end{cases} \tag{8}$$

$$-A_i\overline{W}_{i-1} + C_i\overline{W}_i - B_i\overline{W}_{i+1} = \overline{F}_i, \quad \overline{W}_K = \begin{pmatrix} 0 \\ 0 \\ .. \\ 0 \end{pmatrix}, \quad \overline{W}_{K+M} = \begin{pmatrix} 0 \\ 0 \\ .. \\ 0 \end{pmatrix}. \tag{9}$$

где $i = K+1, ..., K+M-1$.

Утверждение 1. Если $\overline{U}^1_i, ..., \overline{U}^n_i$ – решения задач (7), $\overline{V}^1_i, ..., \overline{V}^n_i$ – решения задач (8), $\overline{W}_i$ – решения задачи (9), а $\overline{Y}_i$ – решения исходной задачи (6) на $(K, K+M)$, тогда

$$\overline{Y}_i = \left(\overline{U}^1_i\overline{U}^2_i ... \overline{U}^n_i\right)\overline{Y}_K + \left(\overline{V}^1_i\overline{V}^2_i ... \overline{V}^n_i\right)\overline{Y}_{K+M} + \overline{W}_i. \tag{10}$$

После подстановки соотношений (10) в исходную систему (6) в точках $K = 0, M, ..., N$, получим систему уравнений относительно параметров $\overline{Y}_K$. Эта система уравнений по своей структуре аналогична (6), имеет меньшую размерность и следующий вид:

$$\begin{cases} \left[C_0 - B_0U_1\right]\overline{Y}_0 - \left[B_0V_1\right]\overline{Y}_M = \overline{F}_0 + B_0\overline{W}_1, & K = 0; \\ -\left[A_KU_{K-1}\right]\overline{Y}_{K-M} + \left[C_K - A_KV_{K-1} - B_KU_{K+1}\right]\overline{Y}_K - \left[B_KV_{K+1}\right]\overline{Y}_{K+M} = \\ = \overline{F}_K + A_K\overline{W}_{K-1} + B_K\overline{W}_{K+1}, & K = M, 2M, ..., N-M; \\ -\left[A_NU_{N-1}\right]\overline{Y}_{N-M} + \left[C_N - A_NV_{N-1}\right]\overline{Y}_N = \overline{F}_N + A_N\overline{W}_{N-1}, & K = N, \end{cases} \tag{11}$$

где U_K и V_K – квадратные матрицы порядка *n*.

Задача (11) решается классическим алгоритмом матричной прогонки [8] на одном процессоре, задачи (7) – (9) решаются независимо в *L* подобластях на *L* процессорах.

Матрицы U_i, V_i и вектор $\overline{W}_i$ на интервалах $(K, K+M)$ вычисляются независимо на *L* процессорах по следующим формулам.

а) Прямой ход прогонки:

$$\begin{aligned} &\alpha_{K+1} = C_{K+1}^{-1}B_{K+1}, \qquad \beta_{K+1} = C_{K+1}^{-1}A_{K+1}, \qquad \overline{\gamma}_{K+1} = C_{K+1}^{-1}\overline{F}_{K+1}. \\ &\alpha_i = \left[C_i - A_i\alpha_{i-1}\right]^{-1}B_i, \qquad \beta_i = \left[C_i - A_i\alpha_{i-1}\right]^{-1}A_i\beta_{i-1}, \\ &\overline{\gamma}_i = \left[C_i - A_i\alpha_{i-1}\right]^{-1}\left[\overline{F}_i + A_i\overline{\gamma}_{i-1}\right], \qquad i = K+2, ..., K+M-1. \end{aligned} \tag{12}$$

б) Обратный ход прогонки:

$$\begin{aligned} &U_{K+M-1} = \beta_{K+M-1}, \qquad V_{K+M-1} = \alpha_{K+M-1}, \qquad \overline{W}_{K+M-1} = \overline{\gamma}_{K+M-1}. \\ &U_i = \alpha_iU_{i+1} + \beta_i, \qquad V_i = \alpha_iV_{i+1}, \qquad \overline{W}_i = \alpha_i\overline{W}_{i+1} + \overline{\gamma}_i, \quad i = K+M-2, ..., K+1. \end{aligned} \tag{13}$$

После вычисления параметров Y_K остальные искомые неизвестные находятся по формуле (10) также независимо на каждом из *L* интервалов на *L* процессорах.

Схема параллельного алгоритма имеет вид: (12) → (13) → (11) → (10).

Утверждение 2. Если для исходной системы (6) выполняются достаточные условия устойчивости метода матричной прогонки по А.А. Самарскому [8]

$$\left\|C_0^{-1}B_0\right\| \le 1, \quad \left\|C_N^{-1}A_N\right\| \le 1, \quad \left\|C_i^{-1}A_i\right\| + \left\|C_i^{-1}B_i\right\| \le 1, \quad i = 1, ..., N-1,$$

причем хотя бы одно из неравенств – строгое, то эти же условия достаточны и для устойчивости метода матричной прогонки при решении системы уравнений (11) относительно параметров $\overline{Y}_K$ (см. [5]).

2.2 Итерационный метод решения задачи

Одним из быстрых итерационных методов решения СЛАУ с симметричной положительно-определенной матрицей является метод сопряженных градиентов (МСГ) [9].

Введение предобуславливания применяется с целью ускорения сходимости итерационного процесса и состоит в том, что исходная СЛАУ

$$Ax = b \tag{14}$$

заменяется на систему уравнений

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

для которой итерационный метод (в нашем случае МСГ) сходится существенно быстрее.

Пусть $C = C^* > 0$. Предположим, что предобуславливатель представлен в виде $C = B^* B$, где B – невырожденная квадратная матрица. Умножим слева обе части системы (15) на B и положим $y = Bx$. В результате придем к эквивалентной системе уравнений

$$\tilde{A}y = \tilde{b}, \tag{16}$$

где $\tilde{A} = (B^*)^{-1} A B^{-1}$, $\tilde{b} = (B^*)^{-1} b$.

Условием выбора предобуславливателя C является следующее

$$cond(\tilde{A}) << cond(A), \quad cond(\tilde{A}) = \frac{\tilde{\lambda}_{\max}}{\tilde{\lambda}_{\min}}, \quad cond(A) = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{17}$$

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 [10]

$$\begin{aligned} & r^0 = b - Ax^0, \quad p^0 = C^{-1} r^0, \quad z^0 = p^0, \\ & x^{k+1} = x^k + \alpha_k p^k, \qquad \alpha_k = \frac{(r^k, z^k)}{(Ap^k, p^k)}, \\ & r^{k+1} = r^k - \alpha_k A p^k, \qquad z^{k+1} = C^{-1} r^{k+1}, \\ & p^{k+1} = z^{k+1} + \beta_k p^k, \qquad \beta_k = \frac{(r^{k+1}, z^{k+1})}{(r^k, z^k)}. \end{aligned} \tag{18}$$

Условием останова итерационного процесса является

$$\frac{\|Ax - b\|}{\|b\|} < \varepsilon. \tag{19}$$

Распараллеливание итерационного метода сопряженных градиентов основано на разбиении матрицы A горизонтальными полосами на m блоков, а вектора решения x 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. 3).

На каждой итерации каждый из m процессоров вычисляет свою часть вектора решения. В случае умножения матрицы A на вектор x каждый из m процессоров умножает свою часть строк матрицы A на вектор x. Host-процессор отвечает за пересылки данных и также вычисляет свою часть вектора решения.

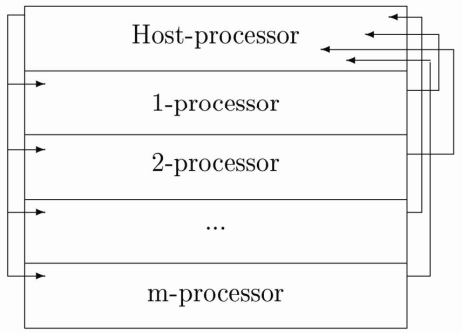


Рис. 3. Схема распределения данных по процессорам

3. Результаты численных экспериментов

Параллельный алгоритм матричной прогонки (ПАМП) и параллельный метод сопряженных градиентов с предобуславливателем (ПМСГ) реализованы на следующих современных многопроцессорных вычислительных системах: многопроцессорном вычислительном комплексе МВС–1000/64 с помощью технологии MPI [11], 4-х ядерном процессоре Intel Core I5-750 (CPU) на языке C++ в среде разработки «Visual Studio» (распараллеливание по потокам данных) и видеоускорителе NVIDIA GeForce GTX 285 (GPU) с помощью технологии CUDA [12].

Многопроцессорный вычислительный комплекс МВС–1000 – российский массивно-параллельный суперкомпьютер кластерного типа с распределенной памятью, установленный в Институте математики и механики УрО РАН.

МВС–1000/64 состоит из 14 2-х процессорных 2-х ядерных модулей AMD Opteron 64 bit (2.6 Ггц), интерфейса GbitEthernet и 112 Гб оперативной памяти.

Технические характеристики вычислительной платформы Intel Core I5-750 с видеоускорителем NVIDIA GeForce GTX 285 приводятся в табл. 1.

Таблица 1. Технические характеристики вычислительной платформы

CPU	4-х ядер. Intel Core I5-750
Частота процессора (ГГц)	2.66
Оперативная память (Гб)	8
Разрядность ОС (Бит)	64
GPU	NVIDIA GeForce GTX 285
Количество процессорных ядер	240
Частота ядра (МГц)	648
Частота процессора (МГц)	1476
Количество видеопамяти (Мб)	1024

С помощью параллельного алгоритма матричной прогонки и предобусловленного метода сопряженных градиентов решена модельная задача о нахождении распределения потенциала в проводящей среде с известным решением (рис. 4).

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

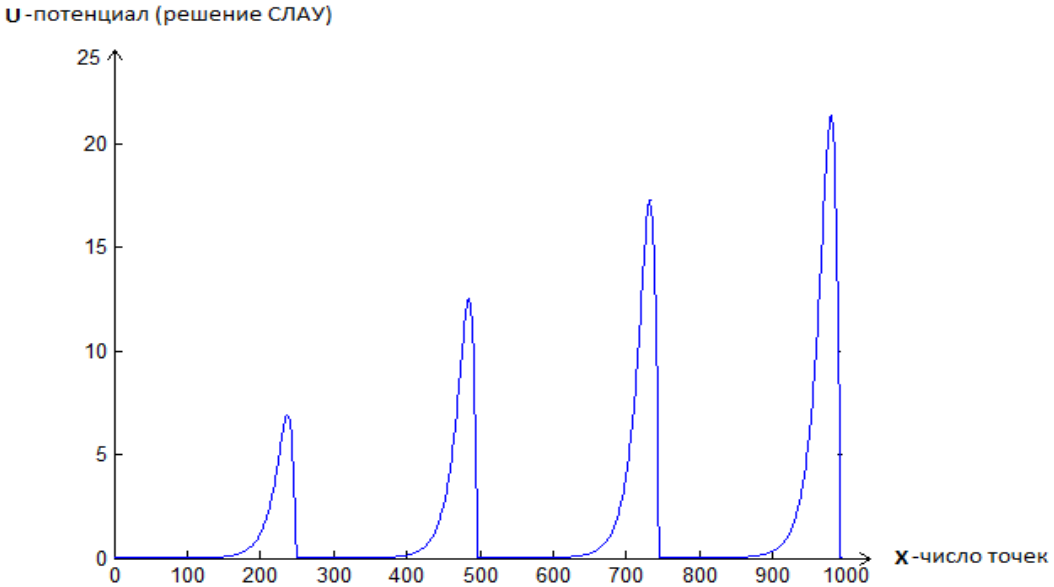


Рис. 4. Решение модельной задачи

После дискретизации задача сводится к решению СЛАУ с плохо обусловленной симметричной положительно-определенной блочно-трехдиагональной матрицей вида (6) размерности 76136×76136 с квадратными блоками порядка 248 (см. рис.1).

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

$$\sigma = \frac{\left\| \bar{Y}^{M} - \bar{Y}^{\Pi} \right\|}{\left\| \bar{Y}^{M} \right\|}, \tag{20}$$

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

Условие (20) выбиралось в качестве критерия останова итерационного ПМСГ при решении модельной 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 (17) находилось число обусловленности матрицы $\tilde{A}$

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

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–1000 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. T_m представляет собой совокупность чистого времени счета и накладных расходов на межпроцессорные обмены $T_m = T_c + T_o$. Число процессоров m соответствует упомянутому разбиению векторов на m частей и разбиению исходной области на m подобластей.

В общем случае эффективность распараллеливания меняется в пределах $0 < E_m < 1$. В идеальном случае при равномерной и сбалансированной загрузке процессоров и минимальном времени обменов между ними E_m близко к единице, но при решении практических задач она уменьшается за счет накладных расходов.

В табл. 2 и 3 приведены времена счета и коэффициенты ускорения и эффективности решения модельной задачи на 4-х ядерном процессоре Intel Core I5-750, видеоускорителе NVIDIA GeForce GTX 285 и многопроцессорном комплексе МВС–1000/64 с помощью параллельного метода сопряженных градиентов с предобуславливателем и параллельного алгоритма матричной прогонки при $\sigma_{ПАМП} \approx 2 \cdot 10^{-7}$.

Заметим, что время решения задачи с помощью метода сопряженных градиентов 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 и 3.

Таблица 2. Решение задачи методом ПМСГ

Вычислитель (число ядер или проц.)	T_m (время, сек.)	S_m либо S	E_m (эффект.)
Intel Core I5-750 (одно ядро)	57	–	–
Intel Core I5-750 (два ядра)	46	1.24	0.62
Intel Core I5-750 (четыре ядра)	36	1.50	0.40
GeForce GTX 285	16	3.56	–
МВС–1000/64 (1 проц.)	120	–	–
МВС–1000/64 (2 проц.)	65	1.85	0.92
МВС–1000/64 (4 проц.)	34	3.53	0.88

Таблица 3. Решения задачи методом ПАМП

Вычислитель (число ядер или проц.)	T_m (время, сек.)	S_m либо S	E_m (эффект.)
Intel Core I5-750 (одно ядро)	52	–	–
Intel Core I5-750 (два ядра)	28	1.86	0.93
Intel Core I5-750 (четыре ядра)	16	3.25	0.81
GeForce GTX 285	10	5.2	–
МВС–1000/64 (1 проц.)	96	–	–
МВС–1000/64 (2 проц.)	60	1.6	0.80
МВС–1000/64 (4 проц.)	31	3.1	0.77

4. Заключение

Для решения СЛАУ с блочно-трехдиагональными матрицами предложены и численно реализованы на многопроцессорных вычислителях различного типа параллельный алгоритм матричной прогонки и параллельный метод сопряженных градиентов с предобуславливателем.

Проведено сравнение времени счета параллельных алгоритмов на видеоускорителе NVIDIA GeForce GTX, 4-х ядерном процессоре Intel Core I5-750 и многопроцессорном вычислительном комплексе МВС–1000/64 при решении модельной задачи о нахождении распределения потенциала на поверхности земли в проводящей среде.

Использование параллельных методов матричной прогонки и сопряженных градиентов с предобуславливателем позволяет достаточно быстро решать задачи с плохо обусловленными матрицами на многопроцессорных вычислителях различного типа, что позволяет рекомендовать данные методы для решения задач электроразведки.

Авторы выражают признательность за полезные советы и обсуждения и внимание к работе члену-корреспонденту РАН В.В. Васину.

Литература

1. Crank J. The Mathematics of Diffusion. Oxford: Clarendon press, 1975.
2. Самарский А.А. Теория разностных схем. М.: Наука, 1983.
3. Тихонов А.Н., Самарский А.А. Уравнения математической физики. М.: Наука, 1966.

4. 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.

5. Акимова Е.Н. Распараллеливание алгоритма матричной прогонки // Математическое моделирование. 1994. Т. 6. № 9. С. 61-67.

6. Акимова Е.Н., Горбачев И.И., Попов В.В. Решение задач многокомпонентной диффузии с помощью алгоритма матричной прогонки // Мат. моделирование. 2005.Т. 17. № 9. С. 85-92.

7. Горбачев И.И., Попов В.В., Акимова Е.Н. Моделирование диффузионного взаимодействия карбонитридных выделений с аустенитной матрицей с учетом возможности изменения их состава // Физика металлов и металловедение. 2006. Т. 102. № 1. С. 22-32.

8. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978.

9. Фаддеев В.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Гос. издат. физ.-мат. литературы, 1963.

10. Амосов А.А. Циркулянтно предобусловленный метод сопряженных градиентов и его применение для численного решения интегрального уравнения переноса излучения // Труды XI Всероссийской школы-семинара «Современные проблемы математического моделирования». Ростов-на-Дону: Издат. Ростов. госун-та, 2005. Выпуск 4. С. 49-65.

11. Баранов А.В., Лацис А.О., Сажин С.В., Храмцов М.Ю. Руководство пользователя системы МВС-1000. URL: http://parallel.ru/mvs/user.html (дата обращения: 15.02.2011).

12. Берилло А. NVIDIA CUDA – неграфические вычисления на графических процессорах. URL: http://www.ixbt.com/video3/cuda-1.shtml (дата обращения: 15.02.2011).

Анализ эффективности распараллеливания решателей пакета ANSYS Multiphysics при моделировании термопрочностных задач в процессах линейной сварки трением

А.Т. Бикмеев[1], Р.К. Газизов[1], В.Ю. Иванов[2], А.А. Касаткин[1], В.В. Латыш[3], С.Ю. Лукащук[1], И.Ш. Насибуллаев[1], К.Р. Юлмухаметов[1], А.М. Ямилева[1]

Уфимский государственный авиационный технический университет[1],
Уфимское моторостроительное производственное объединение[2],
ГУП Научное конструкторско-технологическое бюро «Искра»[3]

Особенностью процесса линейной сварки трением (ЛСТ) является быстротечность процесса, сопровождаемая большими градиентами температуры и напряжений. Моделирование этого процесса в пакете ANSYS Multiphysics требует использование конечных элементов малого размера, а также малого шага по времени, что, в свою очередь, приводит к необходимости использования многоядерных и кластерных вычислительных систем и возможностей параллельных решателей пакета. Как показывают результаты этих расчетов, далеко не все решатели одинаково эффективны.

Введение

При производстве современных лопаток для авиационных двигателей и их объединении в блиски требуется использовать новые методы соединения разных частей изделия. Наиболее широко используются линейная сварка трением.

Сварка трением – это разновидность сварки давлением, при которой нагрев осуществляется трением, вызванным перемещением друг относительно друга соединяемых частей свариваемого изделия [1]. Процесс линейной сварки трением (ЛСТ) осуществляется возвратно-поступательным движением частей, подлежащих свариванию, с частотой порядка 60 Гц и амплитудой до 3-х мм, сжимаемых для образования плотного контакта.

Согласно Вайрису и Фросту [2] в процессе линейной сварки трением выделяют четыре стадии (фазы): начальная, переходная, равновесная и завершающая. На начальной стадии свариваемые изделия приводятся в контакт под давлением и начинается их относительное движение, сопровождаемое нагревом. В течение этого процесса область контакта увеличивается за счет изнашивания шероховатостей. При достаточном тепловыделении за счет трения материал на границе становится мягче, и большие неровности начинают стачиваться на границе контакта – наступает переходная стадия процесса. Площадь контакта становится равной площади сечения образцов и образуется мягкий пластический слой, который уже не может сдерживать прижимающую нагрузку. Затем наступает равновесная стадия, характеризующаяся осевым укорачиванием вследствие вытеснения пластичного материала из зоны контакта, образуется грат. На завершающей стадии механическое движение завершается и к образцам прикладывается дополнительное давление (проковка) для образования сварного соединения.

При математическом моделировании процесса линейной сварки необходимо учитывать следующие особенности процесса:

- Быстротечность процесса – весь процесс линейной сварки трением составляет около 2 сек. За это время происходит нагрев на 500-600 С и напряжения достигают предела текучести (100 МПа). Высокая скорость процесса – частота колебаний брусков составляет порядка 50 Гц при амплитуде колебаний порядка 2 мм. Для сходимости расчетных методов необходимо выбирать очень маленький шаг по времени (10^{-4}...10^{-5}

сек.). Время расчета возрастает за счет большого количества итераций по времени (1000 итераций за 1 шаг, необходимость разбивки всего процесса на несколько шагов).

- Ограниченное количество симметрий модели. Бруски имеют только зеркальную симметрию в плоскости движения, что позволяет уменьшить количество элементов, участвующих в расчете, только вдвое. При расчетах моделей, приближенных к реальности (форма лопатки) симметрия отсутствует. Это означает, что необходимо моделировать полную трехмерную задачу с мелкой неравномерной сеткой.

Таким образом, для моделирования процесса ЛСТ требуются большие вычислительные ресурсы, включая возможность проведения параллельных вычислений на многопроцессорных системах.

В данной работе было проведено математическое моделирование второй стадии линейной сварки трением в пакете ANSYS Multiphysics. Проведена оценка эффективности различных решателей на разных задачах.

1. Постановка задачи

Математическое моделирование процесса линейной сварки трением является сложной задачей, поскольку в рамках этого процесса происходит несколько физических явлений:

1. неравномерное движение тел (используются законы механики);
2. скольжение одного тела по поверхности другого (используются законы механики);
3. нагрев и остывание (используются законы термодинамики);
4. упруго-пластические деформации (используются законы теории упругости и термодинамики).

В связи с этим часто всю задачу разбивают на отдельные части по физическим явлениям и моделируют их по отдельности.

При деформации тела его точки смещаются относительно первоначального положения $\mathbf{r}^{(0)} = \{x_i^{(0)}\}$. Для описания деформаций вводят вектор деформаций $u_i = x_i - x_i^{(0)}$ [3]. В теории упругости для описания деформаций принято использовать не вектор, а тензор деформаций, определяемый следующим образом:

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right) \tag{1}$$

В деформированном теле возникают силы, стремящиеся вернуть его в исходное состояние. Эти силы называют внутренними напряжениями и описывают при помощи тензора напряжений. Если на тело действует сила $\mathbf{F} = \{F_i\}$, то компоненты тензора внутренних напряжений для тела в равновесном состоянии можно получить из уравнения:

$$\sum_{k=1}^{3} \frac{\partial \sigma_{ik}}{\partial x_k} + F_i = 0 \tag{2}$$

Связь между компонентами вектора напряжений и деформаций задается выражениями:

$$\sigma_{ik} = K\varepsilon_{ll}\delta_{ik} + 2\mu\left(\varepsilon_{ik} - \frac{1}{3}\delta_{ik}\varepsilon_{ll}\right);\ \ \varepsilon_{ik} = \frac{1}{9K}\delta_{ik}\sigma_{ll} + \frac{1}{2\mu}\left(\sigma_{ik} - \frac{1}{3}\delta_{ik}\sigma_{ll}\right)$$

здесь K – модуль всестороннего сжатия, μ – модуль сдвига.

Условие равновесия тела, в котором деформации вызваны силами приложенными к его поверхности в векторной форме имеет вид:

$$\mathrm{grad}\cdot \mathrm{div}\,\mathbf{u} + (1-2\nu)\Delta\mathbf{u} = 0\,, \tag{3}$$

здесь ν – коэффициент Пуассона, а внешние силы входят в условие в качестве граничных условий.

Для деформаций, изменяющихся со временем уравнение (2) примет вид [4]:

$$\sum_{k=1}^{3} \frac{\partial \sigma_{ik}}{\partial x_k} + F_i = \rho \frac{\partial^2 u_i}{\partial t^2}$$

При деформировании тела, сопровождающемся изменением температуры (как вследствие внешнего нагрева, так и в результате самой деформации) тензор внутренних напряжений задается выражением [3]:

$$\sigma_{ik} = -K\alpha(T - T_0)\delta_{ik} + Ku_{ll}\delta_{ik} + 2\mu\left(\varepsilon_{ik} - \frac{1}{3}\delta_{ik}\varepsilon_{ll}\right).$$

где α – коэффициент теплового расширения. Если тело нагрето не равномерно, то появляются объемные силы и условие равновесия (3) приобретает вид [83]:

$$\frac{3(1-\nu)}{1+\nu}\operatorname{grad}\operatorname{div}\mathbf{u} - \frac{3(1-2\nu)}{2(1+\nu)}\mathbf{rot}\,\mathbf{rot}\,\mathbf{u} = \alpha\nabla T$$

Распределение температуры в объеме образца определяется уравнением теплопроводности:

$$\frac{\partial T}{\partial t} = \operatorname{div}(a\nabla T) + f$$

где f – внешний источник тепла. В процессе линейной сварки трением тепло выделяется за счет трения. Закон трения выберем в форме Амотона-Кулона:

$$F_{fr} = \lambda(T)R_n$$

здесь λ(T) – коэффициент трения, зависящий от температуры, R_n – сила реакции опоры.

Для материалов в справочниках обычно указываются такие параметры как модуль юнга E и коэффициент Пуассона ν. Запишем выражения, определяющие связь между этими параметрами и модулями сдвига и всестороннего сжатия [3]:

$$E = \frac{9K\mu}{3K+\mu},\ \nu = \frac{1}{2}\frac{3K-2\mu}{3K+\mu}$$

В ходе работы было исследовано три задачи: нагрев образцов за счет трения (термо-структурная), нагрев одного образца движущимся источником тепла (тепловая) и образец внутри зажима (структурная).

1.1 Геометрия образцов

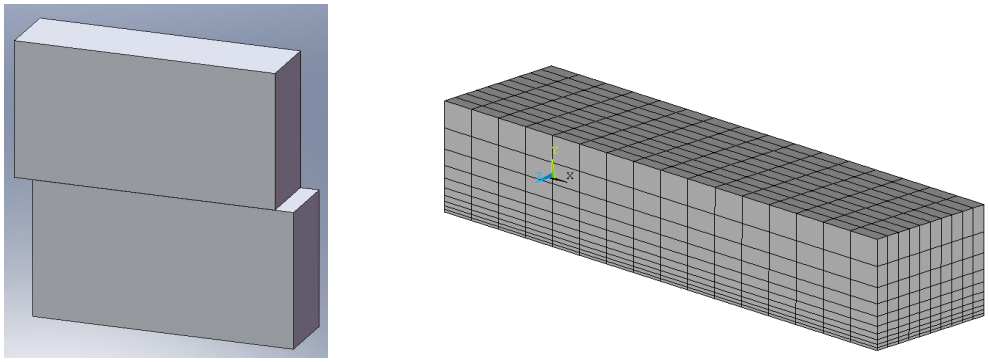


Рис. 1. Геометрия задачи (слева) и расчетная сетка (справа)

Исследуются термо-упругие деформации образцов размером 26мм×35мм×13мм из сплава сплава Ti6Al4V (Рис. 1). Для повышения точности расчетов без увеличения числа конечных элементов была использована неравномерная сетка вдоль оси Oy (т.к. область нагрева располагается в тонком слое вблизи контакта, то рассматривается слой меньшей толщины в 5 мм), в двух других направлениях сетка равномерная. Рассматривается адиабатический процесс (замкнутая система).

1.2 Термо-структурная задача

Геометрическая модель состоит из двух одинаковых брусков, стоящих один на другом.

Верхний образец скользит по поверхности нижнего образца вдоль длинной стороны, нижний образец неподвижен. Для задания параметров скольжения между брусками создается

контакт. Тип контакта – закрытый и скользящий, без взаимного проникновения. Коэффициент трения равен 0.3.

Как в качестве объемного элемента используется 3D 20-ти узловой тип элемента SOLID 226 [5].

Нижний брусок фиксируется своей нижней гранью. На верхний брусок фиксации не наложено, но все его точки движутся по периодическому закону $x = a \cdot \sin(\omega \cdot t)$ с амплитудой $a = 2мм,$ частотой $f = 50Гц.$

На верхнюю грань верхнего образца задано постоянное давление 100 МПа.

Рассматривается изотропный упругий материал. В модели материала используются параметры сплава Ti6Al4V [6].

1.3 Тепловая задача

Рассматривалась модель нагрева с тепловым потоком, имитирующим тепловыделение при трении. Тепловой поток задан на каждую из плоскостей контакта по формуле:

$$q = \frac{\mu F A \omega}{2W} \frac{|\cos(\omega t)|}{L - A|\sin(\omega t)|} \Phi(x,t),$$

где $\Phi(x,t) = \frac{1}{2}[\mathrm{sgn}(x + A\sin(\omega t)) + \mathrm{sgn}(L - A\sin(\omega t) - x)].$

Таким образом, тепловой поток, заданный на контактных гранях, пульсирует во времени и колеблется вдоль оси Ox.

В данной задаче используется переменный источник тепла, зависящий от времени по периодическому закону. В качестве конечных элементов выбраны тепловые 20-ти точечные элементы SOLID 90. Использовались те же параметры материала, что и в предыдущей задаче.

1.4 Структурная задача

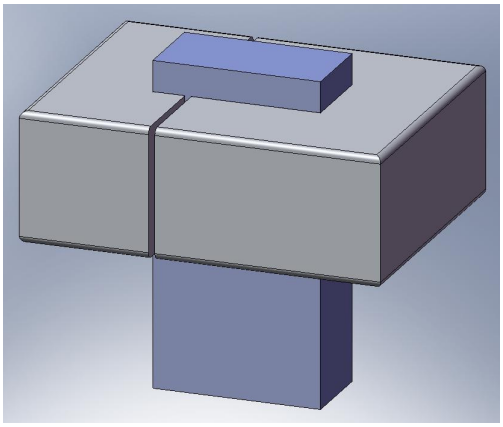


Рис. 2. Геометрия структурной задачи

Структурная задача рассматривалась на примере статического зажима образца (Рис. 2). Здесь рассматривалась наиболее простая конструкция зажима: две прямоугольные скобы сжимают образец вдоль более длинной стороны. Для этого на две противоположные грани зажима действует давление 100 МПа. Помимо этого, такое же давление задано на верхнюю грань верхнего образца, нижняя грань нижнего образца закреплена по вертикали.

Как для зажима, так и для образца задавались те же параметры материала, как и ранее. Для расчетов использовались структурные 20-ти точечные элементы SOLID 186.

2. Результаты моделирования

Нами были получены поля температур и напряжений (Рис. 3). При рассмотрении распределения температуры вдоль контрольных линий, обозначенных на рис. 3, можно

отметить, что перепад температуры в области контакта достигает величины порядка 500 K (Рис. 4). Вследствие такой неравномерности нагрева контакта течение материала начнется в небольших локальных областях.

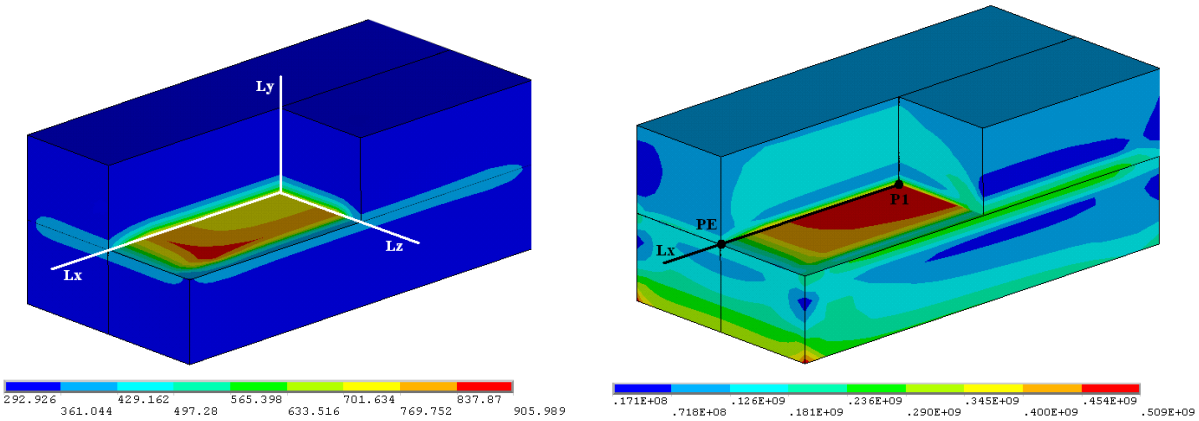


Рис. 3. Распределение поля температур (слева) и поля напряжений фон Мизеса (справа) в конечный момент времени 0.1 с

При рассмотрении напряжений следует, прежде всего, отметить, что максимальное напряжение наблюдается в плоскости контакта. В достаточно большой области контакта напряжения остаются постоянными, но вдоль границ их величина резко уменьшается вследствие падения температуры (рис. 6).

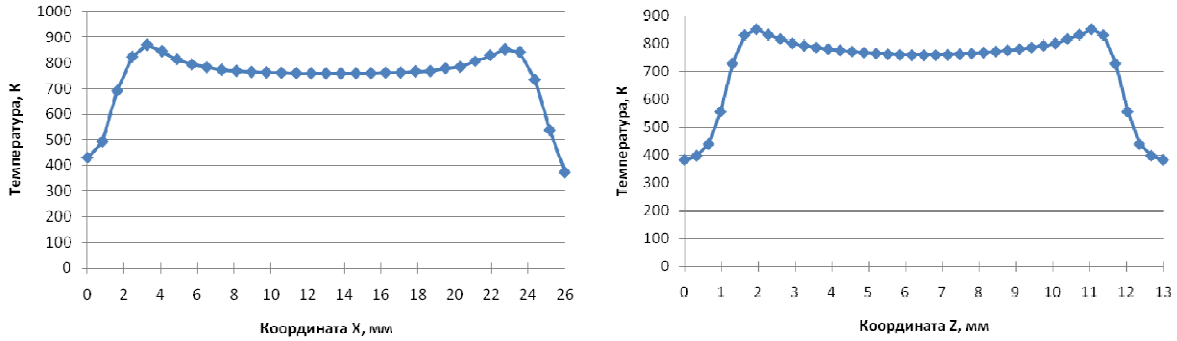


Рис. 4. Распределение температуры в плоскости контакта вдоль направления движения (слева) и перпендикулярно направлению движения (справа) в конечный момент времени 0.1 с

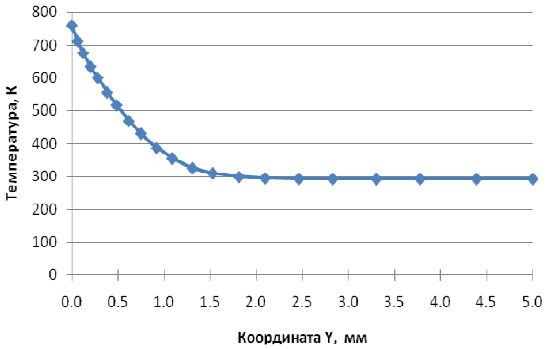


Рис. 5. Распределение температуры по вертикали в центре контакта в конечный момент времени 0.1 с

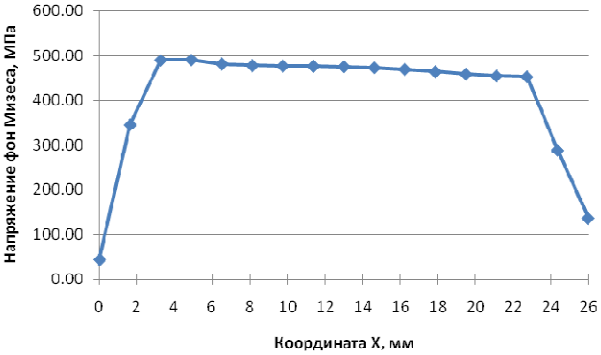


Рис. 6. Распределение напряжения фон Мизеса в плоскости контакта вдоль направления движения в конечный момент времени 0.1 с

Температурная задача дала в целом результаты, аналогичные предыдущей задаче. Так как в формуле заданного теплового потока не учитывается зависимость от координаты Z, то в тепловой задаче распределение тепла вдоль соответствующей оси не отличается от

полученного в предыдущей задаче, вдоль остальных направлений результаты очень близки к полученным в задаче с трением.

В результате структурной задачи получили поле напряжений фон Мизеса, показанное на рисунке 7. Видно, что напряжения распределяются вдоль более длинной грани, так как более существенным оказывается давление со стороны боковых частей скоб, направленное вдоль более короткой боковой стороны образца. Напряжения концентрируются на углах образцов, где давление действует вдоль двух взаимноперпендикулярных направлений.

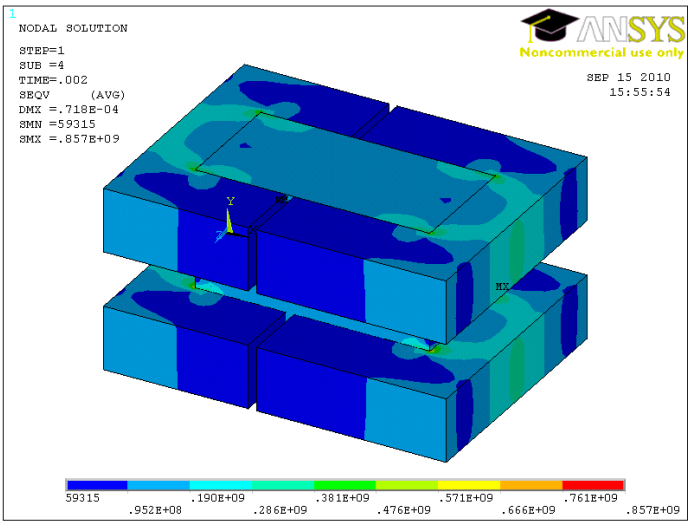


Рис. 7. Распределение поля напряжений фон Мизеса

3. Оценка эффективности решателей

Рассматривались следующие решатели: Sparse, JCG, AMG, ICCG, PCG.

3.1 Решатели ANSYS Mechanical APDL

При решении термоупругих задач необходимо учитывать взаимное влияние температурного поля и напряженно-деформированного состояния материала. При этом модифицируется уравнение баланса энергии (или энтропии). Кроме того, в связь напряжений и деформаций добавляется тепловое расширение. В результате после дискретизации появляется связь пространственных степеней свободы с температурными.

Связь степеней свободы различных типов может быть учтена различными способами. Сильная связь (strong coupling в терминологии ANSYS, также называемая иногда матричной или полной связью) означает учет взаимодействия через общесистемную матрицу, когда в каждом уравнении присутствуют как температурные, так и структурные переменные.

При слабой связи (weak coupling, связь через нагрузки) учет взаимосвязей производится путем добавления слагаемых в правую часть системы. При этом каждый элемент правой части зависит от переменных обоих типов, но их значения берутся с предыдущей итерации, что может приводить к проблемам со сходимостью для сложных термоупругих задач. Матрица для задачи со слабой связью является симметричной, что облегчает решение системы и приводит к значительному снижению времени расчета.

В результате дискретизации уравнений на основе метода конечных элементов получается нелинейная система уравнений вида [7]

$$[K]\{u\} = \{F^a\},$$

где $\{u\}$ – вектор степеней свободы (неизвестных), $[K]$ – матрица коэффициентов, в общем случае зависящих от $\{u\}$, $\{F^a\}$ – вектор нагрузок.

Для поиска решения в пакете применяется метод Ньютона-Рафсона решения систем нелинейных уравнений. Базовый метод представляет собой итеративную процедуру:

$$[K_i^T]\{\Delta u_i\} = \{F^a\} - \{F_i^{nr}\},$$
$$\{u_{i+1}\} = \{u_i\} + \{\Delta u_i\},$$

где индекс i определяет номер итерации установления равновесия (equilibrium iteration в терминологии ANSYS), $[K_i^T]$ – матрица Якоби (касательная матрица), составленная из производных от элементов $[K]$, $\{F_i^{nr}\}$ – вектор возвращающих нагрузок (внутренние нагрузки на элементы). Коэффициенты $[K_i^T]$ и $\{F_i^{nr}\}$ вычисляются при значениях степеней свободы, взятых с текущей итерации $\{u_i\}$.

Сходимость процесса определяется по норме вектора невязок $\{R\} = \{F^a\} - \{F_i^{nr}\}$ и вектора изменения степеней свободы $\{\Delta u_i\}$ (команда **CNVTOL**). Может быть также ограничено количество итераций равновесия (команда **NEQIT**).

На каждой итерации равновесия требуется один раз решить линейную систему уравнений. Так как количество неизвестных определяется количеством степеней свободы задачи, размерность системы достаточно велика. Успешно решать такие системы помогает тот факт, что возникающие матрицы являются сильно разреженными (структура матрицы определяется связями узлов в конечных элементах). Для решения таких систем применяются хорошо разработанные методы вычислительной линейной алгебры. На них основаны различные решатели, присутствующие в любом инженерном вычислительном пакете.

Для решения полученной системы в пакете применяется несколько методов. Рассмотрим их последовательно.

Прямой (точный) метод (sparse direct solver) основан на LU-разложении матрицы. Метод активируется командой **EQSLV, SPARSE** или автоматически. Имеется версия решателя для систем с распределенной памятью – **DSPARSE**. Максимальное ускорение расчета достигается, когда разделенная на узлы задача полностью помещается в оперативную память.

Помимо прямого метода в пакете доступен ряд итерационных методов. Они, как правило, требуют меньшего объема памяти и вычислительного времени, лучше распараллеливаются, но требуют более тщательной настройки. В частности, для плохо обусловленных матриц итерационный метод может сильно замедлиться, вообще не сойтись или даже получить неверное решение (зависит от задачи, конкретного метода и настроек решателя).

В основном это методы, основанные на методе сопряженных градиентов с различными предобуславливателями. Решатель **JCG** использует предобуславливатель Якоби (диагональная часть матрицы $[K]$). Активируется командой **EQSLV, JCG**.

Решатель **ICCG** строит предобуславливатель на основе неполного разложения Холецкого, может работать с несимметричными матрицами.

Решатель **PCG** работает только с вещественными симметрическими матрицами, но содержит очень эффективный алгоритм предобуславливания. При этом достигается максимальная скорость вычислений и возможность работы с плохо обусловленными матрицами. Используемый объём оперативной памяти примерно в 2 раза больше, чем для JCG.

Распределенные версии решателей **– DPCG** и **DJCG**, пригодны для запуска на системах с распределенной памятью. Математические формулировки при этом не изменяются, добавляется межузловой обмен данными и параллельное построение самих матриц.

Решатель **AMG** основан на алгебраическом многосеточном методе и является одним из самых эффективных для плохо обусловленных матриц. Решатель более требователен к оперативной памяти, чем PCG.

4.2 Сравнение эффективности решателей

4.2.1 Термо-структурная задача

Один узел кластера состоит из 2-х процессоров, содержащих 4 ядра. Тесты для SMP проводились на одном узле для разного числа ядер (от 1 до 8). Расчеты для MPI (версия HP-MPI) запускались на разном числе узлов.

Для термо-структурной задачи с сильной связью в силу несимметричности матрицы доступны только решатели JCG, PCG и SPARSE, при этом оба итерационных метода в этом случае не поддерживают распараллеливание с распределенной памятью и запускались только для SMP. В результате было получено, что итерационные методы практически не ускоряются. Прямой метод ускоряется, но не более чем в 2 раза.

Термо-структурная задача со слабой связью распараллеливается лучше. Однако использование более 5 ядер на узле при любом решателе не дает прирост производительности, скорее всего, из-за интенсивной работы решателя с памятью (количество неизвестных в каждом уравнении довольно велико) и из-за перестройки матрицы после каждой итерации, которая выполняется последовательно. Наиболее эффективным оказывается распараллеливание на 2 узла по 4-5 ядер, при этом большее ускорение показывает решатель JCG.

Решатель SPARSE показывает ускорение на уровне 2.5 при использовании 6 ядер и больше на двух узлах (Рис. 10).

На рисунках 8 – 10 показаны зависимости ускорения от количества узлов и суммарного числа ядер для наиболее эффективных решателей.

В Таблице 1 указаны времена расчетов термо-структурной задачи для некоторых решателей, а также используемая ими память. Видно, что для малого числа ядер метод PCG считает лучше, но с увеличением количества ядер наиболее быстрым решателем становится JCG. Отметим, что метод JCG использует значительно меньше оперативной памяти, по сравнению с другими решателями. Отметим, что при использовании более чем 8 ядер используемая память на ядро медленно уменьшается.

Таблица 1. Время расчета и размер используемой памяти для термо-структурной задачи для различных решателей (со слабой связью)

Метод	Время (1 ядро), ч.	Лучшее время, ч.	Размер (1x1), МБ	Размер на 1 ядро (1x8), МБ
JCG	8,35	**2,32** (2x4)	**83,6**	**26,6**
PCG	**5,16**	3,53 (3x5)	107	67,9
SPARSE	7,9	3,23 (2x5)	754	350

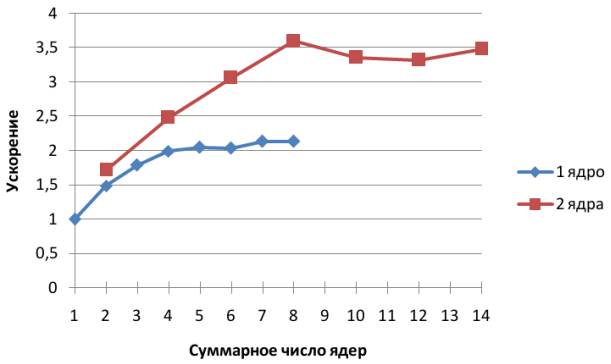


Рис. 8. Термо-структурная задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (8,35 ч) для решателя JCG (Weak) / MPI.

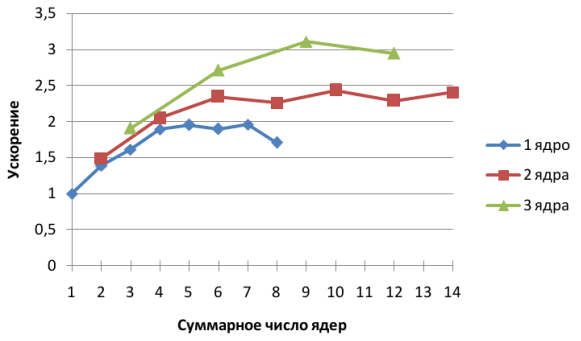


Рис. 9. Термо-структурная задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (5,16 ч) для решателя PCG (Weak) / MPI

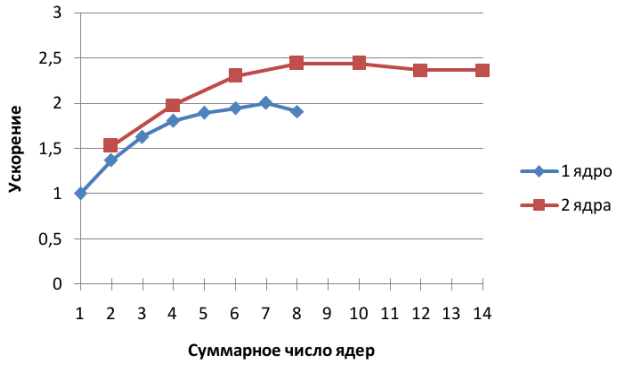


Рис. 10. Термо-структурная задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (7,9 ч) для решателя Sparse (Weak) / MPI

4.2.2 Тепловая задача

Тепловая задача считается на несколько порядков быстрее и имеет значительно более низкую вычислительную сложность по сравнению с термо-структурной задачей, что влияет на более повышение ускорения. Тепловая задача распараллеливается лучше термо-структурной. Ее распараллеливание ограничивается только её малым размером (см., например, Рис. 13, где справа изображен график ускорения для задачи с количеством элементов в 8 раз большим, чем для задачи слева).

В Таблице 2 показаны времена расчетов тепловой задачи для некоторых решателей, а также используемая ими память. Наиболее эффективным и по времени расчета и по используемой памяти оказался метод JCG. Метод PCG показал плохое распараллеливание и экономичное использование памяти. Метод SPARSE распараллеливает задачу лучше, но использует на порядок больше памяти, чем другие решатели.

Таблица 2. Время расчета и размер используемой памяти для тепловой задачи для различных решателей

Метод	Время (1 ядро), сек.	Лучшее время, сек.	Размер (1x1), МБ	Размер на 1 ядро (1x8), МБ
JCG	**207**	**77** (1x6)	**15,7**	**10,0**
PCG	367	185 (1x7)	19,5	12,9
SPARSE	251	102 (2x4)	251	168

Метод PCG с SMP показал низкое ускорение (до 1,4 раза с максимумом на 4 ядрах). С MPI метод JCG показывает ускорение до 2,7 для конфигурации 1x6 (Рис. 11), а PCG – до 2,0 для конфигурации 1x7 (Рис. 12). Метод Sparse показал максимальное ускорение 2,5 для конфигурации с двумя узлами, однако, для задачи с большим количеством элементов, как было отмечено выше, ускорение возрастает (Рис. 13).

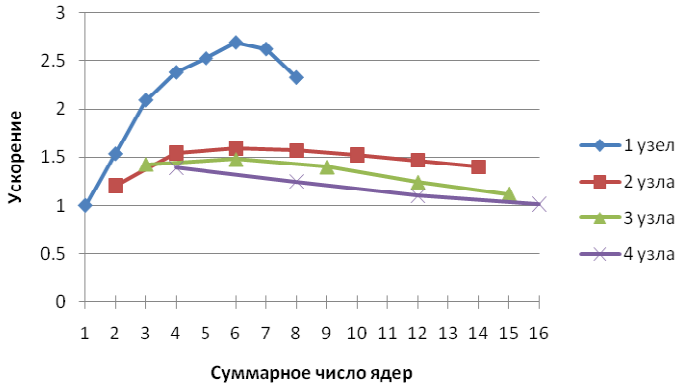


Рис. 11. Тепловая задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (269 с) для решателя JCG / MPI

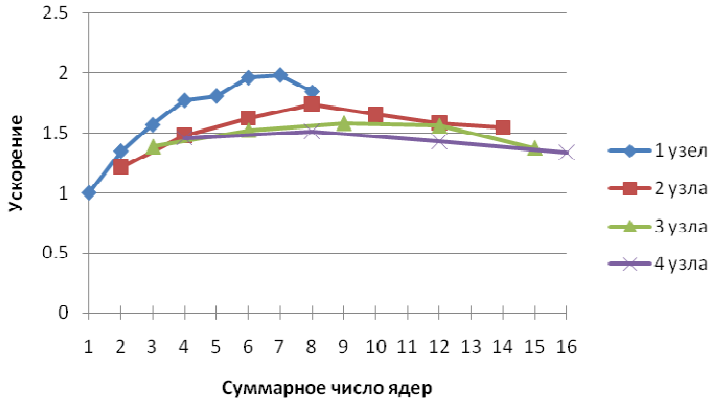


Рис. 12. Тепловая задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (253 с) для решателя PCG / MPI

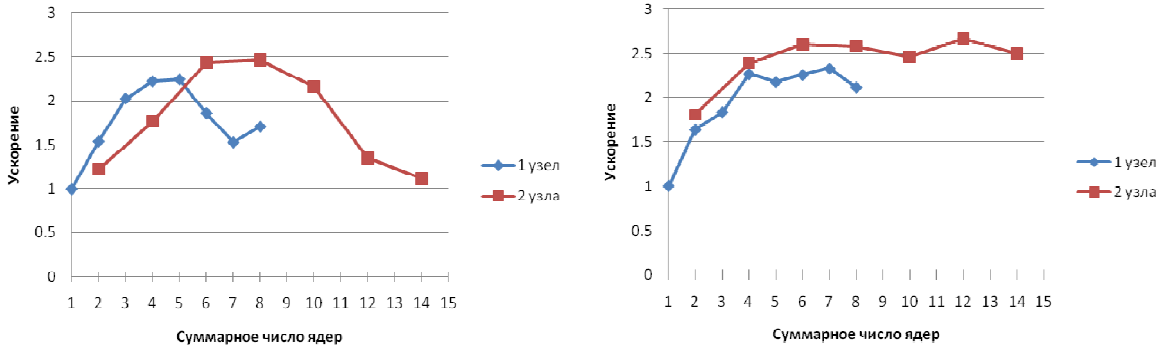


Рис. 13. Тепловая задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (251 с) для решателя Sparse / MPI. Справа задача в увеличенным в 8 раз числом элементов

4.2.3 Структурная задача

Структурную задачу возможно решить только с помощью прямого метода Sparse. Из-за большого количества контактных и структурных элементов (общее количество 20336 элементов) решатель требует большого количества памяти (3,3 ГБ для одного узла и 637 МБ на ядро для конфигурации 1x8). На Рис. 14 представлена зависимость ускорения от суммарного количества ядер. Падение ускорения на одном узле 6 и более ядер связано с тем, что задача не помещалась полностью в оперативную память и система подключала более медленную память подкачки. Для двух и более узлов количество доступной памяти выросло, что и сказалось на росте ускорения. Отметим хорошее ускорение (до 5,7 раз) при использовании 3 и 4 узлов. При времени выполнения задачи 43 мин. на одном узле, наилучшее время составило 7,6 мин.

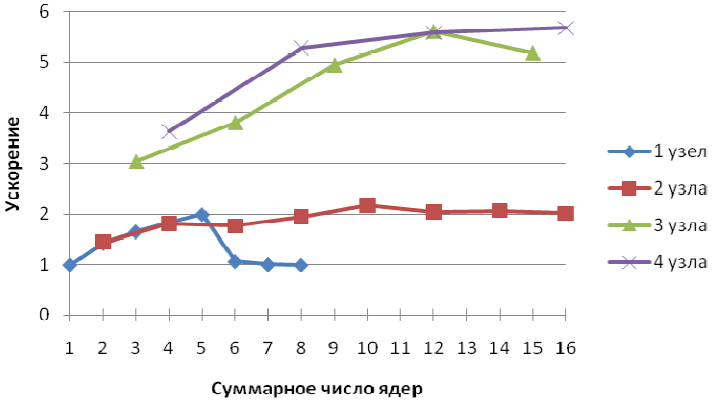


Рис. 14. Структурная задача: зависимость ускорения от суммарного числа ядер относительно расчета на одном ядре (0,72 ч) для решателя Sparse / MPI

4.2.4 Расчет на GPU

В новой версии ANSYS Multiphysics 13.0 добавлена возможность использования для вычислений графических процессоров Nvidia. Пока поддерживается только использования одного ускорителя совместно с SMP-версиями решателей SPARSE, PCG и JCG.

В качестве тестовой конфигурации была взята двухпроцессорная рабочая станция Fujitsu Siemens Computers V840 (процессоры Opteron 2214 Dual-Core, 2.2 GHz, объем оперативной памяти 8 GB, PC2-5300 FB-DIMM ECC) с графическим ускорителем NVIDIA Tesla C1060.

Решалась аналогичная предыдущим термо-структурная задача (трение брусков с разогревом) с 9600 20-узловыми элементами (10 временных шагов, что достаточно для быстрой оценки).

На рисунке 15 представлена зависимость ускорения от количества ядер для трёх различных решателей. Больше всего выигрывает от использования GPU точный решатель Sparse. Наибольшее ускорение демонстрирует решатель PCG, но использование GPU ускоряет его лишь на 20-36% (для 4 и 1 ядра соответственно). Скорее всего, это связано с быстрым выполнением ускоряющейся части кода по сравнению с остающимися последовательными фрагментами, что подтверждается наблюдением за загрузкой CPU и GPU (минуты между итерациями равновесия работает только одно ядро без GPU).

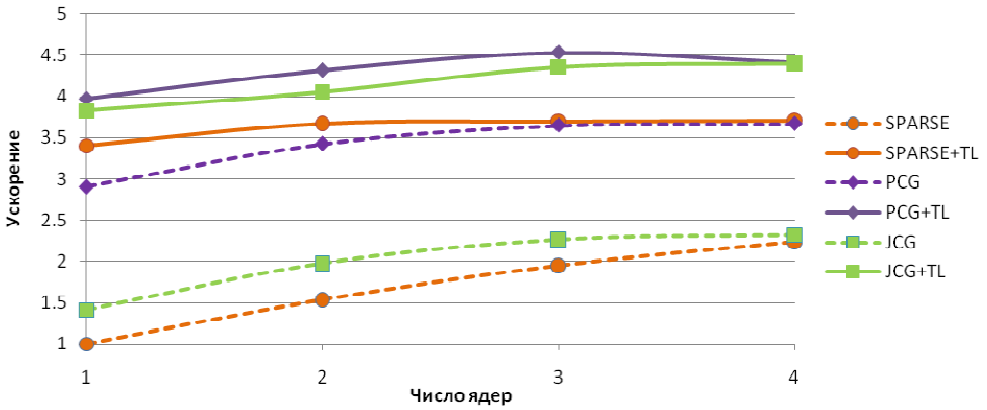


Рис. 15. Зависимость ускорения от числа ядер и относительно расчета на одном ядре (3,6 ч) для решателя SPARSE

Заключение

Была исследована эффективность различных решателей Ansys Multiphysics для трех типовых задач, связанных с моделированием процесса линейной сварки трением. Было показано, что наиболее эффективными для тепловой и термо-структурной задач при слабой связи степеней свободы оказываются решатели JCG и PCG, а для структурной задачи наибольшую эффективность показал решатель Sparse. Расчеты на GPU показал прирост ускорения по сравнению с CPU.

Благодарности

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 от 9.04.2010 г. «О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства».

Литература

1. Вилль В.И. Сварка металлов трением. – М.: Машиностроение, 1970.
2. A.Vairis, M.Frost. On the extrusion stage of linear friction welding of Ti6Al4V. Material Science and Engineering. A 271, (1999), pp 477—484.
3. Ландау Л. Д., Лифшиц Е. М. Теоретическая физика в 10 т., Т.VII Теория упругости — М.: Наука, 1987. – 248 с.
4. Jeongho Ahn, David E. Stewart Dynamic frictionless contact in linear viscoelasticity // IMA Journal of Numerical Analysis (2009) 29, pp.43–71.
5. Basic Analysis Guide // Release 12.1, November 2009, ANSYS Inc.
6. ГОСТ Титановый сплав ВТ6
7. Theory Reference for the Mechanical APDL and Mechanical Applications // Release 12.1, November 2009, ANSYS Inc.

Параллельный предобуславливатель SSOR для решения задач электромагнетизма в частотной области *

Д.С. Бутюгин[1]

Институт Вычислительной Математики и Математической Геофизики СО РАН[1]

В работе рассматриваются подходы к распараллеливанию предобуславливателя симметричной последовательной верхней релаксации (SSOR) в модификации Айзенштата, используемый при итерационном решении СЛАУ, возникающих в результате аппроксимации соответствующих вариационных задач электромагнетизма. Распараллеливание предобуславливателя SSOR основано на декомпозиции расчетной области, в том числе и алгебраической, с совместным переупорядочиванием матрицы. Отдельное внимание уделено обеспечению высокой производительности на NUMA-архитектурах. Результаты проведенной серии численных экспериментов демонстрируют производительность и масштабируемость представленных алгоритмов.

1. Введение

Задача моделирования трехмерных электромагнитных полей в частотной области возникает при расчетах различных волновых устройств, таких как волноводы, антенны, микроволновые устройства и других. Данная задача является вычислительно сложной, поскольку требует решения систем линейных алгебраических уравнений высоких порядков (10^6–10^7). Особый интерес вызывает возможность эффективного использования итерационных алгоритмов для решения данных систем. Однако это требует использования эффективных предобуславливателей, что, в ряде случаев, вызывает проблемы задействования вычислительных мощностей на многоядерных системах, поскольку многие предобуславливатели имеют проблемы с масштабируемостью.

В данной работе рассматриваются подходы к распараллеливанию и оптимизации предобуславливателя симметричной последовательной верхней релаксации (SSOR) в модификации Айзенштата [1], используемого совместно с итерационными методами сопряженных градиентов (CG), сопряженных невязок (CR) и другими при решении ряда модельных задач электромагнетизма. Распараллеливание предобуславливателя SSOR основано на декомпозиции расчетной области с совместным переупорядочиванием матрицы. Также рассматривается алгебраическая декомпозиция области при использовании библиотеки METIS [2]. Отдельное внимание при реализации матрично-векторных операций уделено обеспечению высокой производительности на NUMA-архитектурах (Non-Uniformed Memory Architecture).

Структура данной работы следующая. В разделе 2 описывается постановка задачи и используемые итерационные алгоритмы. Раздел 3 содержит описание подхода к распараллеливанию предобуславливателя SSOR, а разделе 4 — некоторые полезные оптимизации предлагаемого алгоритма. В разделе 5 приведены результаты численных экспериментов. Наконец, в последнем разделе обсуждаются полученные результаты а также рассматриваются возможные пути дальнейших исследований.

2. Постановка задачи

Решение разреженных систем линейных алгебраических уравнений вида

$$Ax = b, \tag{1}$$

*Работа выполнена при поддержке РФФИ (грант 08-01-00526)

итерационными методами в подпространствах Крылова требует проведения умножений матрицы A на различные векторы. Ниже приведены схемы вычислений в алгоритмах сопряженных градиентов (случай $q = 0$) и сопряженных невязок ($q = 1$):

$$\begin{aligned} p_0 = r_0 &= f - Au_0, \\ \alpha_j &= \langle A^q r_j, r_j\rangle / \langle A^q p_j, Ap_j\rangle, \\ u_{j+1} &= u_j + \alpha_j p_j, \\ r_{j+1} &= r_j - \alpha_j A p_j, \\ \beta_j &= \langle A^q r_{j+1}, r_{j+1}\rangle / \langle A^q r_j, r_j\rangle, \\ p_{j+1} &= r_{j+1} + \beta_j p_j, \end{aligned}$$

где $\langle u, v\rangle$ — скалярное произведение векторов u и v.

Для увеличения производительности итерационных решателей и ускорения их сходимости, как правило, применяют одно из следующих предобуславливаний системы:

$$\begin{aligned} B^{-1}Ax &= B^{-1}b, \\ B_L^{-1}AB_U^{-1}B_U x &= B_L^{-1}b, \\ AB^{-1}Bx &= b, \end{aligned} \tag{2}$$

где $B_L B_U = B$, а B — предобуславливающая матрица, "близкая" в некотором роде исходной матрице A. В рамках данной работы рассматривается предобуславливатель симметричной последовательной верхней релаксации:

$$B_L = (L + G)G^{-\frac{1}{2}}, \quad B_U = G^{-\frac{1}{2}}(L^T + G), \quad G = \frac{1}{\omega}D, \tag{3}$$

где $A = L + D + L^T$ — симметричная матрица системы (1), а $0 < \omega < 2$ — параметр релаксации. Известна модификация данного предобуславливателя, позволяющая свести умножение на исходную матрицу и решение двух приведенных треугольных систем уравнений к решению двух вспомогательных треугольных систем (подробнее см. [1]): умножение на матрицу $\bar{A} = B_L^{-1}AB_U^{-1}$ вычисляется как

$$\begin{aligned} y &= (I + \bar{L}^T)^{-1}x, \\ \bar{A}x &= (I + \bar{L})^{-1}\left(x - (2I - \bar{D})y\right) + y, \end{aligned} \tag{4}$$

где $\bar{L} = G^{-1/2}LG^{-1/2}$, $\bar{D} = G^{-1/2}DG^{-1/2} = \omega I$, I — единичная матрица.

В то время как распараллеливание векторно-векторных операций в данных итерационных решателях представляет из себя довольно простую задачу, распараллеливание предобуславливателя SSOR, как обычного так и в модификации Айзенштата, сопряжено с определенными трудностями. Сложность эта связана с необходимостью решения треугольных систем, имеющих портрет исходной матрицы. Матрицы, получающиеся в результате конечно-элементных и конечно-разностных аппроксимаций дифференциальных уравнений имеют обычно, с одной стороны, небольшое число элементов на строку и, с другой стороны, сложные зависимости между переменными. Все это не позволяет реализовать эффективное распараллеливание в общем случае. Можно отметить, что большинство пакетов итерационных решателей не поддерживают параллельное предобуславливание с помощью SSOR (например, PETSc и др.), а те пакеты, которые имеют распараллеленное решение треугольных систем (такие, как Intel MKL), как правило показывают невысокую масштабируемость на реальных задачах. В рамках данной работы предлагаются подходы и ряд оптимизаций, позволяющие получить масштабируемый SSOR на задачах электромагнетизма.

3. Параллелизация SSOR

Основная проблема при разработке масштабируемого предобуславливателя SSOR связана с тем, что при решении треугольных систем уравнений имеются нетривиальные зависимости по данным. Например, нижне-треугольной системе с некоторой матрицей L можно сопоставить граф, вершинами которого являются столбцы матрицы, а ребрами — ненулевые элементы матрицы так, что граф $G = (V, E)$, где множество вершин $V = \{1, 2, \ldots, n\}$, n — порядок матрицы A, а множество ребер $E = \{(u, v) : u \in V, v \in V, [L]_{u,v} \neq 0\}$. Тогда граф G является ориентированным и показывает зависимости между элементами вектора неизвестных x при решении системы $Lx = y$. Транспонированный граф G^T при этом указывает зависимости между элементами вектора неизвестных x при решении верхне-треугольной системы $L^T x = y$.

Для того, чтобы решить проблему с произвольной структурой графа G и невозможностью в полной мере использовать возможности мультипроцессорных систем при решении треугольных систем, можно воспользоваться идеей алгоритма вложенных сечений графа G [2]. Алгоритм вложенных сечений основан на следующей процедуре: среди вершин V графа G ищется разделяющее множество $S \subset V$, предпочтительно небольшого по сравнению с V размера, такое что оставшиеся вершины графа делятся на два множества V_1 и V_2, причем $V_1 \bigcap V_2 = \emptyset$ и $\forall u \in V_1, v \in V_2 : (u, v) \notin E, (v, u) \notin E$. Далее процедура повторяется для каждой из двух полученных частей графа до тех пор, пока либо не будет достигнут малый размер получившихся частей, либо пока не будет получено достаточное количество частей. Далее осуществляется перенумерация вершин графа по следующему принципу. Сначала рекурсивно нумеруются вершины в V_1, затем — в V_2, и в последнюю очередь нумеруются вершины S.

Алгоритм вложенных сечений графа в данной работе предлагается использовать следующим образом. Рассмотрим граф G для нижне-треугольной части L матрицы A. Построим многоуровневую декомпозицию матрицы G. Пусть $\sigma(i)$ — номер вершины i в новой нумерации. Рассмотрим матрицу перестановок P:

$$P = \begin{bmatrix} e_{\sigma(1)} \\ e_{\sigma(2)} \\ \vdots \\ e_{\sigma(n)} \end{bmatrix}, \tag{5}$$

где e_j — вектор длины n, все компоненты которого равны нулю за исключением компоненты j, равной единице. Тогда матрица $P^T AP$ будет иметь следующую структуру:

$$P^T AP = \begin{bmatrix} D_1 & 0 & B_1^T \\ 0 & D_2 & B_2^T \\ B_1 & B_2 & D_3 \end{bmatrix}. \tag{6}$$

Здесь блоки D_1 и D_2 соответствуют множествам вершин V_1 и V_2, блок D_3 — разделяющему множеству S, а подматрицы B_1 и B_2 — ребрам между V_1 и S и V_2 и S соответственно.

Теперь вместо решения системы (1) можно решать систему

$$\hat{A}\hat{x} = \hat{b}, \quad \hat{A} = P^T AP, \quad \hat{x} = P^T x, \quad \hat{b} = P^T b. \tag{7}$$

Такой алгоритм, как правило, используется для получения системы, у которой число ненулевых элементов в LU-разложении меньше, чем в исходной системе. Для этого необходимо

на нижнем уровне алгоритма вложенных сечений использовать некоторую процедуру, которая минимизирует заполнение LU-факторов, например алгоритм минимальных степеней (AMD) [2]. Однако для нас интерес представляет структура матрицы $\hat{A}$ и, в частности, вид матрицы системы $(I + \bar{L})u = f$, требующей решения в предобуславливателе SSOR с модификацией Айзенштата. Ее структура представлена ниже:

$$\begin{bmatrix} I + \bar{L}_{D_1} & 0 & 0 \\ 0 & I + \bar{L}_{D_2} & 0 \\ \bar{B}_1 & \bar{B}_2 & I + \bar{L}_{D_3} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \tag{8}$$

Отсюда видно, что вычисление коэффициентов u_1 и u_2 можно производить независимо, а коэффициенты u_3 определять уже после вычисления u_1 и u_2. Аналогично в случае системы с матрицей $(I + \bar{L}^T)$ сначала определяются коэффициенты u_3, а затем u_1 и u_2 могут быть определены независимо друг от друга.

Один из вариантов алгоритма вложенных сечений реализован в библиотеке METIS [2]. Однако следует отметить, что алгоритм вложенных сечений требует довольно существенных вычислительных ресурсов. Применение его может быть оправдано, например, в случае многоуровневых итерационных процессов, когда внутренние итерации обращают один и тот же предобуславливатель. В таком случае алгоритм вложенных сечений необходимо запустить только один раз в начале итераций, после чего полученное разбиение можно использовать многократно при решении предобуславливающей системы с различными правыми частями. Примером такого многоуровневого процесса может служить предобусловленное решение системы с седловой точкой, предложенное в [3]. В данной работе решается система вида

$$\begin{bmatrix} A - k_0^2 M & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{9}$$

с предобуславливателем

$$\mathcal{P}_{M,L} = \begin{bmatrix} A + (1 - k_0^2)M & 0 \\ 0 & L \end{bmatrix}. \tag{10}$$

При этом описанную технику можно применить для решения систем с матрицами L и $\mathcal{P}_M = A + (1 - k_0^2)M$.

В случае, если подход с алгоритмом вложенных сечений не подходит по тем или иным причинам, можно предложить альтернативный подход для конечно-элементных схем. Базисные функции конечно-элементных подпространств пространства H^{rot} соответствуют геометрическим объектам конечно-элементной сетки — узлам, ребрам, граням и объемам [4] (тетраэдрам, шестигранникам, призмам и т.д. в зависимости от типа сетки). В таком случае разделяющее множество имеет очевидную геометрическую интерпретацию — оно соответствует элементам сетки, лежащим на полигональной поверхности, делящей расчетную область на два не пересекающихся объема. Тогда возможно реализовать альтернативные, чисто геометрические, алгоритмы декомпозиции матрицы на блоки. Например, можно на каждом этапе в текущей расчетной области проводить плоскость по границам элементов сетки, делящую ее на две примерно равные подобласти, причем так, чтобы сечение области плоскостью имело, по возможности, меньшую площадь. После этого данная процедура рекурсивно применяется к двум получившимся подобластям до тех пор, пока не будет получено достаточное число подобластей. Далее все конечные элементы нумеруются в соответствии с уже изложенным выше принципом: на каждом уровне рекурсии сначала нумеруются

элементы в получившихся подобластях, а затем уже — элементы, лежащие на разделяющей плоскости. Данный подход отличается относительной простотой в случае расчетных областей с несложной конфигурацией, когда легко проводить разделяющие плоскости. Кроме того, дополнительный плюс состоит в том, что основные временные затраты, связанные с разделением области на подобласти и разделяющие множества, не зависят от порядка используемых базисных функций, а зависят только от количества элементов сетки.

Таким образом, для параллелизации SSOR выполняются следующие действия. Для графа исходной матрицы запускается алгоритм вложенных сечений либо алгоритм геометрической декомпозиции области. Получаемые части рекурсивно делятся до тех пор, пока не будет получено необходимое число частей нижнего уровня (например, большее либо равное числу потоков). Затем выполняется перенумерация вершин графа, и с помощью перестановки $\sigma(i)$ вычисляется новая матрица $\hat{A} = P^T A P$. В блочном виде эта матрица будет иметь вид

$$\hat{A} = \begin{bmatrix} D_1 & 0 & B_{3,1}^T & & & & B_{7,1}^T & & B_{k,1}^T \\ 0 & D_2 & B_{3,2}^T & & 0 & & B_{7,2}^T & \cdots & B_{k,2}^T \\ B_{3,1} & B_{3,2} & D_3 & & & & B_{7,3}^T & & B_{k,3}^T \\ & & & D_4 & 0 & B_{6,4}^T & B_{7,4}^T & & B_{k,4}^T \\ & 0 & & 0 & D_5 & B_{6,5}^T & B_{7,5}^T & \cdots & B_{k,5}^T \\ & & & B_{6,4} & B_{6,5} & D_6 & B_{7,6}^T & & B_{k,6}^T \\ B_{7,1} & B_{7,2} & B_{7,3} & B_{7,4} & B_{7,5} & B_{7,6} & D_7 & \cdots & B_{k,7}^T \\ & \vdots & & & \vdots & & \vdots & & \vdots \\ B_{k,1} & B_{k,2} & B_{k,3} & B_{k,4} & B_{k,5} & B_{k,6} & B_{k,7} & \cdots & D_k \end{bmatrix}. \qquad (11)$$

При этом некоторые из блоков $B_{i,j}$ могут оказаться нулевыми. Далее предлагается сконденсировать граф исходной матрицы так, чтобы вершинами сконденсированного графа стали разделители и полученные на самом нижнем уровне алгоритма части графа. Пусть сконденсированный граф обозначен как $G' = \{V', E'\}$, причем количество вершин в нем будет равно k. Ориентировав ребра этого графа естественным образом (задав направление ребра от вершины с большим номером к вершине с меньшим номером для нижне-треугольных систем и от вершины с меньшим номером к вершине с большим номером для верхне-треугольных), получим граф зависимостей блоков. Каждый из блоков D_i и соответствующие ему неизвестные вектора u назначается некоторому потоку, после чего граф G' используется для отслеживания зависимостей между блоками.

Описанный подход к параллелизации SSOR, по сравнению с обычной параллелизацией решения треугольных систем для исходной матрицы, имеет то преимущество, что каждый поток на нижнем уровне обрабатывает большой блок как одно целое, а мощность разделителей предполагается небольшой по сравнению с общим числом вершин. В этом случае существенно снижаются затраты на синхронизацию потоков, что повышает производительность и масштабируемость.

4. Предлагаемые оптимизации

В первую очередь, необходимо отметить следующую особенность задачи распараллеливания и оптимизации предобуславливателя SSOR в модификации Айзенштата при использовании итерационных решателей: такое итерационное решение оказывается bandwidth-

limited, то есть существенно ограничено пропускной способностью шины данных системы. Действительно, легко видеть, что за одну итерацию алгоритмов CG и CR требуется один раз прочитать всю матрицу системы (для решения двух треугольных систем), и несколько раз прочитать и записать различные векторы. Таким образом, возможные оптимизации алгоритма можно разделить на две категории: оптимизации доступа к памяти и оптимизации, направленные на повышение производительности алгоритма на NUMA-системах, приобретающих все большую популярность в последнее время. Данные системы отличаются тем, что имеют несколько блоков памяти и ее контроллеров, что позволяет разным процессорным сокетам одновременно читать различные участки памяти, повышая эффективную пропускную способность шины данных [5]. Однако перед обсуждением оптимизаций необходимо определить форматы данных, в которых будет храниться матрица либо ее блоки.

4.1. Форматы хранения данных

В простейшем случае, матрицу A системы удобно хранить в формате CSR (Compressed Sparse Row, сжатый строчный формат). При этом требуется хранить следующую информацию:

N — порядок системы;

$vals(N_Z)$ — хранит ненулевые элементы матрицы;

$cols(N_Z)$ — для каждого ненулевого элемента матрицы содержит номер столбца, в котором находится соответствующий элемент;

$rowInd(N+1)$ — i-й элемент массива содержит индекс первого элемента i-й строки в массиве $vals$, N-й элемент (полагается нумерация строк и столбцов с 0) содержит число ненулевых элементов в матрице плюс один;

N_Z — число ненулевых элементов матрицы, может быть вычислено как $N_Z = rowInd[N] - rowInd[0]$.

В более общем случае, возможна следующая модификация: вместо массива $rowInd(N+1)$ хранятся два массива $rowBegin(N)$ и $rowEnd(N)$ — соответственно, индексы начала и конца заданной строки в массивах $vals$ и $cols$. Такой формат более удобен тем, что позволяет легко вырезать подматрицы из матрицы (например, нижне- и верхне-треугольные части) путем вычисления значений массивов $rowBegin$ и $rowEnd$. Помимо этого, он позволяет менять местоположение строк в массивах $vals$ и $cols$, что может пригодиться в дальнейшем. Простейший пример кода, который выполняет умножение матрицы на вектор в данном формате, представлен на рисунке 1.

```
void multiply(int N, const int* rowBegin, const int* rowEnd,
    const int* cols, const double* vals,
    const double* x, double* y) {
        int i, j;
        for(i = 0; i < N; ++i)
            for(y[i] = 0, j = rowBegin[i]; j < rowEnd[i]; ++j)
                y[i] += vals[j] * x[cols[j]];
}
```

Рис. 1. Алгоритм умножения матрицы на вектор в расширенном формате CSR

4.2. Оптимизации обращений к памяти

Первый этап состоит в явном вычислении матриц $\hat{L}$, $\hat{D}$ и $\hat{U}$, $\hat{L} + \hat{D} + \hat{U} = \hat{A} = P^T AP$. Несмотря на то, что при этом затраты памяти увеличиваются, это дает возможность проведения дальнейших оптимизаций. Во-первых, при решении треугольной системы $(I+\bar{L})x = y$

серийный алгоритм проводит последовательные обращения к элементам всех массивов, кроме y, однако и для последнего обращения идут по возрастанию индексов элементов. Данное свойство позволяет существенно увеличить производительность, так как большинство аппаратных оптимизаций работы с кэшем процессора ориентированы на последовательное обращение к данным. Неприятность состоит в том, что процесс решения системы $(I+\bar{U})x = y$ не отличается такой регулярностью. Предлагаемое решение проблемы состоит в том, чтобы переставить местами строки в массивах $vals$ и $cols$ для матрицы $\bar{U}$ так, чтобы первой шла последняя строка, далее — предпоследняя, и так далее. Таким образом, если изначально элементы массивов были расположены следующим образом:

$$vals = \left(\begin{array}{cccc|c|cccc} a_{1,c_{1,1}} & a_{1,c_{1,2}} & \dots & a_{1,c_{1,k_1}} & \dots & a_{N,c_{N,1}} & a_{N,c_{N,2}} & \dots & a_{N,c_{N,k_N}} \end{array} \right),$$

$$cols = \left(\begin{array}{cccc|c|cccc} c_{1,1} & c_{1,2} & \dots & c_{1,k_1} & \dots & c_{N,1} & c_{N,2} & \dots & c_{N,k_N} \end{array} \right),$$

то после перестановки элементы будут расположены как показано ниже:

$$vals = \left(\begin{array}{cccc|c|cccc} a_{N,c_{N,1}} & a_{N,c_{N,2}} & \dots & a_{N,c_{N,k_N}} & \dots & a_{1,c_{1,1}} & a_{1,c_{1,2}} & \dots & a_{1,c_{1,k_1}} \end{array} \right),$$

$$cols = \left(\begin{array}{cccc|c|cccc} c_{N,1} & c_{N,2} & \dots & c_{N,k_N} & \dots & c_{1,1} & c_{1,2} & \dots & c_{1,k_1} \end{array} \right).$$

Тогда процедура решения треугольной системы для матриц вида $(I + \bar{U})$ в простейшем случае будет выглядеть как показано на рисунке 2. Как видно из кода, при этом достигается последовательное обращение к данным матрицы $\bar{U}$.

```
void solveU(int N, const int* rowBegin, const int* rowEnd,
    const int* cols, const double* vals,
    double* x, const double* y) {
        int i, j;
        for(i = 0; i < N; ++i)
            for(x[N-i-1] = y[N-i-1], j = rowBegin[i]; j < rowEnd[i]; ++j)
                x[N-i-1] -= vals[j] * x[cols[j]];
}
```

Рис. 2. Алгоритм решения верхне-треугольных систем для SSOR

4.3. NUMA-оптимизации

Как уже отмечалось выше, NUMA-системы (системы с несколькими сокетами) имеют несколько контроллеров памяти. При страничной организации памяти каждая страница физически относится к памяти только одного сокета. Поэтому, при необходимости обращения другого сокета к такой странице он осуществляет такой доступ посредством обращения к сокету, фактически владеющим страницей. При этом в ядре Linux по-умолчанию используется принцип привязки страниц к сокетам, называемый first-touch, то есть, первый сокет, который обращается к еще не привязанной странице, предварительно созданной системным вызовом типа *mmap*, привязывает эту страницу к себе [5].

Из приведенного выше описания принципов работы NUMA-систем виден их существенный недостаток: если данные инициализируются в одном потоке, а затем используются в нескольких, то фактически данные оказываются привязанными к одному сокету, что может привести к снижению максимальной скорости чтения и записи данных по сравнению со случаем их равномерной привязки к разным сокетами системы. Для преодоления указанной сложности предлагается следующее.

Во-первых, необходимо осуществить статическую привязку потоков к блокам матрицы, получаемых в алгоритме вложенных сечений. Это требуется для того, чтобы на каждой итерации одни и те же потоки читали одни и те же блоки матрицы, что позволит привязать эти блоки к соответствующим сокетам и, соответственно, разместить их в памяти, "близкой" к текущему потоку. Произвести статическую привязку предлагается следующим образом. В начале выделяются блоки, не имеющие зависимостей от других блоков. Эти блоки делятся равномерно между потоками, либо по числу строк, либо по числу ненулевых элементов. После этого данные блоки исключаются из рассмотрения и выделяются блоки, которые не имеют зависимостей, кроме как от уже рассмотренных блоков. Далее процесс повторяется, пока все блоки не будут назначены каким-либо потокам. В результате мы получим назначение блоков потоками, которое обеспечивает достаточно равномерную загрузку потоков.

Во-вторых, необходимо осуществить правильную привязку данных блоков матрицы и частей векторов соответствующим потокам. Добиться этого несложно, просто выделяя требуемую память с помощью вызова *malloc* и организовав копирование в массивы матриц $\bar{L}$ и $\bar{U}$, а также инициализацию элементов векторов в параллельном режиме таким образом, что каждый поток копирует и инициализирует только блоки, относящиеся к нему после статического назначения. В этом случае сработает принцип first touch, и страницы окажутся привязанными к необходимым сокетам. С учетом того, что на архитектурах x86 и x86-64 доступны страницы размером 4 килобайта и 2 мегабайта, можно заключить, что при большом объеме данных матрицы A (гигабайты и десятки гигабайт) возможно достичь высокой точности распределения данных по сокетам.

5. Численные эксперименты

Для подтверждения теоретических выводов, сделанных в рамках данной работы, а также для экспериментального тестирования производительности изложенных выше подходов был проведен ряд численных экспериментов на методических задачах. В качестве расчетной области рассматривался параллелепипед размера $1000 \times 100 \times 100$ мм с квазиструктурированной тетраэдральной сеткой, полученной из разбиения расчетной области на кубы с ребром 1 мм и дальнейшего разбиения кубов на 6 тетраэдров. В качестве задачи предлагалось решение уравнения Пуассона, которое в алгебраическом виде соответствует обращению нижнего блока предобуславливателя $\mathcal{P}_{M,L}$. В случае прямоугольной равномерной сетки и конечно-элементной аппроксимации вариационной задачи для данного уравнения узловыми конечными элементами первого порядка получается матрица следующего вида:

$$L_{i,j} = \mathcal{C} \cdot \begin{cases} 6, & i = j, \\ -1, & i \neq j, \end{cases} \tag{12}$$

где $\mathcal{C}$ — некоторая константа, связанная с объемом элементов сетки. Далее решалось матричное уравнение

$$Lx = f, \tag{13}$$

со случайным вектором $x : 0 \leq x_i \leq 1$ при помощи метода сопряженных градиентов (CG). Производилась геометрическая декомпозиция расчетной области путем деления ее на 16 частей по оси Ox (так что суммарное число частей, включая разделители, оказывалось равным 31). Параметр релаксации ω был выбран равным 1.85, критерием остановки итераций служило условие $|f - Lx| < \varepsilon |f|$, $\varepsilon = 10^{-7}$. Общее число итераций при этом оказывалось равным 65. Итерационный решатель и предобуславливатель SSOR были распараллелены с использованием OpenMP. Результаты численных экспериментов представлены ниже.

Для проведения численных экспериментов использовались следующие серверы:

- Intel Xeon X5670, 2.93 ГГц, 2 сокета x 6 ядер (всего 12 ядер), включен Hyper Threading;

- Intel Xeon X7560, 2.27 ГГц, 4 сокета x 8 ядер (всего 32 ядра).

В качестве библиотеки OpenMP использовалась библиотека, входящая в состав компилятора Intel Composer XE 2011.

Для сравнительного тестирования производительности был также реализован обычный метод сопряженных градиентов с предобуславливателем SSOR с использованием библиотеки Intel MKL 10.3. В частности, из данной библиотеки использовались функции BLAS уровня 1, а также функции решения треугольных разреженных систем из Sparse BLAS уровня 2. Все эти функции распараллелены внутри библиотеки Intel MKL [6]. Для того, чтобы обеспечить такое же число итераций, как и у предлагаемого в данной работе алгоритма, в случае MKL SSOR также производилось переупорядочивание матрицы.

Предобуславливатель	KMP_AFFINITY	Число потоков						
		1	2	4	6	8	12	24
Par SSOR	compact,0,0	15.8	14.2	11.4	11.7	11.8	12.0	8.26
Par SSOR	compact,1,0	16.0	11.9	11.4	11.6	92.4	7.90	8.44
Par SSOR	scatter,0,0	15.9	10.0	8.07	7.91	7.78	8.03	8.41
MKL SSOR	compact,0,0	22.9	26.4	25.4	26.6	26.6	26.6	26.6
MKL SSOR	compact,1,0	22.1	20.7	21.3	21.6	19.7	18.3	18.4
MKL SSOR	scatter,0,0	21.9	17.7	17.3	17.7	17.9	18.5	18.4

Таблица 1. Время работы решателей, Intel Xeon X5670 @ 2.93 GHz

Предобуславливатель	KMP_AFFINITY	Число потоков					
		1	2	4	8	16	32
Par SSOR	compact,0,0	30.4	18.9	15.5	14.8	10.2	8.56
Par SSOR	scatter,0,0	30.4	17.1	9.69	7.48	8.56	8.57
MKL SSOR	compact,0,0	40.6	34.5	33.5	33.4	28.9	26.3
MKL SSOR	scatter,0,0	40.6	31.8	27.1	26.1	26.3	26.3

Таблица 2. Время работы решателей, Intel Xeon X7560 @ 2.27 GHz

В таблицах 1 и 2 представлены результаты сравнительного тестирования предлагаемого алгоритма (Par SSOR) и алгоритма, реализованного с использованием библиотеки Intel MKL (MKL SSOR). В таблицах приведены времена работы решателей в секундах в зависимости от числа потоков, а также в зависимости от установки переменной окружения KMP_AFFINITY, определяющей распределение потоков OpenMP по сокетам, ядрам и логическим процессорам (в случае включенного режима Hyper Threading) системы. На рисунках 3 и 4 приведены соответствующие графики.

Из полученных данных видно, что алгоритм в действительности является ограниченным по пропускной способности шины данных: при небольшом числе потоков бóльшую

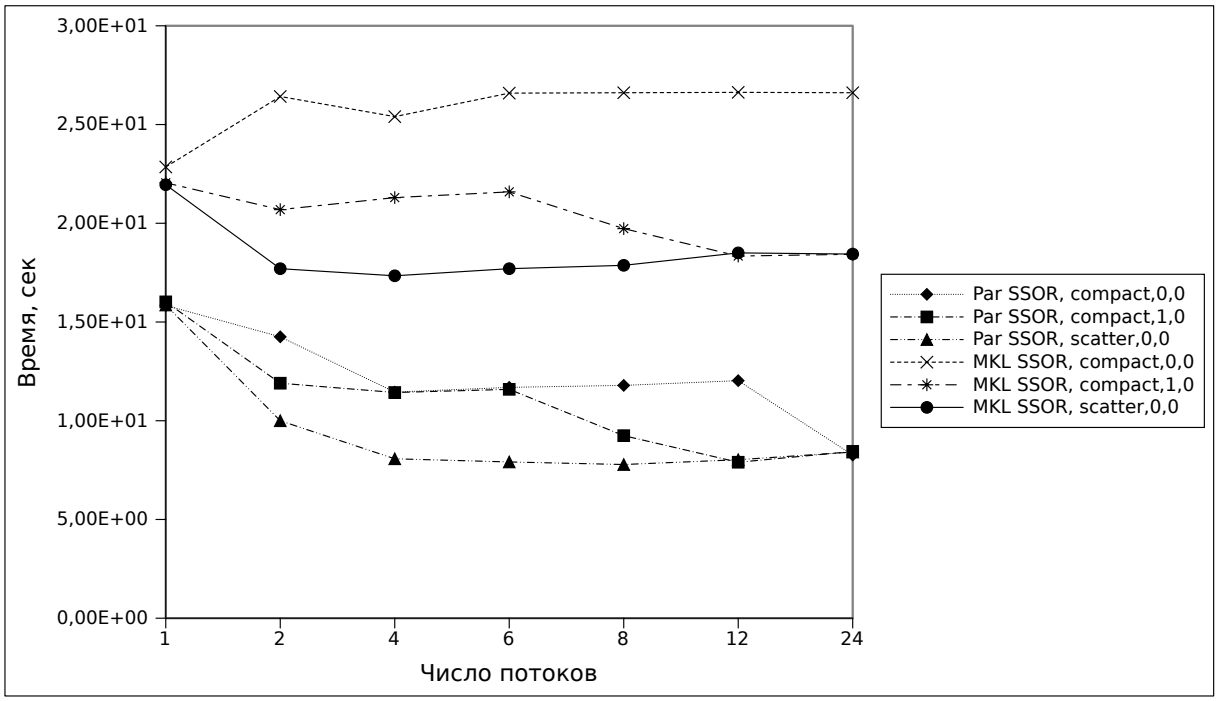


Рис. 3. Сравнение производительности решателей, Intel Xeon X5670 @ 2.93 GHz

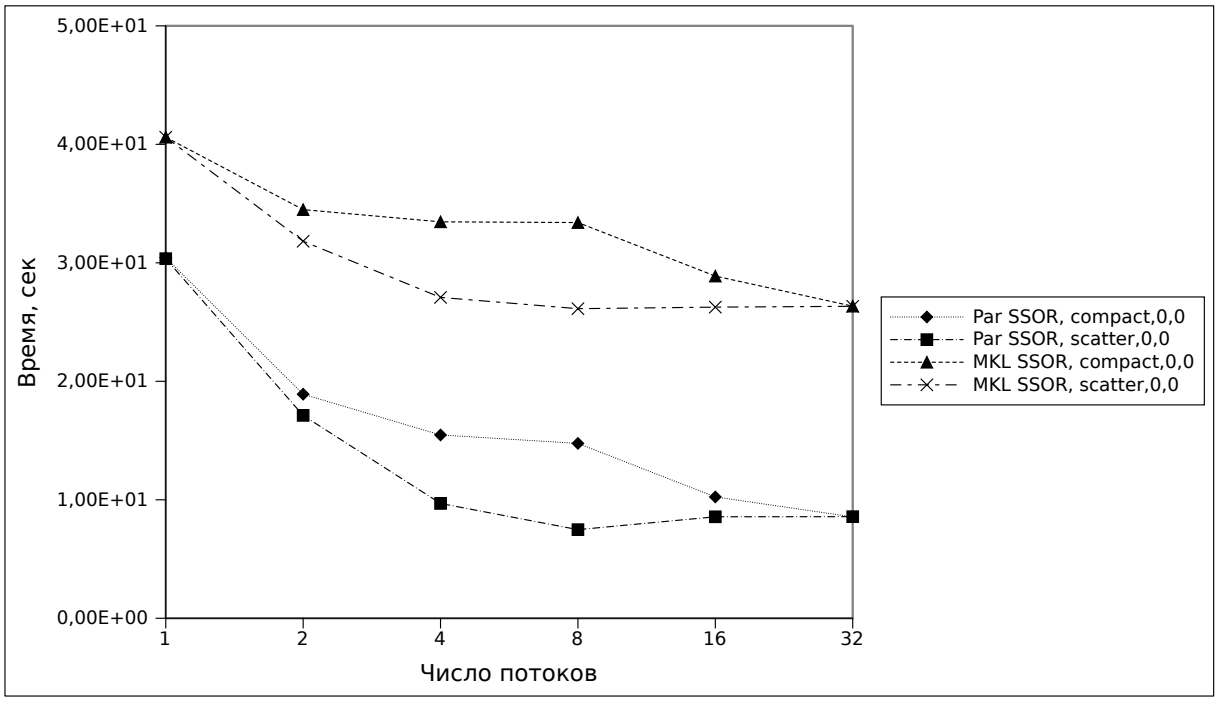


Рис. 4. Сравнение производительности решателей, Intel Xeon X7560 @ 2.27 GHz

производительность имеют варианты расположения потоков на разных сокетах системы. Кроме того, видно, что слишком большое число потоков приводит к снижению производительности, поскольку они начинают конкурировать за шину данных. В итоге, алгоритм показывает оптимальную производительность в случае, когда на один сокет приходится приблизительно 2 потока.

Также можно отметить высокую производительность полученного алгоритма, а также его довольно хорошую масштабируемость, по сравнению с масштабируемостью алгоритма

при использовании библиотеки Intel MKL. В наилучшем варианте, алгоритм показывает в 2.6 раза лучшую масштабируемость, и в 3.5 раза лучшую производительность. При этом максимальная достигнутая масштабируемость алгоритма составляет 4.1 раза на Intel Xeon X7560 на 8 потоках.

6. Заключение

В работе предложен подход к получению масштабируемого алгоритма решения треугольных систем для предобуславливателя SSOR. Описан ряд оптимизаций, позволяющих повысить производительность алгоритма. Приведенные численные эксперименты показывают, что алгоритм демонстрирует масштабируемость до 4-х раз на некоторых из доступных в настоящее время многопроцессорных машин с общей памятью и существенно эффективнее реализации с использованием параллельных функций решения треугольных систем библиотеки Intel MKL.

Отметим также, что хотя в работе рассматривается случай симметричной матрицы A, предложенный подход может быть применен для любых матриц с симметричным портретом. Более того, данный подход может быть использован для построения кластерной версии предобуславливателя SSOR, однако эта тема требует дальнейших исследований. Также одним из возможных направлений является исследование вопроса построения наиболее эффективного метода отслеживания зависимостей между блоками во время работы программы, когда некоторые из блоков вектора неизвестных оказываются вычисленными.

Литература

1. Ильин В.П. Методы и технологии конечных элементов. Новосибирск: Изд-во ИВМиМГ СО РАН, 2007. 370 с.

2. 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.

3. Greif C., Schötzau D. Preconditioners for the discretized time-harmonic Maxwell equations in mixed form // Numer. Linear Algebra Appl. 2007. Vol. 14, P. 281–297.

4. Bossavit A. Computational Electromagnetism. Variational Formulations, Complementarity, Edge Elements. Academic Press, San Diego, CA, USA, 1998.

5. Optimizing Software Applications for NUMA:
URL: http://software.intel.com/en-us/articles/optimizing-software-applications-for-numa

6. 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

Система поддержки принятия решений при выборе параллельного аппаратно-программного комплекса для построения областей достижимости летательного аппарата

А.В. Быстров, А.П. Карпенко, О.Г. Козлова, В.А. Федин

МГТУ им. Н.Э.Баумана

Приводится постановка задача приближенного построения области достижимости динамической системы. Рассматриваются численные методы решения этой задачи на кластерных вычислительных системах, графических процессорных устройствах и нейросетевых ускорителях. Обсуждаются основные принципы разработки системы поддержки принятия решений, предназначенной для оптимального проектирования аппаратно-программного комплекса, обеспечивающего параллельное построение границ области достижимости летательного аппарата с заданной точностью и за заданное время.

1. Введение

Для приближенного построения области достижимости летательного аппарата (ОДЛА) могут быть использованы методы на основе многократного интегрирования исходной модельной системы обыкновенных дифференциальных уравнений (ОДУ), близкие алгоритмы, использующие полиномиальную или нейросетевую аппроксимацию правых частей этих уравнений, алгоритмы приближенного построения аппроксимаций границ ОДЛА и т.д. [1].

Указанные методы могут быть реализованы на многопроцессорных вычислительных системах с общей или распределенной памятью, системах на основе графических процессорных устройств, а также на нейросетевых ускорителях [2,3]. При этом каждому из методов может быть поставлено в соответствие несколько алгоритмов, каждый из которых может быть несколькими способами отображен на архитектуру каждой из рассматриваемых вычислительных систем. В силу многообразия вариантов состава целевого аппаратно-программного комплекса, реализующего построение ОДЛА на борту летального аппарата, возникает задача разработки программной системы для автоматизированного синтеза этого комплекса. В работе данная система рассматривается, как система поддержки принятия решений (СППР).

Полагается, что СППР функционирует на инструментальном гетерогенном вычислительном комплексе, который включает в себя все указанные классы вычислительных систем. Это дает возможность корректной оценки времени решения задачи, а также позволяет выполнять на инструментальном комплексе отладку целевых программ.

2. Постановка задачи

Рассмотрим динамическую систему

$$\dot{X} = F(t, X, U),\ X(0) = X^0,\ t \in [0,T], \qquad (1)$$

где $X = X(t)$ – n-мерный вектор фазовых переменных системы, $U = U(t)$ – m-мерный вектор управлений, X^0 – n-мерный вектор начальных условий, $t \in [0,T]$, $F(t,X,U)$ – n-мерная вектор-функция. Задано множество допустимых управлений D_U, так что $U \in D_U \subset L_U[0,T]$, где $L_U[0,T]$ - некоторое пространство m-мерных функций, определенных на интервале $[0,T]$, например, пространство кусочно-постоянных функций [4].

Среди компонентов вектора фазовых переменных $X=(x_1,x_2,...,x_n)$ выделим $\nu \le n$ переменных. Не ограничивая общности, положим, что эти переменные образуют ν-мерный вектор $Y=(x_1,x_2,...,x_\nu)^T$.

Областью достижимости $D=D(T,X^0)$ системы (1) назовем множество всех возможных значений вектора $Y(T)$, которые принимаются на решениях системы (1) при начальных условиях X^0 и выполнении условия $U\in D_U$. Ставится задача приближенного построения области достижимости D.

В области достижимости D может находиться одна или несколько, вообще говоря, динамических запрещенных областей $\Omega_i(t)$, $i\in[1:\kappa]$. Очевидно, что в результате область достижимости D трансформируется. Сохраним за трансформированной препятствиями областью достижимости прежнее обозначение D.

Для решения поставленной задачи, очевидно, достаточно построить границу соответствующей области достижимости. В некоторых случаях могут быть известны множества управлений $D_U^\Gamma \in L_U^\Gamma[0,T]\subseteq L_U[0,T]$, которые приводят систему (1) на границу Γ области достижимости D [4]. В этом случае задача сводятся к построению границы Γ. Здесь $L_U^\Gamma[0,T]$ - некоторое подпространство функционального пространства $L_U[0,T]$.

2. Методы решения задачи

Метод мультифиниша. Покроем множество D_U некоторой сеткой с узлами $U_1,U_2,...,U_M$. Поставим в соответствие системе (1) совокупность M систем ОДУ с указанными управлениями:

$$\begin{cases} \dot{X}_1 = F(t,X_1,U_1),\ X_1(0)=X^0, \\ \qquad\qquad ... \\ \dot{X}_M = F(t,X_M,U_M),\ X_M(0)=X^0. \end{cases} \tag{2}$$

Тогда схему приближенного построения области достижимости D методом мультифиниша можно представить в следующем виде.

1) С помощью того ли иного метода интегрирования совокупности систем (2) находим множество точек $\{Y_i(T)\}$, представляющее собой дискретную аппроксимацию области D. Запоминаем полученные наборы значений $(U_i,Y_i(T))$; $i\in[1:M]$.

2) Во множестве $\{Y_i(T)\}$ находим граничные точки $\{Z_j(T)\}$, представляющие собой дискретную аппроксимацию Γ_d границы Γ области достижимости. Запоминаем соответствующие наборы значений $(U_j,Z_j(T))$; $j\in[1:\varsigma]$.

3) На основе точек $\{Z_j(T)\}$ строим подходящую непрерывную аппроксимацию Γ_c границы Γ.

Модифицированный метод мультифиниша использует комбинацию рассмотренного метода мультифиниша с методом аппроксимации векторного поля системы (1), т.е. подходящую аппроксимацию вектор-функции $F(t,X,U)$.

Общая схема метода имеет следующий вид.

1). Покрываем множество $[0,T]\times D_X\times D_U$ некоторой сеткой $\Delta=\{t_a,x_{i,b},u_{k,c}\}$ с узлами $(t_a,x_{i,b},u_{k,c})$; $i\in[1:n]$, $k\in[1:m]$.

2). Во всех узлах сетки Δ вычисляем значения функции $F(t_a,X_b,U_c)$ и запомним эти значения.

3). Строим некоторую функцию $\tilde{F}(t,X,U)$, аппроксимирующую функцию $F(t,X,U)$.

4) По схеме метода мультифиниша интегрируем совокупность систем ОДУ (2), используя в качестве требуемых значений функции $F(t,X,U)$ значения функции $\tilde{F}(t,X,U)$. Строим дискретную Γ_d, а затем непрерывную аппроксимацию Γ_c границы Γ.

Рассматривается два класса методов построения функции $\tilde{F}(t,X,U)$ - методы на основе полиномиальной МНК-аппроксимации функции $F(t,X,U)$ и методы нейросетевой аппроксимации этой функции [5]. Отметим, что из числа методов первого класса в алгоритмическом отношении удобно использовать композицию одномерных полиномов (метод Брандона) [6].

Для интегрирования совокупности систем ОДУ (2) предполагается использование явных, неявных, а также соответствующих нейросетевых методов интегрирования.

Для построения непрерывной границы Γ_c имеется в виду использование полиномиальной или нейросетевой аппроксимацию границы Γ_d.

Далее полагается, что известен класс управлений, приводящих систему (1) на границу области достижимости Γ, и что этот класс представляет собой релейные управления с не более, чем одной точкой переключения каждого из компонентов вектора $U(t)$ [4].

3. Распараллеливание вычислений

В методе мультифиниша при приближенном построении границы Γ естественным образом выделяется два этапа:

1) интегрирование системы ОДУ (2) и построение дискретной аппроксимации Γ_d границы области достижимости Γ;

2) построение непрерывной аппроксимации Γ_c этой границы.

Этап 1. Для решения задачи первого этапа рассматривается использование многопроцессорной вычислительной системы с распределенной памятью (вычислительного кластера), графического процессорного устройства (ГПУ) с архитектурой CUDA, а также нейросетевой вычислительной системы (НСВС).

Схемы распараллеливания вычислений первого этапа и особенности отображения метода мультифиниша на архитектуры вычислительного кластера и ГПУ рассмотрены в наших работах [2,3]. В этих же работах приведены соответствующие результаты исследования эффективности распараллеливания.

Этап 2. Построение границы Γ_c также может быть выполнено на вычислительных кластерах, ГПУ и на НСВС. Использование с этой целью НСВС рассмотрено в наших работах [7, 8].

В модифицированном методе мультифиниша аналогично выделяется три этапа.

1) построение функции $\tilde{F}(t,X,U)$, аппроксимирующей функцию $F(t,X,U)$;

2) интегрирование системы (2) и построение дискретной аппроксимации Γ_d границы области достижимости Γ с использованием функции $\tilde{F}(t,X,U)$ вместо функции $F(t,X,U)$;

3) построение непрерывной аппроксимации Γ_c этой границы.

Среди указанных этапов новым, по сравнению с этапами метода мультифиниша, является только первый этап. На этом этапе полиномиальная МНК-аппроксимация функции $F(t,X,U)$ может быть реализована на вычислительных кластерах и ГПУ, а нейросетевая аппроксимация – на указанных системах, а также на НСВС.

Метод мультифиниша можно считать частным случаем модифицированного метода мультифиниша, когда $\tilde{F}(t,X,U) = F(t,X,U)$. Поэтому будем далее говорить только о последнем из этих методов и называть его просто «метод мультифиниша».

4. Основные функции системы поддержки принятия решений

Для приближенного построения границ ОДЛА может быть использована иерархическая совокупность численных методов, представленная на рисунке 1. Здесь полагается, что «Методы построения границы Γ_d» включают в себя и методы интегрирования систем ОДУ (2).

Рис. 1. Иерархия методов приближенного построения границы ОДЛА

Обозначим $\mathbf{M}_1 = \{M_{1,i}\}$ - рассматриваемую совокупность методов аппроксимации функции $F(t, X, U)$. Аналогичные обозначения $\mathbf{M}_2$, $\mathbf{M}_3$ введем для совокупностей методов построения границ Γ_d, Γ_c соответственно.

Каждому из методов $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$ можно поставить в соответствие некоторую совокупность алгоритмов $\mathbf{A}_{i,j} = \{A_{i,j,k}(\mathbf{P}_{i,j,k})\}$, где $A_{i,j,k}$ - k-й алгоритм, реализующий метод $M_{i,j}$, а $\mathbf{P}_{i,j,k}$ - вектор свободных параметров этого алгоритма. В случае алгоритма $A_{1,j,k}$, например, этот вектор может включать в себя порядок аппроксимирующего полинома; в случае алгоритма $A_{2,j,k}$ - число узлов сетки, покрывающей множество D_U; в случае алгоритма $A_{3,j,k}$ - также порядок аппроксимирующего полинома.

Назовем проектируемую бортовую вычислительную систему «целевой», а вычислительные системы, на которых функционирует рассматриваемая СППР, «инструментальными». В качестве вариантов реализации целевой ЭВМ рассмотрим следующие классы систем: вычислительный кластер $\mathbf{C}_1 = \{C_1(\mathbf{P}_1)\}$, ГПУ $\mathbf{C}_2 = \{C_2(\mathbf{P}_2)\}$ и НСВС $\mathbf{C}_3 = \{C_3(\mathbf{P}_3)\}$, где $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ - векторы свободных параметров соответствующих ЭВМ (например, число процессоров для вычислительного кластера, число мультипроцессоров для ГПУ, число нейрочипов для НСВС).

Множество вариантов отображения алгоритма $A_{i,j,k}(\mathbf{P}_{i,j,k})$ на архитектуру системы $C_l(\mathbf{P}_l)$ обозначим

$$\mathbf{E}(A_{i,j,k}(\mathbf{P}_{i,j,k}), C_l(\mathbf{P}_l)) = \{E_m(A_{i,j,k}(\mathbf{P}_{i,j,k}), C_l(\mathbf{P}_l))\}.$$

Задача оптимального проектирования аппаратно-программного комплекса, обеспечивающего построение областей достижимости летательного аппарата за заданное время с заданной точностью, состоит в многокритериальном выборе методов M_{1,i^*}, M_{2,j^*}, M_{3,k^*}, соответствующих алгоритмов $A_{1,i^*,l^*}(\mathbf{P}^*_{1,i,j})$, $A_{2,k^*,l^*}(\mathbf{P}^*_{2,k,l})$, $A_{3,m^*,n^*}(\mathbf{P}^*_{3,m,n})$, целевой ЭВМ $C_{q^*}(\mathbf{P}^*_q)$ и, наконец, в выборе вариантов отображения указанных алгоритмов на архитектуру выбранной ЭВМ.

Процесс решения задачи оптимального проектирования является итерационным и включает в себя несколько вложенных циклов:

- выбор методов $M_{1,i}$, $M_{2,j}$, $M_{3,k}$;

- выбор соответствующих алгоритмов $A_{1,i,l}$, $A_{2,j,m}$, $A_{3,k,n}$;

- выбор значений параметров $\mathbf{P}_{1,i,l}$, $\mathbf{P}_{2,j,m}$, $\mathbf{P}_{3,k,n}$ этих алгоритмов;

- выбор класса целевой вычислительной системы $\mathbf{C}_q$;

- выбор значений вектора параметров $\mathbf{P}_q$ и, тем самым, целевой системы $C_q(\mathbf{P}_q)$;

- выбор вариантов отображения алгоритмов $A_{1,i,l}(\mathbf{P}_{1,i,l})$, $A_{2,j,m}(\mathbf{P}_{2,j,m})$, $A_{3,k,n}(\mathbf{P}_{3,k,n})$ на вычислительную систему $C_q(\mathbf{P}_q)$;

- построение аппроксимации границы Γ_c ОДЛА на инструментальной вычислительной системе класса $\mathbf{C}_q$ с помощью выбранных методов, алгоритмов и их параметров;

- оценка точности полученной аппроксимации;

- в случае неудовлетворительной точности, последовательные попытки изменить значения параметров алгоритмов, выбрать другой алгоритм и, наконец, - другой метод;

- оценка времени решения задачи на системе $C_q(\mathbf{P}_q)$;

- в случае неудовлетворительного времени решения задачи, последовательные попытки изменить значения параметров системы и класса системы.

Отметим, что для корректной оценки времени приближенного построения ОДЛА на целевом аппаратно-программном комплексе, в инструментальный вычислительный комплекс требуется включить вычислительный кластер, ГПУ и НСВС. Гетерогенность инструментального вычислительного комплекса позволяет одновременно проводить на нем отладку целевого программного обеспечения.

Для вычислительного кластера методика оценки времени решения задачи на целевой ЭВМ рассмотрена в нашей работе [2]. Методика предполагает получение аналитической зависимости этого времени от параметров вычислительной системы и последующее экспериментальное уточнение указанной зависимости.

Разрабатываемая СППР призвана облегчить лицу, принимающему решения (ЛПР), или группе лиц, принимающих решения (ГПР), решение указанной задачи оптимального проектирования. Основной задачей, которую должна решать эта СППР, является многокритериальная оценка вариантов параллельного аппаратно-программного комплекса. В качестве критериев качества решения при этом могут использоваться вес вычислительной системы, ее габариты, энергопотребление, стоимость и т.д.

5. Оценка точности аппроксимации границ ОДЛА

Для вычисления значений указанных выше критериев оптимальности нужно, прежде всего, оценить точность аппроксимации границы Γ области достижимости границей Γ_c с помощью выбранных методов и алгоритмов. Аналитическая оценка этой точности не удается. Поэтому предлагается определение этой точности с помощью вычислительного эксперимента на соответствующей вычислительной системе инструментального комплекса.

Рассмотрим прежде частный случай, который определяется следующими соглашениями [4].

1) Имеет место плоское движение ЛА, которое определяется компонентой $u_1(t) = \gamma_c = const$ вектора управления $U(t)$, где γ_c.- угол наклона траектории; $\gamma_c \in [0, \pi]$.

2) Известна структура оптимальных управлений $u_2(t), u_3(t)$, приводящих на границы ОДЛА. Точнее говоря, дальняя граница достигается при управлениях

$$u_2 = \begin{cases} \pm 1, & t \le \tau_f, \\ 0, & t > \tau_f, \end{cases} \quad u_3 = const = 1,$$

ближняя граница - при управлениях

$$u_2 = \begin{cases} \mp 1, & t \le \tau_n, \\ \pm 1, & t > \tau_n, \end{cases} \quad u_3 = const = -1,$$

а боковая граница – при управлениях

$$u_2 = const = \pm 1, \quad u_3 = \begin{cases} \pm 1, & t \le \tau_s, \\ \mp 1, & t > \tau_s, \end{cases}$$

Здесь $\tau_f \in [0,T]$ – момент времени переключения управления $u_2 = u_2(t)$ при построении дальней границы ОДЛА, $\tau_n \in [0,T]$ – аналогичный момент времени при построении ближней границы, $\tau_s \in [0,T]$ – момент времени переключения управления $u_3 = u_3(t)$ при построении боковой границы ОДЛА.

Положим, что тем или иным способом для управлений $U_1, U_2, ..., U_M$ получены наборы точек $\{Z_i^f(T)\}$, $\{Z_k^n(T)\}$, $\{Z_l^s(T)\}$, представляющие собой, соответственно, дискретные аппроксимации Γ_d^f, Γ_d^n, Γ_d^s дальней, ближней и боковой границ ОДЛА. И пусть на этой основе построены некоторые непрерывные аппроксимацию Γ_c^f, Γ_c^n, Γ_c^s указанных границ.

Схема метода оценки точности аппроксимации, к примеру, границы области достижимости Γ^f границей Γ_c^f имеет следующий вид.

1) В прямоугольнике $[0,T] \times [0,\pi]$ генерируем K псевдослучайных равномерно распределенных точек $(\tau_{F,1}, \gamma_{c,1})$, $(\tau_{F,2}, \gamma_{c,2}), \ldots, (\tau_{F,k}, \gamma_{c,K})$ и определяем соответствующие управления $u_1(t) = \gamma_{c,i}$, $u_2(\tau_{f,i})$, $u_3 = 1$, $i \in [1:K]$.

2) При каждом из указанных управлений интегрируем систему ОДУ (1), находим декартовы координаты точки $Z_i^f(T)$, а затем ее полярные координаты $(\varphi_i^f(T), \chi_i^f(T), \rho_i^f(T))$, где $\varphi_i^f(T)$, $\chi_i^f(T)$, $\rho_i^f(T)$ - азимут, угол места и расстояние от начала координат до точки $Z_i^f(T)$ соответственно.

3) В направлении $(\varphi_i^f(T), \chi_i^f(T))$ определяем расстояние $\tilde{\rho}_i^f(T)$ от начала координат до точки на границе Γ_c^f.

4) Вычисляем величину $\varepsilon_i^f(T) = \left| \rho_i^f(T) - \tilde{\rho}_i^f(T) \right|$ - погрешность аппроксимации границы ОДЛА в направлении $(\varphi_i^f(T), \chi_i^f(T))$.

5) В качестве искомой оценки точности аппроксимации принимаем, например, оценку математического ожидания погрешности или оценку максимальной погрешности:

$$\bar{\varepsilon}^f(T) = \frac{1}{K} \sum_{i=1}^{K} \varepsilon_i^f(T); \quad \varepsilon_{\max}^f(T) = \max_{i \in [1:K]} \varepsilon_i^f(T).$$

6. Пример

Рассмотрим летальный аппарат, движение центра масс которого в земной неподвижной системе координат $0x_r y_r z_r$ описывается системой нелинейных дифференциальных уравнений (3), где v – скорость летательного аппарата; Θ – угол наклона траектории; Ψ – угол поворота траектории; x_r, y_r, z_r – геометрические координаты центра масс летательного аппарата (y_r - высота); $u_1 = \gamma_C$ – скоростной угол крена, $\gamma_c \in [0,\pi]$; $n_T = n_T^{\max} u_2$ – тангенциальная перегрузка, $|n_T| \le n_T^{\max}$, $|u_2| \le 1$; $n = n^{\max} u_3$ – нормальная перегрузка, $|n| \le n^{\max}$, $|u_3| \le 1$; g – ускорение свободного падения [4].

Движение летательного аппарата рассматривается на интервале времени $t \in [0,T]$. Заданы начальные условия $v(0) = v^0$, $\Theta(0) = \Theta^0$, $\Psi(0) = \Psi^0$, $x_r(0) = x_r^0$, $y_r(0) = y_r^0$, $z_r(0) = z_r^0$.

$$\begin{cases} \dot{v} = g \cdot (n_T^{\max} u_2 - \sin\Theta), \\ \dot{\Theta} = g/v \cdot (n^{\max} u_3 \cos u_1 - \cos\Theta), \\ \dot{\Psi} = -gn^{\max} u_3 \cdot \sin u_1 / v \cdot \cos\Theta, \\ \dot{x}_r = v \cdot \cos\Theta \cdot \cos\Psi, \\ \dot{y}_r = v \cdot \sin\Theta, \\ \dot{z}_r = -v \cdot \cos\Theta \cdot \sin\Psi. \end{cases} \quad (3)$$

Положим, что относительно управлений u_1, u_2, u_3 действуют соглашения, указанные в предыдущем разделе.

Построение области достижимости летательного аппарата, описываемого системой ОДУ (3), выполнено на ГПУ NVIDIA GeForce 9600 GT. В качестве среды разработки системы использована Microsoft Visual Studio 2008 и технология OpenCL. Схему параллельных вычислений покажем на примере построения дальней границы ОДЛА. Покроем прямоугольник $[0,T]\times[0,\pi]$ равномерной сеткой Δ_f с $N_1 \times N_2 = M$ узлами $(\tau_{f,i}, \gamma_{c,j})$, где $i \in [1:N_1]$, $i \in [1:N_2]$. Разобьем узлы этой сетки на $N_{blocs} = N_2 \dfrac{N_1}{L}$ блоков по L узлов в каждом и поставим в соответствие каждому из блоков поток с идентификатором (k, j), где $k = iL$. Таким образом, потоку с идентификатором (k, j) соответствуют значения $\tau_f = \tau_{f,l}$, $\tau_{f,(l+1)}$, …, $\tau_{f,(l+L)}$ и значение $\gamma_c = \gamma_{c,j}$. Здесь $l = kL + 1$.

На рисунке 2 в качестве примера изображена область достижимости летательного аппарата (3) при двух стационарных сферических препятствиях Ω_1, Ω_2, построенная при начальных условиях Z=1000 м, X=0, Y=0, V=330 м/с, Θ=0, Ψ=0 и T=10 с. Соответствующие области запрещенных управлений (управлений, которые порождают траектории летательного аппарата, пересекающие хотя бы одно из препятствий) приведены на рисунке 3.

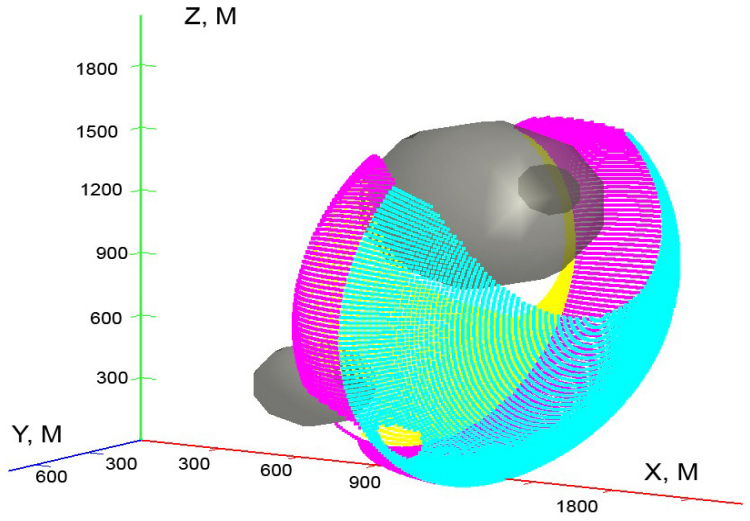


Рис. 2. Область достижимости летательного аппарата при двух стационарных сферических препятствиях

Для ситуации, когда в области достижимости летального аппарата (3) имеется одно стационарное сферическое препятствие, выполнено исследование производительности параллельных вычислений. Некоторые результаты исследований представлены на рисунке 4, который показывает, что при большом числе узлов сетки Δ_f ускорение вычислений достигает пяти (по сравнению с вычислениями на центральном процессоре системы).

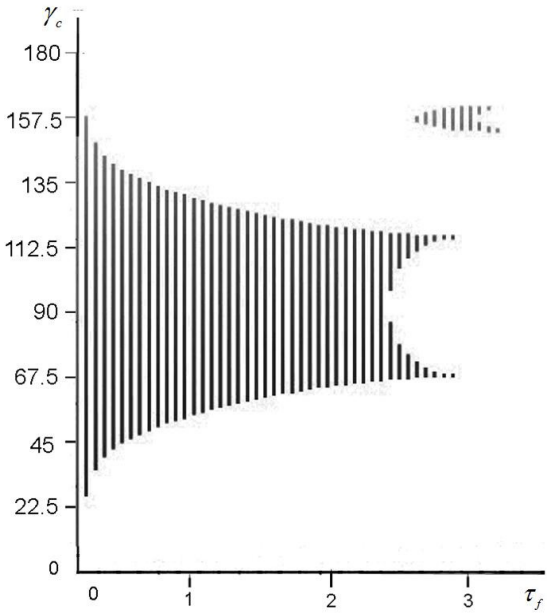


Рис. 3. Области запрещенных управлений для задачи с двумя стационарными сферическими препятствиями

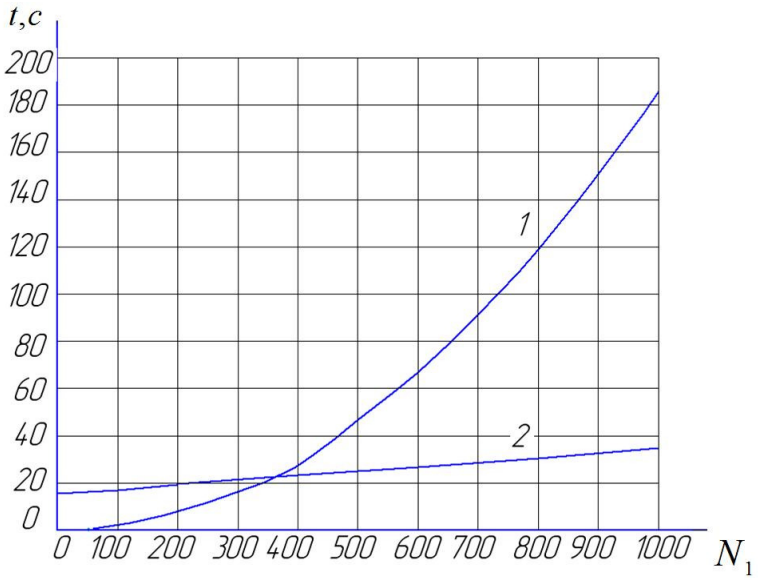


Рис. 4. Время построения дальней границы ОДЛА в функции числа узлов N_1 при $N_2 = 256$:
1 – GPU; 2 - CPU

Заключение

В работе изложены основные принципы построения системы поддержки принятия решений, предназначенной для оптимального проектирования аппаратно-программного комплекса, обеспечивающего параллельное построение границ области достижимости летательного аппарата с заданной точностью и за заданное время.

В качестве целевой вычислительной системы предполагается использование кластерной системы, системы на основе графических процессорных устройствах или системы на основе и нейросетевых ускорителей. Инструментальная вычислительная система полагается гетерогенной, включающей в себя указанные типы параллельных вычислительных систем.

Рассмотрена иерархическая совокупность численных методов приближенного построения границ ОДЛА, включающая в себя различные методы аппроксимации векторного поля

модельной системы обыкновенных дифференциальных уравнений, методы построения дискретной аппроксимации границы ОДЛА, а также методы построения непрерывной аппроксимации этой границы.

В настоящее время ведутся работы по реализации рассматриваемой системы поддержки принятия решений.

Авторы благодарят д.т.н., проф. МГТУ им. Н.Э.Баумана Воронова Е.М. и к.т.н. Карпунина А.А. за постановку задачи приближенного построение границ области достижимости летательного аппарата и плодотворные многократные обсуждения возможных подходов к ее решению.

Литература

1. Воронов Е.М., Карпенко А.П., Козлова О.Г., Федин В.А. Численные методы построения области достижимости динамической системы // Вестник МГТУ. Сер. “Приборостроение”. 2010. №2(79). С. 3-19.
2. Воронов Е. М., Карпенко А. П., Федин В. А. Параллельное построение множества достижимости высокоманевренного летательного аппарата методом "мультифиниша" // Параллельные вычислительные технологии (ПаВТ’2010): Труды международной научной конференции / ЮУрГУ. Челябинск: ЮУрГУ, 2010. С. 113-120.
3. Витюков Ф.А., Домашнев В.К., Карпенко А.П., Федин В.А. Построение области достижимости динамической системы на NVIDIA и AMD графических процессорах // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи 2010: Труды международной суперкомпьютерной конференции / МГУ. Москва: МГУ, 2010. С. 635-641.
4. Воронов, Е.М., Карпунин А.А. Алгоритм оценки границ области достижимости летательного аппарата с учетом тяги // Вестник МГТУ. Сер. Приборостроение. 2007. №4(69). С. 81-99.
5. Саймон Хайкин. Нейронные сети: полный курс. 2-е издание. Вильямс, 2006. С. 281-286.
6. Хохлов, С.Ф., Школа О.И. Применение метода Брандона для обработки экспериментальных данных // Вопросы химии и химической технологов. 1973. Вып. 28. С. 204-207.
7. Козлова О.Г. Нейросетевая аппроксимация границы области достижимости летательного аппарата. 1. Двухмерный случай // Наука и образование: электронное научно- техническое издание. 2009. №7.

   URL http://technomag.edu.ru/doc/129990.html (дата обращения: 11.12.2010).
8. Козлова О.Г. Нейросетевая аппроксимация границы области достижимости летательного аппарата. 2. Трехмерный случай // Наука и образование: электронное научно- техническое издание. 2009. №8.

   URL http://technomag.edu.ru/doc/130282.html (дата обращения: 11.12.2010).

Использование HPC технологий для решения пространственных задач мультифизики

В.А. Васильев[1], М.В. Крапошин[2], А.Ю. Ницкий[1], А.В. Юскин[2]

Челябинский Государственный Университет[1]
НИЦ «Курчатовский Институт»[2]

В работе описывается результат разработки программного комплекса MCF для решения сопряженных задач гидроупругости различными способами. Рассмотрены существующие способы решения слабосвязанных и сильносвязанных задач. Из возможных методов сопряжения выбраны и описаны итерационные методы, как наиболее простые для реализации и предоставляющие широкие возможности масштабирования программного обеспечения. Рассматривалось связывание гидродинамического модуля OpenFOAM (движение турбулентной (RAS и LES) несжимаемой, слабосжимаемой и сжимаемой жидкостей) и двух моделей динамики конструкции: метод конечных объёмов (OpenFOAM) и метод разложения по собственным формам и частотам (UZOR). Проведена оценка производительности вычислительных машин ТТИ ЮФО и ЧелГУ.

Введение

В настоящее время все большую популярность завоевывают численные методы для решения задач мультифизики, позволяющие исследовать взаимосвязанное влияние двух или более феноменов в рамках единой модели, например:

- задачи гидро- и аэро- упргуости;
- задачи магнитогидродинамики;
- задачи сопряженного теплообмена (в том числе многофазные с фазовыми превращениями и с реагирующими потоками);
- связанные задачи теплогидродинамики и пространственной нейтронной кинетики.

Большинство из этих классов задач предполагает либо использование нескольких расчетных областей, либо системы уравнений, в которых разделение различных сред описывается разрывом свойств. Это приводит к необходимости использования сложных и по возможности максимально устойчивых численных методов. Сегодня развитие этого направления идёт двумя путями:

- домашние (т. н. in-hose) коды, основным достоинством которых является использование устойчивых численных схем и высокопроизводительных алгоритмов. Эти преимущества зачастую достигаются в ущерб общности постановки задачи (преднамеренно сужается класс решаемых задач) и за счет ограничения использования разработанного программного обеспечения (при освоении требуется присутствие самого разработчика, зачастую отсутствует развитая пользовательская документация и описание исходного кода, программы пишутся для конкретного аппаратного и системного обеспечения).
- коммерческие коды, которые почти всегда позиционируются как многоцелевые (General Purpose) и принципиально должны быть способны решать любые задачи. При таком подходе разработчики зачастую отказываются от монолитных моделей и прибегают к решениям с разделением расчетных областей, что отрицательно сказывается на устойчивости численных схем и производительности алгоритмов (особенно при параллельных вычислениях). Вторым важным минусом является высокая стоимость программного обеспечения [1], которая при использовании большого числа ядер (что является скорее правилом, чем исключением при выполнении промышленных расчетов) вполне может сравняться не только со стоимостью оборудования, но и вполне может стать сопоставимой с ценой ***полномасштабного эксперимента***.

В данной работе авторами предпринимается попытка разработки программного обеспечения для решения задач гидроупругости несколькими способами. Были рассмотрены существующие способы решения таких задач, равно как и их классификация, которая неотъемлемо связана с численным схемами, применяемых в таких задачах. Задачи делятся на сильносвязанные и

слабосвязанные. Первые отличаются от вторых сильным влиянием одной области на другую и интенсивным обменом импульса. Из возможных методов сопряжения слабосвязанных и сильносвязанных в работе выбраны итерационные методы, как наиболее простые для реализации и предоставляющие широкие возможности масштабирования программного обеспечения. В качестве основных модулей было выбрано открытое программное обеспечение, которое сегодня завоевывает всё большую популярность в мировой научной среде [2].

1. Разработанная численная модель

Рассматривалось связывание гидродинамического модуля OpenFOAM[3,4] (движение турбулентной (RAS и LES) несжимаемой, слабосжимаемой и сжимаемой жидкостей) и двух моделей динамики конструкции:

- реализованной в коде OpenFOAM на основе метода конечных объёмов [5];
- разложение по собственным формам и частотам [6].

В результате был разработан программный комплекс MCF для решения связных задач средствами свободного программного обеспечения, который позволяет связывать любые коды с открытым исходным кодом без изменения основных алгоритмов. Среди рассмотренных схем связывания были реализованы следующие:

- итерационная для слабосвязанных задач;
- итерационная с релаксацией решения динамики конструкции;
- итерационная с релаксацией решения динамики конструкции и адаптивным коэффициентом релаксации (метод Эйткена).

Для решения были отобраны следующие задачи:

- течение в каверне с эластичным дном и подвижной крышкой [7];
- колебания жгута в спутном следе за цилиндром [8];
- колебания диска на тонкой ножке в радиальном зазоре 230мкм [1];
- автоколебания в поворотно-золотниковом клапане [9].

Математическая модель гидродинамики включала в себя уравнения: неразрывности (1), сохранения количества движения (2), сохранения энергии (3). Тензор напряжений представлен в линейной форме и с учетом использования модели широкомасштабных вихрей выглядит следующим образом (4), (5). Для разрешения подсеточного масштаба использовалось уравнение транспорта кинетической энергии (6) и соотношение для вычисления подсеточной вязкости (7).

$$\frac{\partial c}{\partial t} + \nabla \cdot (cU) = 0 \tag{1}$$

$$\frac{\partial cU}{\partial t} + \nabla \cdot (cUU - y) = -\nabla p \tag{2}$$

$$\frac{\partial ch}{\partial t} + \nabla \cdot \left(cUh - б^{Eff} \nabla h\right) = \frac{Dp}{Dt} \tag{3}$$

$$y = м^{Eff} \left[\nabla U + (\nabla U)^T\right] - \frac{2}{3} м^{Eff} \nabla \cdot U \tag{4}$$

$$м^{Eff} = м^{SGS} + м^{lam} \tag{5}$$

$$\frac{\partial ck}{\partial t} + \nabla \cdot \left(cUk - D_k{}^{Eff} \nabla k\right) = G - \frac{2}{3} c \nabla \cdot Uk - \frac{c_e ck^{3/2}}{Д} \tag{6}$$

$$м^{SGS} = c_k c \sqrt{k} Д \tag{7}$$

Как уже упоминалось выше, механическая модель была представлена в двух видах:

- метод конечных объёмов, уравнение (8);
- методом конечных элементов (с использованием разложения по собственным формам и частотам), уравнение (9).

$$\frac{\partial^2 (cD)}{\partial t^2} - \nabla \cdot \left(м \nabla D + м (\nabla D)^T + л tr(e) I\right) = cf \tag{8}$$

$$M\ddot{q} + C\dot{q} + Kq = Q \tag{9}$$

В общем случае систему уравнений, описывающую взаимодействие структуры и жидкости можно представить в следующем виде:

$$Y = F(X) \tag{10}$$

$$X = S(Y) \tag{11}$$

Где $\boldsymbol{F}$ и $\boldsymbol{S}$ условно обозначают систему уравнений жидкости и структуры соответственно. В (10,11) $\boldsymbol{Y}$ — вектор нагрузок на общей поверхности взаимодействия (также называется смоченной), а $\boldsymbol{X}$ — вектор её перемещений. Систему уравнений (10, 11) необходимо дополнить соотношениями баланса на границе взаимодействия жидкость-структура:

$$\begin{gathered} D_{\Gamma_F} = D_{\Gamma_S} \\ t_{\Gamma_S} = \left(pn - 2\mu\nabla \cdot \frac{1}{2}\left(\nabla U + (\nabla U)^T\right) \right)_{\Gamma_F} \end{gathered} \tag{12}$$

Также (10, 11) представляют в виде, удобном для поиска решения:

$$\begin{aligned} r_F(X,Y) &= Y - F(X) = 0 \\ r_S(X,Y) &= X - S(Y) = 0 \end{aligned} \tag{13}$$

Способ решения такой системы варьируется в зависимости от класса самой задачи (сильно или слабо связанная) и выбранного метода связывания. Условно сильно связанная задача отличается от слабосвязанной тем, насколько велико взаимодействие (обмен импульсом) между жидкостью и потоком. Обычно сильносвязанные задачи характеризуются малой жесткостью конструкции, плотностью жидкости близкой или большей чем плотность структуры, несжимаемым поведением самой жидкости (бесконечная скорость звука).

Для связывания в слабосопряженных задачах использовалась смещенная схема интегрирования (см. рисунок 1). Для сильно связанных задач использовалась сильносвязанная схема с постоянным и адаптивным коэффициентами релаксации, см. рисунок 2).

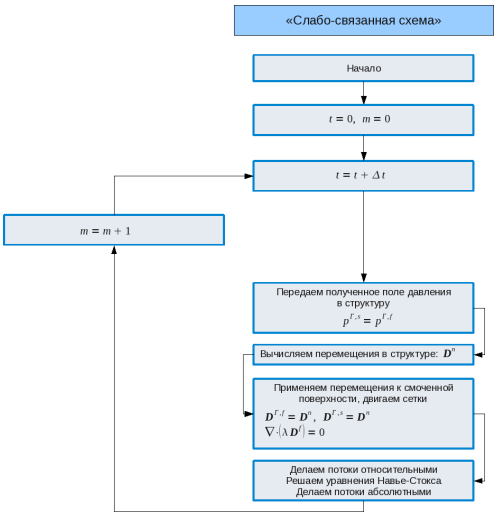


Рис. 1. Смещенная схема алгоритма интегрирования для слабосвязанных задач.

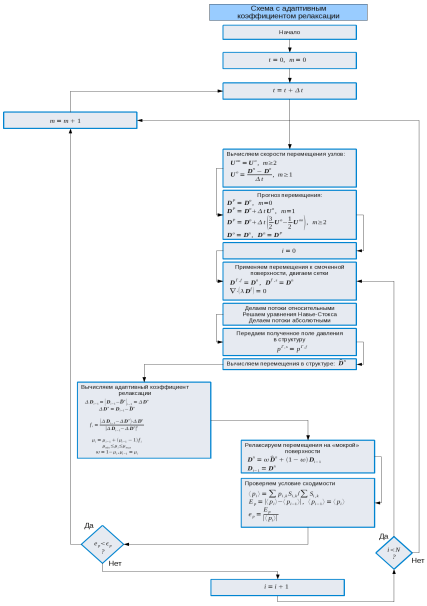


Рис. 2. Схема «сильного» связывания с использование техники релаксации с адаптивным коэффициентом (метод Эйткена)

Как в случае с сильносвязанными задачами, так и со слабосвязанными необходимо было организовать сбор данных при передаче распределений давлений от жидкости конструкции и перемещений границы проточной части от структуры. Для этого использовалась иерархическая модель обмена данными — один вычислительный узел (с номером 0) выбирается ведущим (или управляющим), а все остальные — ведомыми. Ведущий узел играет активную роль, опрашивая ведомые, последние являются пассивными и выполняют команды управляющего процессора. Сбор данных с процессоров (полей давления, площадей и нормалей граней) осуществлялся по схеме,

представленной на рисунке 3 (при взаимодействии OpenFOAM и UZOR) и включал в себя следующие этапы:

- Подготовительный этап. Выполняется один раз при инициализации программы — производится сбор информации о топологии вычислительной сети и разбиении «смоченной» поверхности по узлам кластера (количество граней поверхности, приходящееся на каждый вычислительный узел), аккумулирование информации на ведущем узле (с номером 0) и последующая раздача всех собранных данных остальным узлам.
- Регулярный этап (получение поля нагрузок на грани поверхности взаимодействия):
  - На каждом процессоре, включая ведущий, вычисляются давления, площади и нормали смоченной границы и сохраняются в специальный массив.
  - На всех процессорах, исключая главный и те, которые не содержат в памяти ни одной грани смоченной поверхности, выполняется команда посылки данных ведущему узлу.
  - Ведущий узел производит опрос всех остальных (исключая те, которые не содержат данных о поверхности обмена взаимодействия), собранные данные добавляются в конец единого массива, вычисляются усилия и передаются в модуль расчета структуры.

Раздача данных (передача перемещений границы проточной части от UZOR к OpenFOAM) осуществлялась схожим образом (см. рисунок 4).

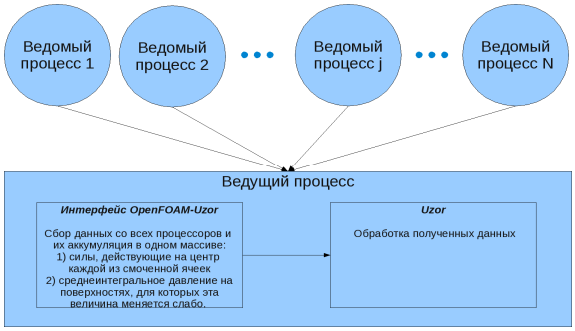


Рис. 3. Схема сбора данных гидродинамической моделей с вычислительных узлов СуперЭВМ

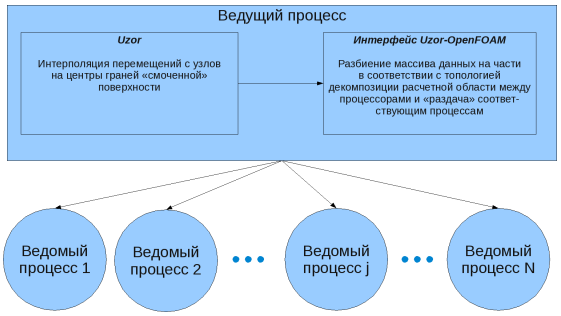


Рис. 4. Схема «раздачи» информации о перемещениях граней «смоченной» поверхности по узлам СуперЭВМ

Расчеты проводились на следующих кластерах:

- Кластер ЧелГУ (12x12 ядер Intel(R) Xeon(R) CPU X5650 @2.67GHz);
- ТТИ ЮФО (программа Университетский Кластер, 128x16 ядер AMD Opteron™ 8356 @2.67GHz).

На первой машине выполнялись расчеты только на 12 ядрах одного узла (по причине невозможности настроить OpenMPI для работы с OpenFOAM). На второй — количество ядер варьировалось от 32 до 256. При расчетах на машине ТТИ ЮФО на 256 ядрах было получено, что коэффициент масштабирования не опускался ниже 30%, для расчетов использовались как HP MPI, так и OpenMPI реализации MPI.

2. Результаты моделирования бенчмарк тестов

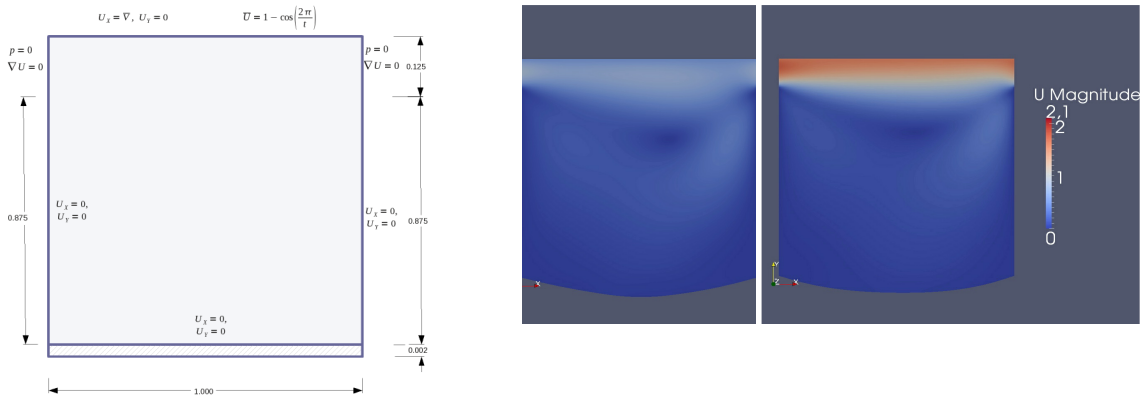
Рассмотрение задач подобного рода позволяет оценить на относительно простых примерах степень пригодности предложенной методики для моделирования мультифизики. Всего рассматривалось две тестовых задачи — течение в каверне с эластичным дном и колебания эластичного жгута в спутном следе за плохо обтекаемым телом.

2.1 Каверна с эластичным дном

Эта задача рассматривается как базовая во многих работах по взаимносвязанным задачам (например, [7]) и является модификацией широко известного теста «каверна», предназначенного для изучения адекватности моделирования кодом течений, вызванных движением стенки. В данном случае отличия от стандартной постановки заключаются в следующем (см. рисунок 5а): 1) скорость движущейся верхней стенки не постоянная, а меняется по периодическому закону от 0 до 2

$\overline{U} = 1 - \cos\left(\frac{2p}{t}\right)$; 2) для того, чтобы объём мог изменяться, вблизи верхней стенки жидкость может покидать и входить в расчетную область; 3) нижняя стенка представляет собой гибкую мембрану, которая может двигаться под действием потока и сопротивляться ему.

Параметры сред следующие: плотность среды — 1 кг/м^3, вязкость — 0.01 м^2/с; плотность структуры — 500 кг/м^3, модуль Юнга — 250Па, коэффициент Пуассона — 0.0. Моделируемое время — 70 секунд. Расчетное поле модуля скорости и перемещений нижней стенки предварительны на рисунке 5б.



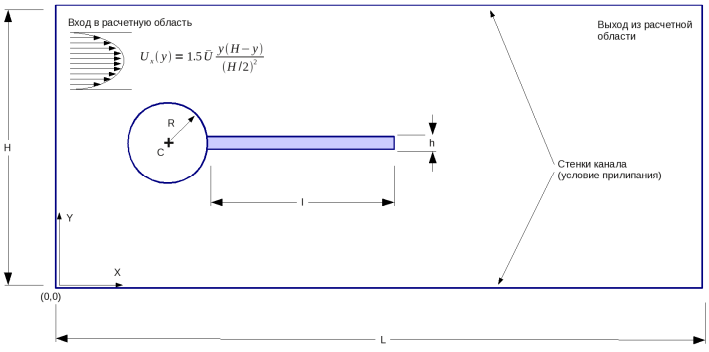
а) б)

Рис. 5. Принципиальная схема задачи (а) и поле модуля скорости в различные моменты времени (б)

2.2 Колебания эластичного бруса в спутном следе за цилиндром

Схематическое изображение расчетной области представлено на рисунке 6а. Область представляет собой прямоугольный канал (задача двумерная), на входе в который задается некоторый профиль скорости, геометрические характеристики расчетной области следующие: а) длина **L** равна 2.5м, высота **H** — 0.41м; б) центр круга (ось цилиндра) расположен в точке **C** с координатам (0.2, 0.2), радиус **R** — 0.05м; в) эластичная балка длиной **l** = 0.35м и высотой **h**=0.02м жестко закреплена на цилиндре, а правый нижний угол имеет координаты (0.6, 0.19); г) контрольная точка, в которой производилось сравнение перемещений расположена на правом конце балки и по середине её высоты — координаты (0.6, 0.2). Поле модуля и смещение балки представлено на рисунке 6б.

а)



б)

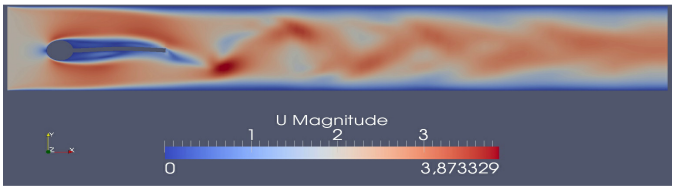


Рис. 6. Принципиальная схема тестовой задачи (а) и поле модуля скорости в режиме развитого течения (б)

Всего рассматривалось два варианта — стационарный при числе Re=20 (FSI1) и Re = 200 (FSI3). Физические параметры жидкой и эластичной сред для этих вариантов сведены в таблицу 1. В первом случае возникновения автоколебаний не наблюдалось и смещения носили стационарный характер, определяемый величиной статичной нагрузки от поля давления. Во втором случае происходило периодическое движение эластичной балки за счет нестационарного поля давления, обусловленного срывом вихрей за обтекаемым цилиндром.

Результаты расчетов сравнивались с данными работы [8]. Получено хорошее качественное и количественное совпадение.

Таблица 1 Физические параметры задачи из [8]

Наименование параметра	Режим FSI1	Режим FSI3
Число Re	20	200
Структура		
$c^S, кг/м^3$	1000	1000
$н^S$	0.4	0.4
$м^S, кг/(мс^2)$	$0.5\ 10^6$	$2.0\ 10^6$
Жидкость		
$c^F, кг/м^3$	1000	1000
$н^F, м^2/с$	0.001	0.001
$\overline{U}, м/с$	0.2	2.0

3. Результаты моделирования реальных геометрий

После тестирования было выполнено моделирование двух реальных геометрий — имитация течения в радиальном зазоре питательного насоса [1] и течение перегретого пара в поворотно-золотниковом клапане по радиальной щели [9]. Эти расчеты позволяют продемонстрировать применимость программного комплекса при решении задач оптимизации конструкций промышленного класса.

3.1 Моделирование течения питательной воды в тонкой щели разгрузочного устройства питательного насоса

Геометрия и расчетная область задачи представлены на рисунке 7. Слабосжимаемая среда (вода при комнатной температуре) поступает по кольцевому каналу снизу (течет вдоль направления oZ, обозначено коричневым цветом на рисунке 7а), затем поворачивает на 90^0 (под желтым диском, рисунок 7а) и попадает в кольцевой зазор толщиной 0.23мм, по которому течет в радиальном направлении. На выходе из зазора поток расширяется, дважды поворачивает и следует к выходу из расчетной области вдоль оси oZ (цилиндрический канал над диском).

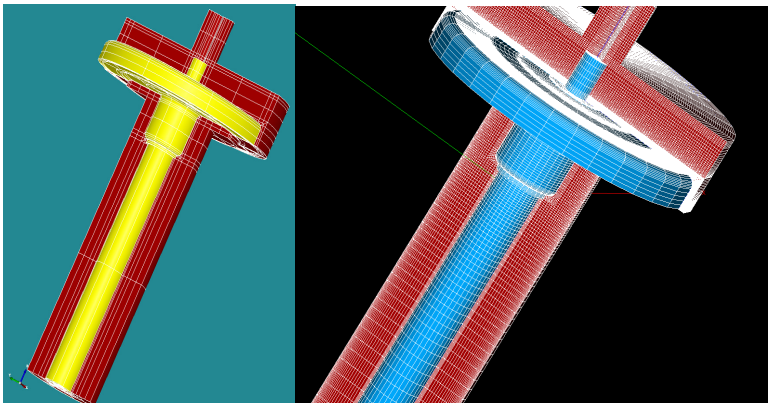
Расчет велся по двум моделям: в одном случае для дискретизации уравнений как гидродинамики, так и механики конструкции использовался метод конечных объёмов (OpenFOAM). В другом случае для получения решения задачи нестационарных колебаний механической системы использовался метод разложения по собственным формам (ПК UZOR [6]). Полный список исследовавшихся режимов представлен в таблице 2. В каждом режиме расчет велся для по крайней мере 1 секунды физического времени либо до получения условно стационарного состояния. Стационарным предполагалось состояние, при котором амплитуда колебаний равна нулю или асимптотически приближается к нему, либо постоянна и не меняется во времени. Амплитуда вычислялась как корень из суммы квадратов смещений по каждому из направлений (X, Y, Z). При решении задачи на входе и выходе расчетной области проточной части задавалось статическое

давление (например, 6атм и 0.1атм), скорость среды рассчитывалась из решения уравнения для давления. Для конструкции граничные условия на поверхности задавались согласно соотношению (12), нижний торец считался консольно закреплённым.

На расчет одного режима уходило (в зависимости от перепада) от 3 до 9 суток (на 12 ядрах). Анализ производительности вычислительной системы дан в разделе 4.

В результате расчетов №№1, 2 и 3 были получены затухающие колебания конструкции с частотой порядка 150Гц, вызванные движением упругого стержня в осевом направлении. По прохождении определенного количества времени колебания пропадали, а поле смещений в конструкции полностью определялось полем давлений в жидкости. Предположительно, причинами затухания являются:

- принципиальная невозможность использования метода конечных объёмов для решения задач *динамики* конструкции; метод расщепления переменных, предложенный в [5] вносит слишком сильное демпфирование в решение, что делает его непригодным для учета динамики, хотя в статике решение будет стремится к аналитическому по мере сгущения сетки;
- некорректность задания начальных граничных условий — в такого рода задачах обычно предполагается, что в момент времени t=0c, конструкция находится в уже предварительно напряженном состоянии, а течение соответствует развитому.



а) б)

Рис. 7. Геометрия (слева, а) и расчетная сетка (справа, б) моделируемого стенда. Коричневым цветом обозначена проточная часть, а желтым (слева) и синим (справа) — конструкция.

Таблица 2. Описание списка исследуемых режимов с

№№ режима	Описание
Расчеты по модели OpenFOAM-OpenFOAM	
1	Базовая тестовая задача. Выбраны три точки для отбора полей давления и перемещений: а) на нижней плоскости диска до зазора; б)на нижней плоскости диска и на максимальном радиусе; в) на ножке стенда. Грубая расчетная сетка включала в себя 90 тысяч контрольных объёмов для проточной части и 150 тысяч контрольных объёмов для механической части. Использовалась RAS-модель турбулентности, были получены затухающие продольные колебания с частотой ~120-140Гц. Давление перед зазором 6.1бар, после — 0.1бар.
2	Аналог случая №1, но с оптимизированной (сгущенной) сеткой проточной части (300 тысяч контрольных объёмов), для дискретизации адвективных слагаемых использовалась смешанная схема. Результаты по сравнению с п.1 не изменились.
3	Аналогично случаю №2, но с использованием LES-модели турбулентности в проточной части задачи. Результаты по сравнению с п.2 не изменились.

Таблица 2. Описание списка исследуемых режимов с

№№ режима	Описание
Расчеты по модели OpenFOAM-UZOR	
4	Всё как в №3, но с моделью UZOR в механической части. По техническим причинам (из-за завышенного критерия интерполяции), было не полное соответствие между точками проточной и механической частей. Были получены автоколебания с частотой 100Гц, перемещениями, соответствующими собственной форме №1 и амплитудой порядка 50мкм.
5	Аналогично №4, но с давлением перед зазором 1.5бар. Получены затухающие колебания конструкции по первой собственной форме с частотой 75Гц.
6	Аналогично №4, но с давлением на входе в щель 20бар. Обнаружены автоколебания конструкции по первой собственной форме с частотой 134Гц и амплитудой 415мкм.
7	Как и в №4, но с заниженным критерием поиска соответствий между узлами гидродинамической и механической частей на смоченной границе (для установления полной связи). Результат количественно и качественно полностью идентичен случаю №4.
8	Как и в №5, но с заниженным критерием поиска соответствий между узлами гидродинамической и механической частей на смоченной границе (для установления полной связи). Результат количественно и качественно полностью идентичен случаю №5.
9	Как в №7, но с $k-ų SST$ моделью турбулентности (вместо LES). В результате получены слабозатухающие колебания с амплитудой, в 2 раза меньше, чем в №7.
10	Как в №9, но с $k-e$ моделью турбулентности. Для данной расчетной сетки не удалось получить устойчивое решение — численная схема после нескольких десятков итераций расходилась.
11	Аналогично случаю №7, но с частотой вывода проб поля в 4 раза выше для более точной оценки частоты колебаний. Результаты полностью идентичны случаю №7.
12	Как в серии №10, но с $realizable k-e$ моделью турбулентности. Получены слабозатухающие колебания с хаотичным изменением амплитуды.
13	Как в режиме №9, но без моделирования турбулентности (ламинарная модель). Возникают незатухающие колебания с амплитудой порядка 20мкм.

По результатам расчетов режимов №№ 4-13 (связка OpenFOAM-UZOR) можно сделать следующие выводы:

1. получено качественно хорошее совпадение с экспериментом:
   a) с повышением перепада давления на щели наблюдается переход от затухающих колебаний к установившемся автоколебаниям (см. рисунки 8, 9, 10);
   b) увеличение перепада давления на тонкой щели приводит к росту собственной частоты системы как и в эксперименте, на ~30% (см. рисунок 11);
   c) амплитуда колебаний меняется вместе с перепадом (чем больше перепад, тем выше амплитуда).
2. Были обнаружены и существенные количественные отличия от эксперимента
   a) расчетная амплитуда в режиме с перепадом 6атм в 2 раза меньше экспериментальной;
   b) в предложенной постановке оказалось невозможным учесть эффект виртуальной массы.

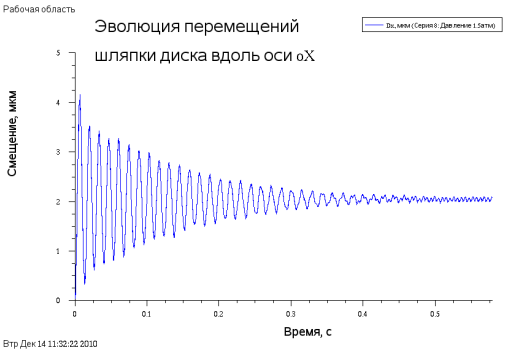


Рис. 8. История перемещений диска при перепаде 1.5атм (вариант расчета №5)

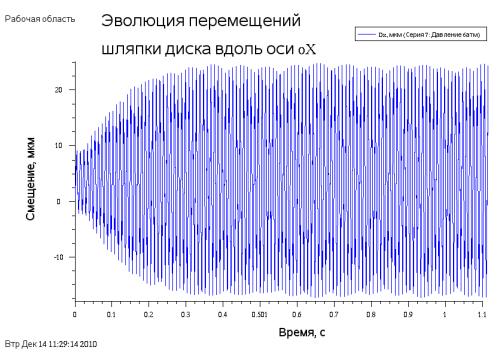


Рис. 9. История перемещений диска при перепаде 6атм (вариант расчета №4)

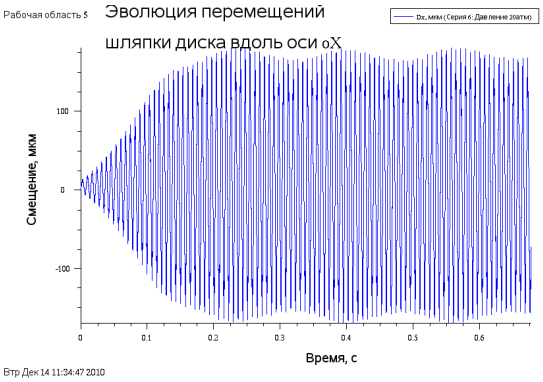


Рис. 10. История перемещений диска при перепаде 20атм (вариант расчета №6)

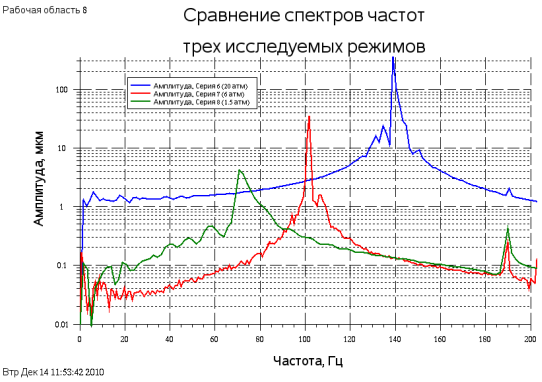


Рис. 11. Спектры частот колебаний диска при перепадах 1.5атм, 6атм и 20атм.

3.2 Моделирование течения перегретого пара в поворотно-золотниковом клапане

В данной задаче рассматривается течение в регулирующем клапане [9], при котором возможно возникновение автоколебаний в определенных режимах. Конструкция и и проточная часть представлены на рисунках 12а,б. Конструктивно клапан такого типа состоит из обязательных трех частей: 1) корпуса, включающего в себя входную и выходные камеры, в которых собирается дросселируемый поток до и после клапана; 2) статичной гильзы — со входными и выходными отверстиями и 3) вращающегося золотника, вставленного в гильзу, с окнами для регулирования расхода.

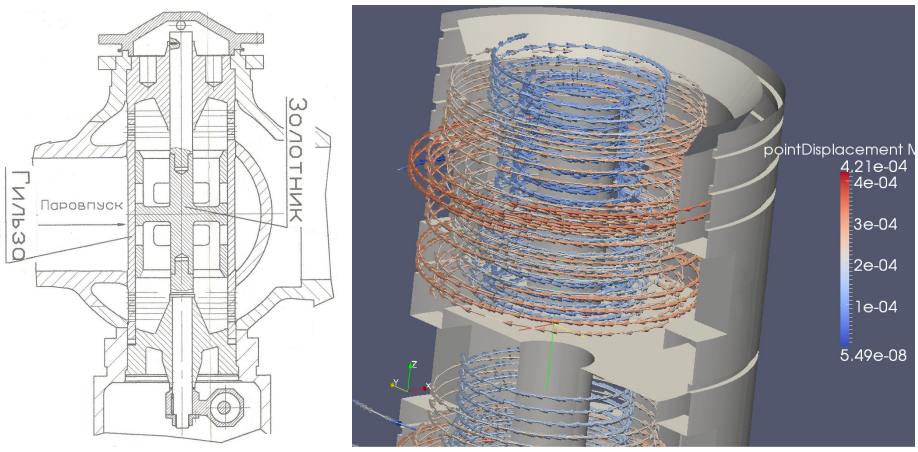
В золотнике сделаны отверстия, находящиеся на одной высотной отметке со входными окнами гильзы. Между гильзой и золотником — кольцевой зазор малой толщины (порядка 150мкм), обеспечивающий протечки теплоносителя при нулевом открытии клапана. Поток, поступая из входной камеры в окна гильза разбивается на три части: а) идущую напрямую к выходным отверстиям гильзы по зазору; б) идущую к окнам золотника по кольцевому зазору; в) и наконец, в случае положительного перекрытия между окнами золотника и гильзы, идущую по золотнику к выходным окнам гильзы.

В определенном смысле эта конструкция напоминает уже рассмотренную в разделе 3.1, но но при этом есть и следующие принципиальные отличия — «мягкая» часть — золотник закреплен жестко с двух сторон, а среда может течь как вверх по цилиндрическому зазору, так и вдоль штока золотника. При такой организации течения потоком в окнах золотника создается гидродинамическая сила, действующая на закрытие клапана и потенциально могущая привести к возникновению автоколебательных процессов.

Проведенные расчеты позволили выявить не только возникновение крутящих и поступательных усилий, вызываемых гидродинамикой потока, но и ряд других эффектов, приводящих к

неудовлетворительному виброакустическому состоянию при эксплуатации арматуры данного типа. Это позволило выявить и указать 6 возможных причин возникновения резонанса:

- Причина №1 – в золотнике действительно обнаружены сложные вихревые течения, которые вместе с высокими скоростями потока могут приводить к возникновению автоколебаний.
- Причина №2 – возникновение крутящего момента, обусловленного гидродинамической силой показано во всех расчетах, и при определенных условиях (эксцентриситет в зазоре, неравномерность или скошенность конструкции) именно вращательные колебания могут быть спусковым крючком для резонансного процесса.
- Причина №3 – наложение на турбулентное течение сверхзвуковых эффектов и взаимодействия распространяющихся волн давления вполне может являться причиной возникновения резонанса.
- Причина №4 – в случае нарушения симметричности конструкции (а это всегда присутствует в реальных изделиях) азимутальные формы колебаний могут приводить к другим формам, вызывающим дальнейшее смещение центра тяжести конструкции и способствующие развитию автоколебательного процесса.
- Причина №5 – контактное взаимодействие элементов конструкции часто приводит к возникновению дискретных составляющих в спектре шума с частотами, кратными некоторому базовому значению; косвенно это предположение может быть подтверждено высокочастотными продольными колебаниями штока золотника (1кГц), которые могут приводить к его биению о втулку гильзы.
- Причина №6 – во всех расчетах было получено существенное снижение температуры теплоносителя в зазоре и золотнике (ниже Ts) а поскольку модель среды однофазная, то эффект конденсации не может быть учтен в расчетах явно, но о его наличии можно судить по локальным температурам в проточной части.



а) б)

Рис. 12. Принципиальная схема конструкции регулирующего клапана (а) и разрез его проточной части с траекториями движения поверхности золотника (б)

4. Результаты оценки производительности вычислительных ресурсов

Анализ производительности выполнялся как для тестовых задач, так и для реальных геометрий (см. рисунки 13,14). В последнем случае отдельное внимание уделялось влиянию затратам на вычисления отдельных блоков программы (см. рисунок 14).

Анализ масштабируемости, выполненный для кластера ЧелГУ (рисунок 13) показал, что коэффициент масштабирования $\left(M = \frac{T_{посл}}{NT_{парал}} \right)$ составляет величину порядка 50%, что является хорошим результатом, поскольку на каждое ядро приходилось по 25 000 контрольных объёмов (300 000 к.о. на всю расчетную область), а эта величина (соотношение числа неизвестных на

вычислительную единицу) является пороговой при параллельных вычислениях.

На кластере ТТИ ЮФО было получено, что даже при использовании свыше 128 вычислительных процессоров (для разрешения проточной части использовалось более 2млн. контрольных объёмов) коэффициент масштабируемости не падал ниже 0.3, что является хорошим показателем. При этом основные затраты на расчет в пределах одной итерации (см. рисунок 14) складываются из:

1. сбора локальных давлений, распределенных по ядрам ЭВМ, на границе сшивки областей гильза - проточная часть;
2. сбора локальных давлений, распределенных по ядрам ЭВМ, на границе сшивки областей золотник - проточная часть;
3. решения уравнений движения для гильзы (метод конечных элементов, МКЭ);
4. решения уравнений движения для золотника (МКЭ);
5. передачи перемещений, полученных в результате движения гильзы в расчетную область проточной части;
6. передачи перемещений, полученных в результате движения золотника в расчетную область проточной части;
7. решения уравнений движения расчетной области проточной части;
8. решения уравнений Навье-Стокса;
9. сохранения промежуточных анализов (перемещения в отдельных точках).

Как следует из приведенного графика, основные затраты уходят на а) решение уравнений движения механики; б) решение уравнения смещения точек расчетной области; в) решение уравнений Навье-Стокса (уравнение для давления). И, если в первом случае есть достаточное пространство для тактической оптимизации (распараллеливание), то в последних двух необходимы: 1) оптимизация процессов выделения и очистки памяти под массивы данных; 2) минимизация межпроцессного обмена данными при решении СЛАУ.

В численном эксперименте движение жидкой среды считалось установившимся, если поток прошел через расчетную область хотя бы один раз (а лучше — несколько, например 10) со скоростью, равной скорости жидкости на входе. Для характерного тестового режима физическое время, за которое поток полностью проходит один раз через расчетную область составляет 0.8-0.9с. При шаге физического времени 10^{-6}с требуется 800 000-900 000 итераций, на каждую из которых затрачивается ~1210мс процессорного времени. Получаем, что для набора удовлетворительной статистики необходимо считать как минимум 10 дней. То же самое можно сказать и относительно других режимов, отметив лишь, что в некоторых из них масштаб времени меньше в 10 раз (поскольку скорости выше в 10 раз) по сравнению с базовым вариантом.

На расчеты 20 режимов регулирующего клапана ушло примерно 3 месяца процессорного времени. Оценочные подсчеты показывают, что при выполнении этих же расчетов на обыкновенных ПЭВМ с четырьмя процессорами потребовалось бы **более 8 лет.**

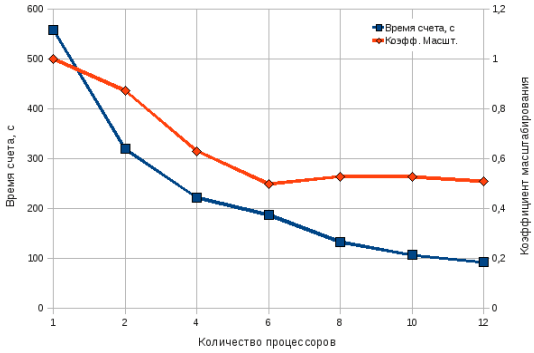


Рис. 13. Анализ масштабируемости кластера ЮУрГУ

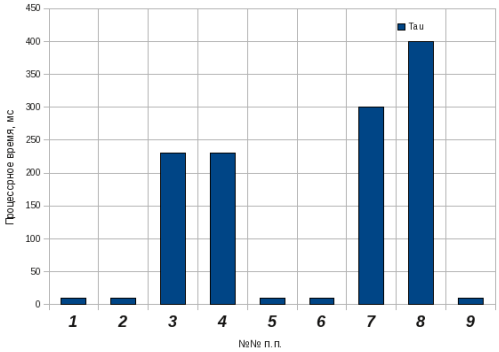


Рис. 14. Затраты процессорного времени на выполнение одного шага интегрирования связной задачи с одной проточной частью и двумя механическими. 128 вычислительных ядер, 2 миллиона контрольных объёмов в проточной части.

5.Выводы

1. Рассмотрены основные различия между открытыми кодами, программами собственной разработки (in-house) и коммерческими платформами общего назначения (General Purpose Codes). Получено, что различия в стратегии создания программного обеспечения определяют классы решаемых задач.

2. В соответствии с проведенным анализом программного обеспечения, сделан выбор в пользу свободных продуктов и их взаимной интеграции. Рассмотрены основные методы сопряжения задач мультифизики, приведены базовые уравнения, определяющие соотношения в этой области.

3. Разработан программный комплекс для решения связных задач теплогидродинамики и механики конструкции, позволяющий сопрягать модели (заложенные в потенциально любых открытых программных сред) друг с другом как методом сильного, так и слабого связывания.

4. Возможности разработанного программного комплекса продемонстрированы на тестовых задачах-бенчмарках и реальных конструкциях тяжелой промышленности. В результате тестирования получено хорошее качественное совпадение с экспериментом, что является основой для принятия решения об адекватности численной модели.

5. Анализ производительности аппаратного обеспечения, использовавшегося при решении рассмотренных выше задач показал удовлетворительную масштабируемость программного комплекса и свидетельствует о потенциально возможных путях ускорения работы программы за счет дополнительных мер по оптимизации алгоритмов и исходного кода.

6. В целом, на основе материалов данной работы можно сказать о готовности современных открытых программных решений к задачам любого уровня сложности, в том числе и промышленного масштаба.

6. Список литературы

1. Васильев В.А., Ницкий А.Ю., Крапошин М.В., Юскин А.В. Исследование возможности параллельных вычислений задач гидроаэродинамики с использованием открытого пакета программ OpenFOAM на кластере «СКИФ Урал» ЮУрГУ // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта - 2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ, 2010. С. 422-430.

2. М.В. Крапошин, О.И. Самоваров, С.В. Стрижак «Опыт использования СПО для провидения расчетов пространственной гидродинамики промышленного масштаба» // Труды конференции "Свободное программное обеспечение — 2010", СПБГПУ, Санкт-Петербург, 2010

3. Open∇FOAM (version 1.6). The Open Source CFD Toolbox, User Guide, OpenCFD Ltd, London, 2009

4. Open∇FOAM (version 1.6). The Open Source CFD Toolbox, Programer Guide, OpenCFD Ltd, London, 2009

5. Jasak, H. and Weller, H.G.: “Application of the Finite Volume Method and Unstructured Meshes to Linear Elasticity”// Comp. Meth. Appl. Mech. Engineering, 2000

6. Киселев А-др.С., Киселев Ал-ей.С., Даничев В.В. Аннотация программы UZOR_1. //ВАНТ, серия: Физика ядерных реакторов, вып.1, 1999, стр.109-113.

7. Jesús Gerardo Valdés Vázquez, «Nonlinear Analysis of Orthotropic Membrane and Shell Structures Including Fluid-Structure Interaction», PhD Thesis // UNIVERSITAT POLITÈCNICA DE CATALUNYA, Barcelona, 2007

8. Stefan Turek, Jaroslav Hron, Mudassar Razzaq, Hilmar Wobker, and Michael Schafer «Numerical Benchmarking of Fluid-Structure Interaction: A comparison of different discretization and solution approaches» // Germany, 2010

9. А-др С. Киселёв, А-ей С. Киселёв, М.В. Крапошин, С.Е. Таршилов «Взаимосвязанные гидравлические и прочностные расчетные исследования конструкции БПК, направленные на улучшение его вибрационных характеристик» // Отчет РНЦ «Курчатовский Институт», Москва, 2009

Визуализация профиля работы программ с памятью[1]

Вад.В. Воеводин

НИВЦ МГУ имени М.В. Ломоносова

Для большинства задач из любой предметной области для получения их эффективной реализации необходимо проводить анализ работы с памятью. В данной работе проводится исследование профиля доступа к памяти на примере некоторых типовых задач. На основе данного исследования с помощью визуализации проводится анализ возможных свойств рассмотренных задач, а также выделяются некоторые общие закономерности. Приводятся примеры визуализации профилей работы с памятью, иллюстрирующие указанные свойства и особенности.

1. Введение

Для многих задач эффективность выполнения программ является очень важным фактором. Чем выше эффективность выбранного решения, тем быстрее программа будет выполнена, тем меньше времени будет потрачено на решение задачи в целом. В области суперкомпьютерных вычислений многие задачи требуют часы, дни, а иногда и недели для решения, поэтому даже незначительное увеличение эффективности может привести к существенному выигрышу в скорости работы программы.

Для повышения эффективности программ на сегодня разработано множество различных средств, однако они всегда нацелены на относительно узкую область: либо на определенный класс вычислительных платформ, либо на предопределенный класс задач, либо на оптимизацию конкретных свойств программ. Для решения проблем общего характера, например, подбора наиболее эффективной платформы для выбранной задачи или эффективной реализации задачи на другой платформе, данные средства плохо подходят. Нужен более общий подход, позволяющий проводить исследование эффективности всего процесса отображения задач на архитектуру вычислительных систем.

Для изучения процесса отображения предлагается на каждом из его этапов выделять набор характеристик, которые могут влиять на эффективность или же помогать в ее исследовании [1, 2]. Все характеристики делятся на три класса: характеристики алгоритма, программы и аппаратуры. Предполагается, что во время отображения задачи и алгоритм, и платформа уже определены, поэтому их характеристики фиксированы. По сути, эти характеристики описывают те рамки, в которых программа должна работать. Для получения эффективной реализации мы можем менять характеристики программы в соответствии с характеристиками алгоритма и аппаратуры. Примерами указанных характеристик из различных классов могут служить следующие параметры:

- Характеристики алгоритма: свойства графа алгоритма [3];
- Характеристики программы: свойства текста программы (структура вхождения циклов, способ индексации массивов и т.д.), коммуникационные свойства (объем передаваемых данных, структура передач, число нитей/процессов);
- Характеристики аппаратуры: строение подсистемы памяти, топология, латентность и пропускная способность сети.

Одним из самых главных факторов, влияющих на эффективность выполнения программы, является эффективность работы с памятью. Эта проблема известна и получила название «стены памяти»: все последние годы скорость работы процессоров растет значительно быстрее скорости работы памяти. Во всех современных вычислительных системах доступ к данным является

[1] Работа выполнена при поддержке грантов РФФИ № 09-07-00168-а и № 10-07-00586-а.

дорогостоящей операцией, поэтому неэффективное использование памяти может очень сильно сказываться на производительности всей выполняемой программы. Вместе с этим, подсистема памяти устроена очень сложно, и ее оптимальное использование является непростой задачей. Кэш-памяти разных уровней, оперативная память, TLB буфер, аппаратный префетчер – каждый из этих компонентов подсистемы памяти обладает целым набором аппаратных характеристик, которые необходимо учитывать для эффективной работы с памятью.

На данный момент разработаны различные программные средства и проводятся исследования, направленные на анализ и повышение эффективности работы с памятью, например, с помощью сбора информации об эффективности использования кэш-памяти [4] или эмуляции работы программы [5]. В данной работе акцент делается на визуальном анализе структуры доступа к памяти, который во многих случаях дает интересную информацию о свойствах программы, и главное – об эффективности ее выполнения. Стоит отметить, что активно визуализация используется и при исследовании других свойств программ, например, коммуникационного профиля параллельных программ [6].

2. Профиль работы с памятью и характеристики программ

Для исследования и анализа эффективности программ важно понимать структуру их профиля работы с памятью. Введем понятие “поток обращений”, под которым будем понимать последовательность обращений к используемым в программе переменным, записанную в том порядке, как это происходило при выполнении программы. Если задействован элемент массива или другой составной переменной, то в поток попадает именно этот элемент, а не вся составная переменная. Таким образом, в поток обращений могут входить скалярные переменные, составные переменные (если в некоторой операции адресуется такая переменная целиком) или отдельные элементы составных переменных.

Если для работы с элементами составной переменной используется некоторый итератор, то для каждого конкретного значения итератора в поток обращений записывается отдельное обращение. В случае итерирования массива в потоке обращений появятся отдельные элементы данного массива. Например, если в программе есть код:

```
for (i=0; i<5; i++) A[i]=0;
```

то поток обращений к массиву `A` будет выглядеть следующим образом: `A[0]`, `A[1]`, `A[2]`, `A[3]`,`A[4]`.

Для изучения профиля работы программ с памятью можно опираться на различные подходы. Один из возможных вариантов – это использование специальных характеристик программы, описывающих локальность используемых данных.

В работе [2] для описания пространственной и временно́й локальности данных программы вводится набор характеристик. Первые две характеристики описывают пространственную локальность, которая отражает среднее расстояние между несколькими последовательными обращениями к памяти. Характеристика *L* показывает среднее расстояние по памяти между всеми соседними элементами потока обращений. Это дает оценку, насколько близко в памяти расположен следующий элемент данных по отношению к текущему. Характеристика *SEQ_L* отражает среднюю длину последовательных элементов потока, являющихся непосредственными соседями в памяти. Данная характеристика позволяет оценить самый хороший случай для пространственной локальности, когда используемые данные следуют в памяти подряд, что дает хорошее использование кэш-памяти.

Для описания свойств временно́й локальности выделяются три характеристики. Временна́я локальность данных программы показывает среднее число обращений по одному адресу в памяти за время исполнения всей программы. Характеристика α равна среднему расстоянию (в терминах потока обращений) до следующего обращения к той же переменной, что показывает интенсивность использования данных. Степень влияния этой характеристики, а также отдельных переменных на эффективность работы программы в целом, позволяет оценить характеристика N_α, которая равна среднему числу обращений к переменным (элементам потока). Чем

больше обращений произведено к некоторой переменной, тем важнее с точки зрения эффективности работы с памятью интенсивность обращений к данной переменной. Степень повторного использования переменных программы показывает характеристика *P*, значение которой изменяется на отрезке [0, 1]. Характеристика *P* принимает крайние значения в следующих случаях: если ко всем переменным программы обращение происходит только один раз, то *P*=0, а если ко всем переменным обращение происходит как минимум дважды, то *P*=1.

Более подробное описание и примеры использования данных характеристик приведены в работе [2].

3. Визуализация профиля работы с памятью

Для изучения качественных характеристик профиля работы программ с памятью хорошие результаты дает его визуализация. Конечно, визуализация не дает возможности получить точные оценки или провести аккуратный детальный анализ, однако она позволяет понять общее строение профиля, как в целом устроена программа, что во многих случаях может дать подсказку пользователю о дальнейших действиях.

В данной работе мы будем рассматривать профили работы с памятью для некоторых типовых алгоритмических структур [1, 2]. Для каждого массива строятся отдельные профили. Все профили на рисунках устроены одинаково: по вертикали расположены линеаризованные индексы элементов массива (т.е. смещение элемента относительно начала массива), по горизонтали – номер обращения в потоке (чем больше этот номер, тем позже было произведено данное обращение в программе).

Стоит сразу отметить, что картинка никак не отражает время, затрачиваемое на обращение к различным компонентам подсистемы памяти (например, кэш-памяти или ОП), которое может сильно отличаться для разных обращений. Данное представление является машинно-независимым, и не всегда позволяет сделать точные выводы о работе реальной программы.

Также, поскольку число обращений в потоке часто весьма велико, то визуализация всего профиля может быть обманчива, и для точного анализа, вообще говоря, необходимо рассматривать отдельные, небольшие фрагменты профиля. Возьмем, например, профиль работы с памятью для вектора в задаче умножения разреженной матрицы на плотный вектор (рис. 1а). На первый взгляд, кажется, что данный профиль отражает полностью случайный характер обращения к элементам вектора, однако при увеличении масштаба видно, что обращения к вектору имеют закономерность (рис. 1б).

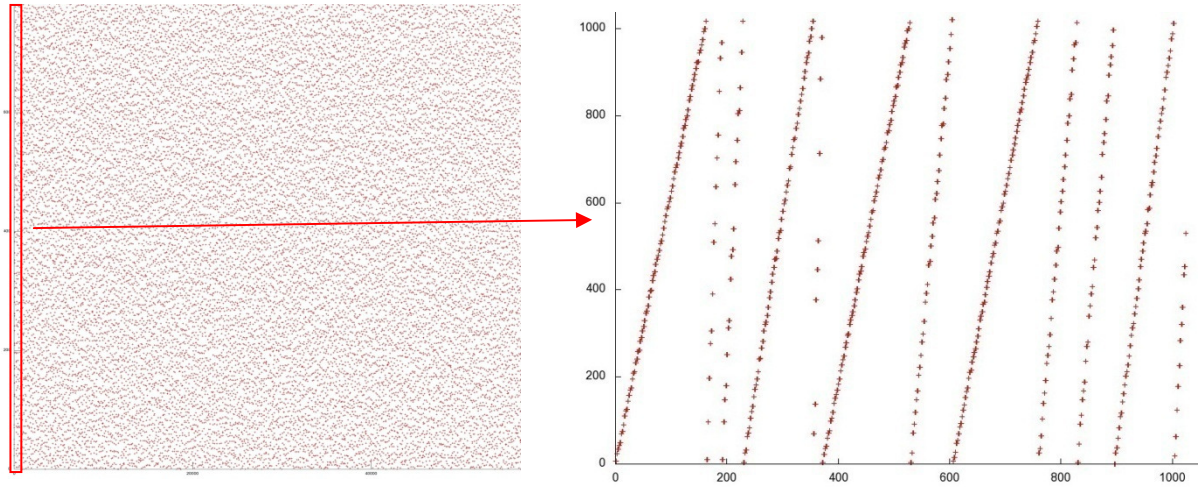


Рис. 1а. Умножение разреженной матрицы на вектор, общий профиль, > 100.000 обращений в память

Рис. 1б. Умножение разреженной матрицы на вектор, увеличенный фрагмент, 1024 обращений в память

Несмотря на то, что при рассмотрении общего профиля точный анализ недоступен, некоторые выводы касательно работы с памятью все же можно делать. Например, график на рис. 2а

позволяет предположить, что для любого элемента массива все обращения к нему расположены близко друг к другу, и при этом элементы массива перебираются последовательно, значит, массив обладает достаточно хорошей пространственной локальностью. На рис. 2б видно, что правая часть профиля портит общую локальность, поскольку обращения происходят ко всем элементам массива, причем порядок обращений похож на случайный. Вместе с этим, число обра-

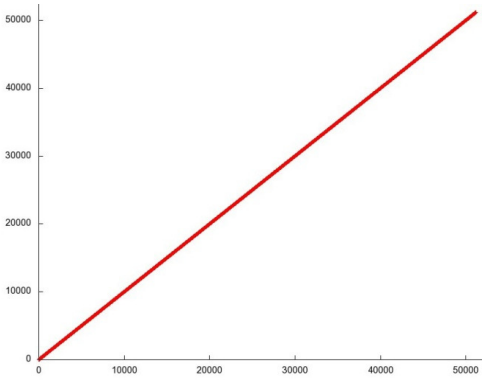


Рис. 2а. Пример профиля с хорошей пространственной локальностью

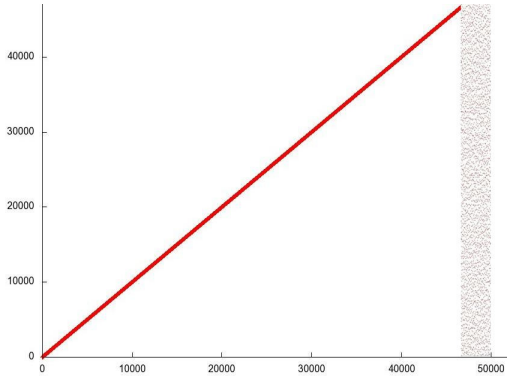


Рис. 2б. Измененный профиль, пространственная локальность ухудшается

щений в этой правой части значительно меньше, чем во всей остальной части, поэтому его влияние на общий профиль будет не так велико.

Рассмотрим более интересный пример – блочное перемножение плотных квадратных матриц (`N` – размерность матриц, `BLOCK` – размер блока):

```
for(i1=0; i1<N; i1+=BLOCK)
   for(j1=0; j1<N; j1+=BLOCK)
      for(k1=0; k1<N; k1+=BLOCK)
         for (i=i1; i<i1+BLOCK; i++)
            for (j=j1; j<j1+BLOCK; j++)
               for(k=k1; k<k1+BLOCK; k++)  {
                  C[i][j]+=A[i][k]*B[k][j];
               }
```

В случае при `N=128, BLOCK=16` профили работы с памятью для массивов `A`, `B` и `C` выглядят следующим образом (рис. 3а,3б,3в).

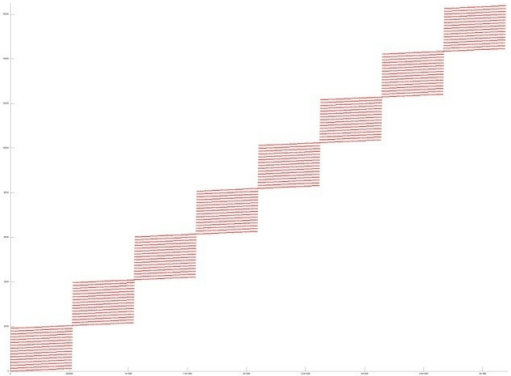


Рис. 3а. Профиль обращений в память к массиву `A`

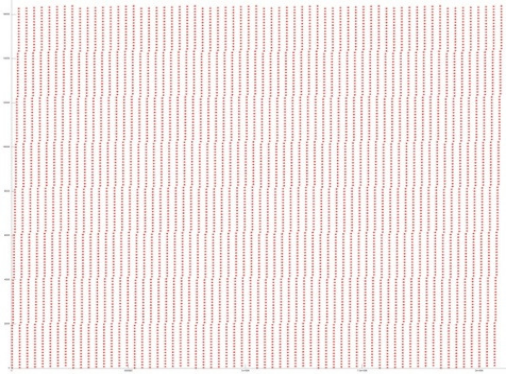


Рис. 3б. Профиль обращений в память к массиву `B`

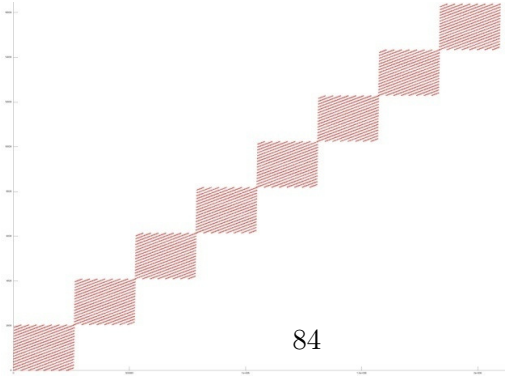


Рис. 3в. Профиль обращений в память к массиву `C`

В данном случае индекс из двумерного линеаризуется в одномерный, с учетом того, что матрица хранится по строкам. По вертикали, как обычно, указан индекс элемента массива, к которому происходит обращение, а по горизонтали – порядковый номер обращения. Данные графики показывают, что профили массивов `A` и `C` похожи – элементы массивов образуют 8 (128/16) блоков, и сначала происходят обращения только к первому блоку, затем – только ко второму и т.д. Это указывает на хорошую локальность обращений, поскольку все обращения к каждому элементу расположены близко друг к другу. В случае массива `B` ситуация иная: из профиля на рис.3б видно, что каждый элемент используется на протяжении всей программы и через большие промежутки времени, а значит, локальность в этом случае, скорее всего, хуже, чем в случае массивов `A` и `C`.

Рассмотрим для более точного анализа небольшие участки профиля. Более подробное изучение профилей массивов `A` и `C` на рис. 4а и рис. 4б позволяет заметить, что локальность данных лучше в случае массива `C`.

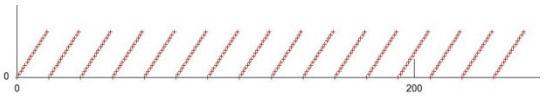


Рис. 4а. Профиль обращений к массиву `A` при выполнении двух внутренних циклов

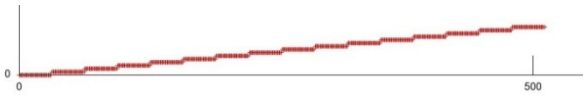


Рис. 4б. Профиль обращений к массиву `C` при выполнении двух внутренних циклов

Сравнивая профили на данных рисунках, полученных при выполнения двух внутренних циклов фрагмента перемножения матриц, видно, что все обращения к элементу массива `C` идут подряд, в то время как обращения к элементам массива `A` происходят с некоторым шагом, что приводит к менее эффективной работе с памятью.

Конечно же, все качественные заключения, полученные при изучении профилей работы с памятью, находят отражение и в соответствующих характеристиках программ. Рассмотрим значения описанных выше характеристик локальности использования данных, и сравним полученные результаты (табл. 1).

Таблица 1. Значения характеристик для задачи блочного перемножения матриц, `N=128`, `BLOCK=16`

Массив	L	SEQ_L	α	N_α	P
A	0.2706	16.0009	0.0006	128	1
B	0.0041	1	0.00007	128	1
C	0.681	512.438	0.0044	256	1

Сейчас не нужно внимательно анализировать абсолютные значения характеристик. Важно, что для каждой из характеристик верно следующее утверждение: чем больше значение характеристики, тем лучше (выше) локальность обращений к данным программы. Иными словами,

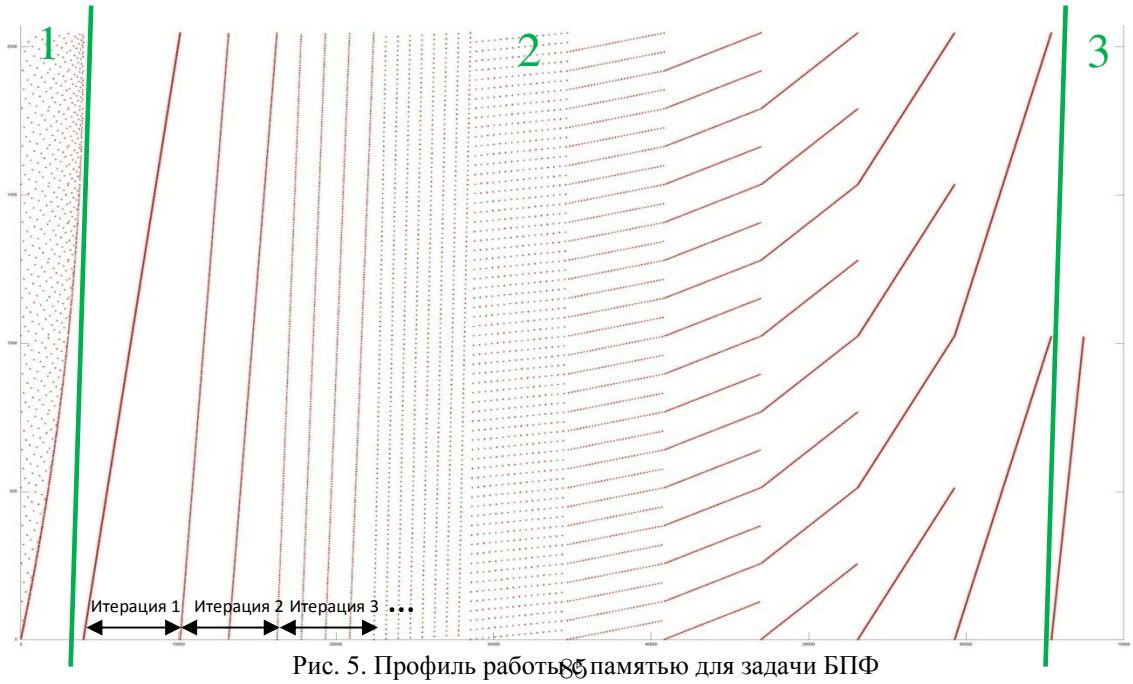


Рис. 5. Профиль работы с памятью для задачи БПФ

выводы, сделанные по профилям на рисунках, подтверждаются значениями характеристик.

Следует еще раз подчеркнуть возможную обманчивость первого восприятия всего профиля целиком. С этой точки зрения, изначальное предположение о том, что, согласно рисункам, работа с массивом `B` организована менее эффективно, чем работа с массивами `A` и `C`, могло оказаться и неверным. Например, программа могла быть устроена так, что к элементам массива `B` происходят последовательные обращения на протяжении многих итераций, в то время, как в каждом блоке обращений к массиву `A` профиль работы с памятью мог носить случайный характер. В такой ситуации локальность обращений к массиву `B` оказалась бы лучше, чем к массиву `A`. Это еще раз подчеркивает, что общая картина профиля не всегда адекватно отражает реальные свойства программы.

Нужно отметить и тот факт, что далеко не всегда обращения к памяти имеют “линейный” характер, как могло показаться из приведенных выше рисунков. Рассмотрим профиль работы с памятью для другой типовой алгоритмической структуры – быстрого преобразования Фурье (рис. 5). Для решения использовался простой нерекурсивный алгоритм. Несмотря на достаточно сложную структуру обращений к памяти, на данном графике легко можно увидеть 3 этапа алгоритма. На этапе 1 происходит начальное преобразование входного массива для последующего расчета. На этапе 2 выполняются основные вычисления, причем отчетливо видно увеличение шага по элементам массива в 2 раза на каждой последующей итерации. Этап 3 соответствует записи результатов в выходной массив.

В заключение рассмотрим еще один интересный пример: умножение разреженной матрицы размером `NxN` на вектор с диагональной схемой хранения. В такой схеме хранения матрицы `i`–й диагональю называется совокупность ненулевых элементов, расположенных на `i`-м слева месте в каждой строке. То есть нулевая диагональ – это совокупность первых в строке ненулевых элементов. В случае если в строке нет `i`-го элемента, в `i`-й диагонали не будет присутствовать ни одного элемента из данной строки. Данная форма хранения матриц используется в тех случаях, когда нужно работать с длинными векторами, а диагональная схема, в данном смысле, имеет большое преимущество перед популярными столбцовой или строчной формами. Алгоритм выглядит следующим образом:

```
for (j = 0; j <= num_diag-1; j++)
   for (i = ip[j]; i <= ip[j+1]-1; i++)
      b[ia[i]] = b[ia[i]]+a[i]*x[ic[i]];
```

В массиве `a` записана матрица по диагоналям (всего в матрице `num_diag` диагоналей), `x` – вектор, `b` – результирующий вектор. Массив `ip` хранит номера строк, отмечающих начало и конец каждой диагонали, а массивы `ic` и `ia` содержат соответственно номера столбцов и строк элементов матрицы.

Рассмотрим профиль обращений к вектору `x`. Данный профиль зависит от того, каким образом формируется разреженная матрица. Для начала рассмотрим такую стратегию выбора не-

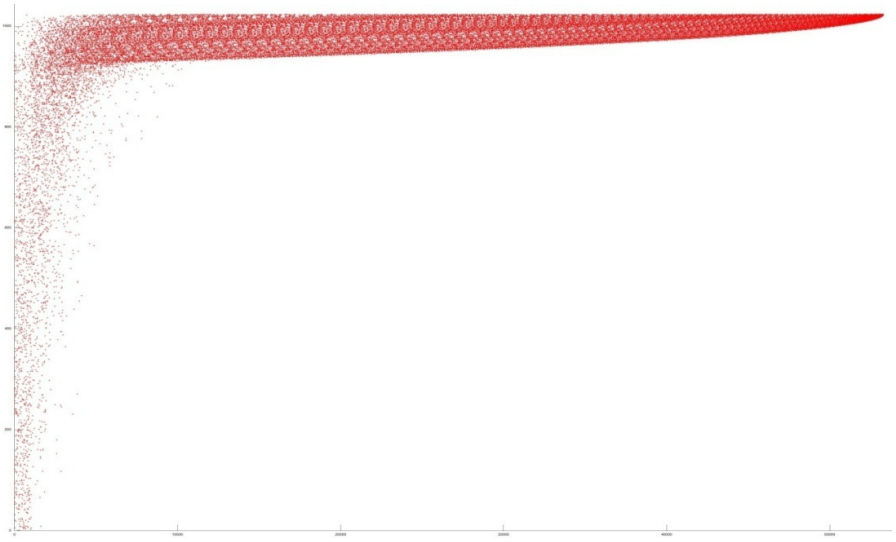


Рис. 6. Профиль работы с памятью для задачи умножения разреженной матрицы на вектор, первый вариант заполнения матрицы

нулевых элементов – для каждой строки случайным образом выбирается число ненулевых элементов в ней. Для каждой строки случайным образом выбирается первый элемент в диапазоне от `0` до `N-K-1`, где `K` – число ненулевых элементов в данной строке; затем выбирается второй элемент в диапазоне от `A1+1` до `N-K-2`, где `A1` – номер столбца первого ненулевого элемента; и т.д. В результате образуется следующий профиль (рис. 6).

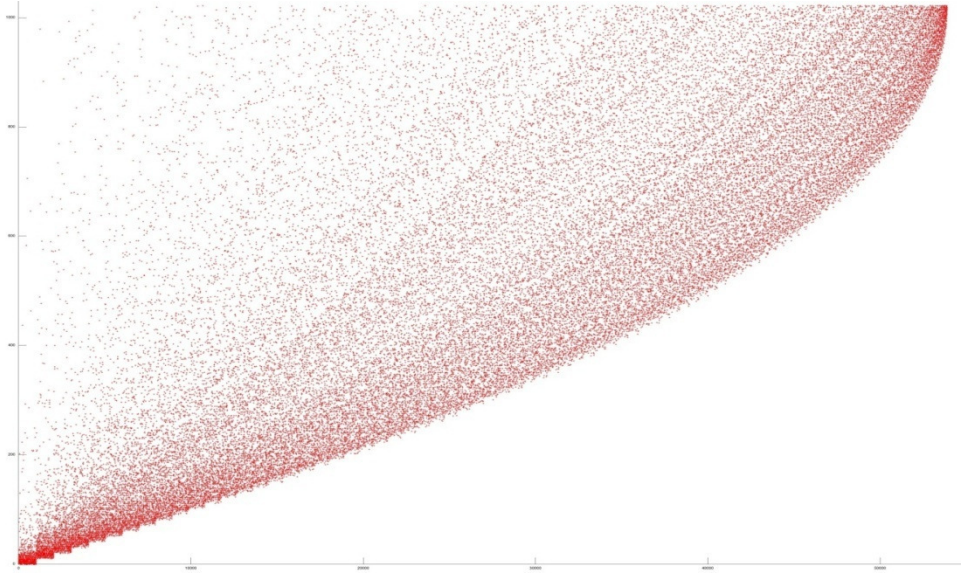


Рис. 7. Профиль работы с памятью для задачи умножения разреженной матрицы на вектор, второй вариант заполнения матрицы

Данный профиль позволяет увидеть, что большинство обращений происходит к элементам, расположенным в конце массива `x`. Следовательно, выбранный способ формирования матрицы приводит к тому, что большинство ненулевых элементов скапливается в последних столбцах матрицы. В случае если желательно получить более равномерное распределение ненулевых элементов в матрице, можно использовать другой способ их выбора: разделим строку на `K` равных частей, и в каждой части случайным образом выберем один элемент. В этом случае обращения к элементам массива `x` будут более равномерными, что и отражает профиль, показанный на рис. 7.

4. Заключение

Все профили программ, приведенных в данной работе, построены с помощью простого подхода: сначала фиксируется поток обращений анализируемой программы к памяти, который затем визуализируется с помощью системы gnuplot. Формирование потока обращений делается путем сохранения после каждой операции с массивом индекса задействованного элемента.

Безусловно, для детального изучения профиля работы программ с памятью необходим полноценный инструментарий, объединяющий глубокий аналитический анализ с развитыми возможностями по визуализации. Данный инструмент должен позволять выбирать для анализа только необходимые переменные и фрагменты кода, легко управлять объемом собираемых данных, иметь гибкие средства фильтрации, агрегирования, позволять менять масштаб и выбирать область для детального исследования. Разработка подобного инструмента стоит в плане работ на будущее, но все текущие исследования выполняются более простыми средствами.

Приведенное исследование показывает, что для обеспечения эффективной работы программ стоит изучать и анализировать профиль работы с памятью. Данный анализ можно проводить различными способами, одним из которых является исследование результатов визуализации профиля. Несмотря на то, что данный способ не позволяет проводить точный анализ, он помогает понять на общем уровне, каким образом происходит работа с памятью в программе, а также сделать некоторые выводы касательно общей эффективности использования памяти.

Список литературы

1. Воеводин Вад.В. Характеристики типовых алгоритмических структур// ПАВТ'2010, Уфа, принято к публикации в журнал ВАК «Вестник ННГУ». Март-апрель 2011, выпуск №2.
2. Воеводин Вад.В. Характеристики работы с памятью и эффективность работы программ // Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2010): материалы X Международной конференции. Пермь, 2010. С. 114-118.
3. Воеводин В.В. Вычислительная математика и структура алгоритмов // М: Изд-во МГУ, 2006. 112 с.
4. Документация по программному инструменту ThreadSpotter компании Acumem. URL: http://acumem.com. (дата обращения 9.02.2011)
5. Корж А.А. Оценочное тестирование современных систем и постановка задачи разработки суперкомпьютеров с перспективной архитектурой // доклад на семинаре «Научный семинар Parallel.ru», НИВЦ МГУ, 2008. URL: http://agora.guru.ru/parallel/files/MemoryWall-1_22-12-08.zip (дата обращения 9.02.2011)
6. Документация по программному средству Vampir. URL: http://www.vampir.eu/ (дата обращения 9.02.2011)

Грид-сервисы в вычислительной химии: достижения и перспективы

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 возможностей применения грид-сервисов для нужд вычислительной химии. На примере ресурсного грид-центра ИПХФ РАН продемонстрирована реализация ряда подобных сервисов для запуска в распределённых вычислительных средах однопроцессорных и параллельных приложений на ведущих российских грид-полигонах (EGI-RDIG, СКИФ-Полигон и Национальная Нанотехнологическая Сеть). Охарактеризованы основные компоненты созданных грид-сервисов – от адаптированных прикладных пакетов до web-ориентированного грид-портала. Описаны особенности их работы в обстановках различных грид-платформ и методы совмещения раздичных грид-сайтов на едином физическом пространстве. Сформулированы перспективы развития применения грид-сервисов в химии, указана их основополагающая роль в иерархическом многомасштабном моделировании материальных объектов от квантового до макроуровней.

1. Введение

Вычислительная и квантовая химия являются одними из наиболее заинтересованных в грид-вычислениях (в том числе на входящих в состав грид-полигонов суперкомпьютерах) отраслями науки. 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.

Наиболее востребованы грид-вычисления (как и суперкомпьютинг) в следующих областях химии, химической физики и близких к ним науках:

- изучение строения вещества;
- строение молекул и структура твердых тел;
- создание материалов с заранее заданными свойствами;
- кинетика и механизм сложных химических реакций;
- химическая физика процессов горения и взрыва;
- газодинамика экстремальных состояний;
- химическая физика процессов образования и модификации полимеров;
- предсказательное моделирование наноструктур и различные нанотехнологии;
- общие проблемы химической физики - и др.

Для проведения крупномасштабных вычислений в области вычислительной и квантовой химии и сопряженных областей науки требуется проведение высокоинтенсивных параллельных и распределенных расчетов. Например, некоторые задачи оптимизации 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.

2. Вычислительная химия как перспективный инструмент иерархического моделирования материальных объектов

Рассмотрим только одну из областей применения вычислительной химии, требующую высокопроизводительных и весьма ресурсоемких расчетов. Сейчас одним из наиболее перспективных и востребованных направлений в области вычислительной химии и смежных наук яв-

ляется разработка материалов с заранее заданными функциональными свойствами, включая как создание принципиально новых материалов, так и расширение функциональных возможностей технологически уже освоенных. Физико-химические и функциональные свойства подобных материалов определяются на микро- и нанометровом пространственном диапазоне масштабов. С физико-химической точки зрения такие материалы и структуры относятся к области молекулярно- и наноструктурированных систем. В связи с этим особую важность приобретают методы компьютерного моделирования молекулярных и наносистем, которые должны быть достаточно быстрыми и точными для возможно более точного прогноза свойств и структуры создаваемых или проектируемых материалов, в том числе под влиянием внешних воздействиях. Вообще, оптимизация наноматериалов, молекулярных и наноструктур в рамках создаваемых нанотехнологий включает исследование поведения свойств широкого круга веществ в многообразных физико-химических условиях. Чисто экспериментальный подход к решению подобных задач практически исчерпан ввиду огромного числа подлежащих оценке соединений и фазовых состояний, а также по временны*м* и экономическим соображениям. В качестве альтернативы прямым экспериментам в молекулярных и нано-технологиях все большее применение находят методы компьютерного моделирования разных уровней строения вещества, которые основаны на использовании высокопроизводительных вычислительных систем: суперкомпьютеров и распределенных вычислительных сетей.

В настоящее время разработано большое количество прикладных пакетов и авторских программ по квантово-механическому и молекулярно-динамическому моделированию атомно-молекулярных систем. Используя различные методы распараллеливания, они позволяют рассчитывать квантово-механические системы, содержащие до нескольких тысяч атомов, и молекулярно-динамические системы, насчитывающие миллионы атомов, в диапазонах времени существования до первых наносекунд. Все это позволяет осуществлять *иерархический мультимасштабный подход* к моделированию всего диапазона материальных объектов: от квантового до макроуровней. При этом на каждом более нижнем уровне вычисляются параметры и переменные, необходимые для построения моделей более высокого уровня. Цели, достигаемые на верхнем уровне иерархического моделирования, определяют задачи моделирования на нижних уровнях. Квантово-механическое моделирование методами *ab initio* проводится для малых кластеров с числом атомов 10-100 и определяет существование возможных в материале фаз, а также описывает электронный энергетический спектр, собственные функции и плотность состояний изолированного кластера при фиксированном положении ядер, потенциальную энергию системы с учетом электронно-ядерных подсистем. На последующем квантово-статистическом уровне используются модели, учитывающие уже окружение кластеров. Затем, на кинетическом уровне исследуется эволюция неравновесных систем из сотен кластеров в различных внешних условиях. На этом уровне также могут использоваться методы молекулярной динамики (с решением уравнений механики) для атомов в роли классических частиц. Становится возможным рассматривать системы от 1000 до до 1 млн. атомов и описывать объекты размером 100-1000 нм^3, а также моделировать работу устройств молекулярных размеров. На этом уровне происходит расчет кинетических коэффициентов, уравнений состояния, равновесная структура, фазовые переходы, неравновесные процессы. Потом можно переходить к мезоуровню и уровню сплошных сред, где рассчитываются вязкость, теплопроводность, коэффициенты трения и другие макроскопические процессы в материалах, а также волновые процессы в элементах. Результаты этой модели применяются уже для определения параметров методов конечных элементов и подобных ему, реализующих модели сплошной среды и модели уровня конструкций (теория механизмов и машин, теория сложных систем). Последовательное иерархическое моделирование материальных объектов в результате позволяет спрогнозировать их свойства на любом уровне, что крайне важно для построения технологических цепочек по производству новых материалов или материалов с измененными свойствами. Как легко видеть, вычислительная химия поддерживает первые уровни подобной цепи построения моделей: моделирование молекулярных и нано-систем, а также биообъектов на квантово-механическом, квантово-статистическом и кинетическом уровнях с использованием методов молекулярной динамики на основе как известных, так и вновь разрабатываемых пакетов прикладных программ, допускающих параллельную вычислительную реализацию. Таким образом, вычислительная химия на основе высокопроизводительных расчетов определяет базис дальнейшего моделирования материальных

объектов и несомненно станет ведущим инструментом как в химии, так и в большом ряде смежных областей науки.

Построение подобных иерархий моделирования, как и крупномасштабные квантово-химические расчеты – лишь одно из основных научных направлений работы вычислительного центра ИПХФ РАН [1-3].

3. Ресурсный грид-центр ИПХФ РАН, структура и возможности

Работы с системами распределенных вычислений в ИПХФ РАН были начаты в 2004-2005 годах и велись по программам Президиума РАН и Федеральным целевым научно-техническим программам, а в настоящее время продолжаются в рамках следующих программ:

- Программа № 13 фундаментальных исследований Президиума РАН на 2009-2011 годы «Проблемы создания национальной научной распределенной информационно-вычислительной среды на основе развития GRID технологий и современных телекоммуникационных сетей», проект «Исследование методов виртуализации вычислительных сред и приложений в области вычислительной химии. Динамическое формирование параллельных программных сред на распределенных ресурсах»;
- Программа № 27 фундаментальных исследований Президиума РАН на 2009-2011 годы «Основы фундаментальных исследований наноматериалов», проект «Самоорганизация наноразмерных материалов и процессы их взаимодействия с адсорбируемыми соединениями: компьютерное моделирование в параллельных и распределенных GRID средах терафлопного уровня»
- Научно-техническая программа Союзного государства РФ-РБ 2009-2010 годы, «Разработка и использование программно-аппаратных средств Грид-технологий перспективных высокопроизводительных (суперкомпьютерных) вычислительных систем семейства «СКИФ» («СКИФ-ГРИД»), проект «Разработка ряда вычислительных сервисов различного уровня (включая объединенный веб-портал) для проведения высокопроизводительных расчетов в распределённых вычислительных средах с использованием наиболее востребованных однопроцессорных и параллельных квантово-химических приложений»;
- Федеральная целевая программа «Развитие инфраструктуры наноиндустрии в Российской Федерации на 2008-2010 годы», инициативный проект «Создание Национальной нанотехнологической сети (ГридННС)»

Основными направлениями исследований авторов стали: 1) адаптация наиболее востребованных прикладных пакетов в области вычислительной (прежде всего квантовой) химии к работе в распределенных средах и обеспечение широкого доступа пользователей к работе с ним с использованием самых различных методов и технологий; 2) развитие ресурсного грид-центра (в виде объединения ресурсных сайтов для нескольких российских грид-полигонов), выступающего как в роли полигона для проведения вычислительных экспериментов в данной области, так и в роли средства для решения реальных фундаментальных и научно-практических задач; 3) развитие новых методов вычислений и организации вычислительных сервисов в условиях распределенных сред. Выбор данных направлений был обусловлен основными стратегиями развития грид-инфраструктур как в России, так и в мире, и позволяет наилучшим образом «приблизить» конечного пользователя (прежде всего – ученого-химика) к широкомасштабному использованию распределенных вычислительных ресурсов и обеспечить возможность решения задач, принципиально трудно разрешимых в настоящее время на единичных вычислительных комплексах.

Для проведения всех работ с распределенными средами основой стал реализованный ранее в ИПХФ РАН ресурсный грид-центр, который объединяет в своем составе полнофункциональные ресурсные сайты следующих российских грид-полигонов:

- узел консорциума EGEE-RDIG (Enable GRID for E-sciencE и Russian Data Intensive GRID, http://www.egee-rdig.ru, с мая 2010 года EGI – European Grid Infrastructure) на основе среды gLite (http://glite.web.cern.ch), виртуальная организация (ВО) RGSTEST;
- сайт категории «А» СКИФ-Полигона (http://skif-grid.botik.ru) на базе промежуточного ПО Unicore (http://www.unicore.eu);
- сайт Национальной Нанотехнологической Сети (ГридННС, http://www.ngrid.ru, виртуальная

организация NanoChem) – среда Globus Toolkit 4, http://www.globus.org)

Данные ресурсные сайты позволяют решать входящие задачи как с использованием адаптированных к распределенным средам прикладных квантово-химических пакетов (см.ниже), так и общего характера (если таковые не требуют предустановки на кластере).

В состав ресурсных сайтов входит также комплекс клиентских интерфейсов различных уровней для взаимодействия адаптированного квантово-химического прикладного ПО с грид-средами. Они позволяют запускать исходящие задачи вычислительной химии на распределенных ресурсах указанных полигонов, обеспечивая возможность формирования заданий, запуск на удаленных сайтах через брокеры ресурсов, мониторинга прохождения заданий, сбора результатов и статистики.

Важнейшей частью ресурсного центра ИПХФ РАН является грид-портал (http://grid.icp.ac.ru, Grid Enabled Chemical Physics), объединяющий высокоуровневые web-интерфейсы в систему грид-сервисов (см. ниже).

Работа в рамках ВО RGSTEST обеспечивает доступ к вычислительным мощностям до 500-700 процессоров и дисковым массивам порядка 8-12 терабайт в нескольких географических зонах (Москва, Дубна, Харьков, Черноголовка и др.). Разнородность узлов данной ВО позволяет легко варьировать параметры запускаемых задач, ориентируясь на различные типы ресурсов. Использование подобного полигона обеспечивает проведение достаточно масштабных вычислительных экспериментов как научного, так и прикладного характеров.

Ресурсный сайт для среды Unicore позволяет выполнять входящие задачи сертифицированных пользователей СКИФ-Полигона, производить мониторинг задач и передавать полученные результаты пользователям. Обеспечена возможность мониторинга состояния сайта извне. Клиентский интерфейс обеспечивает запуски исходящих задач через среду Unicore на собственном ресурсном сайте ИПХФ (в роли удаленного ресурса - https://unicorgw.icp.ac.ru:8080) и на доступных (через брокер ресурсов ИПС (https://testbed.botik.ru:9999) ресурсных узлах СКИФ-Полигона – в основном ИПС РАН, СевКавГУ, СКИФ-МГУ, Cyberia (Томск), СКИФ-Аврора (Челябинск) и др.

В рамках создаваемой с 2010 года Национальной Нанотехнологической Сети (ГридННС) ресурсному сайту ИПХФ предоставлен доступ к вычислительному полигону с общим числом CPU более 8000 (http://mon.ngrid.ru/stats?page=usage) и большим количеством виртуальных организаций, в том числе поддерживающих квантово-химические расчеты. Отметим, что в рамках ГридННС ИПХФ создал и возглавил виртуальную организацию ВО **Nanochem** для проведения крупномасштабных квантово-химических расчетов.

В настоящее время (на 1.12.2010) ресурсный грид-центр ИПХФ реализован следующим образом. Установлен и настроен вычислительный кластер – на базе системы пакетной обработки заданий PBS/Torque Resource Manager подключены 6 расчетных узлов (с текущей пиковой производительностью до 172 GFLOPS – 6x2x4 ядра, Intel Xeon 3,6 ГГц) с увеличением их количества в течение конца 2010-начала 2011 до 9, а затем до 13 расчетных узлов (по мере передачи узлов из основного вычислительного кластера ИПХФ РАН в состав грид-сайта). Управляющая и расчетная сеть – Gigabit-Ethernet. Ресурсный сайт доступен для вычислений и мониторинга извне в постоянном режиме (24/7/365). На расчетных узлах установлен Scientific Linux 5.4 Boron, поскольку наличие данного дистрибутива требуется для расчетных узлов ресурсного сайта gLite, а для других распределенных сред выбор ОС (варианты Linux) не принципиален. На 2-х управляющих машинах установлен гипервизор KVM (Kernel-based Virtual Machine, http://www.linux-kvm.org), выбранный из ряда прочих гипервизоров благодаря простоте администрирования, минимуму накладных расходов ресурсов и устойчивости работы под нагрузкой. После установки гипервизора на управляющих машинах были установлены следующие сервисные виртуальные машины:

1. для Computing Element среды gLite 3.1 (полигон EGEE-RDIG) – ScientificLinux 4.5 и соответствующие сервисы типа lcg-ce, сервисов авторизации и мониторинга;
2. для среды Unicore 6.2 (сайт СКИФ-Полигона) – ОС Ubuntu 9.10, сервисы – шлюз (Gateway); серверный контейнер (Unicore/X), интерфейс к целевой системе (TSI), авторизационный сервис-пользовательская база данных – XUUDB$, пользовательский интерфейс (UI)

3. для среды Globus Toolkit 4 (полигон ГридННС) – ОС CentOS 5.4, сервисы MDS, GRAM, GridFTP, RFT, User Interface.

Выбор соответствующих ОС для управляющих узлов обусловлен либо требованиями дистрибутивов распределенного ПО (как, например, для gLite), либо рекомендациями разработчиков, либо простотой администрирования. На все управляющие машины были установлены серверные компоненты пакета управления заданиями PBS/Torque (http://www.clusterresources.com), тогда как на расчетные узлы была установлена клиентская часть данного пакета. Поскольку все распределенные middleware требуют наличия своих собственных очередей PBS, было принято решение о настройке трех одновременно работающих экземпляров даемона 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 вычислительных экспериментов работ и отладки прикладного ПО это вполне приемлемо. Более детально механизм размещения ресурсных сайтов на едином физическом пространстве кластера описан здесь [6].

4. Основные варианты грид-сервисов для решения задач вычислительной химии

4.1 Работа в распределенных средах с прикладными пакетами вычислительной химии и авторскими программами

Авторами проводилась экспериментальная проверка и апробация возможности использования грид-ресурсов для реальных расчетов на стандартных прикладных пакетах программ (в том числе и параллельных), используемых в вычислительной химии, а также различных авторских программ, разработанных в ИПХФ и НЦЧ РАН. Особый интерес имеет адаптация этих программ для распределенных вычислений на максимуме доступных ресурсов российских и международных грид-инфраструктур.

Для адаптации во всех упомянутых распределенных вычислительных средах (см. выше) были выбраны следующие, наиболее востребованные пользователями ИПХФ прикладные программные пакеты:

- **GAMESS-US** (http://www.msg.ameslab.gov/GAMESS) - одна из самых популярных программ для теоретического исследования свойств химических систем, уступает по известности лишь комплексу Gaussian, позволяет рассчитывать энергию, 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;
- **VASP** (Vienna University, http://cms.mpi.univie.ac.at/vasp) и **PWscf** (http://www.pwscf.org, Plane-Wave Self-Consistent Field) – предназначены для моделирования объема и поверхности твердых тел в рамках неэмпирических подходов, основанных на применении функционалов плотности с использованием периодических граничных условий с базисами на плоских волнах. VASP позволяет проводить оптимизацию структуры и выполнять моделирование в рамках молекулярной динамики. Программный комплекс VASP необходим для моделирования процессов на поверхности и в объеме твердых тел (прежде всего катализа и ионной проводимости). Моделирование на квантово-механическом уровне осуществляется для малых кластеров с числом атомов 10-100, определяющих существование возможных в материале фаз. 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.
- **Gaussian-03** (http://www.gaussian.com) - самое популярное средство выполнения квантово-химических расчетов среди основной массы химиков. Основные причины этого - широта

охвата реализованных квантово-химических методик, высокая эффективность и удобный интерфейс пользователя. Современные версии комплекса программ Gaussian расширением спектра поддерживаемых квантово-химических методов и их модификаций. Комплекс программ Gaussian 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. Основное направление, в котором развиваются версии, это развитие методов расчета сверхбольших молекулярных систем. Однако, использование пакета в распределенных средах затруднено лицензионными ограничениями.

- **Dalton-2** (http://www.kjemi.uio.no/software/dalton/dalton.htm) – позволяет рассчитывать синглет-синглетные возбуждения, а также электронные структуры, вращательные и колебательные спектры молекул, учитывать релятивистские эффекты и эффект сольватации;
- **CPMD** (http://www.cpmd.org) – расчеты в области молекулярной динамики;
- **NAMD** (University of Illinois at Urbana-Champaign, Computational Biophysics Group, http://www.ks.uiuc.edu/Research/namd) – хорошо масштабируемая молекулярно-динамическая программа. Одна из наиболее быстрых при параллельном вычислении на большом числе процессоров. Программа активно используется в ИПХФ РАН для расчетов мицеллы (micelle - коллоидная частица, несущая электрический заряд и объединяющая в себе несколько крупных молекул);
- Авторские программы (разработки ИПХФ), включающие многопараметрические задачи из области квантовой химии и молекулярной динамики.

Для всего выбранного ПО был проведен детальный анализ модульной структуры квантово-химического кода и изучены особенности работы различных реализаций однопроцессорных и параллельных версий, определены стратегии реализации выбранных типов квантово-химических вычислений применительно к различным распределенным средам.

Для большинства выбранных прикладных пакетов созданы и протестированы на реальных задачах низкоуровневые интерфейсы для запуска их в распределенных вычислительных средах. Данные интерфейсы включают набор скриптов по формированию исходящих заданий, запуску через брокер ресурсов на удаленных узлах, мониторингу выполнения задач, возвращению полученных результатов с удаленных ресурсов и «сборку» окончательных результатов на интерфейсе пользователя. Реализованы интерфейсы для однопроцессорных и параллельных (SMP, сокетные, MPI-1,2) вариантов указанного ПО. На ресурсном грид-узле ИПХФ, использованном в качестве удаленного распределенного ресурса, проведены запуски указанного прикладного ПО через инфраструктуры ВО RGSTEST (EGEE-RDIG) и СКИФ-Полигона. Запуски всего адаптированного ПО проводились в разных режимах и конфигурациях (с разным количеством востребованных процессоров и использованием разных вариантов параллельных расчетов). Были изучены варианты совмещения различных вариантов распараллеливания (например, SMP+MPI) вычислений применительно к некоторым прикладным пакетам (пакеты Dalton-2 и CPMD). После ряда вычислительных экспериментов была проведена коррекция созданных низкоуровневых интерфейсов и окончательная оптимизация их для распределенных сред. Были скорректированы проблемы запуска и работы параллельных (SMP, сокетные, MPI-1,2) вариантов указанного ПО на различных типах ресурсных узлов (разные пакетные системы PBS и параллельные среды). В настоящее время ресурсные сайты ИПХФ позволяют решать в качестве **входящих** задачи вычислительной химии с использованием всех указанных квантово-химических пакетов.

4.2 Реализация грид-сервисов в виде высокоуровневых web-интерфейсов

Составной частью ресурсного центра ИПХФ РАН является грид-портал, объединяющий грид-и web сервисы. В его рамках сформированы высокоуровневые web-интерфейсы, позволяющий более эффективно использовать все преимущества грид-расчетов. Эта среда позволяет пользователям получить доступ к грид-ресурсам и сервисам, вызывать и настраивать их с помощью web-браузера. Архитектура грид-портала основана на идее, что портальная система является контейнером для низкоуровневых пользовательских интерфейсов, обеспечивающих работу с грид-службами. Преимущество данной архитектуры в том, что она достаточно легко позволяет встраивать в портал интерфейсы новых грид-служб и изменять существующие. Пор-

тальные сервисы контролируют и визуализируют пользовательский интерфейс. В ИПХФ РАН был сформирован грид-портал (http://grid.icp.ac.ru, Grid Enabled Chemical Physics – GECP), включающий WWW интерфейсы к следующим прикладным пакетам:

1. Квантово-химический комплекс GAMESS-US, методы *ab initio* которого могут использовать параллельные вычисления;
2. Вычисление многопараметрических функций, под которымиой следует понимать целый класс задач химической физики, обладающих свойством параллелизма по данным (Data Parallel).

Данные web-интерфейсы позволяют определять входные параметры и условия (включая загрузку данных, создание и редакцию конфигурационных файлов, работу с сертификатами пользователя), формировать сложные первичные файлы запуска, производить (при условии авторизации пользователя) запуск данного ПО в распределенных средах, осуществлять мониторинг выполнения заданий и сбор результатов. Интегрирована также технология работы через web-интерфейс с «пучками» независимых заданий на «нарезаемых» областях данных, а также с использованием метода формирования «виртуальных контейнеров» для пакета GAMESS-US (см.ниже). Заметим, что основная часть программного кода web-интерфейсов не связана напрямую с выбранной распределенной средой, поэтому они подключены ко всем трем вышеупомянутым грид-полигонам (EGI-RDIG, СКИФ-Полигон, ГридННС) и, соответственно, поддерживает три распределенные среды (gLite, Unicore, Globus GT4).

Созданные web-интерфейсы значительно снижают трудоемкость работы пользователя в части формирования задач и работы с первичными данными и значительно облегчают последующую работу с пакетами в распределенных средах.

4.3 Решение грид-задач с применением методов «виртуальных контейнеров»

Следует отметить, что большинство прикладных пакетов вычислительной химии (как, впрочем, и прикладных пакетов из других областей науки и техники) отличаются сложностью конфигураций и повышенными требованиями к среде выполнения, особенно для проведения параллельных расчетов. Обычно эта проблема решается путем создания *виртуальных организаций*, т.е. объединением через распределенные среды во многом однотипных (по установленному программному обеспечению и настройкам) вычислительных ресурсов. Для них выбранные прикладные пакеты (вместе со средствами конфигурирования и настройки) распространяются из единого репозитория (как, например, для прикладных пакетов ЦЕРНа – Atlas, CMC, Alice и т.п.). В большинстве же случаев неподготовленный ресурсный сайт не имеет нужного заранее установленного прикладного ПО или хотя бы не сконфигурирован должным образом, поэтому запуск непредустановленных сложных прикладных пакетов обычно для таких ресурсов оканчивается неудачей. Поэтому в общем случае необходима ручная или полуавтоматическая перенастройка ресурсных узлов распределенных сред, включающая установку собственно пакетов, конфигурирование центрального узла и расчетных узлов (настройка переменных окружения, общих NFS ресурсов, PBS очередей), установка дополнительных системных библиотек и исполняемых файлов (включая параллельные среды типа Mpich-2). При условии этого возможны запуски пакетов на распределенных ресурсах (как это и сделано на ресурсных сайтах ИПХФ для решения входящих задач).

Для частичного решения данной проблемы авторами был разработан метод создания виртуальных перемещаемых программных «контейнеров». «Контейнер», включающий собственно прикладной пакет, набор необходимых системных файлов и библиотек, скрипты по развертыванию и настройке среды исполнения, файлы данных и конфигурационные файлы, доставляется на удаленный ресурсный узел грид-среды стандартными средствами распределенного middleware. Применение таких «контейнеров» позволяет передавать заранее настроенную среду как единое задание, не требующее дополнительного конфигурирования и сложной процедуры установки и настройки, производимых, как правило, вручную администратором кластеров. «Контейнер» по прибытии на ресурсный узел производит развертывание пакета и необходимых системных библиотек, настройку среды исполнения (включая параллельную среду), запуск задания, по его окончании проводится отправка результатов на пользовательский интерфейс и «очистка» среды исполнения, т.е. приведение ресурса в первоначальное состояние. Так могут

быть решены проблемы установки, настройки, несовместимости с операционной системой и другими программами, разрешаются конфликты одинаковых приложений. Более детально этот метод описан в статье авторов в этом же сборнике трудов (Волохов и др., «Динамически формируемые параллельные среды…»).

4.4 Работа с «пучками» формально независимых заданий

Для решения широкого класса многопараметрических задач вычислительной химии с использованием грид-технологий был создан метод запуска “пучков” независимых заданий для использования всех доступных ресурсов распределенной среды. Как говорилось ранее [1,3,5], в области химической физики существует класс задач, требующих перебора большого количества параметров. При этом полная задача разбивается на огромное количество независимых подзадач (каждая определяется группой значений совокупности параметров). Задача автоматизации процесса разбиения полной задачи на фрагменты важна и определяет удобство пользования системой. Типичный пример - фундаментальная задача в теории элементарных химических процессов: туннельные реакции под воздействием электромагнитного излучения. Параметрами являются частота и амплитуда излучения. Задача имеет высокую вычислительную сложность, однако вычисления в каждой точке сетки в ней происходят независимо друг от друга, поэтому оказалось возможным разбить область вычислений на множество непересекающихся подобластей и для каждой из них запускать задачу на различных процессорах.

Была разработана методика запуска задач и получения результатов методом запуска “пучков” заданий на всех доступных ресурсах выбранной распределенной среды. На языке Perl написан комплекс программ для запуска “пучков” заданий и получения результатов счета с использованием пользовательских интерфейсов (UI) сред gLite, Unicore, Globus GT4. Для решения многопараметрических задач квантовой химии были разработаны методы формирования «пучков» 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, формирование пулов независимых заданий, создание очередей запуска и отправки заданий на брокер ресурсов. После запуска периодически запускаемые (средствами ОС, например по cron) скрипты ведут мониторинг выполнения заданий, контроль таймаутов, перезапуск неудачных заданий и сбор результатов выполненных заданий (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.

В настоящее время изменена стратегия работы с «пучками». Теперь данные о разбиении расчетной области сохраняются в базе данных (MySQL) и используются в оперативном режиме для формирования временного файла начальных данных при генерации исполняемого задания. Это позволило резко снизить нагрузку на файловую систему сервера и увеличить скорость работы. Для работы с большим количеством грид-заданий (для «пучков») переделан механизм контроля «нарезанных» данных и состояния отдельных задач. Для оптимизации времени доступа и скорости просмотра все задания записываются в три таблицы базы данных mySQL: ожидающие запуска, отправленные и посчитанные. Эти таблицы являются общими для всех зарегистрированных пользователей портала. Соответственно, для каждой таблицы заданий написан соответствующий монитор на языке Perl, который может запускаться в качестве сервиса как после заполнения формы запроса пользователем, так и с использованием системных средств (например, утилитой cron). Таблица заданий, ожидающих запуска, заполняется после того, как пользователь определил все параметры, как для формирования файлов данных, так и файлов запуска, а также, загрузил свои личные сертификаты и получил прокси-сертификат для запуска заданий в грид-средах.

Монитор запуска заданий просматривает последовательно таблицу заданий, ожидающих запуска, и при обнаружении записи формирует команду обращения к брокеру ресурсов соответствующего полигона. Если получен положительный ответ от брокера ресурсов, то текущая запись перемещается в таблицу отправленных заданий. Монитор запуска прекращает свою работу, если таблица пуста. Однако система периодически порождает запуск монитора, если он отсутствует в списке процессов. Аналогично работает монитор проверки статуса

запущенного задания с таблицей отправленных заданий. В результате записи перемещаются в таблицу завершенных заданий, с которыми работает монитор возврата файлов результатов по описанному выше алгоритму, при этом соответствующая запись удаляется. Файлы результатов пользователя, получают имена, содержащие индекс запуска, и накапливаются в директории портала указанного проекта. Данная технология облегчает проведение непрерывного мониторинга запуска заданий, включая контроль таймаутов, перезапуск неудачных заданий и т.п.

Главным недостатком механизма работы с «пучками» заданий являются ограниченные возможности пользователя по мониторингу собственных заданий в средах Unicore и Globus GT4. В целом же, описанный новый, полностью асинхронный механизм запуска «пучка» заданий требует от пользователя только выбора ресурса по желанию, что возможно через механизмы web-портала.

5. Заключение

Авторами описаны использование некоторых технологий грид-вычислений применительно к приложениям вычислительной химии. Наши работы позволили создать достаточно полный ряд грид-сервисов для проведения крупномасштабных расчетов в области вычислительной химии. Это позволило достигнуть нового уровня расчетов в области вычислительной химии:

- создан комплекс адаптированных к различным грид-средам (gLite, Unicore, Globus GT4) прикладных программных пакетов вычислительной химии с интерфейсами различного уровня (от низкоуровневых интерфейсов вплоть до Web-портала);
- разработаны новые методики вычислений (методы формирования «пучков» независимых заданий, метод «виртуальных контейнеров» и т.д.) в распределенных и параллельных средах применительно к прикладному ПО вычислительной химии;
- создан ресурсный центр (включающий ресурсные узлы полигонов EGI-RDIG, СКИФ-Полигона, ГридННС, а также web-портал) для проведения вычислительных экспериментов в этой предметной области, объединяющий как ресурсы для решения входящих заданий в средах gLite, Unicore, Globus GT4, так и пользовательские интерфейсы к этим распределенным средам для решения исходящих задач на внешних ресурсах;

В результате выполнения всего проекта создан вычислительный центр, позволяющий проводить масштабные расчеты в области вычислительной химии в распределенных средах на крупномасштабных полигонах (в перспективе до 10^4 CPU на узлах многотерафлопного масштаба). На ряде реальных задач продемонстрирована применимость созданных ресурсов для решения крупномасштабных химических задач на высокопроизводительных вычислительных полигонах. Это 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, фармацевтика, разработка топливных элементов и прочие близкие отрасли наук.

Литература

1. Волохов В.М., Варламов Д.А., Пивушков А.В., Покатович Г.А., Сурков Н.Ф. *Технологии грид-в вычислительной химии* // «Вычислительные методы и программирование», М.: МГУ, 2010, т.11, № 1, с.175-182
2. 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
3. В.М. Волохов, Д.А. Варламов, А.В. Волохов, А.В. Пивушков, Г.А. Покатович, Н.Ф. Сурков *Использование грид-полигонов для решения больших задач вычислительной химии* // «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (20-25 сентября 2010 г., г. Новороссийск). – М.: Изд-во МГУ, 2010, с.107-111

4. Д.А. Варламов, В.М. Волохов, Н.Ф. Сурков, А.В. Пивушков *Виртуализация вычислительной среды в грид* // «Параллельные вычислительные технологии 2010» ПаВТ-2010, (Уфа, март 2010), Челябинск, изд-во ЮУрГУ, с.63-70

5. В.М. Волохов, А.В. Пивушков, Н.Ф. Сурков, Д.А. Варламов, А.В. Волохов *Новые методы решения задач вычислительной химии в распределенных средах* // «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (20-25 сентября 2010 г., г. Новороссийск). – М.: Изд-во МГУ, 2010, с.181-184

6. 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;

Динамически формируемые параллельные среды в условиях грид-полигонов, проблемы и решения

В.М. Волохов[1], Д.А. Варламов[1,2], А.В. Пивушков[1], А.В. Волохов[1], Н.Ф. Сурков[1]

[1] Институт проблем химической физики РАН
[2] Институт экспериментальной минералогии РАН

Статья посвящена анализу проблем выполнения сложносконфигурированных параллельных прикладных пакетов в распределенных средах и описанию некоторых способов разрешения подобных проблем. Описаны методы создания и запуска в различных грид-средах (gLite, Unicore, Globus GT4) динамически формируемых («виртуальных») параллельных сред исполнения для обеспечения запуска параллельных прикладных задач. Разработанные методы позволяют запускать сложные, обычно требующие предустановки и настройки кластера, параллельные приложения на несконфигурированных заранее произвольных ресурсах грид-полигонов. Приведен пример реализации данного метода для квантово-химического пакета GAMESS-US на различных грид-полигонах в условиях всех трех вышеупомянутых сред.

1. Введение

На основе долговременного опыта работы в различных распределенных средах - gLite, Unicore, Globus GT4 [1] в условиях основных российских грид-полигонов (EGEE(EGI)-RDIG, Национальная Нанотехнологическая Сеть, СКИФ-Полигон) авторами ранее [2,4] был сделан вывод, что значительными препятствиями (среди многих прочих) на пути применения грид-технологий в вычислительной химии (а в целом – для запуска в грид-средах любых сложно сконфигурированных прикладных пакетов) являются следующие проблемы:

- грид-полигоны могут содержать в своем составе разнородные ресурсные узлы (обуславливается разнообразием операционных систем, применением различных типов сетевых сервисов, способами реализации параллельных сред, системами управления задачами, политиками безопасности и т.п.);
- для многих ресурсоемких параллельных приложений необходимо создавать вычислительную среду, состоящую из «приватных» конфигурационных настроек, дополнительных служб, специфичных параллельных сред, специализированных мест хранения данных и прочих компонентов, что плохо коррелирует с особенностями настройки кластеров в целом;
- невозможно (или избыточная трудоемко) перенастраивать работающие вычислительные ресурсы (особенно суперкомпьютеры или кластеры класса “production farms”) для целей распределенных вычислений под нужды конкретных прикладных пакетов.

Одной из стратегий решения этих проблем (или хотя бы части из них) может стать применение различных технологий виртуализации вычислительных ресурсов и приложений.

В 2009-2010 годах в рамках Программы фундаментальных исследований Президиума РАН № 13 на 2009-2011 годы «Проблемы создания национальной научной распределенной информационно-вычислительной среды на основе развития грид-технологий и современных телекоммуникационных сетей» авторами были продолжены исследования по применимости различных методов виртуализации в условиях распределенных сред. Среди прочих задач проекта была поставлена цель изучить и применить на реальных расчетах в области химии два класса технологий виртуализации расчетов, включая:

- адаптация ряда квантово-химических пакетов к работе в условиях разных распределенных сред (gLite, Unicore, Globus Toolkit), в том числе с использованием динамически формируемых (виртуальных) переносимых параллельных сред;
- создание и применение виртуальных машин на существующих ресурсных узлах различных распределенных сред с целью расширения их функциональности (решение различных прикладных задач на базе разных программных архитектур, разделение ресурсов, повышение

безопасности, вычислительные эксперименты) и отработки устойчивости узлов;

Термин «виртуализация» был использован авторами в двух основных смыслах: виртуализация вычислительных грид-ресурсов и сервисов с использованием виртуальных машин (ВМ) и виртуализация вычислительного объекта (приложения), перемещаемого в грид-среде. Потребность в виртуализации приложений для распределенных вычислительных сред продиктована необходимостью создания и поддержки стандартных механизмов взаимодействия между приложениями и вычислительными ресурсами (сервисами), одинаковых со стороны ресурса (поставщика сервисов) и со стороны приложения (вернее, созданного для него интерфейса) вне зависимости от настроек конкретного вычислительного ресурса.

Основные результаты в области виртуализации ресурсов были описаны авторами в других публикациях [4-6], в данной же статье более детально рассмотрен один из разработанных авторами методов виртуализации вычислительного объекта (параллельного приложения), перемещаемого и выполняемого в грид-среде. Метод основан на реализации виртуальной, динамически формируемой параллельной MPI среды, передаваемой на ресурсные узлы грид-полигонов в форме «виртуального контейнера». Метод включает предварительную адаптацию (и иногда рекомпиляцию) программы (прикладного пакета) для работы в роли приложения в составе «контейнера», создание прототипа виртуальной параллельной среды исполнения, формирование собственно «контейнера» и процесс выполнения его как исходящего грид-задания на неподготовленном удаленном ресурсном грид-узле (т.е без предустановки и настройки собственно прикладного пакета). Для чего это надо? Конечный пользователь грид-среды перестает быть зависим от наличия на ресурсных узлах предустановленных прикладных пакетов (что во многом зависит от администраторов кластеров), используемых параллельных сред, особенностей настроек кластеров. В идеале возможен запуск большинства сложно сконфигурированных прикладных пакетов без привязки к конкретным ресурсным узлам, что всегда являлось ахиллесовой пятой грид-сред.

2. Анализ механизма выполнения стандартных грид-заданий в различных распределенных средах (для параллельных приложений)

Сегодня для эффективной и комфортной работы прикладных пакетов даже в условиях локального кластера требуется создание целой информационно-вычислительной инфраструктуры, включающей вспомогательные приложения, службы, сетевые сервисы, хранилища данных и прочие компоненты, которые зачастую плохо совместимы с режимами работы ресурсного узла в целом. Для ряда пакетов прикладного ПО и сервисов нужно создавать комплексные среды с необходимым набором приложений и политиками безопасности. Ключевыми требованиями являются скорость и простота предоставления таких сред, их тщательная изоляция друг от друга, квотирование вычислительных ресурсов для каждой среды, независимость от базовых настроек узла. Зачастую все это необходимо делать без прерывания работы узлов и остановки вычислительной среды, особенно в рамках суперкомпьютеров и кластеров класса «production farms», т.е. ресурсных узлов, не допускающих остановок и переконфигурирования системы.

Другой фундаментальной проблемой решения задач (особенно параллельных) в условиях распределенных вычислений является необходимость виртуализации программных сред для исходящих задач. Например, для проведения параллельных вычислений требуется наличие установленной на ресурсных узлах какой-либо системы параллельного программирования (MPI, OpenMP и др.) или предустановленных специфичных математических библиотек. Широко используемые в настоящее время пакеты прикладных программ (ППП) вычислительной химии (GAMESS, Gaussian, NAMD и др.), как, впрочем, и большинство инженерных пакетов (ANSYS, Abacus, FlowVision и т.п.), отличаются сложностью конфигураций и повышенными требованиями к среде выполнения, особенно для проведения параллельных расчетов. Они требуют обязательной настройки большого количества переменных окружения операционной системы до запуска параллельного приложения на каждом из использующихся процессоров. Такая настройка, как правило, осуществляется в два этапа:

– при ручной (или полуавтоматической) установке ПО системным администратором на каждом узле ресурсного сайта на уровне операционной системы (например, при формировании

сайтов виртуальной организации, требующей единых настроек ПО);
– при настройке соответствующих скриптов запуска задания для каждого пользователя, согласно требованиям как приложения, так и системы параллельного программирования;
– при установке пакетов из центрального репозитория, в рамках виртуальной организации.

При этом традиционный подход со статическим линкованием необходимых библиотек (не говоря уже о динамическом варианте) к исполняемому модулю (пакету) часто не способен создать *полностью работоспособное* параллельное задание на произвольном ресурсе среды грид, поскольку на подобном ресурсе попросту могут отсутствовать необходимые системные файлы или не поддерживаться необходимая приложению параллельная среда.

Для решения данной проблемы авторами был проведен анализ процедуры исполнения типичного параллельного задания на ресурсном грид-узле (для сред gLite, Unicore, Globus GT4), позволивший определить требования к создаваемому виртуальному образу среды исполнения, а также принципиальную возможность формирования динамической среды исполнения для тестируемых ресурсов. Также был сделан анализ систем параллельного программирования для выбора оптимального виртуального образа среды исполнения параллельного приложения на грид-ресурсах. В качестве базового пакета для разработки виртуального образа среды исполнения параллельного приложения был выбрана среда Mpich-2. Детальнее этот анализ и последующая работа с библиотеками MPI-2 описан здесь [2,4]. Отметим, что на данном этапе работ были введены некоторые ограничения для используемых программных сред:

- использованы только аппаратные архитектуры x86 и em64t;
- на расчетных узлах грид-ресурсов предполагается использование операционной системы Linux (клоны на базе RedHat – собственно RedHat, ScientificLinux, Fedora и т.п.), что связано с особенностями размещения системного ПО, хотя принципиальных ограничений на использование других ветвей Linux дистрибутивов нет;
- по стандарту настройки ресурсных грид-узлов для коммуникации между расчетными узлами используется интерфейс TCP/IP и беспарольный доступ по ssh (включая копирование файлов), возможна поддержка NFS ресурсов;
- некоторые версии прикладных пакетов с целью повышения производительности вычислений имеют привязку к сетевым продуктам конкретных производителей и используют поставляемые этими производителями низкоуровневые драйверы. Нами пока такие версии, несмотря на их высокую эффективность, использоваться не будут.

После анализа процедуры выполнения грид-задания в разных распределенных средах и выбора среды параллельных вычислений была разработана технология создания динамически формируемых образов исполняемых сред, или виртуальных «контейнеров». Был сформирован перемещаемый программный пакет MPI-2. Полученный пакет далее использовался в качестве базового прототипа для разработки виртуального образа среды исполнения конкретных параллельных приложений.

В качестве тестового приложения был использована программа вычисления числа π (‘cpi.c’) из пакета Mpich-2, правильность работы которой легко проверяется в параллельной среде с различным количеством узлов. Исходная тестовая программа была доработана с учетом особенностей запуска прикладных приложений на грид-узлах, был получен ее исполняемый модуль и скрипты запуска с использованием библиотек MPI-2. Тестовый модуль и перемещаемый пакет MPI-2 были собраны и упакованы в единый «контейнер», для запуска которого в тестируемых грид-средах была разработана серия низкоуровневых скриптов пользовательского интерфейса.

Была принята следующая схема запуска: на удаленный ресурсный грид-узел сети через брокер ресурсов (или непосредственно – как возможно в Globus) передается главный скрипт и упакованный «контейнер», содержащий исполняемые файлы, необходимые системные библиотеки, файлы конфигурации и данных. Далее главный скрипт выполняет (упрощенно) следующую последовательность шагов: сбор информации о текущем узле грид, распаковка «контейнера» в директории псевдопользователя грид-среды и перемещение файлов в общедоступную область, настройка среды, запуск сервера mpd (с правами mapped-user) на стартовом узле и проведение его тестирования, распределение необходимых файлов по списку свободных узлов, запуск «кольца» серверов mpd, запуск параллельного приложения и его работа как обычного

распределенного задания с последующей передачей результатов на брокер ресурсов и пользовательский интерфейс, удаление всех библиотек и временных файлов со всех узлов. Более детально последовательность работы «контейнера» описана здесь [2-4]

Тестовый вариант «контейнера» (на примере нескольких простых параллельных задач) был отлажен на ресурсном грид-центре ИПХФ (его сайты использовались в качестве удаленного ресурса) для сред gLite, Unicore, Globus. Дальнейшее тестирование было проведено на ресурсных узлах RDIG в рамках ВО RGSTEST (узлы НИИЯФ МГУ), а также на узлах СКИФ-Полигона и ГридННС, что показало применимость данного метода для большинства ресурсных сайтов данных сетей.

Разработка системы динамического компилирования приложения на ресурсном узле и инсталляции дополнительных библиотек на данном этапе работ не рассматривалась.

3. Реализация динамически формируемой параллельной среды исполнения для запуска квантово-химического пакета GAMESS-US

Для проведения реальных технологических испытаний разработанного метода на примере прикладного пакета был выбран квантово-химический пакет GAMESS-US.

GAMESS-US (http://www.msg.ameslab.gov/GAMESS) – одна из наиболее популярных программ для теоретического исследования свойств химических систем, уступает по известности лишь комплексу Gaussian, позволяет рассчитывать энергию, 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. Основные программные модули GAMESS-US поддерживают параллельный режим вычислений как на многопроцессорных компьютерах, так и на кластерах. Пакет отличается сложностью установки и конфигурации, а также требует нестандартных настроек параллельной среды вычислений.

Работы по распараллеливанию GAMESS-US начались еще в 1991 году. Однако использование методов передачи сообщений MPI получило применение только с 1999 г., когда в пакете GAMESS-US была реализована модель интерфейса с распределенным размещение данных (DDI – Data Distributed Interface). Последняя версия интерфейса DDI, которая была оптимизирована для многопроцессорных SMP-архитектур общего вида, особенно работающих с памятью в стиле System V, была выпущена только в мае 2004 г. В настоящее время практически все *ab initio* методы, включенные в пакет GAMESS, могут использовать параллельные вычисления.

Интерфейс DDI использует в качестве базовой *сокетную* TCP/IP модель межпроцессорных коммуникаций. Использование такого метода распараллеливания для работы на локальном кластере достаточно эффективно и довольно просто в конфигурации, но при работе в грид-средах возникает ряд принципиальных проблем: а) необходимо заранее явно указывать используемые расчетные узлы (что обычно нереально); б) неправильно оценивается загруженность расчетных узлов (учитывается только первый расчетный узел); в) отсутствует возможность контроля выполнения удаленной задачи средствами middleware распределенной среды; (г) на ряде современных кластеров (например «Чебышёв» в НИВЦ МГУ) сокетная модель неработоспособна из-за политик безопасности кластера.

Конфигурации же GAMESS-US с использованием библиотеки MPI авторами пакета разработаны только для ряда мейнфреймов известных производителей (Cray, IBM, SGI). До последнего времени конфигурации с MPI не рекомендовались, и желающим предлагалось экспериментировать с такими конфигурациями самостоятельно.

Для работы с пакетом GAMESS-US в грид-среде на ресурсных узлах ИПХФ РАН первоначально была установлена последняя наиболее широко распространенная версия Mpich-1.2.7 (см.выше), которая является реализацией стандарта MPI-1. Достоинством данной версии является то, что она явно включает интерфейс Globus-2, основанный на Globus Runtime System, что было бы эффективно для запуска Globus заданий. Однако получить работоспособную конфигурацию пакета GAMESS-US для MPI-1 не удалось по двум основным причинам:

- из-за особенностей запуска исполняемого задания GAMESS-US, которая осуществляется скриптом, активизирующем более 150 переменных окружения. На главном узле среда соз-

дается правильно, но механизм передачи переменных окружения на подчиненные узлы в библиотеке Mpich-1 стандартно отсутствует. В пакет Mpich был включен безопасный сервер (“secure server”), одной из задач которого являлась ликвидация этого недостатка. Но из-за неполной совместимости с операционными системами ряда RedHat (типа ScientificLinux) эту функцию безопасного сервера использовать не удалось. При запуске задания на локальном узле пакет Mpich-1 использует команды оболочки такие, как ‘.’ , `eval`, `exec`, которые не наследуют среду окружения запускающего процесса, что ведет к краху дочерних процессов;

- особенности реализации команды запуска параллельных заданий `mpirun` пакета Mpich-1. Запуск заданий на главном и подчиненных узлах существенно различаются — строки команды удаленного запуска (`rsh` или `ssh`) на подчиненных узлах дополняются служебными переменными. В результате стандартное расположение передаваемых в командной строке аргументов задания GAMESS-US нарушается и не распознается, что ведет к краху запуска.

В ИПХФ РАН с целью расширения функциональности применения пакета GAMESS-US в грид-сетях была поставлена задача разработки оригинальной конфигурации и сборки из исходных текстов исполняемого файла пакета GAMESS-US с использованием библиотеки MPI-2.

После установки библиотек MPI стандарта 2.0 (версия 1.0.3 пакета Mpich2) была проведена соответствующая модификация конфигурационных скриптов пакета GAMESS-US (`compddi, comp, compall, lked`), а также программных модулей `ddi_init.c` и `ddi_base.h`. Был полностью переписан соответствующий раздел в запускающем скрипте `rungms`, который сначала запускает кольцо серверов `mpd`, а затем уже и само задание. После сборки исполняемого файла было проведено его тестирование на включенных в пакет GAMESS-US примерах файлов данных и получено совпадение результатов. Запуск параллельных заданий осуществляется командой `mpiexec`, которая не имеет указанных выше недостатков команды `mpirun` (библиотеки MPI-1).

Использование модифицированной авторами библиотеки MPI позволило впервые из отредактированных исходных текстов свободно распространяемого квантово-химического пакета GAMESS-US получить исполняемое задание для работы в параллельной среде под управлением MPI. Выбранный авторами подход по созданию виртуального образа среды исполнения на основе перемещаемого пакета MPI-2 показал свою продуктивность и в этом случае. Была проведена компиляция модифицированных исходных кодов GAMESS-US с использованием библиотек MPI-2 и получен бинарный пакет. Затем была создана система компоновки необходимых системных файлов (библиотеки, исполняемые системные файлы), собственно модифицированного GAMESS-US, сопутствующих конфигурационных файлов и настроечных скриптов, файлов данных в единый «контейнер», выступающий в роли исходящего задания распределенной среды. Запуск подобного контейнера аналогичен описанному выше для прототипа.

Серия первичных запусков (с использованием внутренних тестовых примеров собственно пакета GAMESS-US) вплоть до получения положительного результата тестов (равно-значность поведения сокетных и MPI вариантов) была проведена на ресурсном грид-сайте ИПХФ РАН (grid-ce.icp.ac.ru) для сред gLite и Unicore (узлы использовались как удаленные ресурсы, т.е. запуск задач шел через грид-инфраструктуру). Дальнейшее успешное тестирование было проведено на удаленных ресурсных узлах RDIG в рамках ВО RGSTEST (узлы НИИЯФ МГУ, lcg38.sinp.msu.ru, среда gLite). Были проведены успешные запуски пакета GAMESS-US с применением данной технологии (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.

Технология запуска пакета GAMESS-US с использованием динамически формируемых параллельные сред исполнения интегрирована в высокоуровневые web-интерфейсы работы с пакетом GAMESS-US для трех вышеописанных грид-полигона, входящих в составе грид-портала ИПХФ РАН (http://grid.icp.ac.ru), что позволяет использовать данную технологию даже неискушенному пользователю.

4. Основные проблемы применения метода «виртуального контейнера» в распределенных средах

Было обнаружено, что на ряде кластеров (например, в Курчатовском РНЦ), предоставленных в качестве грид-ресурсов, запрещено или сильно ограничено использование скриптовых языков, что, естественно, противодействует запуску пришедшего на ресурс «виртуального контейнера». Для разрешения данной проблемы нами были проведены работы по переводу всех действий по развертыванию и настройке «контейнеров» в полностью бинарные исполняемые программы, которые действуют аналогично, но при этом не требуют доступа к shell языкам. Таким образом, впервые разработана технология запуска GAMESS-US в грид-средах в виде единого откомпилированного бинарного файла. При этом входящая задача порождает единственный процесс, который распаковывает библиотеки и бинарные системные файлы, собственно прикладной пакет, файлы данных, настраивает среду исполнения, в том числе параллельную среду mpich2, запускает параллельные процессы GAMESS-US, собирает полученные результаты, удаляет «мусор» и отправляет выходные данные на пользовательский интерфейс грид-среды.

На части доступных нам кластеров выявлено, что в качестве политики безопасности запрещена передача файлов между расчетными узлами по беспарольному ssh, что в какой-то мере может быть решено использованием общедоступных NFS ресурсов, но в таком случае «контейнер» должен конфигурироваться для каждого приложения по своему.

Не совсем корректно проводится мониторинг выполнения грид-задания, что осложняет контроль пользователя.

5. Выводы и перспективы

В результате применения описанной технологии пользователь получает единое *виртуальное* 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 (с использованием серии скриптов или в форме единичного бинарного процесса) на всех доступных пользователю полигона узлах ресурса грид, 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. Заметим, что данная технология применима для запуска подобных приложений и в условиях локальных кластеров без необходимости настройки расчетных узлов.

В более далекой перспективе одним из вариантов данной технологии является использование (по аналогии с описанным «контейнером») виртуальных машин (ВМ) как исходящих распределенных заданий, что позволит гарантировать пользователю необходимое качество обслуживания, не затрагивающее при этом работу основных служб ресурсных узлов. Таким образом, пользователю распределенной среды может быть предоставлена полностью изолированная виртуальная вычислительная среда, по своим свойствам не уступающая физическому серверу, в которой может быть предоставлен любой его собственный вычислительный сервис. Приложения, реализованные в ВМ, в этом случае абсолютно не зависят от операционной системы и окружения, в котором ВМ выполняется. Пользователь получает возможность создать образ виртуальной машины с предустановленной операционной системой и полностью сконфигурированными приложениями, нацеленной на решение конкретной задачи. Этот образ затем передается на распределенный ресурс и исполняется там как грид-приложение, не требуя настройки

данного узла под конкретные задачи. Это существенно облегчает адаптацию прикладного ПО для работы в распределенных средах. Дополнительным плюсом служит то, что данные технологии в принципе позволяют запускать образы виртуальных машин с операционными системами, отличными от установленных на ресурсах (например, Windows ВМ на Linux-кластере). Следует отметить потенциальные недостатки данного метода. Это размеры передаваемых заданий (может достигать первых гигабайтов), «накладные» расходы на виртуализацию (до 15-20% от мощности ресурса, при оптимальном конфигурации они могут быть снижены до уровня 5-7%), необходимость установки и настройки гипервизоров ВМ на расчетных узлах.

6. Заключение

Разработан метод создания динамически формируемых параллельных вычислительных сред в виде перемещаемых «виртуальных контейнеров», которые содержат: «персональные» копии необходимых системных файлов и библиотек, скрипты по настройке операционной системы, необходимые файловые «деревья», собственно приложение, файлы данных и т.п.. После динамического создания «контейнера» на пользовательском интерфейсе он средствами распределенной среды через брокер ресурсов как обычное грид-задание доставляется на удаленный ресурсный узел, «разворачивается» там, настраивает среду узла «под себя» и запускается как обычное параллельное приложение. По окончании работы приложения происходит «очистка» среды выполнения, возврат результатов на пользовательский интерфейс и приведение расчетных узлов в исходное состояние. Созданы два варианта «контейнеров»: с использованием скриптовых языков и как единого бинарного задания. В настоящее время этот метод применим для исполнения на ресурсных узлах, поддерживающих ОС системы Linux, т.е. типичных кластерах, интегрированных в грид-среды.

В качестве примера использован классический квантово-химический пакет GAMESS-US, для которого создан работоспособный «виртуальный контейнер» для сред gLite, Unicore, Globus GT4. Метод создания «контейнеров» интегрирован в высокоуровневые web-интерфейсы пакета GAMESS-US в составе грид-портала ИПХФ и реализован для трех российских грид-полигонов. Нет принципиальных ограничений для создания подобных «контейнеров» для других прикладных пакетов, требующих специфических параметров окружения и нестандартных настроек параллельных сред выполнения. Большинство проблем применения подобных «контейнеров» связано с политиками безопасности на ресурсных узлах, которые могут быть разрешены в рамках виртуальных организаций (ВО) различных грид-полигонов

Применение данного метода виртуализации приложений позволяет существенно расширить круг доступных грид-ресурсов для выполнения на них сложно сконфигурированных прикладных пакетов.

Литература

1. Волохов В.М., Варламов Д.А., Пивушков А.В., Покатович Г.А., Сурков Н.Ф. *Технологии грид в вычислительной химии* // «Вычислительные методы и программирование», М.: МГУ, 2010, т.11, № 1, с.175-182
2. Волохов В.М., Варламов Д.А., Сурков Н.Ф., Пивушков А.В. *Виртуальные вычислительные среды: использование на GRID полигонах* // Вестник ЮУрГУ, серия «Математическое моделирование и программирование», 2009, № 17 (150), вып. 3, с.24-35.
3. Д.А. Варламов, В.М. Волохов, А.В. Пивушков, Н.Ф. Сурков *Виртуализация параллельных приложений квантовой химии для запуска на ресурсных узлах распределенных сред* // 3-я Межд. науч. конф. «Суперкомпьютерные системы и их применение» SSA’2010, Минск, май 2010; Минск: ОИПИ НАН Беларуси, т.2, с.12-16
4. Д.А.Варламов, В.М.Волохов, Н.Ф.Сурков, А.В.Пивушков *Виртуализация вычислительной среды в грид* // "Параллельные вычислительные технологии 2010" ПаВТ-2010, (Уфа, март 2010), Челябинск, изд-во ЮУрГУ, с.63-70

5. В.М. Волохов, А.В. Пивушков, Н.Ф. Сурков, Д.А. Варламов, А.В. Волохов *Новые методы решения задач вычислительной химии в распределенных средах* // «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи» (20-25 сентября 2010 г., г. Новороссийск). – М.: Изд-во МГУ, 2010, с.181-184
6. 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

Применение технологии CUDA для задач голосовой биометрии на примере построения универсальной фоновой модели диктора

В.В. Габдуллин, А.И. Капустин, А.И. Королев

ООО «Центр Речевых Технологий»

В работе рассмотрено применение технологии CUDA для задач голосовой биометрии, разработаны параллельные алгоритмы вычисления универсальной фоновой модели диктора (UBM – universal background model), исполняемые на многопроцессорных системах и на графических процессорах (GPU) видеокарт NVIDIA с поддержкой технологии CUDA. Были проведены эксперименты для различных параметров модели, по их результатам был проведен сравнительный анализ скорости расчета UBM модели, точности вычислений (для GPU есть ограничения точности) и представлен прогноз ускорения расчетов при использовании нескольких видеоускорителей.

1. Введение

Задачи, связанные с определением пользователя по голосу, можно разделить на идентификацию и верификацию. В первом случае, задача состоит в классификации речевого сигнала, при этом каждый класс соответствует одному человеку, зарегистрированному в системе. Во втором случае, задача представляет собой бинарную классификацию. Более формально, задача верификации представляет собой проверку двух гипотез:

H_0: фразу Y произнес диктор S

и

H_1: фразу Y произнес НЕ диктор S.

Оптимальной проверкой для выбора одной из двух гипотез является отношение правдоподобия. При этом процедура принятия решения выглядит следующим образом:

$$\frac{p(Y \mid H_0)}{p(Y \mid H_1)} \begin{cases} \geq \theta, & H_0 \\ < \theta, & \overline{H}_0, \end{cases} \tag{1}$$

где p(Y|H)– функция плотности вероятности для гипотезы H, оцененная на речевом сегменте Y, а θ – порог принятия решения. Математически гипотеза H может быть определена моделью λ, которая характеризует диктора S в пространстве признаков.

В последнее время для верификации личности диктора по голосу применяется модель гауссовых смесей (GMM – gaussian mixture model) с использованием так называемой, универсальной фоновой модели (UBM – universal background model). Это наиболее широко распространенный и теоретически обоснованный подход, основанный на аппарате математической статистики. В данном методе, сложное распределение моделируется с помощью взвешенной суммы плотностей многомерных нормальных распределений (компонент смеси).

Для гипотезы H_1 строится универсальная фоновая модель, цель которой характеризовать всех возможных говорящих во всех возможных контекстах. Данная модель обучается на очень большом количестве речевых данных, сбалансированных по гендерному типу, а также по оборудованию и стандартным условиям [1].

Для существующих систем идентификации используются базы речевых данных в несколько сотен часов. При этом, обучение UBM модели может длиться не одну неделю на современном CPU, а существенное увеличение размера базы становится практически невозможным. Реализация параллельных алгоритмов обучения UBM модели может существенно ускорить этот процесс. Используя второй закон Амдала можно оценить максимальное ускорение вычислений для современного 4-х ядерного процессора (s=4), если 90% программы (β=0,1) будет выполняться параллельно[2]:

$$R = \frac{s}{\beta s + (1-\beta)} = \frac{4}{0{,}4 + 0{,}9} \approx 3{,}1\,. \tag{2}$$

Это существенное ускорение, однако, на сегодняшний день, для параллельных вычислений широко применяются графические процессоры общего назначения (GPGPU – general-purpose graphics processing units). На сайте компании NVIDIA представлены графики роста производительности CPU и GPU за последние несколько лет (рис. 1), которые иллюстрируют значительное превосходство современных GPU над CPU, а также перспективы развития этого направления[3].

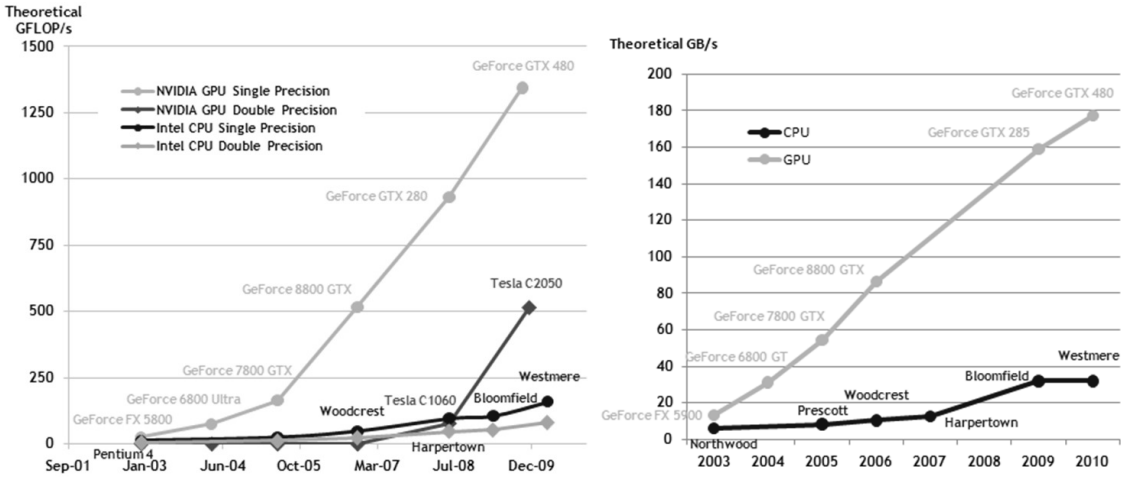


Рис. 1. Количество операций с плавающей запятой в секунду для CPU и GPU

Как видно из графика (рис. 1), максимальная производительность видеокарты GTX480 приближается к 1,5 TFLOP/s, в то время как самые быстрые процессоры только приближаются к отметке в 200 GFLOP/s. Теоретически, ускорение вычислений может составить до 40 раз по сравнению с последовательной версией алгоритма, если тот же код будет выполняться на GPU.

В статье представлены реализации параллельных алгоритмов выполняемые как на CPU, так и на GPGPU компании NVIDIA с поддержкой технологии CUDA (Compute Unified Device Architecture), основным преимуществом которой является ее простота – все программы пишутся на «расширенном» языке C, наличие хорошей документации, набор готовых инструментов, включающих профайлер, набор готовых библиотек, а также кроссплатформенность [4].

2. Алгоритм обучения GMM-UBM

Для D-мерного вектора признаков x, функция плотности распределения описывается следующей функцией:

$$p(x \mid \lambda) = \sum_{i=1}^{M} \omega_i p_i(x), \tag{3}$$

где $\boldsymbol{M}$ – количество компонент, $\boldsymbol{\omega_i}$ – вес $\boldsymbol{i}$-ой компоненты, а $\boldsymbol{p_i(x)}$- плотность распределения каждой компоненты, которая представляет собой D-мерный Гауссиан:

$$p_i(x) = \frac{1}{(2\pi)^{D/2}\left|\sigma_i\right|^{1/2}} \exp\left\{-1/2(x-\mu_i)'\sigma_i^{-1}(x-\mu_i)\right\}, \tag{4}$$

где μ_i – вектор математического ожидания, σ – ковариационная матрица.

Плотность распределения смеси Гауссиан полностью описывается параметрами компонент и значениями весов и представляются как $\lambda = \{\omega_i, \mu_i, \sigma_i\}$ [1].

Для определения параметров модели λ существуют несколько методов, наиболее распространенным из которых является метод максимального правдоподобия. Задача метода состоит в нахождении по заданным обучающим данным таких параметров модели, при которых функция правдоподобия модели достигает максимума.

Для последовательности из T обучающих векторов $X = \{x_1, x_2 ... x_T\}$, функция правдоподобия может быть записана как [1]:

$$p(X \mid \lambda) = \sum_{t=1}^{T} p(x_t \mid \lambda). \quad (5)$$

Прямая максимизация выражения (5) невозможна, так как функция от параметров λ нелинейная. Однако, приближенные значения могут быть получены с помощью алгоритма EM (expectation-maximization) используемого в математической статистике для нахождения оценок максимального правдоподобия параметров вероятностных моделей. Каждая итерация алгоритма состоит из двух шагов. На первом шаге (expectation) вычисляется ожидаемое значение функции правдоподобия (апостериорная вероятность того, что обучающий объект x_t получен из i-й компоненты смеси) [1, 5]:

$$\Pr(i \mid x_t) = \frac{\omega_i p_i(x_t)}{\sum_{j=1}^{M} \omega_j p_j(x_t)}. \quad (6)$$

На втором шаге (maximization) вычисляется оценка максимального правдоподобия для каждой компоненты модели:

$$n_i = \sum_{t=1}^{T} \Pr(i \mid x_t), \quad (7)$$

$$E_i(x) = \frac{1}{n_i} \sum_{t=1}^{T} \Pr(i \mid x_t) x_t, \quad (8)$$

$$E_i(x^2) = \frac{1}{n_i} \sum_{t=1}^{T} \Pr(i \mid x_t) x_t^2. \quad (9)$$

Затем вычисляются новые параметры модели, которые используются на первом шаге следующей итерации алгоритма [5]:

$$\omega_i = n_i / T, \quad (10)$$

$$\mu_i = E_i(x) + \mu_i, \quad (11)$$

$$\sigma_i^2 = E_i(x^2) + (\sigma_i^2 + \mu_i^2) - \mu_i^2. \quad (12)$$

Алгоритм выполняется до сходимости, но на практике часто ограничивают максимальное количество итераций.

3. Параллельное обучение UBM

Для построения UBM модели требуется большое количество обучающих векторов, при этом одна итерация может рассчитываться несколько часов, а общее количество итераций, необходимых для обеспечения сходимости, для этих данных составляет несколько десятков.

На втором этапе EM алгоритма для каждой из компонент вычисляется оценка максимального правдоподобия путем суммирования $\Pr(i \mid x_t)$. Таким образом, самый простой и надежный способ реализации параллельного алгоритма – разделить последовательность обучающих векторов на N частей и каждую из них вычислять отдельно, а затем сложить. Итоговая сумма, при этом, не изменится, но при этом практически 100% кода может выполняться параллельно. Для CPU – это идеальный вариант, расчет отдельных частей можно запустить в отдельных потоках, которые не требуют синхронизации и работают каждый со своими данными (рис. 2).

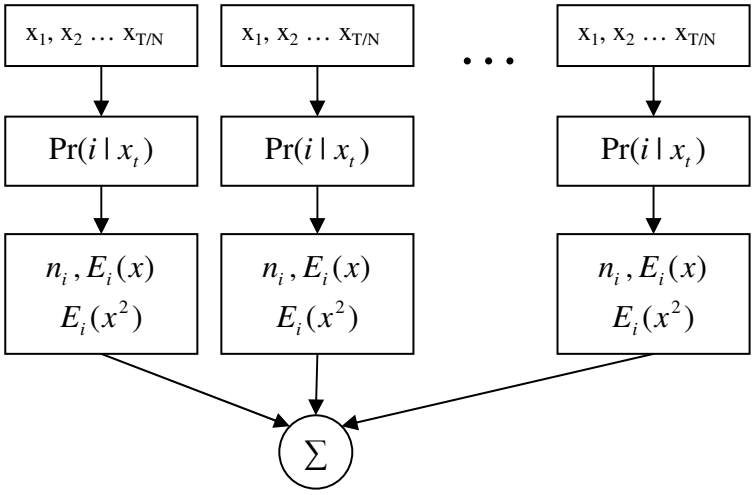


Рис. 2. Параллельное обучение UBM

Прямая реализация такого подхода на GPU может не дать ожидаемого ускорения, связано это с серьезными отличиями между GPU и CPU. CUDA строится на концепции, что GPU выступает в роли массивно-параллельного сопроцессора к CPU. Для решения задач CUDA использует очень большое количество параллельно выполняемых нитей (threads), при этом очень важно понимать, что между нитями на CPU и нитями на GPU есть принципиальные различия:

- нити на GPU обладают крайне небольшой стоимостью создания, управления и уничтожения (контекст нити минимален, все регистры распределены заранее);
- для эффективной загрузки GPU необходимо использовать много тысяч отдельных нитей, в то время как для CPU обычно достаточно 10-20 нитей.

Нити разбиваются на группы по 32 элемента, называемые warp'ами. Только нити в пределах одного warp'а выполняются физически одновременно. Нити из разных warp'ов могут находиться на разных стадиях выполнения программы, при этом управление warp'ами прозрачно осуществляет сам GPU[4]. Все запущенные на выполнение нити организованы в следующую иерархию (рис 3).

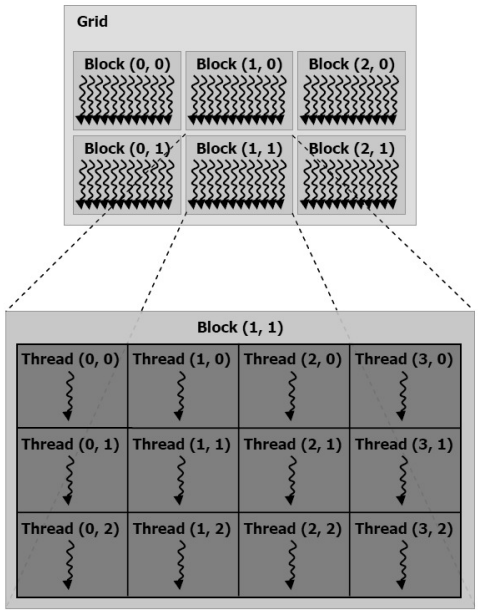


Рис. 3. Иерархия нитей в CUDA

Верхний уровень иерархии – сетка (grid) – соответствует всем нитям, выполняющим ядро (функцию, выполняемую на GPU) и представляет собой одномерный или двумерный массив блоков (block). Каждый блок, в свою очередь, состоит из нитей, организованных в одномерный, двухмерный или трехмерный массив. При этом все блоки, образующие сетку, имеют одинако-

вую размерность, а все нити могут взаимодействовать между собой только в пределах блока. Каждый блок получает в свое распоряжение определенный объем быстрой разделяемой памяти, которую все нити блока могут совместно использовать[3]. Отсюда вытекает еще одна особенность: организация памяти и работа с ней.

Основным местом для размещения и хранения большого объема данных, для обработки ядрами является, глобальная память, которая размещается в DRAM GPU. Поскольку глобальная память расположена вне GPU, то естественно, что она обладает высокой латентностью. Крайне важным является использование возможности GPU объединять несколько запросов к глобальной памяти в один. Также необходимо минимизировать количество обращений к глобальной памяти, за счет использования разделяемой памяти, размещенной непосредственно в самом мультипроцессоре[4].

Каждому блоку в сетке доступно 16 кБ быстрой разделяемой памяти (для GPU поколения Fermi – 48 кБ). Рассмотрим подробнее выражение (6). Для каждого обучающего вектора $\boldsymbol{x}$ рассчитывается $Pr(i|x_t)$. Если распределить вычисление между нитями так же как для CPU (рис. 2), то в пределах одного блока не получится использовать разделяемую память, т.к. каждая нить будет работать с независимыми данными[3]. В рамках технологии CUDA правильнее будет разделить вычисление разных обучающих векторов по блокам сетки. При этом каждый блок будет вычислять $Pr(i|x_t)$ для одного вектора, а нити будут обрабатывать каждый свою компоненту $\omega_i p_i(x_t)$. При такой организации вычислений можно будет существенно сократить обращение каждого потока к глобальной памяти, за счет разделяемой.

Результат вычислений необходимо суммировать, однако разделяемая память доступна только в пределах блока, следовательно, накопление необходимо производить в глобальной памяти. Для текущей версии CUDA (3.2) поддержка атомарной функции сложения доступна только для целых чисел, следовательно, для накопления значений с плавающей запятой придется разделить этап расчета (6) и этап накопления статистики (7, 8, 9) на 2 разных ядра, сохраняя промежуточные данные в глобальной памяти. На рисунке изображена структура обучения UBM на GPU.

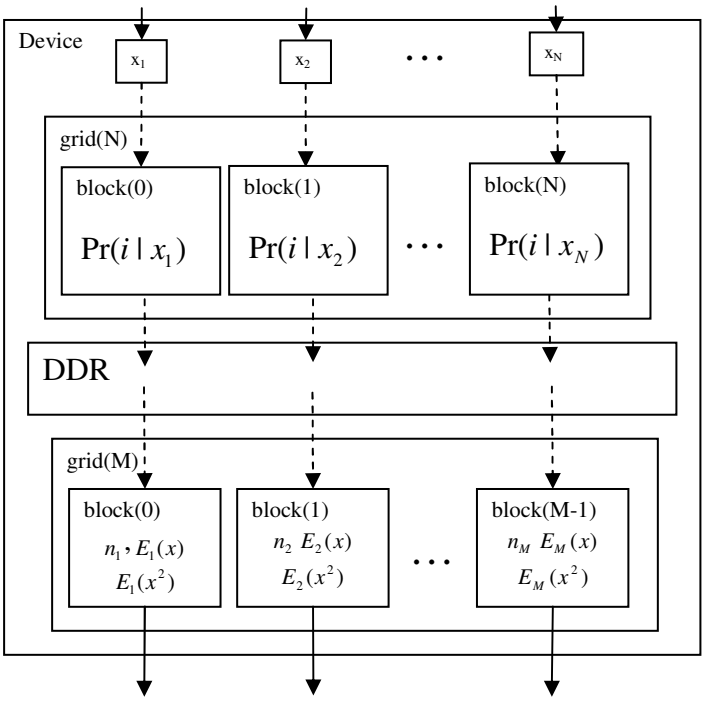


Рис. 4. Структура обучения UBM на GPU

Размер ковариационной матрицы σ и векторов математического ожидания μ_i составляет M*D*4 байта. Т.е. для 512 компонент и 60 признаков получится 122880 байт для каждого из параметров модели, следовательно, в быструю память можно поместить только обучающий вектор и промежуточные результаты вычислений $\omega_i p_i(x_t)$. Для вычислений в каждом блоке, в

этом случае, задействуется M*16+D байт разделяемой памяти. Максимальное количество нитей в блоке для GPU поколения Fermi составляет 1024, а максимум разделяемой памяти на блок 48 кБ. Прямое использование данного параллельного алгоритма позволит вычислять UBM для 1024 компонент, а небольшая модификация позволит получить UBM для 2048 компонент. Для более ранних поколений GPU компании NVIDIA существует ограничение в 512 потоков и 16 кБ памяти, в этом случае максимум 512 компонент может быть получено на GPU. На сегодняшний момент наиболее часто используют 512 компонент (количество признаков при этом варьируется), связано это, во многом, с долгим вычислением UBM и малым количеством данных для обучения.

4. Результаты испытаний

Сравнительные испытания проводились на 2-х испытательных стендах с видеокартами, поддерживающими технологию CUDA. Конфигурации стендов представлены в таблице 1.

Таблица 1. Конфигурации испытательных стендов

Блок	**X2_GTX285**	**X16_GTX480**
Процессор	Intel Core2 Duo E6550	Intel Xeon E5630 Intel Xeon E5630
Число ядер	2	16 (8 + hyper threading)
Объем RAM	2 ГБ	48 ГБ
Видеокарта	GeForce GTX 285	GeForce GTX 480
Число ядер CUDA	240	480
Разделяемая память	16 кБ	48 кБ

4.1. Сравнение быстродействия алгоритмов

Сравнения по быстродействию производились для 100 файлов содержащих в среднем по 10000 обучающих векторов с 39 признаками в каждом, общий объем данных, при этом, составил 150 Мб. Количество итераций было ограничено 10-ю. Из формул (4, 6, 7, 8 и 9) можно оценить количество операций с плавающей точкой (FLOP) необходимое для расчетов и общий объем данных участвующих в вычислениях. Для расчета UBM размерном 1024 гауссоиды необходимо ~8 TFLOP, а объем данных задействованных в вычислениях составляет ~9,6 ТБ. Видно, что объем данных сопоставим с количеством операций, следовательно, «узким местом» будет пропускная способность памяти, т.к. в быструю память GPU не помещается ничего кроме промежуточного результата расчетов. Всего для оценки скорости вычислений было проведено 3 испытания:

1. Размер UBM фиксировался 512-ю компонентами смеси. Изменялось количество потоков обработки для реализации на CPU от 1 до 16.
2. Размер UBM фиксировался 1024-я компонентами смеси. Изменялось количество потоков обработки для реализации на CPU от 1 до 16. Видеокарта GeForce GTX 285 не использовалась в этом испытании из-за ограничения по объему разделяемой памяти.
3. Для 16 потоков на CPU менялось количество компонент смеси для UBM от 32 до 1024.

В таблице 2 представлены результаты 1 и 2 эксперимента. В колонке «время» указано общее время работы программы, в колонке «расчеты», время, потраченное непосредственно на вычисления.

Таблица 2. Результаты испытания

	UBM 512 Xeon E5630		**UBM 512 Core 2 E6550**		**UBM 1024 Xeon E5630**	
Потоки	**Время, с**	**Расчеты, с**	**Время, с**	**Расчеты, с**	**Время, с**	**Расчеты, с**
1	1727	1716	2134	2111	3337	3328
2	725	718	1116	1105	1424	1416
4	423	415	1115	1094	842	830
8	235	225	1087	1054	458	441
16	165,7	155	1093	1043	352	333
	UBM 512 GTX 480		**UBM 512 GTX 285**		**UBM 1024 GTX 480**	
UBM	45,5	23,3	72,7	53,1	72,3	57,5

Из таблицы видно, что для видеокарты GTX 480 накладные расходы на чтение файлов и перенос обучающих векторов в память GPU составили 50%, при этом вычисления прошли почти в 74 раз быстрее, чем на одном ядре процессора Xeon E5630. Итоговое ускорение от реализации параллельных алгоритмов в первом испытании для 16 потоков оказалось равным 11 раз, При этом во втором испытании вычислениях видеокарта оказалась в 7,6 раз быстрее, чем 2 мощных 4-х ядерных процессора.

На видеокарте GeForce GTX 480 установлена память DDR5 с пропускной способностью 177 ГБ/с, следовательно, для расчетов теоретически понадобится минимум 54,2 секунды. На стенде X16_GTX480 установлена память DDR3-1066 работающая в 3-х канальном режиме, теоретическая пропускная способность при этом может достигать 25,6 ГБ/с, следовательно на CPU время затраченное на вычисления оставит порядка 376 секунд. Сравнивая теоретическое время расчетов с результатами испытания, можно оценить эффективность алгоритмов, для видеокарты эффективность использования памяти составила 94%, в то время как на процессоре за счет того, что на каждый из 16 потоков приходилось около 0,75 МБ cache-памяти, время оказалось меньше теоретического расчета. Эффективность использования вычислительных ресурсов GPU и CPU оказывается намного ниже, за счет того, что данные не помещаются целиком в быстрой памяти. Одно ядро процессора Xeon E5630 обладает вычислительным ресурсом в 40 GFLOPS, следовательно, 8 ядер в секунду могут производить 320 GFLOP, эффективность использования ресурса при этом составляет ~7,5% (30% без учета SSE команд). Эффективность использования вычислительных ядер GPU составляет ~12%.

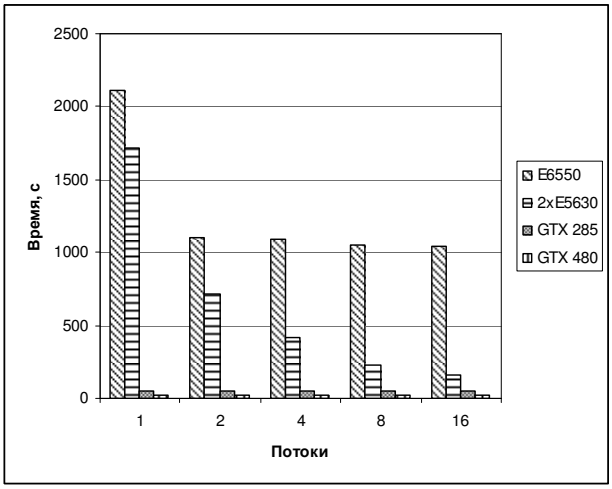


Рис. 5. Время выполнения расчетов

На рис. 5 изображена диаграмма, отражающая результаты испытания. Видно, что с ростом числа потоков вычисления ускоряются нелинейно, если между 1 и 2 потоками разница ровно 2 раза, то при изменении числа потоков с 8 до 16, для 2-х процессорной системы, ускорение составило всего 1,4 раза.

В таблице 3 представлены результаты последнего испытания, отображающие зависимость скорости вычислений от размера UBM модели и сложности вычислений. На CPU вычисления производились в 16 потоков, число нитей на GPU варьировалось от размера UBM.

Таблица 3. Результаты последнего испытания

	Xeon E5630		**GTX 480**		**GTX 285**	
UBM	**Время, с**	**Расчеты, с**	**Время, с**	**Расчеты, с**	**Время, с**	**Расчеты, с**
32	14,55	10,07	26,5	11,6	39,8	23,1
64	24,9	19,8	29,7	13,5	42	24,8
128	44,5	39,6	30,7	14,8	44,5	27,3
256	84,4	78,2	35	18,5	50,7	33,5
512	165,7	155	45,5	23,3	72,7	53,1
1024	352	333	72,3	57,5	–	–

Результат испытания виден на графике (рис. 6). С увеличением размера UBM модели, время выполнения расчетов увеличивается, однако зависимость нелинейная.

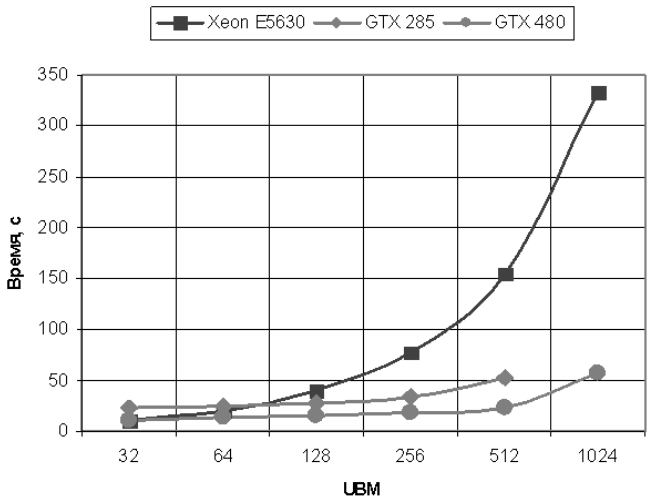


Рис. 5. Время выполнения расчетов

В начале, скорость растет плавно, даже для 2-х процессорной системы. Однако при большом количестве компонент время для CPU начинает расти нелинейно, это связано с заполнением кэш-памяти процессора, а так же с ограничением скорости RAM. Время вычислений на GPU вначале растет нелинейно, т.к. используются не все доступные вычислительные ядра. Время начинает расти линейно, когда количество нитей становится больше количества ядер GPU.

Испытания показали большую эффективность GPU при построении UBM модели, однако существуют серьезные ограничения в точности вычислений с использованием видеокарт. В частности GTX 285 имеет погрешность в последнем знаке при вычислениях с одинарной точностью, и вычисления с двойной точностью производятся только для 64-х бит. Тип данных long double не реализован на GPU, а для построения UBM существенную роль играет точность, т.к. объем данных может достигать десятков гигабайт.

4.2.Сравнение точности

Для сравнения точности GMM-UBM модель обучалась на базе мужских и женских голосов. Размер базы составил 146 часов (15,7 млн. обучающих векторов). Использовалось 512 компонент UBM, длина обучающих векторов составила 39 признаков. Обучение с использованием 20 итераций EM алгоритма длилось 17 часов 30 минут на одном ядре процессора, 2 часа 25 минут с применением параллельной версии алгоритма на 16 потоках и 28 минут на видеокарте GeForce GTX 480 (при этом сами расчеты заняли всего 20 минут). Полученные модели испытывались на 4-х базах: мужские и женские голоса, записанные с микрофона, а так же мужские и женские голоса, записанные в плохих условиях.

В качестве критерия качества использовалась величина равновероятной ошибки EER, которая определяется как значение функций FR (false reject) и FA (false accept) в точке, где они принимают одинаковые значения [6]:

$$EER = FR(\theta) = FA(\theta), \tag{13}$$

где θ – порог принятия решения алгоритма идентификации.

Функция FR определяется как вероятность ложного отклонения «своей» фонограммы:

$$FR(\theta) = \frac{N_{imposter(error)}(\theta)}{N_{target}} \cdot 100\%, \tag{14}$$

где N_{target} – количество сравнений вида «свой-свой»;

$N_{imposter(error)}(\theta)$ – количество сравнений вида «свой-свой», идентифицированных как «свой-чужой», в зависимости от порога.

Функция FA – вероятность ложного принятия «чужой» фонограммы.

$$FA(\theta) = \frac{N_{target(error)}(\theta)}{N_{imposter}} \cdot 100\% , \quad (15)$$

где $N_{imposter}$ – количество сравнений вида «свой-чужой»;

$N_{target(error)}(\theta)$ – количество сравнений вида «свой-чужой», идентифицированных как «свой-свой», в зависимости от порога.

Однопоточный и параллельные алгоритмы показали абсолютно идентичный результат на различных данных, так что в эксперименте проверялось отличие результатов между версиями на CPU и на GPU.

Т.к. вычисления с двойной точностью на видеокарте выполняются в несколько раз медленнее, чем на процессоре [3], часть вычислений было решено перевести на одинарную точность. Было реализовано две версии алгоритма на GPU:

1. Все вычисления производились с одинарной точностью;

2. Все этапы вычисления функции $p_i(x_t)$ с одинарной точностью (основная вычислительная нагрузка), остальные этапы обучения – с двойной точностью.

Алгоритм с одинарной точностью для большого количества обучающих векторов неприменим, т.к. при вычислении $\Pr(i \mid x_t)$ отдельные компоненты могут обращаться в 0, однако для базы, используемой в испытаниях, хватает одинарной точности.

Таблица 4. EER на тестовых базах

База	**EER(CPU, float), %**	**EER(GPU, float),%**	**EER(GPU, double), %**
Микрофон мужчины	4,4	4,2	4,2
Микрофон женщины	3,4	4,4	4,2
Телефон мужчины	12,6	11,6	11,6
Телефон женщины	10,5	10,4	10,6

В таблице 4 приведены результаты испытания для алгоритма, реализованного на CPU и алгоритмов на GPU. На некоторых базах UBM, обученная на GPU, показала значение EER лучше, чем обученная на CPU, на других хуже. Результаты приведены для двух различных каналов: микрофонный, с достаточно хорошим качеством, и аналоговый телефон, с сильными уровнем шума в нем.

В среднем ошибка почти не изменилась, все значения лежат в пределах доверительного интервала — для телефонного канала он составляет 0,5%, для телефонного 1%, при этом увеличив размер базы в 20 раз (316 млн. обучающих векторов) удалось снизить EER для микрофонного канала до ~2,4%.

5. Заключение

Применение технологии CUDA для обучения GMM-UBM модели диктора показало высокую эффективность использования GPGPU вычислений для задач голосовой биометрии. Видеокарта GeForce GTX480 позволяет получать UBM в 46 раз быстрее, чем последовательный алгоритм, выполняемый на CPU, и в 6 раз быстрее параллельного алгоритма на 2-х процессорной 16-и ядерной системе, за счет высокой пропускной способности памяти. При этом снижение точности вычислений не приводит к существенному снижению надежности биометрической системы, а сокращение времени расчетов позволяет использовать большой объем данных для обучения UBM и, как следствие, увеличить надежность системы почти в 2 раза.

В дальнейшем, засчет оптимизация чтения данных с жесткого диска, можно существенно сократить время обучения UBM на GPU, т.к. для 512 компонент GMM затраты на чтение обу-

чающих векторов составляют 50% от общего времени вычислений. Использование нескольких видеокарт позволит линейно наращивать скорость вычислений, т.к. каждое устройства сможет производить независимые вычисления над собственными обучающими векторами. Так же можно повысить эффективность алгоритма на GPU за счет помещения части данных в память констант и быструю память.

Литература

1. Reynolds D.A., Quatieri T.F., Dunn R.B., Speaker Verification Using Adapted Gaussian Mixture Models, Digital Signal Process. 10 (2000), pp 19-41.
2. В.В.Воеводин "Вычислительная математика и структура алгоритмов."**.-**М.: Изд-во МГУ, 2006.-112 с.
3. NVIDIA CUDA C Programming Guide [Электронный ресурс].– режим доступа: http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf
4. Боресков А.В. Харламов А.А. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010, – 232с.: ил. ISBN 978-5-94074-578-5
5. I.H. Witten, E. Frank Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). - Morgan Kaufmann, 2005 ISBN 0-12-088407-0
6. G.R. Doddington, M.A. Przybochi, A.F. Martin D.A. Reynolds. The NIST Speaker Recognition Evaluation – Overview, Methodology, Systems, Results, Perspective. p13.

Современные методы разработки программ для 3D-моделирования задач плазмодинамики (плазменной мультифизики)*

В.А. Гасилов, Г.А. Багдасаров, А.С. Болдарев, С.В. Дьяченко, Е.Л. Карташева, О.Г. Ольховская, С.Н. Болдырев, И.В. Гасилова

Институт прикладной математики им. М.В. Келдыша РАН

Представлен опыт использования современных технологий коллективной разработки прикладных программ, накопленный при создании в ИПМ им. М.В. Келдыша РАН комплекса программ MARPLE, предметной областью которого являются течения ионизированного газа (плазмы), взаимодействующего с магнитным полем. Комплекс MARPLE сконструирован как программное обеспечение для высокопроизводительных параллельных вычислительных систем. Дано общее описание инфраструктуры комплекса MARPLE. Комплекс основан на модели радиационной плазмодинамики, описывающей совокупность нелинейных физических процессов (плазменная "мультифизика"). Рассмотрено моделирование сжатия токонесущей плазменной оболочки собственным магнитным полем.

1. Введение

Компьютерное моделирование — один из важнейших методов анализа и оптимизации в тех случаях, когда исследования выполняются на основе комплексных физических моделей, называемых моделями "мультифизики". В вычислительной физике плазмы модели такого рода востребованы в весьма высокой степени. Появление высокопроизводительной параллельной техники открывает возможность создания численных кодов, которые могут быть использованы для прогноза результатов экспериментов с высокотемпературной плазмой, выполняемых в целях фундаментальных исследований, управляемого термоядерного синтеза и развития наукоемких технологий.

Темпы исследований в современной физике плазмы очень высокие. Для сопровождения экспериментов необходимо создание программного обеспечения (ПО), которое можно быстро адаптировать к изменившимся условиям и которое содержит средства, упрощающие работу с комплексными моделями, алгоритмами и данными различной структуры. Область применения "параллельного" кода должна быть широкой, чтобы сделать его востребованным для многих исследователей и тем самым оправдать затраты на его создание. Поэтому качество современного ПО для научных исследований должно быть на уровне "индустриальных" кодов, так же как и поддержка труда разработчиков и сопровождение программ. Сложные исследовательские коды, как правило, создаются путем коллективной разработки. Перечисленные требования заставляют обратить особое внимание на архитектуру кода и технологии программирования.

В статье описывается процесс разработки прикладного программного комплекса (кода) MARPLE [1], созданного к настоящему времени в ИПМ им. М.В. Келдыша РАН. Код предназначен для трехмерного (3D) моделирования процессов в импульсной высокотемпературной плазме. Он представляет собой совокупность солверов систем уравнений базовых физико-математических моделей и вычислительной среды (инфраструктуры) для проведения расчетов начально-краевых задач математической физики на массивно-параллельных системах с распределенной памятью.

*Работа выполнена при поддержке программы 3-ОМН РАН, грантов РФФИ 09-01-12109-офи-м, 09-02-01532-а, 10-02-00063-а, 10-02-00449-а, а также Исследовательского Центра Грама (Франция). Расчеты выполнены на компьютерах СКИФ МГУ и МВС-100К МСЦ РАН.

В основу кода MARPLE положена модель плазмы С.И. Брагинского [2], включающая трехмерное описание гидродинамики плазмы в магнитном поле. Энергетический баланс плазмы учитывает эффект “двухтемпературности”, т.е. энергетическую неравновесность электронов и ионов. Учтен ряд важных эффектов, необходимых для описания высокочастотных процессов (эффект Холла, возникновение термоэлектродвижущих сил и т.д.). Модель дополнена описанием лучистого обмена энергией, играющего важную роль в высокотемпературной плазме [3].

В предметную область кода MARPLE в первую очередь входят задачи, связанные с сильноточными импульсными разрядами [4]. Для их решения основная система уравнений может быть дополнена моделью протяженного по времени плазмообразования [5], электротехническим уравнением для электрического тока во внешней электрической цепи (в типовых задачах импульсной энергетики полная цепь включает генератор, подающие линии и разрядную камеру с плазмой) и расчетно-экспериментальными базами данных в форме функциональных или табличных зависимостей (нами использовались данные, полученные с помощью кода TERMOS [6], и широкодиапазонные уравнения состояния вещества [7]).

Реализация этих моделей и организация вычислительных экспериментов осуществляется посредством современных численных методов. Моделирование сложной геометрии экспериментальных и промышленных установок, а также пространственно-разномасштабных физических процессов, выполняется на расчетных сетках нерегулярной структуры, в том числе блочных. Допустимы сетки, содержащие разнородные (тетраэдры, пирамиды, гексаэдры и треугольные призмы) и криволинейные элементы. Для описания таких сеток используется аппарат топологических комплексов [8]. Разработанные и реализованные в MARPLE структуры данных предназначены для генерации трехмерных расчетных сеток, построения на них разностных схем и поддержки распределенных вычислений.

Разработаны оригинальные методики для численного решения уравнений в частных производных на таких сетках. Фундаментальным требованием к технике аппроксимации является консервативность результирующей компьютерной модели. Для бездиссипативной подсистемы полной системы уравнений МГД-модели использовано семейство полностью явных схем, основанных на реконструкциях решения TVD (ограничение полной вариации) [9] повышенной точности с коррекцией потоков. Для подсистемы, описывающей диссипативные процессы, применяются неявные конечно-элементные схемы на основе метода Галеркина с разрывными базисными функциями [10]. Построенные схемы используют средние значения величин по ячейкам вместо узловых значений величин, что позволяет отказаться от рассмотрения трехмерных контрольных объемов сложной формы вокруг узлов сетки; в качестве ячеек консервативности выступают непосредственно ячейки расчетной сетки.

Для поддержки работы с разнообразными физическими моделями, в том числе для обеспечения возможности их модификаций, используется универсальный принцип суммарной аппроксимации, выражающийся в организации вычислений путем последовательного учета физических процессов. Данный принцип делает возможным построение алгоритма продвижения по времени, допускающего независимый расчет всех уравнений, включенных в основную систему. Более того, алгоритм допускает использование на одном временном шаге явных (для гиперболической подсистемы) и неявных (для параболической/эллиптической подсистем) аппроксимаций по времени. Для повышения порядка аппроксимации по времени используется двухступенчатая схема “предиктор-корректор”.

Для выполнения расчетов на массивно-параллельных системах с распределенной памятью применяется геометрический параллелизм, основанный на декомпозиции расчетной области (разбиении ее на вычислительные подобласти, каждый из которых обрабатывается отдельным процессорным ядром системы).

В настоящее время код находится в стадии тестирования и пробной эксплуатации.

2. Технологии разработки кода MARPLE

В разработке любой программы можно выделить следующие этапы:

- анализ требований,
- проектирование,
- реализация,
- тестирование,
- выпуск и поддержка.

Анализ требований к программе является самым важным этапом разработки. Здесь закладывается фундамент будущей программы. Должны быть учтены не только текущие требования к программе, но возможность ее модификации в будущем, от чего зависит ее жизненный цикл. Главным результатом этого этапа является список согласованных требований к программе — спецификация.

Проектирование представляет собой формализацию результатов анализа, при которой требования к программе формулируются на языке предметной области в виде алгоритмов (процедурное проектирование) или диаграмм взаимодействующих друг с другом объектов, являющихся экземплярами соответствующих классов, в свою очередь образующих иерархию (объектно-ориентированное проектирование). На этом этапе выбирается язык разработки, вырабатываются правила программирования и оформления кода. Подробно о проектировании и о выбранном нами стиле проектирования рассказано в разделе 2.2.

Реализация заключается в переносе формально записанных требований к программе (в виде алгоритмов или объектов/классов) на какой-либо язык программирования. В отличие от предыдущих этапов, которые выполняют один-два человека (менеджер, главный программист), реализация осуществляется большим количеством разработчиков.

Координацию действий отдельных разработчиков в нашем Проекте мы осуществляли с помощью известных, но малоиспользуемых в научной среде инструментов коллективной разработки, таких как система контроля версий, системы управления заданиями и ошибками, средства общения разработчиков (форум, списки рассылки, wiki). Интегрированные системы управления разработкой предоставляют единый интерфейс для перечисленных средств. Средствам коллективной разработки посвящен раздел 2.3.

Тестирование является, наверное, самым сложным этапом разработки. В общем случае оно включает верификацию и валидацию. Верификация разработанного ПО — это формальная проверка работоспособности программы на различных входных данных, часто не имеющих физического смысла. Этот этап очень важен для проверки базовых модулей, фундамента программы. Валидация ПО является специфической процедурой как относительно типа программы, так и начальных требований, и подробно обсуждается на этапе анализа требований. Валидация исследовательских кодов заключается в проведении тестовых расчетов и сравнении полученного решения с известными данными — аналитическими оценками, численными решениями других авторов, результатами экспериментов и наблюдений. Валидация MARPLE и ему подобных исследовательских кодов весьма важна, во-первых, поскольку они предназначены для предсказания результов еще не состоявшегося эксперимента, во-вторых, ввиду того, что многие модели физики плазмы сами находятся в стадии развития, в том числе и на основе вычислительных экспериментов.

Тестирование программных комплексов выполняется пошагово: тестирование компонент, интеграция, тестирование комплекса. Для каждой компоненты и комплекса необходимо выполнить и верификацию, и валидацию.

Поддержка предлагает обратную связь с пользователями программы: составление документации, обучение, исправление ошибок, улучшения. Мы считаем, что для исследовательских кодов наиболее важным является написание качественной документации для кода в целом, для применяемых алгоритмов и схем, а также для программной реализации. Инструменты для составления и поддержки документации обсуждаются в разделе 2.3.

2.1. Модели разработки

Перечисленные этапы разработки в том или ином виде присущи процессу создания любого ПО, от простейшего калькулятора до операционной системы, разница проявляется лишь во времени, затрачиваемом на каждый этап, и в последовательности этапов.

Нами была выбрана итерационная модель разработки, как наиболее рациональная для создания исследовательского кода. Она использует в качестве контролирующего механизма обратную связь, а не планирование, и предназначена для задач с меняющимися или неполными требованиями. В итерационных моделях последовательность этапов разработки повторяется несколько раз: каждый следующий цикл уточняет, совершенствует и расширяет уже имеющийся программный продукт. Это позволяет не только выделить действительно важные требования к продукту, но и изменять и/или исправлять архитектуру программы.

Водопадная модель, подразумевающая однократное последовательное выполнение этапов разработки, в нашем случае неприменима, потому что требования к коду могут меняться на протяжении жизненного цикла программы. Эта модель разработки ПО пригодна для создания систем с фиксированным набором требований.

Техника экстремального программирования, одна из самых известных модификаций итерационной модели разработки, в которой для ускорения итерационного процесса используются такие приемы, как разработка через тестирование, парное программирование, рефакторинг, коллективное владение кодом и другие, также для нас не подходит, поскольку невозможно организовать разработку через тестирование ввиду сложности и слабой формализованности физических тестов. В то же время идеи коллективного владения кодом (каждый разработчик может вносить изменения в любой участок кода) и рефакторинг (рецензирование кода для улучшения структуры и устранения избыточности) не противоречат итерационной модели и успешно используются в нашей работе.

2.2. Объектно-ориентированное проектирование

Основным и универсальным принципом борьбы со сложностью является принцип “разделяй и властвуй”. Разбить большую сложную задачу на ряд более мелких подзадач можно двумя путями: алгоритмическое разбиение, основанное на порядке выполнения процедур, и объектное, когда выделяются объекты и субъекты некоторых действий. Исходя из специфики решения исследовательских задач в определенных сложных пространственных областях мы выбрали объектно-ориентированный (ОО) подход. Многообразие геометрических данных наиболее естественным образом описывается на языке объектов. Кроме того, ОО-проектирование предоставляет удобные средства организации кода для обеспечения многовариантности вычислительных моделей и методов. Таким образом, в нашем проекте, наряду с геометрическими объектами используются объекты вычислительные — солверы, аппроксимации, граничные условия и др.

Объектной декомпозиции задачи соответствуют два первых шага разработки программной системы — ОО анализ и проектирование. В основе ОО-проектирования лежит представление о том, что программную систему необходимо проектировать как совокупность взаимодействующих друг с другом объектов, рассматривая каждый объект как экземпляр определенного класса, причем классы образуют иерархию. Для наглядного представления и проектирования взаимодействия объектов и иерархий классов используется унифицированный язык моделирования (Unified Modeling Language — UML). Подробно структура классов MARPLE представлена в разделе 3.1.

2.2.1. Объектно-ориентированное программирование

Для создания программ по объектной модели необходима соответствующая поддержка со стороны языка разработки. Для того, чтобы язык программирования мог считаться

объектно-ориентированным, необходимо выполнение следующих условий [11]:

- поддержка объектов, т.е. абстракции данных, имеющих интерфейс в виде именованных операций и собственные данные, с ограничением доступа к ним;
- объекты должны принадлежать к соответствующим типам (классам);
- типы (классы) могут наследовать атрибуты супертипов (суперклассов).

Для проекта MARPLE наиболее подходящей технологией нам представляется ОО программирование на языке C++. Наш опыт использования языка C++ показал, что аппарат производных классов и виртуальных функций является весьма эффективным средством для решения прикладных вычислительных задач с использованием техники нерегулярных расчетных сеток.

Адаптируемая к конкретным условиям архитектура ПО, базирующаяся на программировании средствами C++, предоставляет возможность быстрого обновления кода и делает его удобочитаемым. Разрабатываемое ПО должно быть совместимо с различными операционными системами (MS Windows, GNU/Linux, BSD, Mac OS), работать на различных платформах (персональные компьютеры, мощные рабочие станции, ЭВМ коллективного пользования — включая параллельный процессинг), для которых имеются компиляторы C++ стандарта 2003 года. Развитое прикладное ПО на языке C и C++ совместимо с перечисленными операционными системами и платформами.

2.3. Коллективная разработка

2.3.1. Стандарты программирования

Одним из основных приемов при коллективной разработке является выработка стандартов программирования и единого стиля оформления кода. Следование этим двум стандартам позволило достичь таких важных целей, как переносимость и коллективное владение кодом. Коллективное владение кодом означает, что любой разработчик может легко разобраться и при необходимости исправить код, написанный другими. Облегчается включение в команду новых разработчиков и освоение ими наработанного материала. Переносимость видится нам еще более важной целью, особенно для исследовательских кодов, целевой платформой которых являются высокопроизводительные кластеры с заранее неизвестной архитектурой и установленным программным обеспечением.

2.3.2. Система сборки

Описание проекта в общем случае представляет из себя список исходных файлов и опций, необходимых компилятору и линковщику для сборки исполняемого файла. Для крупного проекта таких опций может потребоваться не один десяток (оптимизирующие опции, подключаемые библиотеки и другие). При коллективном создании ПО разработчиками могут использоваться различные аппаратные и программные платформы, а также инструменты для сборки. Все это порождает огромное разнообразие в необходимых опциях и даже в самих компиляторах и линковщиках. Поддержка проектных файлов для всех используемых конфигураций в актуальном состоянии не представляется возможным. Для решения этой проблемы существуют кросс-платформенные системы сборки, такие как GNU Autotools, SCons/waf, qmake, CMake и другие, которые предоставляют унифицированное описание проекта в том или ином виде.

Стандартной системой сборки проекта MARPLE была выбрана система CMake. Файл проекта CMake представляет собой простой текстовый файл с инструкциями на специальном достаточно простом языке макросов. CMake поставляется с большим количеством поисковых макросов, сильно упрощающих запись зависимостей проекта от внешних библиотек и компонентов. Имеется графический интерфейс, предоставляющий наглядный доступ ко всевозможным опциям проекта.

CMake не занимается непосредственно сборкой, а лишь создает файлы управления сборкой для различных платформ и сред разработки. Таким образом, разработчики могут использовать привычные им средства (IDE, компиляторы и другие) для работы с проектом независимо друг от друга.

2.3.3. Система управления версиями

Системы управления версиями (VCS — Version Control System) предназначены для ведения истории развития электронных документов. Они позволяют хранить несколько версий одного и того же документа, легко получать доступ к другим версиям и находить отличия между версиями, определять авторство сделанных изменений. VCS отслеживают конфликты, возникающие при изменении одного файла несколькими разработчиками, и предлагают средства их решения. Поддержка ветвления и слияния процесса разработки открывает еще больше возможностей как для разработчиков, так и для пользователей. Разработчики могут создавать ветки кода для экспериментирования — например, для реализации новой возможности или рефакторинга кода — не мешая другим разработчикам.

В качестве основной VCS для исходных кодов проекта MARPLE была выбрана система Subversion. Это современная и при этом стабильная система, обладающая многими полезными свойствами (кросс-платформенность, атомарность транзакций, поддержка свойств файлов и др.) и подходящая для разработки программ небольшим коллективом.

Репозиторий проекта MARPLE включает общепринятую иерархию каталогов для хранения исходных файлов и файлов проекта:

- trunk — основная ветка проекта (общая версия, основная для разработчиков);
- branches — альтернативные ветви проекта (специальные рабочие версии, которые планируется в дальнейшем объединить с основной веткой проекта);
- tags — неизменяемые, официально выпущенные и протестированные сборки MARPLE, которые можно считать релиз-версиями.

2.3.4. Документация

Для MARPLE, как и для большинства исследовательских кодов, характерно большое число настроечных параметров, заданных во внешних файлах или внутри программного кода. Наличие подробного руководства для запуска, с примерами, является необходимым условием для распространения кода как за пределами команды разработчиков, так и для внутреннего пользования.

Описание различных программных решений является необходимым условием для коллективного владения кодом разработчиками. Это помогает при отладке и локализации ошибок, рефакторинге, ускоряет обучение новых разработчиков. Документирование программных решений часто оформляется в самом коде в виде комментариев. При следовании определенным правилам оформления комментариев становится возможным автоматически извлечь из них информацию и сформировать полное описание программного кода в распространенных форматах (HTML, RTF и другие).

Мы пользуемся кросс-платформенной системой Doxygen, которая является общепринятым программным средством для автоматического создания документации по исходному коду. Doxygen создает документацию на основе комментариев к исходному коду программы, оформленных в специальном стиле, и может быть настроен на извлечение и сохранение структуры программы (графы зависимостей программных объектов, диаграммы классов и исходных кодов с гиперссылками). Doxygen поддерживает множество языков программирования (C, C++, Fortran, Python, Java и др.) и имеет встроенную поддержку создания документации в формате HTML, LaTeX, man, RTF и XML. Опыт разработки MARPLE свидетельствует в пользу формата HTML — работа в нем значительно проще для объемной документации с множеством перекрестных ссылок.

2.3.5. Система управления проектом

Система управления проектом — это комплекс программ, включающий приложения для планирования задач, составления расписания, контроля цены и управления бюджетом, распределения ресурсов, совместной работы, общения, быстрого управления, документирования и администрирования системы. Различают несколько типов таких систем: desktop- и web-ориентированные, персональные, одно- и многопользовательские, а также интегрированные. На рынке ПО представлено огромное число реализаций каждого типа систем управления проектом, как коммерческих, так и имеющихся в свободном доступе. Выбор конкретного инструмента зависит от многих факторов: численности и распределения коллектива разработчиков, требований к разрабатываемой системе, имеющихся ресурсов и других.

В качестве системы управления проектом MARPLE была выбрана web-ориентированная система trac, предназначенная для разработки ПО небольшим коллективом. К важным для нас преимуществам этой системы можно отнести простоту установки, конфигурирования, администрирования, гибкость, наличие большого числа специальных плагинов, интеграцию с любыми современными VCS, простоту освоения и работы.

Для организации работы выбранных нами средств (Subversion, trac) был создан специальный GNU/Linux-сервер, предоставляющий доступ к хранилищу исходных текстов (репозиторию) MARPLE и системе trac. Наиболее используемыми нашим коллективом возможностями trac стали: интеграция с Subversion, позволяющая быстро просматривать историю кода и конкретные изменения, совместное редактирование документов (wiki) для оформления отчетов, и управление заданиями/ошибками (билетами). Последнее для нас оказалось чрезвычайно эффективным средством, помогая разрешению проблем, требующих участия нескольких разработчиков.

3. Архитектура кода MARPLE

Проект разработки кода для проведения параллельных расчетов начально-краевых задач математической физики в трехмерной постановке был начат в ИММ РАН и продолжается по настоящее время в ИПМ им.М.В. Келдыша РАН [1]. Разработка кода изначально велась с применением обсуждавшихся в разделе 2 технологий разработки.

3.1. Проектирование кода MARPLE

Основные идеи, заложенные при создании MARPLE (как абстрактной вычислительной среды, так и солверных компонентов) это:

- реализация большого числа сервисных функций (ввод-вывод данных, операции с расчетными сетками, поддержка параллельных вычислений, динамическая работа с вычислительными объектами) на уровне вычислительной среды, что позволяет создавать физические солверы с минимальными затратами ресурсов и обеспечивает унификацию их интерфейсов;
- автоматическое управление динамическим созданием и уничтожением вычислительных объектов разных типов, таких как солверы уровней вычислительного домена, физической подобласти и элементарные солверы, аппроксимаций, граничных условий.

В соответствии с этими идеями и принципами объектно-ориентированного проектирования предметная область была разбита на модули: аппроксимация, работа с сетками, солверы, граничные условия и уравнения состояний. Иерархия классов в модулях, а также отношения между ними, представлены на рис. 1 (показаны только существенные классы и отношения, рамками выделены модули).

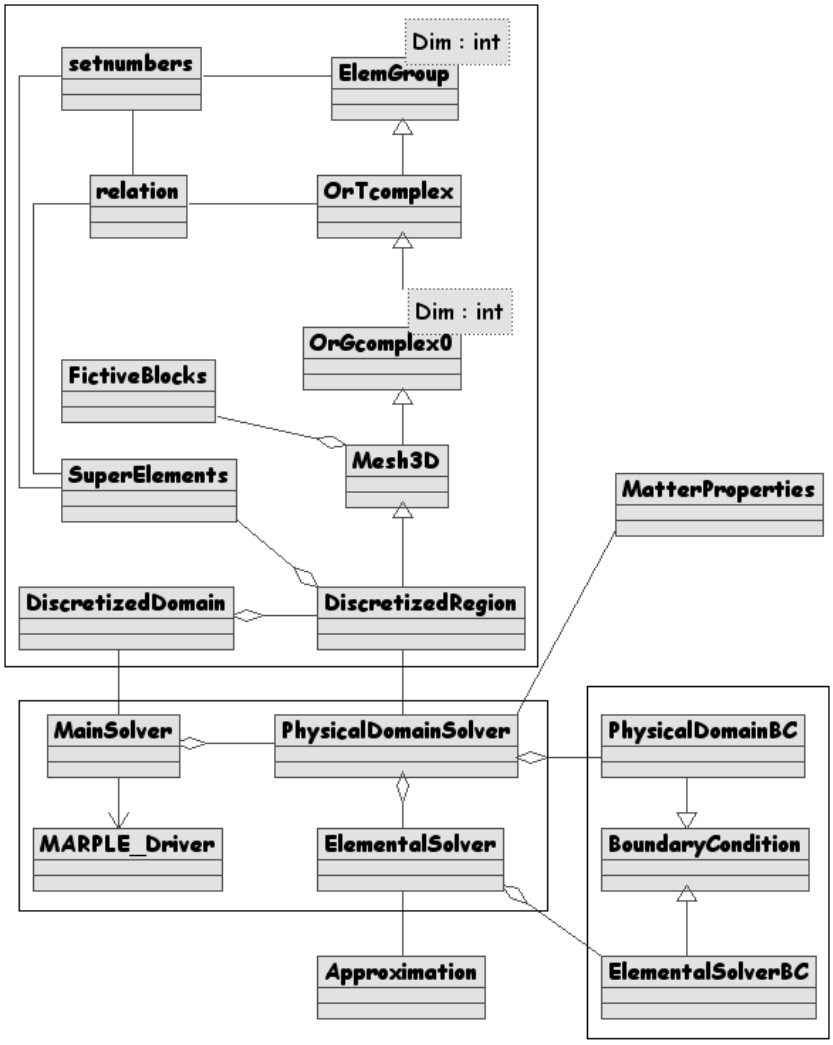


Рис. 1. Диаграмма классов UML: архитектура MARPLE.

- главные солверы (MainSolver) — совокупность физических солверов, выполняют операции верхнего уровня с физическими подобластями, реализуют схему суммарной аппроксимации и схемы аппроксимации по времени;
- физические солверы (PhysicalDomainSolver) — совокупность элементарных солверов, решают систему уравнений в одной физической подобласти;
- элементарные солверы (ElementalSolver), отвечают за решения какого-либо уравнения или системы уравнений;
- элементарные граничные условия (ElementalSolverBC), свои для каждого элементарного солвера, учитывают специфику решаемых элементарным солвером уравнений на границе;
- сложные граничные условия (PhysicalDomainBC) уровня физической подобласти, используются только в том случае, когда не удается разложить заданное граничное условие на множество элементарных граничных условий, находящихся ниже в иерархии (ElementalSolverBC);
- аппроксимации (Approximation), предоставляют унифицированный доступ ко всем необходимым для работы элементарных солверов аппроксимациям;
- свойства вещества (MatterProperties), предоставляют унифицированный доступ к уравнениям состояния для каждой физической подобласти.

Выделение аппроксимаций в отдельный модуль с унифицированным интерфейсом позволяет различным вычислительным объектам (солверам и самим аппроксимациям) пользоваться одними и теми же объектами для схожих операций. Совместно используемая база аппроксимаций MARPLE обеспечивает автоматизированное управление коллективным доступом к вычислительным объектам, предотвращение излишних затрат памяти и ее возможных утечек, возможность динамических замен вычислительных объектов, упрощение и унификацию сборки кода.

В коде используются смешанные сетки нерегулярной структуры. Многоуровневая иерархия классов дикретной модели, представленная на рис. 1, основана на аппарате топологических комплексов [8] для описания таких сеток. Обобщенная реализация классов позволяет применять их для описания дискретизаций областей любой размерности.

Поддержка геометрического параллелизма встроена в дискретную модель в виде блоков фиктивных элементов (FictiveBlocks и Mesh3D на рис. 1) и средств эффективного обновления расчетных величин в этих блоках (актуализации). Фиктивный блок каждого вычислительного домена содержит сеточные элементы из соседних доменов со специальной маркировкой, позволяющей вычислительным объектам отличать "свои" элементы от соседних. Фиктивные блоки также применяются нами для описания различных симметрий в расчетных областях, например, периодических граничных условий.

Наличие нескольких физических подобластей в расчетной области также отражено в иерархии классов дискретной модели (DiscretizedRegion и DiscretizedDomain на рис. 1). Объект класса DiscretizedRegion описывает дискретизацию одной физической подобласти на одном вычислительном домене. Для постановки граничных условий между различными физическими подобластями, а также для выделения границ расчетной области, применяются т.н. "суперэлементы", представляющие собой списки сеточных элементов, принадлежащих границам.

Для обеспечения баланса загрузки вычислительных узлов на каждом из них размещаются фрагменты всех имеющихся в расчетной области физических подобластей. Для этого расчетная сетка с M физических подобластей при расчете на N вычислительных узлах разбивается на $M \cdot N$ кусков (каждая физическая подобласть разбивается на N кусков). Список дискретизаций физических подобластей содержится в объекте класса DiscretizedDomain и обрабатывается в главном солвере (объекте класса MainSolver).

3.2. Сторонние разработки

Использование открытых и широко распространенных сторонних программ (библиотек) при разработке крупных исследовательских кодов избавляет разработчиков от необходимости тратить время на программирование и отладку вспомогательных систем, применяемых для решения конкретных задач, отделяемых от основного кода. В проекте MARPLE используется несколько сторонних библиотек для осуществления таких функций, как декомпозиция расчетной области, параллельное решение систем алгебраических уравнений, компрессия данных и их визуализация.

Для более компактного хранения внешних данных (файлы с сеткой, выходные файлы для визуализации, файлы для сохранения/восстановления расчета) нами используется библиотека zlib. Библиотека написана на языке C, и работа с ней из кода C++ недостаточно прозрачна и подвержена ошибкам. Имеющиеся в свободном доступе классы-обертки (zipstream, gzstream) предоставляют стандартный для C++ способ ввода/вывода данных через потоки.

В рамках проекта MARPLE разработан специальный набор утилит для подготовки к расчетам распределенных сеточных данных. Для эффективной декомпозиции сеточных графов в нем используется один из наиболее известных и широко применяемых инструментов для решения такого рода задач — библиотека ParMETIS.

Для решения в распределенном режиме больших систем линейных алгебраических

уравнений, получаемых при построении неявных схем, мы используем библиотеку Aztec. В ней реализованы параллельные итерационные алгоритмы, основанные на методах подпространства Крылова, применяемые при решении больших линейных систем, которые могут быть плохо обусловлены или получены при моделировании нестационарных процессов. В свою очередь эта библиотека использует функции фактически стандартных пакетов BLAS и LAPACK, которые зачастую предустановлены в вычислительных системах производителем и оптимизированы под конкретную архитектуру и набор компиляторов. Разработанные нами классы обеспечивают работу с распределенными разреженными матрицами и упрощают взаимодействие вычислительных модулей (солверов) с функциями библиотеки Aztec, которая также написана на языке C.

В качестве основы для реализации наших методов и алгоритмов на многопроцессорных системах используется широко распространенная среда параллельного программирования MPI. Набор функций, необходимых для работы кода MARPLE, также поддерживается специальными классами-обертками, которые помимо обеспечения интерфейса, более свойственного языку C++, значительно упрощают взаимодействие разработчиков остальных модулей со средой MPI. При проектировании и написании любого модуля должна обеспечиваться принципиально корректная работа кода в режиме распределенных данных и его выполнения на большой многопроцесорной системе. При этом многоуровневые иерархии солверных и сеточных классов практически полностью скрывают от разработчика вычислительного модуля технические детали параллелизма.

4. Моделирование сжатия проволочной сборки токовым импульсом

Эксперименты с самосжимающимися разрядами (пинчами) проводятся на сильноточных генераторах в ряде лабораторий России, США, Англии и Франции. Их целью является получение мощного импульса рентгеновского излучения (мощностью от нескольких ТВт до сотен ТВт при длительности от нескольких нс до десятков нс). Такие импульсы могут применяться для нагрева термоядерной мишени в схемах инерционного УТС, в нанотехнологиях и для фундаментальных исследований в области лабораторной астрофизики, изучения экстремальных состояний вещества и др. Источником плазмы в этих экспериментах может служить проволочная сборка — конструкция из тонких металлических проволок или полимерных нитей, натянутых между электродами. Высота и диаметр сборки имеют сантиметровые размеры, диаметр проволок — несколько микрон. На электроды подается мощный импульс тока (амплитуда от 1 до 20 МА, время нарастания от 100 нс до 1 мкс). Под действием этого импульса проволочки нагреваются, испаряются и переходят в плазменное состояние. Возникающее азимутальное магнитное поле сжимает плазму к оси сборки, где в момент максимального сжатия происходит быстрый переход кинетической энергии в энергию излучения.

В данном примере использовалась постановка задачи, приведенная в [4]. Проволочная сборка — “цилиндр”, составленный из 240 алюминиевых проволок (диаметр 10.4 мкм, высота 50 мм). Диаметр сборки 140 мм, диаметр разрядной камеры 150 мм. Расчет выполнен при заданном токе генератора (амплитуда 6 МА, время нарастания 700 нс). Уравнения состояния, транспортные коэффициенты и оптические свойства использованы в форме таблиц [6].

Для расчета генерации плазмы при взрыве проволок сильноточным импульсом использовалась нестационарная модель плазмообразования [5]. Проволочная сборка в расчете заменяется эффективной “плазмообразующей поверхностью”, а неоднородность потока плазмы моделируется стохастическим возмущением (до 5%) темпа появления плазмы.

Расчетная сетка — блочная, из смешанных элементов (шестигранники и треугольные призмы), построена в цилиндрическом секторе 60°, состоит из приблизительно 4 млн. ячеек. Средний линейный размер ячейки $h_\phi \approx 600 - 400$ мкм, $h_r \approx 375$ мкм и $h_z = 400$ мкм.

На вертикальных плоских гранях сектора были заданы периодические граничные условия, что эквивалентно полномасштабному 3D моделированию в цилиндре. Такой подход представляется весьма перспективным для экономичных расчетов на подобных сетках.

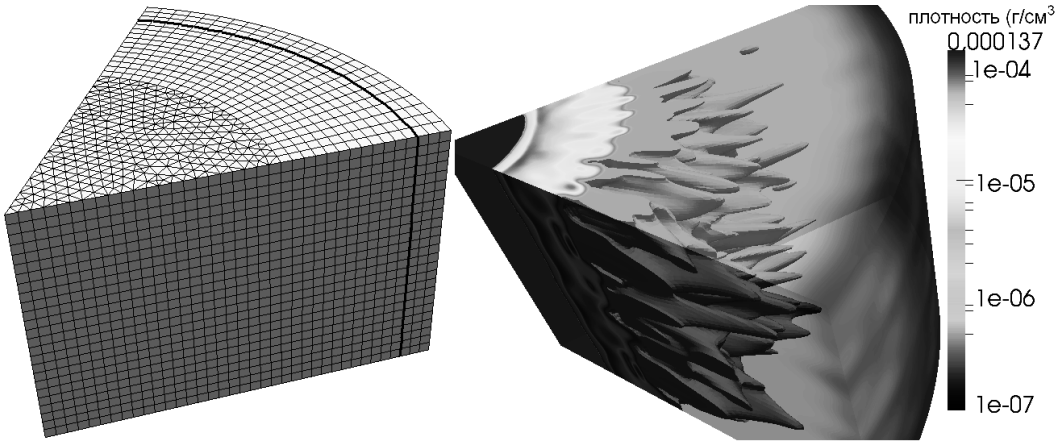


Рис. 2. Слева: пример блочной сетки в цилиндрическом секторе (7 тыс.ячеек), справа: развитие неустойчивости на периферии плазменной оболочки (4 млн. ячеек).

Гидродинамическая неустойчивость, возникающая при сжатии пинчей, может оказать заметное влияние на достижимые параметры. Исследование таких неустойчивостей — одна из важных задач 3D-моделирования, т.к. неустойчивость плазмы, ускоряемой электродинамически, является существенно трехмерным эффектом.

В данном примере на периферии пинча можно видеть развитую стадию неустойчивости плазмы. Существенные возмущения формы оболочки в виде отклонений от цилиндрически симметричного движения вещества могут замедлить образование финальной структуры, снизить температуру и плотность пинчевой плазмы. С другой стороны, турбулизация течения плазмы вызывает повышенную диссипацию электрической энергии токового импульса, вследствие чего растет температура вещества. Таким образом, расчет показывает результат конкуренции различных, притом, нелинейных, процессов. Анализ расчетных данных показал, что в результате моделирования были получены физически адекватные результаты, совпадающие с экспериментальными данными по таким показателям, как мощность излучения, время сжатия, размер, плотность и температура пинча. Хорошее соответствие с результатами экспериментов показали также расчеты пинчей при других начальных и краевых условиях. Полученные при апробации кода данные свидетельствуют в пользу того, что построенная 3D модель Z-пинча является состоятельной для данного класса задач и может быть использована для предсказания результатов будущих экспериментов.

5. Заключение

Рассказано о современных технологиях разработки крупных программных комплексов, представлены различные инструментальные средства, используемые разработчиками для повышения эффективности и скорости создания программ.

Описанные методы программирования и инструментальные средства использовались для разработки программного кода MARPLE, предназначенного для проведения параллельных расчетов прикладных начально-краевых задач. Использование смешанных сеток нерегулярной структуры обеспечивает возможность вычислений в расчетных областях нетривиальной геометрии.

Разработанный код MARPLE применен для моделирования сжатия токовым импульсом алюминиевой проволочной сборки — одной из типовых задач импульсной энергетики. Полученные результаты качественно верно воспроизводят динамику плазмы и основные

особенности сжатия проволочных сборок.

План развития кода предусматривает включение в него новых моделей импульсной плазмы, а также оптимизацию процессов вычислений и обработки данных. В перспективе предусматривается подготовка кода к использованию на вычислительных системах сверхвысокой (тера- и эксафлопной) производительности, которые в предстоящее десятилетие будут использоваться в научных исследованиях.

Литература

1. Gasilov V., Dyachenko S., Olkhovskaya O., Boldarev A., Kartasheva E., Boldyrev S. Object-Oriented Programming and Parallel Computing in RMHD Simulations // IOS Press: Advances in Parallel Computing, Vol. 15, 2008.

2. Брагинский С.И. Явления переноса в плазме // "Вопросы теории плазмы" (под ред. М.А. Леонтовича) — М., Атомиздат, вып.1, с.183-272, 1963.

3. Chandresekhar S. Radiative transfer // Oxford, 1951.

4. Фортов В.Е. Экстремальные состояния вещества // М.: ФИЗМАТЛИТ, 2009.

5. Александров В.В. и др. Динамика гетерогенного лайнера с затянутым плазмообразованием // ФИЗИКА ПЛАЗМЫ, т. 27, № 2, 2001.

6. Никифоров А.Ф., Новиков В.Г., Уваров В.Б. Квантово-статистические модели высокотемпературной плазмы // М.: ФИЗМАТЛИТ, 2000.

7. Ломоносов И.В., Фортов В.Е. и др. Метастабильные состояния жидкого металла при электрическом взрыве // ТВТ, т.39, №5, с.728-742, 2001.

8. Kartasheva E., et al. An Implicit Complexes Framework for Heterogeneous Objects Modelling, in Heterogeneous Objects Modelling and Applications // Lecture Notes in Computer Science, vol. 4889, Springer-Verlag, pp.1-41, 2008.

9. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений // М.: ФИЗМАТЛИТ, 2001.

10. Cockburn B., Shu C.W. The local DG method for time-dependent convection-diffusion systems // SIAM Journal of Numerical Analysis, 35 (6): pp.2440-2463, 1998.

11. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ // 2-е изд., СПб.: "Невский диалект", 2001.

Реализация на высокопроизводительных вычислительных системах математической модели ветровых течений в Керченском проливе*

В.Н. Дацюк[1], Л.А. Крукиер[1], А.Л. Чикин[2]

Южно-Российский региональный центр информатизации[1]
Южный научный центр РАН[2]

Представленная математическая модель описывается уравнениями движения несжимаемой вязкой жидкости. Задача решается конечно-разностными методами с использованием противопотоковых схем и численно реализована на высокопроизводительных вычислительных системах. Данная модель позволяет рассчитывать течения в случае изменения береговой линии, а ее трехмерность дает возможность вычислять скорости на различных горизонтах. Проведено сравнение эффективности параллельной реализации модели на различных вычислительных системах.

Введение

Одним из основных этапов решения задач, связанных с моделированием гидрофизических процессов в водоемах, является расчет гидродинамических параметров течения. Определенный интерес представляют водоемы с большой неоднородностью поля глубин. Под большой неоднородностью глубин понимается наличие как глубоководной области, так и обширных мелководных районов (прибрежная зона, внутренние заливы и лиманы), глубина в которых соразмерна с перепадом уровня воды. В судоходной части Керченского пролива глубины составляют 8 – 10 м, а в южной его части до 19 м. В то же время в прибрежных районах, Таманском и Динском заливах глубина составляет 0,5 – 4 м [1]. Такое распределение глубин позволяет говорить о большой их неоднородности. В настоящее время при расчете параметров течений в водоемах с указанной топологией дна обычно используются специальные координатные системы (σ-координаты), возможно использование криволинейных сеток или проведение предварительного преобразования исходной нерегулярной области в регулярную. Однако, данные методики заметно усложняют как саму систему дифференциальных уравнений, так и ее численное решение.

В связи с этим для численного исследования течений в Керченском проливе использовалась разработанная автором двухслойная модель гидродинамики [2]. Достоинство данного подхода заключается в том, что решение подобных задач происходит без предварительного преобразования расчетной области, используя конечно-разностные методы с применением равномерных прямоугольных сеток. Хотя результаты расчетов нельзя считать высокоточными, их можно использовать в качестве разведочного анализа или начального приближения в более сложных моделях.

Постановка задачи

Суть подхода в следующем. Проведем горизонтальную секущую плоскость P, отстоящую от невозмущенной поверхности водоема P0 на глубине, равной максимальной глубине мелководья $h(x, y)$ (рис.1), и разобьем всю область моделирования на два слоя – верхний – I и нижний слой II. Таким образом, I –все мелководье и верхняя часть глубоководного слоя, слой II – содержит глубоководную область. Предполагается, что эффект осушения из-за сгона воды может присутствовать только в мелководных районах и не может происходить в глубоководной части.

* Работа выполнена при финансовой поддержке РФФИ. Грант 09-01-00023-а.

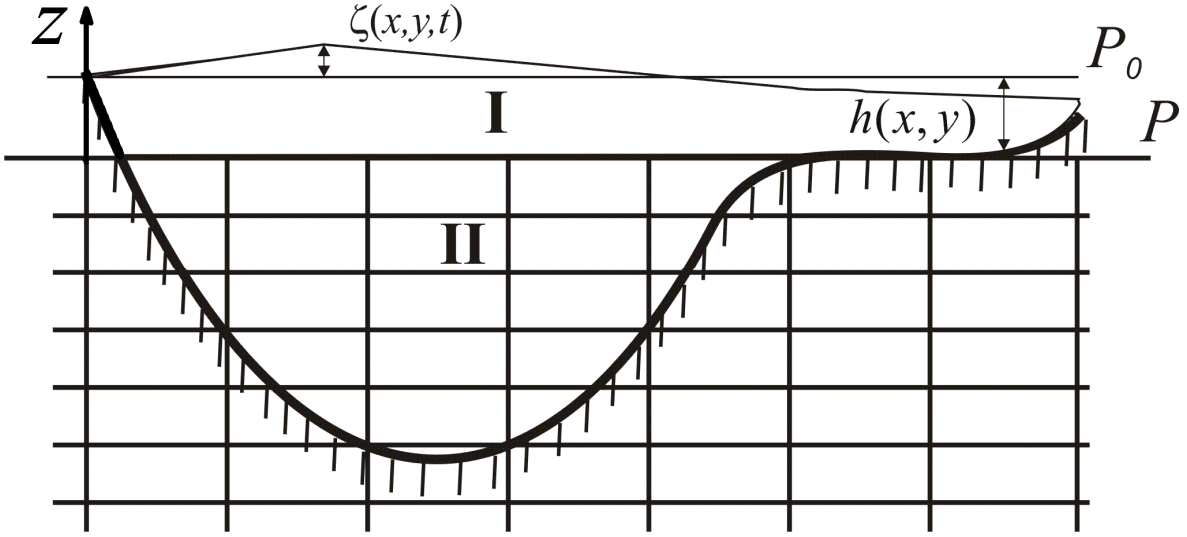


Рис. 1. Вертикальный разрез исследуемого водоема

На движение воды в слое I влияет ветер, а движение в слое II инициируется как движением слоя I, так и градиентами давления.

Движение воды в слое I задается уравнениями мелкой воды

$$\frac{\partial u_s}{\partial t}+u_s\frac{\partial u_s}{\partial x}+v_s\frac{\partial u_s}{\partial y}-\Omega v_s=-g\frac{\partial \zeta}{\partial x}+\nu_{xy}\left(\frac{\partial^2 u_s}{\partial x^2}+\frac{\partial^2 u_s}{\partial y^2}\right)+$$

$$+\frac{\tau_{sx}}{H}-\frac{\tau_{bx}}{H}+F_x(x,y), \qquad (1)$$

$$\frac{\partial v_s}{\partial t}+u_s\frac{\partial v_s}{\partial x}+v_s\frac{\partial v_s}{\partial y}+\Omega u_s=-g\frac{\partial \zeta}{\partial y}+\nu_{xy}\left(\frac{\partial^2 v_s}{\partial x^2}+\frac{\partial^2 v_s}{\partial y^2}\right)+$$

$$+\frac{\tau_{sy}}{H}-\frac{\tau_{by}}{H}+F_y(x,y), \qquad (2)$$

$$\frac{\partial \zeta}{\partial t}+\frac{\partial H u_s}{\partial x}+\frac{\partial H v_s}{\partial y}=0. \qquad (3)$$

В уравнениях (1)–(3) $u_s=\frac{1}{H}\int_{-h}^{\zeta}u\,dz$, $v_s=\frac{1}{H}\int_{-h}^{\zeta}v\,dz$, $H=h+\zeta$; $h=h(x,y)$ – глубина мелководного слоя; $u_s=u_s(x,y,t)$, $v_s=v_s(x,y,t)$ – скорости в слое I; функции $F_x(x,y)$ и $F_y(x,y)$ описывают взаимодействие верхнего и нижнего слоев между собой; τ_{sx},τ_{sy} – проекции на оси OX и OY силы трения ветра о поверхность водоема; τ_{bx},τ_{by} – проекции на оси OX и OY силы трения жидкости о дно (или о глубоководный слой воды). Эти величины зависят от скорости ветра $\bar{W}_B=\{W_x;W_y\}$ и течения $\bar{W}_T=\{u_s;v_s\}$ и определяются так [3]:

$\bar{\tau}_s=\gamma|\bar{W}_B|\bar{W}_B$, $\bar{\tau}_b=\beta|\bar{W}_T|\bar{W}_T$,

где $|\bar{W}_B|=\sqrt{W_x^2+W_y^2}$, $|\bar{W}_T|=\sqrt{u_s^2+v_s^2}$, $\beta(x,y)$ – коэффициент трения верхнего слоя жидкости о дно (или о глубоководный слой); γ – коэффициент трения ветра о слой I.

Движение воды в слое II задается уравнениями движения несжимаемой вязкой жидкости

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}-\Omega v=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu_{xy}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+ \\ +\frac{\partial}{\partial z}\left(\nu_z\frac{\partial u}{\partial z}\right), \tag{4}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}+\Omega u=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu_{xy}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+ \\ +\frac{\partial}{\partial z}\left(\nu_z\frac{\partial v}{\partial z}\right), \tag{5}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0. \tag{6}$$

К системе уравнений (4) –(6) добавляется уравнение гидростатического давления

$$p=g\rho(\zeta-z)+p_a\,. \tag{7}$$

В (4) – (7) u, v, w – компоненты скорости; x, y, z, t – пространственные переменные и время соответственно; ν_{xy}, ν_z – коэффициенты горизонтальной и вертикальной вязкости соответственно; ρ – плотность воды; $g=9{,}8$ м/с^2 – ускорение силы тяжести; p_a – атмосферное давление.

Приведенные системы замыкаются соответствующими граничными условиями. Вдоль твердой границы ставятся условия скольжения или задаются скорости втекания или вытекания воды. На границе между слоями $\partial\Omega_l$ ставится условие равенства скоростей

$$u\big|_{\partial\Omega_l}=u_s, v\big|_{\partial\Omega_l}=v_s\,.$$

Функции $F_x(x,y)$ и $F_y(x,y)$, описывающие взаимодействие I и II слоя, задаются следующим образом:

$$F_x(x,y)=\frac{uw}{H}\bigg|_{\partial\Omega_l},\ F_y(x,y)=\frac{vw}{H}\bigg|_{\partial\Omega_l}\,.$$

Вычислительные эксперименты

Особенностью моделирования гидродинамики Керченского пролива является наличие косы Тузла. Тузлинский нанос – это песчаная полоса, полупогруженная в илы. Коса растет в результате типичной аккумуляции береговых наносов. В 1925 году во время сильного южного шторма в море прорвало косу вблизи Таманского берега, и она превратилась в остров, отделенный от мыса Тузла проливом, который достиг через некоторое время ширины нескольких километров. Средняя глубина пролива между мысом Тузла (Тамань) и надземной частью косы не превышает 0,3-0,6 м. На расстоянии 2,5 км от мыса имеется искусственная промоина глубиной порядка 1,3 м и шириной не более 30 м. В 2003 г. со стороны Таманского полуострова на месте прежней косы была частично насыпана дамба.

Характерной ситуацией в Керченском проливе является смена течений с Азовоморского на Черноморское или наоборот. С помощью построенной математической модели было численно исследована возможная динамика изменения течения с Азовоморского на Черноморское.

В течение первого часа направление течения изменилось, в основном, в примыкающих к проливу заливах. Затем Черноморское течение начинает формироваться в южной части пролива, но течения в протоках вокруг о. Тузла носят северный характер (рис.2).

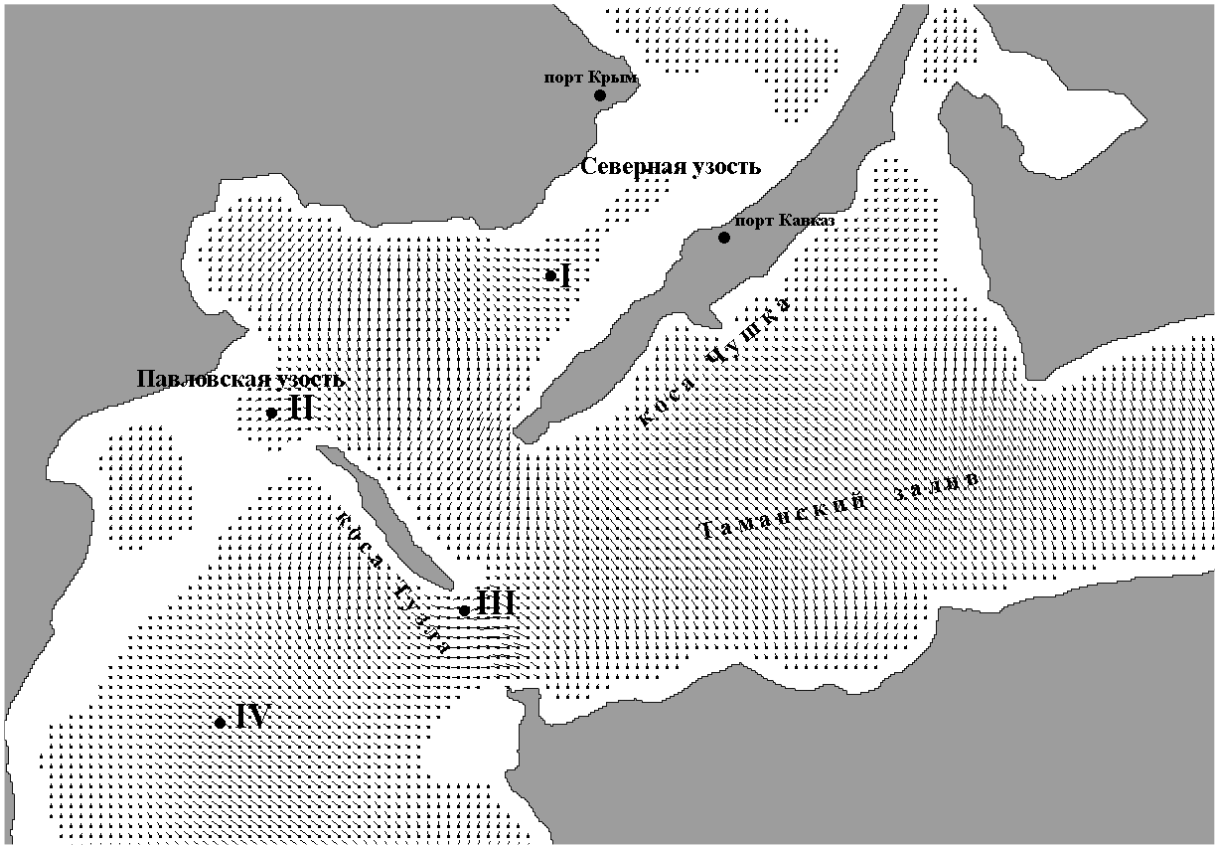


Рис. 2. Картина течения в Керченском проливе через 2 часа после смены ветра с северного на южное направление

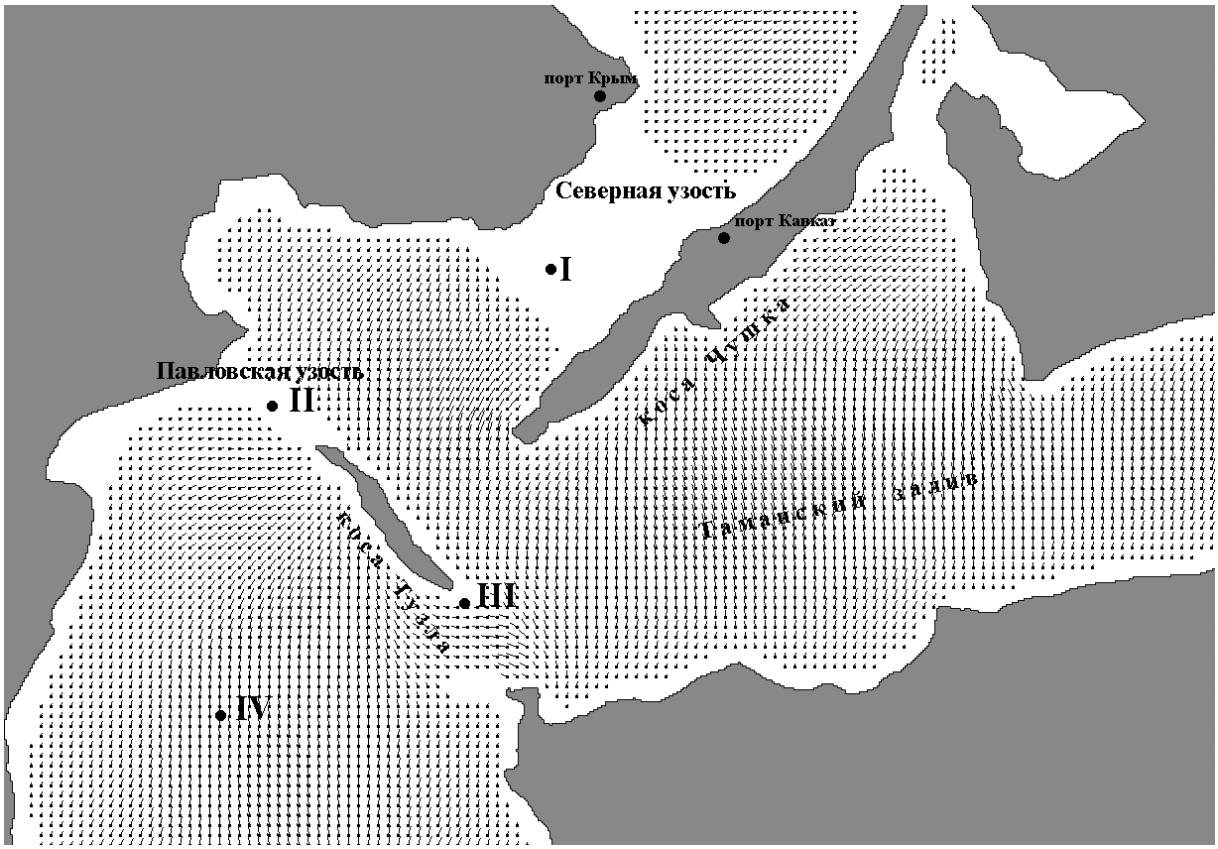


Рис. 3. Картина течения в Керченском проливе через 3 часа после смены ветра с северного на южное направление

Через три часа после смены направления ветра Черноморское течение занимает почти всю южную часть пролива (рис.3). Исключение составляет течение через створ между о. Тузла и Таманским полуостровом. Очень слабые течения наблюдаются в северной и Павловской узостях. К концу седьмого часа Черноморское течение полностью формируется.

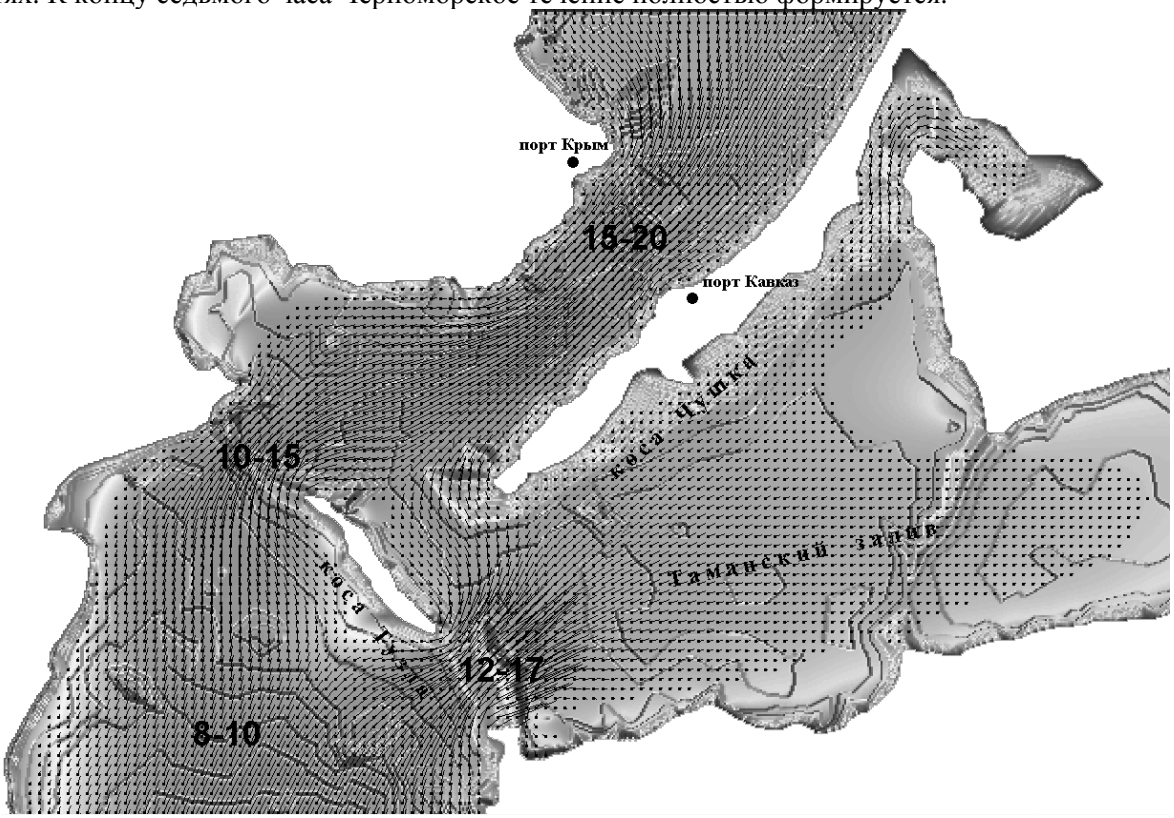


Рис. 4. Поле скоростей в Керченском проливе при отсутствии дамбы вдоль косы Тузла

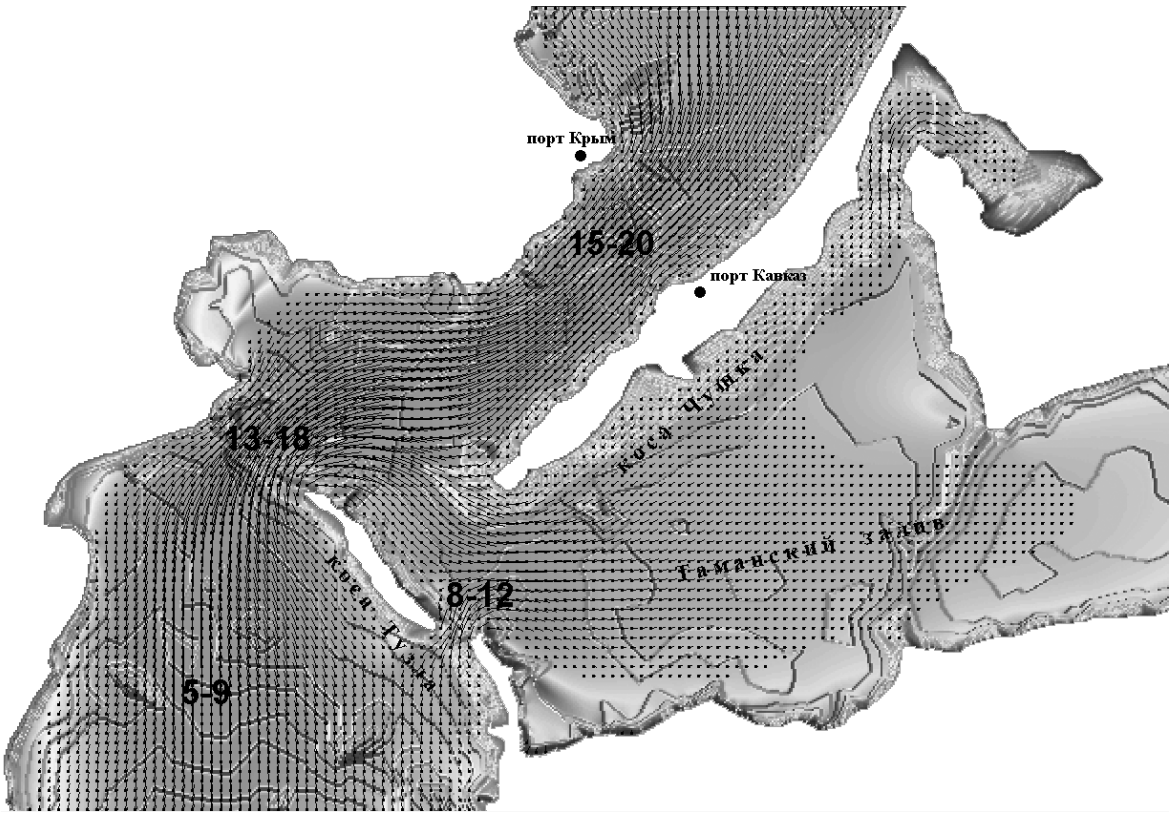


Рис. 5. Поле скоростей в Керченском проливе при наличии дамбы вдоль косы Тузла

С помощью построенной математической модели была численно исследована возможная картина течений в центральной части пролива при отсутствии или наличии дамбы. Так, например, при действии юго-западного ветра в отсутствии дамбы примерно половина объема воды, проходящей через станцию III, нагоняется в Таманский залив, а другая половина движется вдоль северной стороны острова Тузла в сторону Азовского моря (рис.5). При наличии дамбы расход воды через станцию III значительно сокращается, и вода в Таманский залив поступает, в основном, с северной стороны о. Тузла, но движется уже в юго-восточном направлении (рис.6).

Анализ численной реализации

Для решения систем линейных алгебраических уравнений, возникающих при дискретизации исходных дифференциальных уравнений, использовалась библиотека параллельных подпрограмм Aztec. В этой библиотеке реализован набор итерационных методов Крылова для решения систем линейных алгебраических уравнений с разреженными матрицами. Распараллеливание выполнено с использованием коммуникационной библиотеки MPI. Особенностью итерационных методов является многократное использование операции умножения матрицы на вектор. Профилирование программы показало, что на эту процедуру приходится более 50% всех вычислительных затрат. Поэтому оптимизации параллельной версии этой подпрограммы было уделено большое внимание. Тем не менее, из-за большого числа коммуникационных операций трудно рассчитывать на высокую степень масштабируемости этой процедуры и соответственно всей программы в целом. Тестирование показало, что использование этих методов предъявляет высокие требования к коммуникационной среде.

На задаче расчета течений в Керченском проливе было протестировано время счета на трех вычислительных системах:

1. на кластере IBMX с 2-х ядерными процессорами Intel Xeon 5160 и коммуникационной сетью DDR Infiniband;
2. на кластере INFINI с процессорами Intel Pentium 4 3.4 Ггц и коммуникационной сетью SDR Infiniband;
3. на рабочей станции QUAD с 4-х ядерным процессором Q6600 с тактовой частотой 2.4 Ггц и оперативной памятью 4Гбайт.

Рассматривалось три размера разностных сеток с шагами по горизонтали 60 м, 40 м, 20 м с максимальным числом неизвестных 626000, 1414000 и 5670000 соответственно. При тестировании расчет проводился для 200 временных шагов.

Как и следовало ожидать, наибольшее ускорение (Sp) прослеживается на кластере IBMX, с коммуникационной сетью DDR Infiniband (Таблица 1). Для самой грубой сетки (h=60м) линейное ускорение на этой платформе сохраняется для числа узлов $n_p \le 4$. В то время как, на кластере с однократной скоростью Infiniband ускорение начинает отклоняться от линейного уже для $n_p \le 2$. И совершенно неоправданным представляется запуск нескольких MPI процессов на многоядерных вычислительных узлах (рабочая станция QUAD).

Таблица 1. Сравнение производительности различных вычислительных платформ.

np	Сетка h=60 м					
	INFINI (сек.)	Sp	IBMX (сек.)	Sp	QUAD (сек.)	Sp
1	1858	1,0	1677	1,0	1203	1,0
2	971	1,9	853	2,0	1008	1,2
3	714	2,6	570	2,9	1421	0,8
4	545	3,4	435	3,9	1648	0,7
5	470	4,0	357	4,7	-	
6	405	4,6	313	5,4	-	
7	362	5,1	273	6,1	-	
	Сетка h=40 м					
1	3896	1,0	3828	1,0	2740	1,0
2	2132	1,8	2015	1,9	2154	1,3
3	1542	2,5	1376	2,8	2865	0,95
4	1212	3,2	1066	3,6	2920	0,93
5	1029	3,8	881	4,3	-	
6	903	4,3	758	5,1	-	
7	802	4,9	681	5,6	-	
	Сетка h=20 м					
1	mem	-	15182	1,0	mem	-
2	mem	-	8097	1,9	mem	-
3	mem	-	5818	2,6	-	
4	mem	-	4503	3,4	-	
5	mem	-	3771	4,0	-	
6	mem	-	3304	4,6	-	
7	mem	-	3098	4,9	-	

На сетке (h=40м) приоритеты в вычислительных системах сохраняются, однако коэффициент ускорения снижается.

На мелкой сетке (h=20м), когда число неизвестных становится очень большим, подобные задачи решаются только на кластере IBMX. При попытке решения этих задач на системах INFINI или QUAD возникают проблемы с памятью.

Численное исследование показало, что при использовании технологии MPI на вычислительной системе QUAD с 4-х ядерным процессором производительность немного возрастает при подключении 2-го ядра, а затем падает при подключении последующих. В то время, как на системах с распределенной памятью INFINI и IBMX наблюдается хороший рост ускорения при увеличении числа вычислительных узлов. Заметим, что на вычислительном кластере IBMX с 2-ядерным процессором на каждом узле запускался только один счетный процесс. Использование второго ядра, так же, как и на системе QUAD приводила к деградации производительности узла.

Заключение

Сравнение результатов расчета с наблюденными данными показало, что представленная математическая модель достаточно адекватно описывает гидродинамику течений в Керченском проливе. Данная модель позволяет рассчитывать течения в случае изменения береговой линии, а ее трехмерность дает возможность вычислять скорости на различных горизонтах.

Проведенное нами исследование ставит под сомнение целесообразность создания кластеров с большим числом ядер на узлах. По крайней мере, нужно отдавать себе отчет, для решения

каких задач будет преимущественно использоваться вычислительная система и насколько эффективно будут выполняться программы на той или иной конфигурации вычислительных узлов.

Литература

1. Карта Керченского пролива М 1:100000. Главное управление навигации и океанографии, С.-Пб, 2003.
2. Чикин А.Л. Об одном из методов расчета параметров течений в водоемах с большой неоднородностью глубин// Водные ресурсы, 2005. Т. 32. № 1. С. 55-60.
3. Филиппов Ю.Г. Об одном способе расчета морских течений //Тр. ГОИН. 1970. Вып. 103. С.87-94.

Оценка сложности стратегий параллельного построения изображений для систем визуализации на суперкомпьютерах*

О.В. Джосан

Московский Государственный Университет имени М.В. Ломоносова

В работе рассматриваются различные стратегии параллельного построения изображений и видеопоследовательностей на суперкомпьютерах для систем визуализации научных данных. Анализируется их вычислительная сложность. Проводится обобщение существующих стратегий для случая произвольного количества коммуникаций на каждом шаге. Приводятся оценки эффективности и масштабируемости стратегии для различных входных параметров задачи. Практическая апробация предложенных методов проведена на суперкомпьютере BlueGene /P.

1. Введение

Метод объемного построения изображений (volume rendering) – это наиболее распространенный подход к визуализации трехмерных данных большого объема, которые получаются в результате крупномасштабного моделирования на суперкомпьютерах. Вычислительные эксперименты такого рода имеют ряд существенных особенностей. Основная особенность – это размер получаемых данных. В работе [1] рассмотрен пример визуализации данных для задачи моделирования несферических частиц. Объем данных, которые визуализируются в эксперименте, составляет 439 гигабайт. Размер данных моделирования турбулентного горения, которые представлены Институтом Institute for Ultra-Scale Visualization [2] в качестве бенчмарка для методов параллельной визуализации данных, составляет порядка 300 гигабайт. Такой объем требует специализированных методов по организации хранения и ввода-вывода данных. Следующей особенностью является распределенное хранение результатов вычислений. При этом число процессоров, на которых проводится эксперимент, может исчисляться десятками тысяч, и в будущем прогнозируется существенный рост этого показателя. Распределенное хранение данных, которые требуется визуализировать, порождает еще одну особенность параллельной визуализации – существенное время, которое требуется на коммуникации между процессорами в процессе построения изображений. Ввиду этих особенностей реализация параллельного построения изображений в реальном времени для крупномасштабных вычислительных экспериментов сейчас является научной проблемой, которая требует разработки новых методов и подходов к такого рода визуализации.

Методы параллельного построения изображений по распределенным данным активно развиваются в течение последних двадцати лет [3]. Существующие алгоритмы традиционно разделяются на три типа по этапу, на котором происходит обмен распределенными данными: sort-first, sort-middle, sort-last [4]. Примеры различных стратегий параллельного построения изображений можно найти в работах [5-10]. Существует ряд программных продуктов, которые поддерживают параллельное построение изображений, в частности можно выделить системы VisIt[13-14] и ParaView [11-12] с ядром параллельного построения изображений IceT [15]. Однако существующие системы не обеспечивают достаточной скорости работы для построения высококачественных изображений и видео в реальном времени.

В данной работе будут рассмотрены только алгоритмы, относящиеся к классу sort-last, в которых каждый процессор визуализирует свою часть данных, и далее осуществляется только обмен изображениями для построения итогового изображения. Алгоритмы класса sort-last включают в себя два шага: шаг построения изображения (image rendering) и шаг компоновки изображения (image compositing). На первом шаге каждый процессор строит свою часть изображения по части данных, которая была получена на нем при вычислениях. Затем на втором

* Работа выполнена при поддержке ФЦП “Научные и научно-педагогические кадры инновационной России” на 2009 - 2013 годы”, гранта РФФИ 11-07-00614-а.

шаге из полученных изображений формируется итоговая картинка. В данной работе детально проанализированы стратегии, применяемые на шаге компоновки изображения для уменьшения времени работы и объема пересылаемых данных.

Самый тривиальный для реализации подход – пересылка всех данных на один процессор. Такой метод условно называется последовательным [17]. Однако этот метод весьма затратный по вычислениям и времени выполнения, а также может быть легко оптимизирован и упрощен, поэтому в реальности не применяется. В терминах MPI проблема компоновки изображения эквивалентна решению reduce-scatter проблемы. Первой оптимизацией является использование топологии виртуального дерева (virtual tree) для пересылки, что позволяет сократить общее время пересылки данных. Такие подходы подробно описаны в работе[18]. Следующим этапом развития алгоритмов компоновки можно назвать появление метода бинарных обменов (binary swap)[19]. Идея алгоритма заключается в пересылке не целого изображения, а только его части. Недостатком такого алгоритма являлось то, что количество процессоров должно было быть степенью 2. Эта проблема была решена в работе [20], где предложен алгоритм на основе 2-3 обменов. Еще одним принципиальным подходом к организации обменов стал сценарий параллельного конвейера (parallel pipelined)[21]. В работе [22] предложена существенная модификация конвейерного алгоритма, названная циклическим разделением (rotate tiling), которая перенаправляет обмены, сокращая количество пересылок.

В настоящее время алгоритмы компоновки изображений развиваются в двух направлениях: 1) применение гибридных подходов, 2) пересылка только значимых пикселей изображения. Примеры гибридных подходов рассмотрены, например, в работах [23] и [24]. Методы выделения только значимых пикселей предложены в работах [25] и [24].

Приведенный обзор показывает, что выбор методов параллельной компоновки изображений достаточно широк. Однако существующие методы решают далеко не все проблемы, актуальные в настоящее время. Так, например, методы с выделением значимых пикселей сталкиваются с проблемой балансировки загрузки, т.к. в этом случае она должна быть динамической, а это означает применение весьма вычислительно сложных алгоритмов.

Большинство реализаций алгоритмов параллельной композиции для ускорения работы использует сжатие данных. В данной задаче требуется алгоритм с невысокой степенью сжатия, однако сжатие должно быть без визуальных потерь и очень быстрым. Методы, используемые в настоящее время, достаточно примитивны. В работе [26] предложен ряд алгоритмов сжатия, которые могли бы более эффективно использоваться для сжатия при компоновке изображений. Некоторые экспериментальные оценки затрат на коммуникации в сетях различной архитектуры получены в работе [27].

Гибридные модели алгоритмов компоновки в большинстве случаев являются статическими и не могут быть динамически адаптированы под конкретную архитектуру или контекст задачи. Интерес представляет анализ обобщенных гибридных моделей и выделение в них параметров, которые в дальнейшем могут быть использованы для динамической адаптации.

Рассмотренные методы по разному решают вопрос, где и в каком виде хранится итоговое изображение. В большинстве случаев финальное изображение храниться после компоновки распределенно. Для отображения на дисплей нужно это изображение передать с вычислительных узлов на интерфейсную машину или записать информацию на жесткие диски. Если бы мы храним изображение распределенно, то в этом случае может эффективно применяться параллельный ввод-вывод. Однако с появлением мультидисплейных комплексов в процессе компоновки появляется еще одна задача по разделению и передаче итоговой картинки на несколько дисплейных устройств. Ее решение также представляет интерес, т.е. может быть интегрировано в процесс компоновки изображения.

В данной работе рассмотрены теоретические и экспериментальные оценки вычислительной сложности различных методов компоновки изображения при параллельном построении. Оценки проводятся с учетом необходимости отображения на мультидисплейном комплексе, соответственно учитываю возможность совмещения компоновки изображения и распределения изображения между отображающими устройствами. Рассмотрено два похода к организации обменов: 1) на каждом шаге задействуются все процессоры, 2) процессоры разбиты на группы и обработка осуществляется в конвейерном режиме. Рассмотрены некоторые гибридные моде-

ли и их параметры, позволяющие адаптировать процесс компоновки в зависимости от особенностей архитектуры вычислительного комплекса и контекста задачи.

2. Оценка сложности методов параллельного построения изображений

Рассмотрим основные параметры, которые необходимо использовать при оценке сложности методов параллельного построения изображений. Пусть итоговое изображение, которое необходимо построить, имеет размер XxY пикселей. Соответственно площадь этого изображения S пикселей (S=X*Y). N - количество процессоров, используемых для построения изображения. $B_{вх}$ и $B_{исх}$ – количество пикселей, которое может быть получено/отправлено процессоров в секунду (пропускная способность). Пусть $B_{вх} = B_{исх} = B$. Путь мультидисплейный комплекс состоит из решетки pxq дисплеев. Общее количество дисплеев – M=p*q. Пусть выполнено условие M<N.

Рассмотрим оценки для последовательного метода, метода бинарных обменов, метода параллельного конвейера, метода циклического разделения. При этом распределение по процессорам, к которым подключены дисплеи, осуществляется последовательно. Также сделано допущение, что время компоновки пренебрежительно мало по сравнению со временем пересылки изображений между процессорами. После этого опишем гибридный подход для мультидисплейных комплексов с использованием балансировки загруженности.

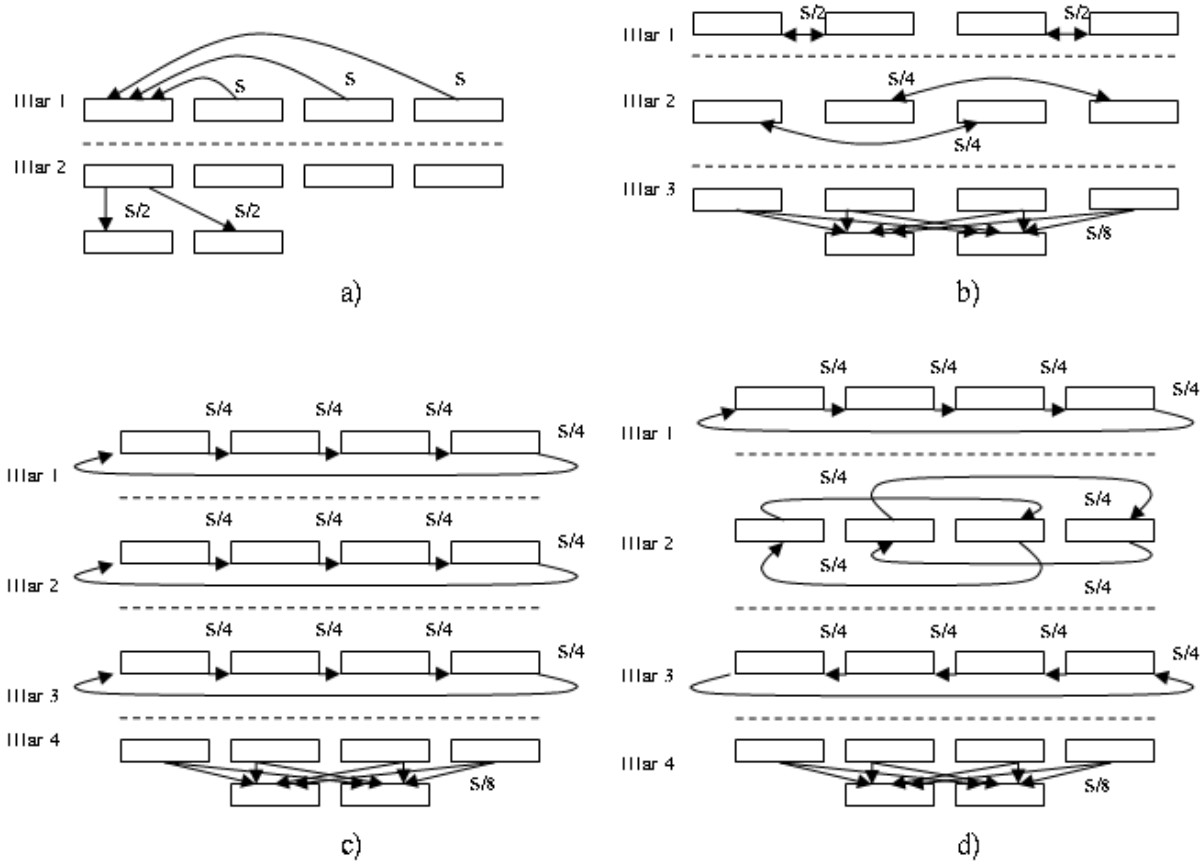


Рис. 1. Схемы обменов при разных стратегиях пересылки для случая N=4, p=1, q=2. a) последовательный метод; b) метод бинарных обменов; c) метод параллельного конвейера; d) метод циклического разделения.

2.1 Оценки сложности для последовательного метода

Суть последовательного метода состоит в том, что все данные передаются на один процессор, далее с него распределяются по процессорам, к которым подключены дисплеи. Сценарий

работы для этого метода для случая N=4, p=1, q=2 проиллюстрирован на Рис.1a. Общий объем передаваемых данных (в пикселях): V_{SS}=S*(N-1)+S =S*N. Поскольку все выполняется последовательно, по время на пересылку вычисляется по формуле: T_{SS}= S*N/B.

2.2 Оценки сложности для метода бинарных обменов и конвейерного метода

Метод бинарных обменов для случая N=4, p=1, q=2 проиллюстрирован на Рис.1b. На первом шаге алгоритма процессоры обмениваются S/2 пикселями изображения, на следующем шаге S/4 и т.д. Количество итераций в этом случае $\log_2 N$. Объем данных, который последовательно передается на i-й итерации, составляет: $V_i=S/2^i$. Время на i-ой итерации составляет $T_i= S/(B*2^i)$. Таким образом суммарное время на сборку и объем передаваемых последовательно данных можно вычислить следующим образом:

$$T_{BS}'= \sum_{i=1}^{\log_2 N} \frac{S}{B*2^i} = \frac{S}{B}\left(1-\frac{1}{N}\right); \quad V_{BS}'= \sum_{i=1}^{\log_2 N} \frac{S}{2^i} = S\left(1-\frac{1}{N}\right);$$

После выполнения алгоритма на каждом процессоре получится S/N пикселей итогового изображения. Поскольку сборка на M процессоров для отображения на дисплеи осуществляется последовательным образом, то количество данных, которое необходимо последовательно передать, можно посчитать как количество данных, которое придет на каждый из M процессоров. Таким образом, V_2=S/M; T_2=S/(B*M). В итоге получаем, что время, которое тратится на построения изображения в формате для мультидисплеев с помощью метода бинарных обменов:

$$T_{BS} = \frac{S}{B}\left(1-\frac{1}{N}\right)+\frac{S}{BM} = \frac{S}{B}\left(1-\frac{1}{N}+\frac{1}{M}\right);$$

Объем передаваемых данных:

$$V_{BS} = S\left(1-\frac{1}{N}+\frac{1}{M}\right);$$

Идея конвейерного метода состоит в том, что на каждой итерации процессор передает и получает S/N пикселей изображения. При этом выполняется N-1 итераций. После этих итераций осуществляется последовательная сборка на M Процессоров. С учетом оценок полученных в работе [22] получаем, что оценка времени работы и объема передаваемых данных с учетом мультидисплейности совпадает с показателями для метода бинарных обменов. Работа метода проиллюстрирована на Рис.1c.

2.3 Оценки сложности для метода циклического разделения

Верхние оценки времени работы алгоритма и объема передаваемых данных получены авторами метода в работе [22]. Используя эти оценки для случая мультидисплейности получаем:

$$V_{RT} = S\left(\frac{1}{N}\sum_{i=1}^{\log_2 N}\left(1-\frac{1}{N}\right)^{i-1} * \frac{1}{2^{i-1}} - \frac{1}{N}+\frac{1}{M}\right);$$

$$T_{RT} = \frac{S}{B}\left(\frac{1}{N}\sum_{i=1}^{\log_2 N}\left(1-\frac{1}{N}\right)^{i-1} * \frac{1}{2^{i-1}} - \frac{1}{N}+\frac{1}{M}\right);$$

Таким образом из проанализированных классических методов сборки изображения метод циклического разделения требует наименьший размер передаваемых последовательно на каждой итерации данных и время его работы меньше остальных. Однако в каждом из перечисленных методов в оценках присутствует слагаемое, получаемое из применения последовательного метода перераспределения изображения для мультидисплейного комплекса. Очевидно, что этот шаг можно оптимизировать, применив гибридную стратегию, в рамках которой компоновка изображения совмещена с распределением его по мультидисплейному комплексу.

3. Метод гибридной сборки с учетом мультидисплейности

3.1 Описание гибридной стратегии сборки для мультидисплейного комплекса

В данной статье автором предлагается стратегия гибридной сборки для параллельной компоновки изображения. Стратегия гибридной сборки представляет собой объединение шагов компоновки изображения и распределения итогового изображения по процессорам для отображения на мультидисплейном комплексе. Возможно применение различных алгоритмов компоновки изображения при таком подходе. Однако различные стратегии будут давать различный результат при исполнении программы на различных параллельных архитектурах. Это связано с тем, что аппаратно не обеспечивается одинаковая скорость при коммуникации между различными процессорами. Для эффективной реализации параллельной компоновки изображения и его распределения возможно использование коллективных операций MPI, в частности операции reduce.

Для использования этой операции требуется обеспечить дополнительное условие, что операция компоновки изображения будет ассоциативной по отношению к порядку, котором происходит сборка изображения. Однако в случае компоновки изображения это условие может быть ослаблено. Достаточно, чтобы ошибка, которая вносится при разном порядке компоновки, была визуально не заметна.

Наибольшего ускорения возможно было бы добиться при использовании неблокурующих вызовов reduce, но такая функциональность будет поддерживаться только в стандарте MPI-3. В текущем стандарте MPI используются блокирующие операции reduce, поэтому компоновка итогового изображения для M дисплеев эквивалентна последовательному выполнению M вызовов reduce для каждого из процессоров, к которому подключен дисплей. Каждый шаг операции reduce проводится для части изображения, которая соответствует текущему дисплею. В общем виде схему выполнения операции можно представить следующим образом (Рис.2).

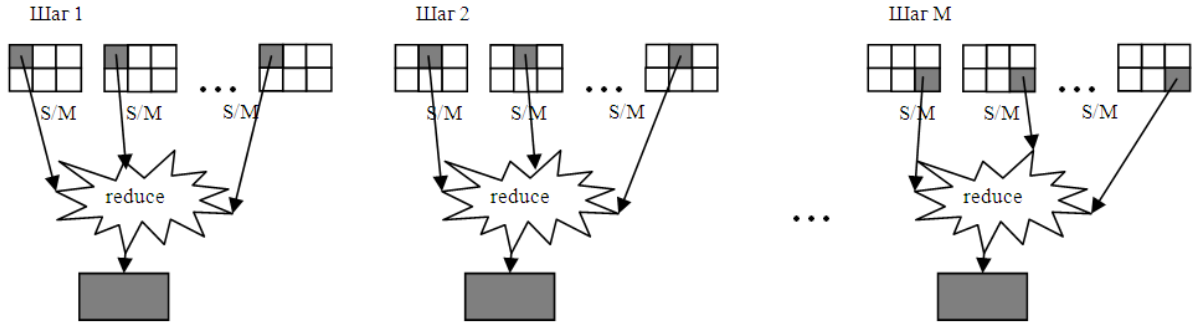


Рис.2. Схема компоновки изображения для мультидисплейного комплекса с использованием MPI-операции reduce.

Теоретически выполнение операции reduce на каждой конкретной параллельной архитектуре будет выполняться быстрее, чем произвольный алгоритм компоновки изображения, т.к. разработчики аппаратного и системного программного обеспечения реализуют базовые функции MPI максимально эффективно для своей архитектуры. В частности, на суперкомпьютере Blue Gene/P используется специальная коммуникационная сеть для выполнения коллективных операций, что позволяет получить существенных выигрыш по времени выполнения по сравнению с самостоятельно реализованными алгоритмами обменов, использующих коммуникации точка-точка. Поэтому получение теоретической оценки ожидаемого времени выполнения затруднительно, т.к. требует детального знания архитектуры конкретной параллельной системы и особенностей реализации операций MPI. Однако эти оценки могут быть получены экспериментально. Соответствующий эксперимент был выполнен на суперкомпьютере Blue Gene/P и его результаты приведены в разделе 4 данной работы.

3.2 Использование балансировки по данным для оптимизации времени выполнения

При визуализации научных данных в получаемом изображении далеко не все пиксели являются значащими. Больш*а*я часть пикселей относится к фоновым и имеет постоянное значение. Для эффективной передачи данных можно использовать различные алгоритмы сжатия перед компоновкой. Однако использование сжатия приводит дисбалансу загруженности процессоров, т.к. у некоторых процессоров окажется часть изображения, где большинство пикселей значащие, а некоторые процессоры получат только фоновые пиксели, которые будут эффективно сжаты.

В качестве одного из подходов к балансировке загруженности процессоров при использовании сжатия может быть применено неравномерное разбиение по площади для частей, которыми обмениваются процессоры. Но такой подход не применим, если в итоге должно быть получено не одно изображение, а несколько изображений для мультидисплейного комплекса.

Требуется предложить способ разбиения данных между процессорами, при котором возможно будет использовать сжатие, при этом будет сохранена равномерная загруженность процессоров и в итоге получится изображение для мультидисплейного комплекса. Для выполнения этой задачи предполагается следующий подход. Предлагается выполнять операцию reduce не для части данных, которые относятся к одному дисплею, а по полосе данных, которая включает в себя данные для q дисплеев в ряду. При таком способе получается более равномерное распределение значащих пикселей. Далее полученное сжатое изображение пересылается на q процессоров с помощью MPI операции scatter.

4. Экспериментальная оценка эффективности и масштабируемости на Blue Gene /P

4.1 Описание эксперимента

Эксперимент проводился на суперкомпьютере Blue Gene/P МГУ. Получения экспериментальных оценок эффективности выполнения была использована реализация 5 методов:

1) последовательный (SS)
2) бинарных обменов (BS)
3) конвейерный метод (RS)
4) метод на основе reduce-вызовов (RED)
5) метод на основе reduce-вызовов с использованием оптимизации по данным (REDopt)

Методы 1) - 3) были реализованы на основе стратегий библиотеки IceT[15]. Эксперимент проводился для визуализации данных молекулярной динамики для молекул с порядком атомов 10^5. Однако в эксперименте оценивалось только время выполнения и эффективность компоновки изображения, поэтому входными данными алгоритма можно считать изображения в S пикселей, полученные на каждом процессоре, по данным, которые у него были.

Проведены следующие эксперименты:

1) анализ времени выполнения на различном количестве процессоров при фиксированном размере изображения и фиксированной сетке мультидисплейного комплекса;
2) анализ времени выполнения при изменяющемся размере изображения;
3) анализ времени выполнения при изменяющейся сетке мультидисплейного комплекса.

В экспериментах использовалось 128, 256, 512, 1024 процессора. Размер изображения менялся от 2000x2000, 1000x1000, 500x500 для каждого дисплея из мультидисплейного комплекса. Сетки мультидисплейного комплекса брались размером 2x2, 2x4 и 4x4.

4.2 Результаты

Графики полученных в вычислительном эксперименте результатов приведены на Рис.3.

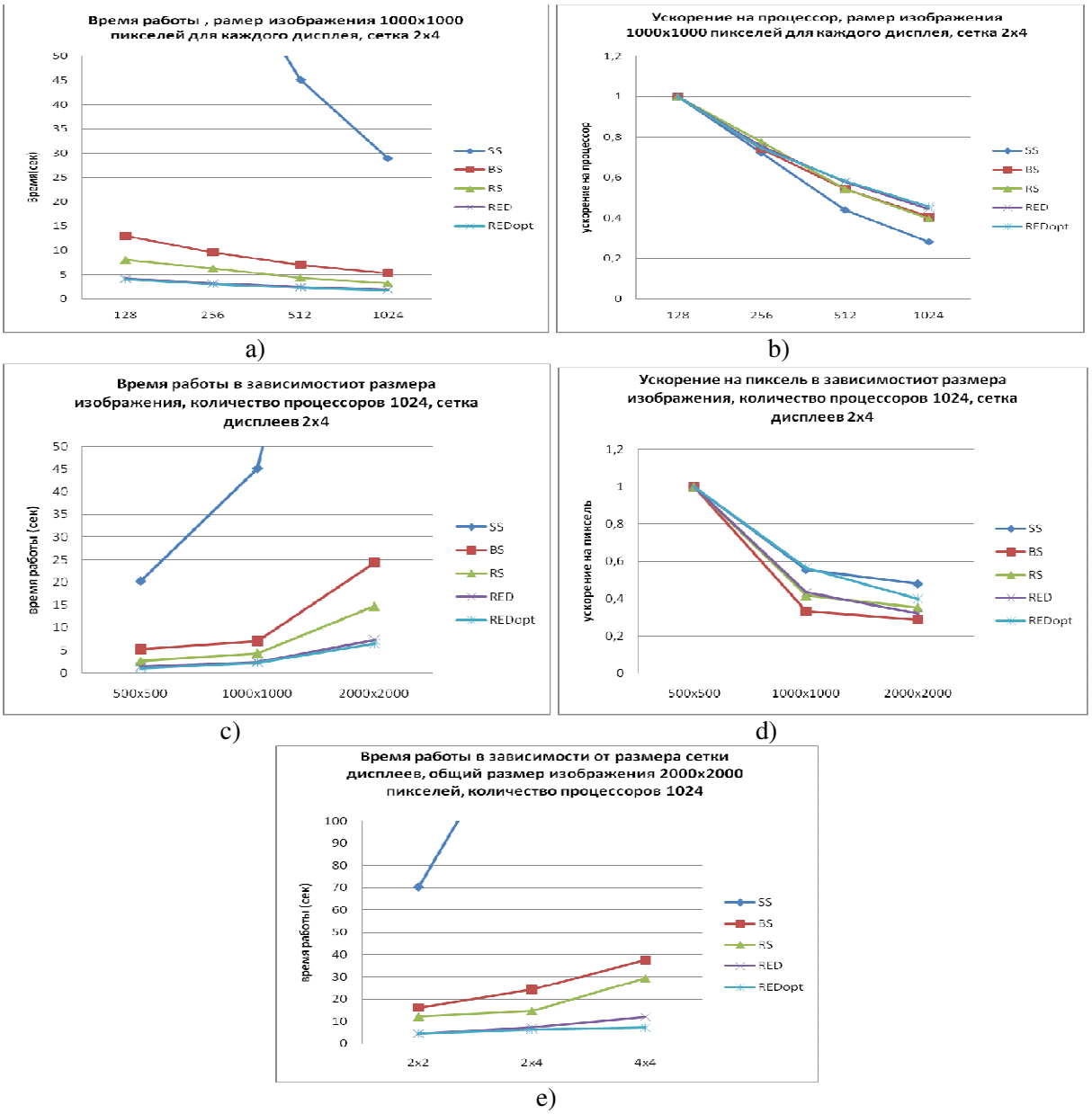


Рис.3. Результаты вычислительных экспериментов на Blue Gene/P.

На Рис.3a показан график времени работы методов при изменении количества процессоров. При этом фиксирован размер изображения для каждого дисплея 1000x1000 пикселей и сетка мультидисплейного комплекса: размер 2x4. На Рис.3b показано ускорение на процессор в этом же эксперименте. Как видно из графика, полученное решение довольно плохо масштабируемо и дает низкую эффективность при большом количестве процессоров. Однако предложенные методы RED и REDopt более эффективны, чем известные ранее методы.

На Рис.3c показано время работы методов при использовании различного размера изображения для каждого из дисплеев. При этом фиксировано количество процессоров – 1024 и сетка мультидисплейного комплекса 2x4. На Рис.3d показано ускорение на пиксель в зависимости от размера изображения. Использование в методе RED оптимизации по данным и сжатия передаваемых данных позволило существенно улучшить показатель по этому параметру.

На Рис.3e показана зависимость времени работы метода от изменения сетки дисплеев. При этом фиксировано значение размера входного изображения 2000x2000 пикселей и количество используемых процессоров: 1024x1024. Как видно из графика оптимизация по данным позволила уменьшить время работы алгоритма RED на сетке из 16 дисплеев в два раза. Необходимо

провести дополнительное исследование на сетке большей размерности и при большем общем размере изображения.

5. Заключение

В работе проведен анализ методов параллельной компоновки изображения при визуализации научных данных большого размера на суперкомпьютерах методом объемного построения изображений. Приводится теоретическая оценка сложности методов для случая, когда на выходе получается не одно изображение, а несколько для отображения на мультидисплейном комплексе. Предложен гибридный метод компоновки изображения, объединяющий стадии компоновки изображения и распределения изображения между несколькими экранами. Предложена оптимизация это метода, позволяющая сократить размеры передаваемых данных и соответственно общее время работы метода. Проведено экспериментальное исследование эффективности предложенного метода и его оптимизации на суперкомпьютере Blue Gene/P. В дальнейших исследованиях предполагается совершенствование метода сжатия данных для ускорения работы программы, исследование масштрабируемости предложенных методов на суперкомпьютере «Ломоносов» и оптимизация предложенных методов для этого суперкомпьютера.

Литература

1. Götz J., Iglberger K., Stürmer M., Rüde U. Direct Numerical Simulation of Particulate Flows on 294912 Processor Cores // Proc. Conf. High Performance Computing, Networking, Storage and Analysis. Washington, DC, USA, pp.1 -11, 2010.
2. The SciDAC Ultra-Scale Visualization Institute. URL:http://vis.cs.ucdavis.edu/Ultravis/datasets/ (дата обращения: 10.01.2011).
3. Survey of Parallel Volume Rendering Algorithms URL:www.hpl.hp.com/research/mmsl/presentations/3d/pdpta98.pdf (дата обращения: 10.01.2011).
4. Mueller C. The sort-first rendering architecture for high-performance graphics // In ACM SIGGRAPH ASIA 2008 courses (SIGGRAPH Asia '08). New York, USA, Article 36 , 11 pages, 2008.
5. Crockett T.W., Orloff T. A MIMD rendering algorithm for distributed memory architectures // Proceedings of the symposium on Parallel rendering -1993, p.35-42, San Jose, California, USA, 1993.
6. Pajarola R. Cluster parallel rendering // In ACM SIGGRAPH ASIA 2008 courses (SIGGRAPH Asia '08). New York, USA, Article 34, 12 pages, 2008.
7. Martin K.M., Geveci B., Ahrens J., Law C. Large Scale Data Visualization Using Parallel Data Streaming // IEEE Comput. Graph. Appl., Vol. 21, pp. 34-41, 2001.
8. Berkant B.C., Cevdet A. Hypergraph-Partitioning-Based Remapping Models for Image-Space-Parallel Direct Volume Rendering of Unstructured Grids // IEEE Trans. Parallel Distrib. Syst., Vol.18, pp. 3-16., 2007.
9. Moloney B., Ament M., Weiskopf D., Möller T. Sort First Parallel Volume Rendering // IEEE Transactions on Visualization and Computer Graphics, Vol.9, 2010.
10. Finkbeiner B., Entezari A., Van De Ville D., Möller T. Efficient volume rendering on the body centered cubic lattice using box splines // Computers & Graphics, Vol. 34(4). pp. 409 - 423, 2010.
11. Cedilnik A., Geveci B., Moreland K., Ahrens J., Favre J. Remote Large Data Visualization in the ParaView Framework // A. Heirich, B. Raffin, L. P. Santos (eds.) editors, Eurographics Parallel Graphics and Visualization 2006, pp. 162--170, 2006.

12. Moreland K., Avila L., Lee Ann Fisk. Parallel Unstructured Volume Rendering in ParaView. // In Visualization and Data Analysis 2007, Proceedings of SPIE-IS&T Electronic Imaging, vol. 6495, pp. 64950F-1–12, 2007.

13. Foulks A., Bergeron R.D. Uncertainty visualization in the VisIt visualization environment // Proceedings Visualization and Data Analysis 2009, Vol. 7243, 2009.

14. VisIt Visualization Tool. URL: https://wci.llnl.gov/codes/visit/ (дата обращения: 10.01.2011).

15. Moreland K. IceT Users' Guide and Reference. Tech Report SAND 2009-3170, June 2009.

16. Peterka T., Goodell D., Ross R., Han-Wei Shen, Rajeev Thakur. A configurable algorithm for parallel image-compositing applications // Proc.Conf. High Performance Computing Networking, Storage and Analysis. New York, USA, Article 4 , 10 pages, 2009.

17. Porter T., Duff T. Compositing digital images // SIGGRAPH Comput. Graph., Vol. 18, pp. 253-259, 1984.

18. Moreland K., Wylie B., Pavlakos C. Sort-last parallel rendering for viewing extremely large data sets on tile displays // In Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics. IEEE Press, Piscataway, USA, pp. 85-92. 2001.

19. Ma K.-L., Painter J. S., Hansen C. D., Krogh M. F. Parallel volume rendering using binary-swap compositing // IEEE Computer Graphics and Applications, vol. 14, №4, pp. 59–68, 1994

20. Yu H., Wang C., Ma K.-L., Massively parallel volume rendering using 2-3 swap image compositing // Proc. Conf. Supercomputing. Piscataway, USA: IEEE Press, pp. 1–11, 2008.

21. Lee T.-Y., Raghavendra C. S., Nicholas J. B. Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers // IEEE Transactions on Visualization and Computer Graphics, vol. 2, no. 3, pp. 202–217, 1996.

22. Lin C.F., Liao S.K., Chung Y.C. A rotate-tiling image compositing method for sort-last parallel volume rendering systems on distributed memory multicomputers // Journal of Information Science and Engineering, pp.643-664, 2004.

23. Nonaka J., Ono K., Miyachi H. Theoretical and Practical Performance and Scalability Analyses of Binary-Swap image Composition Method on IBM Blue Gene/L // The 1st International Workshop on Super Visualization (IWSV), June 7, 2008, Kos Is., Greece, 2008.

24. Takeuchi A., Ino F., Hagihara K. An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors // Parallel Comput., vol. 29, no. 11-12, pp. 1745–1762, 2003.

25. Peterka T., Goodell D., Ross R., Shen H.-W., Thakur R. A Configurable Algorithm for Parallel Image-Compositing Applications // Proc. Conf. High Performance Computing, Networking, Storage, and Analysis, Portland, Oregon, USA, 2009. Preprint ANL/MCS-P1624-0509, May 2009.

26. Джосан О.В., Мурынин А.Б., Попова Н.Н. Метод визуализации многомерных динамических данных на многопроцессорных комплексах // Вестник компьютерных и информационных технологий. 2009. № 8. сс. 8-12.

27. Корж А.А., Макагон Д.В., Оценка минимальных требований к аппаратуре и топологии при построении высокоскоростных коммуникационных сетей для суперкомпьютеров с общей памятью // Вычислительные методы и программирование: новые вычислительные технологии. 2008. Т. 9. № 2. сс. 26-31.

Параллельный код AstroChemHydro для моделирования химико–динамических процессов в межзвездной среде*

М.А. Еремин[1], В.Н. Любимов[1], Е.О. Васильев[2]

Волгоградский государственный университет[1],
Южный федеральный университет[2]

Разработан и реализован параллельный трехмерный численный код для моделирования химической и динамической эволюции межзвездной среды. С использованием разработанного комплекса программ было выполнено численное моделирование физико-химической эволюции облака межзвездного газа, взаимодействующего с плоской ударной волной. Проведенные расчеты позволяют проследить превращения химических реагентов в результате динамического взаимодействия облака с ударной волной и изучить кинетику молекулярного водорода в зависимости от различного набора параметров.

1. Введение

Наблюдения являются основным способом получения информации об астрофизических объектах, однако астрономические наблюдения осложнены целым рядом факторов: во-первых, удаленностью объектов изучения, во-вторых, поглощением излучения в межзвездной среде на пыли, и в-третьих, поглощением в атмосфере Земли. В отличие от других разделов физики, в астрофизике проведение экспериментов за редкими исключениями невозможно, что обусловлено, с одной стороны огромными пространственными и временными масштабами, и с другой, специфическими процессами, протекающими в астрофизических системах, которые зачастую не воспроизводимы в земных условиях.

В последние двадцать лет в связи с прогрессом в области компьютерных технологий роль численного моделирования в астрофизике резко возросла, поскольку именно оно позволяет проследить эволюцию различных астрофизических систем исходя из “первых принципов”. Например, компьютерное моделирование широко применяется для верификации космологических моделей, исследования эволюции галактик и т.д. Для корректной интерпретации астрономических наблюдений, а прежде всего это данные о химическом составе, необходимо самосогласованное моделирование динамической и химической эволюции межзвездного газа. Подобного рода задачи являются чрезвычайно сложными и ресурсоемкими. Это связано, например, с тем, что характерные времена протекания химических реакций и гидродинамических процессов отличаются на много порядков. Моделирование динамики химически реагирующих газов, предъявляет особые требования и к объему необходимой оперативной памяти. Отметим, что учет переноса излучения, совершенно необходимый в подобного рода задачах, до сих пор выполняется в рамках достаточно грубых приближений! В межзвездной среде взаимодействие ударных волн, джетов и звездного ветра с облаками межзвездного газа имеет место при больших числах Рейнольдса, и является турбулентым. Таким образом, моделирование процессов, протекающих в астрофизических системах, может быть выполнено только с применением современных численных схем высокого разрешения и технологий параллельных вычислений.

Укажем на тот факт, что пространственное распределение межзвездной среды (МЗС) крайне неоднородно, вещество в МЗС сосредоточено в областях различной плотности и температуры: газовые и газово-пылевые облака, глобулы Бока, гигантские молекулярные

*Работа поддержана грантом ВолгУ 70-2011 – а/ВолГУ, и грантами РФФИ N08-02-009933, N09-02-97019, N09-02-97021, а также ФЦП Рособразование госконтракт П1248.

облака, теплая межоблачная среда и зоны HII и т.д. Все это приводит к существенным различиям в физико-химических свойствах различных составляющих межзвездной среды. Например, температура газа в плотных молекулярных облаках, являющимися центрами активного звездообразования, составляет всего несколько Кельвин, в то время как в горячем корональном газе, образующемся при взрывах сверхновых, температура межзвездного газа достигает нескольких милллионов Кельвин, различие плотности газа в этих составляющих межзвездной среды также оценивается как минимум в шесть порядков [1–3]!

Газ в межзвездной среде находится в основном в двух фазах: в теплой разреженной фазе с температурой порядка 10^4 К и холодной плотной фазе с температурой около 100 К [4,5]. Наибольшая часть массы межзвездного вещества сосредоточена в облаках различной плотности и размера, динамика и взаимодействие которых оказывают значительное влияние на эволюцию межзвездной среды.

Для описания эволюции МЗС важно иметь модели, адекватно описывающие динамику облаков. На эволюцию облака помимо других факторов может оказывать значительное влияние его взаимодействие с ударными волнами, образующимися, например, при вспышках сверхновых. Кроме того, химическая и динамическая трансформации облака могут быть обусловлены его прохождением через галактическую ударную волну. В результате этого взаимодействия может значительно меняться не только форма и размер, но и химический состав облака. Химические реакции, в свою очередь, также могут оказывать сильное влияние на динамику процесса, являясь эффективным способом отвода тепла.

Основными целями нашей работы являются:

1. Разработка и реализация параллельного трехмерного кода AstroChemHydro для моделирования процессов в межзвездной среде с учетом химических реакций.

2. Численное моделирование физико-химической эволюции облака межзвездного газа, взаимодействующего с плоской ударной волной.

2. Постановка задачи

2.1. Уравнения газодинамики смеси газов

Система уравнений, описывающая течение смеси химически реагирующих газов в одножидкостном приближении, записывается в виде [6, 7]:

$$\frac{\partial \rho X_i}{\partial t} + \nabla \cdot (\rho \mathbf{u} X_i) = \rho \sigma_i \,, i = 1, ..., n_s, \quad (1)$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \,, \quad (2)$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla \mathrm{p} \,, \quad (3)$$

$$\frac{\partial E}{\partial t} + \nabla \cdot ([E + \mathrm{p}]\mathbf{u}) = -L \,. \quad (4)$$

Здесь ρ, p, $\mathbf{u} = \{u, v, w\}$ – плотность, давление и вектор скорости смеси газов соответственно, $X_i = \rho_i / \rho$ – массовая доля (концентрация) i-го газа, σ_i – массовая скорость производства i-го компонента, определяемая набором химических реакций, n_s – количество реагентов в смеси, $E = \rho \left(e + \frac{\mathbf{u}^2}{2} \right)$ – объемная энергия, e – удельная внутренняя энергия, $L = \Lambda - \Gamma$ – функция неравновесных тепловых потерь, $\Lambda = \Lambda(T, \rho^{\{j\}})$ – скорость (темп) объемного охлаждения, $\Gamma = \Gamma(\nu, \rho^{\{j\}})$ – скорость объемного нагрева. Отметим, что в химическом равновесии все σ_i тождественно равны нулю.

Плотность i-го компонента определяется через его атомный вес A_i и концентрацию n_i по формуле:

$$\rho_i = A_i m_p n_i \,,$$

где m_p – масса протона.

Массовые доли газов, входящих в состав смеси, удовлетворяют соотношению нормировки:

$$\sum_{i=1}^{n_s} X_i = 1. \tag{5}$$

Давление p определяется уравнением состояния, которое связывает давление и температуру газа с плотностью и удельной тепловой энергией. Для замыкания системы (1)–(4) в данной модели будем использовать уравнение состояния в форме уравнения состояния для идеального газа:

$$p = \rho e(\gamma - 1) = \frac{\rho k_B T}{\mu m_p}, \tag{6}$$

где γ – показатель адиабаты (для одноатомного газа $\gamma = 5/3$), k_B — постоянная Больцмана, m_p – масса протона. Средняя молекулярная масса смеси μ вычисляется следующим образом

$$\frac{1}{\mu} = \sum_{k=1}^{n_s} \frac{X_k}{A_k}\,. \tag{7}$$

Система уравнений, описывающая невязкие течения многокомпонентной смеси химически реагирующих газов (1)–(4) записывалась в безразмерной форме с использованием нескольких базисных величин. Перечислим наиболее важные размерные параметры задачи: $L_0 = 1$ пк – характерный пространственный масштаб, $T_0 = 10^4$ K — температура, $n_0 = 0.1\,\text{г/см}^3$ – концентрация, $c_{s0} \simeq 12$ км/с — адиабатическая скорость звука, $t_0 \approx 10^5$ лет — характерное время задачи.

2.2. Описание химической модели

Уравнения для моделирования газовых течений с химическими реакциями представляют собой нестационарные трехмерные уравнения для плотности газа, плотностей химических компонентов, скорости и энергии. В общем виде уравнения сохранения массы химических компонентов можно записать в виде:

$$\frac{\partial \rho X_i}{\partial t} = -\nabla \cdot (\rho \mathbf{u} X_i) - \nabla(\rho \mathbf{v}_{d,i} X_i) + \rho \sigma_i\,. \tag{8}$$

При условии малой диффузии отдельных компонентов ($u \gg v_{d,i}$) уравнение сохранения массы i-го компонента можно рассматривать отдельно как перенос пассивного компонента (первое слагаемое) и химические превращения компонента (последнее слагаемое).

Массовая скорость производства i-го компонента зависит от скорости его образования F_i и разрушения D_i , так что

$$\sigma_i = F_i - D_i X_i,\ i = 1, ..., n_s\,.$$

В соответствии с идеей метода расщепления по физическим процессам, решение системы уравнений (8) строится как совокупность решений уравнений адвекции и химической кинетики. Таким образом, для переноса каждого из химических компонентов записывается отдельное уравнение непрерывности и отдельно решается система уравнений химической кинетики:

$$\frac{dX_i}{dt} = F_i - D_i X_i,\ i = 1, ..., n_s\,. \tag{9}$$

Отметим, что скорости образования и разрушения являются функциями, сильно зависящими от температуры, так что система уравнений химической кинетики, как правило, является жестокой. Уравнения переноса (адвекции) химических компонентов решаются в строгом

Таблица 1. Процессы охлаждения и нагрева, включенные в модель

процесс	ссылка
H возбуждение	[13]
H столкновительная ионизация	[13]
H^+ рекомбинация	[13]
He возбуждение	[13]
He столкновительная ионизация	[13]
He^+ рекомбинация	[13]
свободно-свободные переходы (H^+, He^+)	[13]
комптоновское взаимодействие с квантами РИ	[13]
H_2 вращательно-колебательные переходы	[10, 12]
H_2 столкновительная диссоциация	[14]
рекомбнация на пылинках	[15]
столкновения газ-пыль	[15]
CII 2326 Å	[15]
OI 6300 Å	[15]
CI (609 μm), тонкая структура, 3 уровня	[12]
CII (158 μm), тонкая структура, 2 уровня	[12]
OI (63 μm), тонкая структура, 3 уровня	[12]
ионизация космическими лучами	[10]
образование H_2 в газе	[16]
образование H_2 на пыли	[12]

соответствии с выбранной численной схемой для переноса плотности газа. Для решения системы жестких дифференциальных уравнений химической кинетики в параллельном коде AstroChemHydro используется стандартный пакет DVODE [8].

Химическая кинетика межзвездного газа включает в себя следующие основные компоненты: H, H^+, H^-, He, He^+, H_2, H_2^+, C, C^+, O и O^+. Для расчета концентрации электронов мы предполагаем выполнение закона сохранения заряда. Массовая доля гелия предполагалась равной $Y_{\rm He} = 0.24$. Обилия углерода и кислорода приняты равными солнечным [9]: $({\rm C/H})_\odot = 2.45 \times 10^{-4}$, $({\rm O/H})_\odot = 4.57 \times 10^{-4}$. Скорости химических реакций для столкновительных и радиационных процессов были взяты из [10, 11]. В системе химических реакций

учтен процесс образования молекулярного водорода на пылинках [12]:

$$H \ + \ H(+\text{пыль}) \ \rightarrow \ H_2 \, ,$$

который является доминирующим каналом образования H_2 в межзвездной среде с металличностью близкой к солнечному значению.

В уравнении для энергии были учтены основные процессы радиационных потерь и нагрева, характерные для межзвездного газа (таблица 1). Отметим, что для скорости охлаждения во вращательно-колебательные переходах молекулярного водорода взята аппроксимация для низкой плотности, $L(n \rightarrow 0)$, из работы [10]. Скорость охлаждения H_2 для любой плотности рассчитывалась с помощью формализма, разработанного в [12].

2.3. Численный метод интегрирования уравнений

Для компьютерного моделирования химико-динамических процессов в межзвездной среде нами была реализована явная численная схема для системы (1)–(4) в декартовой системе координат. При построении численной схемы использовался метод расщепления по физическим процессам, согласно которому численное решение строится как решение уравнений, описывающих различные физические процессы.

В данном разделе кратко опишем алгоритм, использованный для численного интегрирования уравнений (1)–(4), описывающих течение смеси химически взаимодействующих газов. Система (1)–(4) может быть записана в виде:

$$\frac{\partial \mathbf{U}}{\partial t} = -\nabla \cdot \mathbf{U} + \mathbf{S}, \tag{10}$$

где $\mathbf{U} = [\rho X_i, \rho, \rho \mathbf{u}, \rho E]^{\mathrm{T}}$ – вектор консервативных переменных, $\mathbf{S} = \mathbf{S}_{react} + \mathbf{S}_{therm}$ – источниковый член, состоящий из двух слагаемых различной природы: $\mathbf{S}_{react}$ – вектор, описывающий изменение концентрации i - го компонента за счет химических реакций, $\mathbf{S}_{therm}$ – источниковое слагаемое, описывающее изменение внутренней энергии смеси за счет неравновесных процессов охлаждения и нагрева.

Для дискретизации системы уравнений (1)–(4) мы использовали метод расщепления Стрэнга. На первом шаге, решалась система уравнений химической кинетики на первой половине временного шага $\Delta t/2$ и учитывалось изменение внутренней энергии. Консервативные переменные модифицировались следующим образом:

$$\mathbf{U}^* = \mathbf{U}^n - \frac{\Delta t}{2} \mathbf{S}^n. \tag{11}$$

На этом шаге происходит изменение плотностей взаимодействующих газов, вызванное химическими реакциями, и внутренней энергии смеси за счет процессов нагрева и охлаждения. Заметим, что плотность смеси предполагается неизменной в течении этого временного шага.

На втором шаге рассчитывалось изменение консервативных величин за счет адвекции и работы сил давления в пренебрежении источниковых слагаемых, так что

$$\mathbf{U}^{**} = \mathbf{U}^* - \Delta t - \nabla \cdot \mathbf{F}^{n+1/2} \, , \tag{12}$$

где $\mathbf{F}^{n+1/2}$ – потоки величин через границы ячеек на шаге $n + 1/2$.

Если в некоторых ячейках решение оказывается нефизичным, то есть плотность смеси или массовая доля какого-либо компонента оказывается отрицательной, то в этих ячейках мы искусственно полагаем плотность равной неотрицательной малой величине, которая определяется пользователем.

На третьем шаге система уравнений химической кинетики интегрировалась на второй половине временного шага $\Delta t/2$ и мы корректировали внутреннюю энергию в соответствии с формулой

$$\mathbf{U}^{n+1} = \mathbf{U}^{**} - \frac{\Delta t}{2}\mathbf{S}^{**}. \tag{13}$$

На четвертом, заключительном шаге, происходит вычисление нового временного шага из условия устойчивости Куранта–Фридрихса–Леви:

$$\Delta t = \mathcal{C} \min_{i,j,k}\left(\frac{\Delta x}{|u_{i,j,k}| + c}, \frac{\Delta y}{|v_{i,j,k}| + c}, \frac{\Delta z}{|w_{i,j,k}| + c}\right), \tag{14}$$

где u, v, w – компоненты вектора скорости, Δx, Δy, Δz – размеры ячеек в x, y и z направлениях соответственно, c–скорость звука, определяемая из уравнения состояния, $\mathcal{C} < 1$.

Обратим внимание на то, что структура уравнений системы (1)–(4) не изменилась после введения уравнений непрерывности для компонент смеси, и несмотря на наличие источниковых слагаемых в правой части, она остается гиперболической. Для уравнений гиперболического типа разработаны схемы высокого разрешения, имеющие высокий порядок аппроксимации в областях гладкого течения и обеспечивающие отсутствие паразитных осцилляций на скачках [17].

В параллельном трехмерном коде AstroChemHydro для численного решения уравнений химической динамики многокомпонентной смеси газов используется явная конечно-объемная схема неубывания полной вариации (TVD). Опишем некоторые особенности реализованной схемы. Во-первых, пространственно-нерасщепленная TVD схема относится к типу MUSCL [18], имеет третий порядок аппроксимации по пространству в областях гладкого течения и первый на скачках. Во-вторых, за счет применения алгоритмов пересчета типа Рунге-Кутта, реализованная численная схема обладает вторым порядком по времени. В-третьих, для вычисления потоков величин через границы дискретных ячеек использовался приближенный метод Хартена–Лакса–ван Леера с учетом контактного разрыва (HLLC) [19].

2.4. Архитектура кода AstroChemHydro

Параллельный численный код AstroChemHydro написан на языке C++ с использованием объектно–ориентированной парадигмы программирования. В основу реализованного программного обеспечения положен модульный принцип, позволяющий легко переходить от расчета одной физической модели к другой. Входные параметры хранятся в отдельных файлах. Объем необходимой машинной памяти определяется требуемым пространственным разрешением и точностью расчета. Результаты расчета записываются через определенные промежутки времени в различные файлы в формате, определяемом пользователем (бинарном или текстовом). Предусмотрена возможность продолжения расчета с любого заданного момента времени.

Особого внимания, на наш взгляд, заслуживает реализация параллельной версии кода AstroChemHydro. Для облегчения процесса создания параллельных трехмерных программ нами была создана библиотека MPIParallel3D для распараллеливания трехмерных вычислительных программ для компьютеров с массивно-параллельной архитектурой. Разработанная библиотека позволяет в полуавтоматическом режиме производить распараллеливание трехмерных расчетных программ для MPP архитектур. Инструмент предназначен в первую очередь для распараллеливания по данным с трехмерной декомпозицией. Библиотека является объектно-ориентированной и реализована в виде класса MPIParallel3D на языке C++.

Класс MPIParallel3D включает методы, позволяющие выполнять основные операции с данными, необходимые в параллельной программе, без обращения к низкоуровневым функциям стандарта MPI [20, 21]:

- Функции для копирования или автоматического распределения между процессами одномерных и многомерных массивов, и обмена границами между ними.
- Коллективные действия над переменными: синхронизация переменных, сравнение, выбор максимума или минимума среди значений на всех процессах и др.
- Функции для вывода в файл распределенных одномерных и многомерных массивов.
- Вспомогательные функции, полезные при отладке параллельной программы – вывод распределенных одномерных и многомерных массивов на консоль, приостановка программы с выводом пользовательского сообщения и другие.

Все функции автоматически производят взаимодействие процессов исходя из параметров декомпозиции, инкапсулированных в классе. Единственным входным параметром является размеры расчетной области, передаваемые в объект класса. Исходя из вида декомпозиции, заданной пользователем (линейная или кубическая), автоматически определяются параметры распределения расчетной области между процессами. В пользовательском режиме помимо трехмерной, возможна двумерная или одномерная декомпозиции.

Основной целью при написании данной библиотеки являлось максимальное упрощение распараллеливания вычислительного кода без необходимости постоянной отладки стандартных операций коммуникации процессов. При этом мы пытались создать наиболее универсальный инструмент, не привязанный к определенной структуре программы или архитектуре вычислительной системы. Таким образом, данный класс можно использовать для распараллеливания большинства программ, в которых предполагается параллельность по данным.

Разработанная библиотека была использована для распараллеливания численного кода AstroChemHydro. Применение данного инструмента значительно упростило и ускорило процесс распараллеливания, даже для человека имеющего слабое представление о технологиях параллельного программирования. Следует отметить высокую эффективность полученной параллельной версии программы, которую удалось достигнуть благодаря использованию различных алгоритмов обмена границами в библиотеке MPIParallel3D. Проведенные тесты показывают, что эффективность параллельной версии кода составляет $70 - 90\%$ в зависимости от параметров задачи.

2.5. Моделирование взаимодействия облака межзвездного газа с ударной волной

В качестве примера применения разработаного кода нами было проведено двумерное численное моделирование физико-химической эволюции облака межзвездного газа, взаимодействующего с плоской ударной волной. Изучение этого процесса с помощью многомерного динамического моделирования с учетом различных физических факторов проводится достаточно давно [22–24], однако совместное химико-динамическое исследование стало возможным только в последние несколько лет. Установленное существенное влияние химических процессов на динамическую и тепловую эволюцию газа указывает на необходимость такого рода исследований. Кроме того, изучая самосогласованно химические превращения мы можем получить физические величины, характеризующие реальные космические объекты, наблюдаемые с помощью оптических и радио- телескопов, интерпретировать наблюдения и понять природу этих объектов.

В представленном примере химические процессы описываются 26 реакциями с участием 11 компонент. Проведенные расчеты позволяют оценить изменение массы молекулярного водорода и других реагентов в результате взаимодействия облака с ударной волной. В задаче взаимодействия ударная волна – облако важен учет молекуляризация водорода на пыли, а в уравнении для энергии были учтены процессы радиационных потерь во вращательно-колебательных линиях молекулярного водорода.

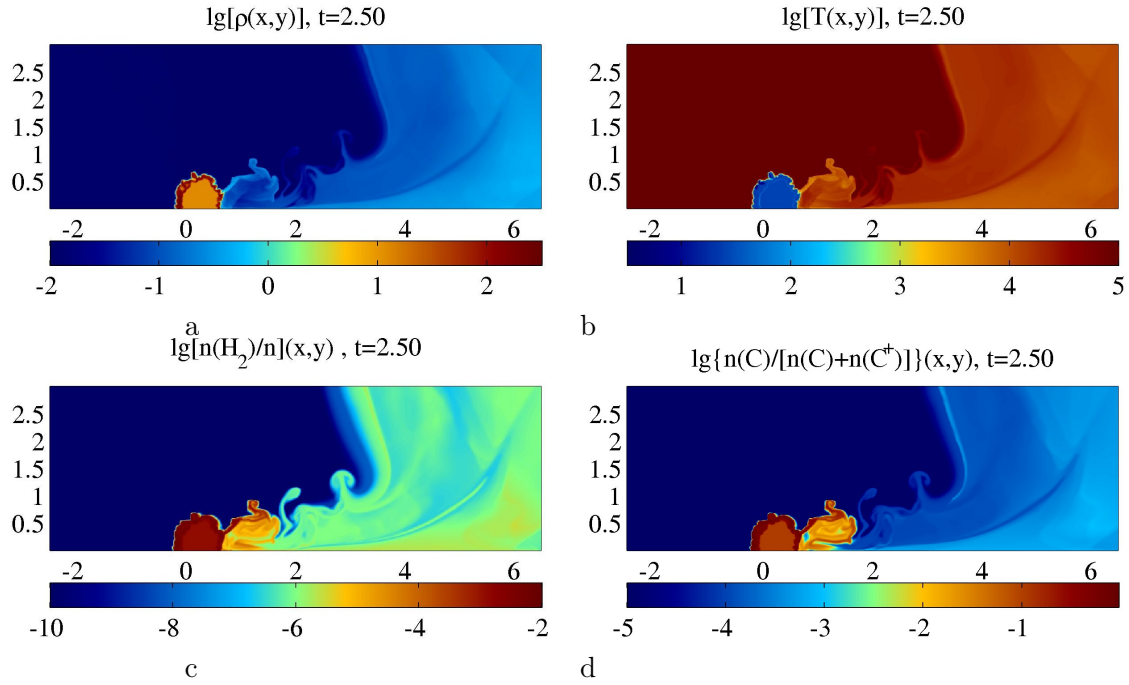


Рис. 1. Распределение логарифма плотности, логарифма температуры, логарифма относительного содержания молекулярного водорода H_2 и логарифма относительного содержания углерода C в момент времени $t = 2.5 \cdot 10^5$ лет для модели 1

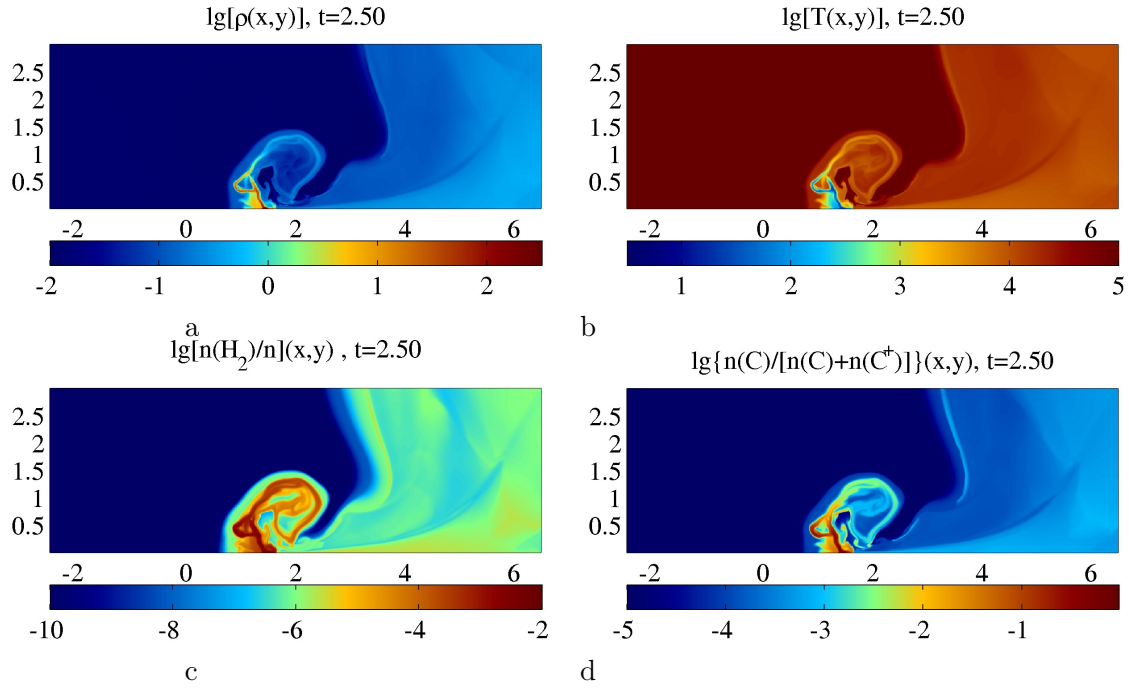
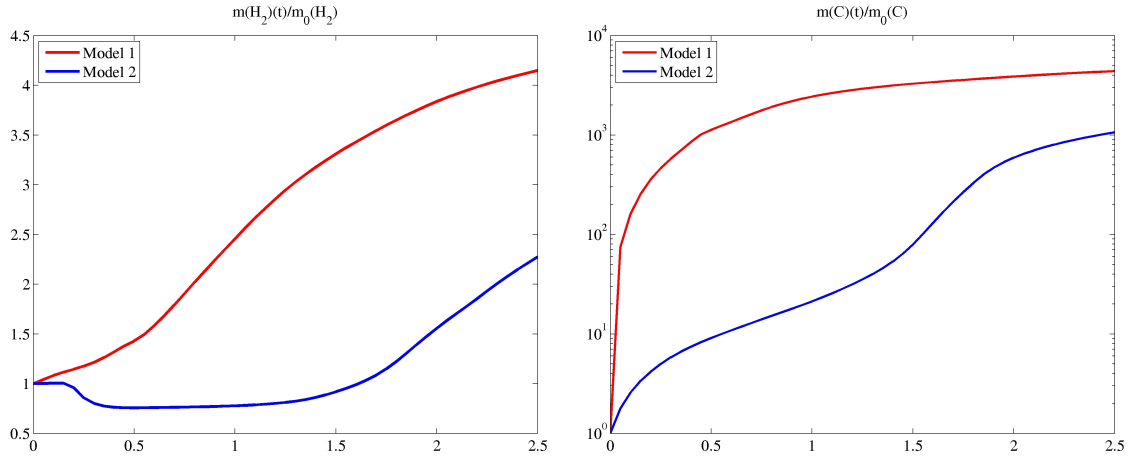


Рис. 2. Распределение логарифма плотности, логарифма температуры, логарифма относительного содержания молекулярного водорода H_2 и логарифма относительного содержания углерода C в момент времени $t = 2.5 \cdot 10^5$ лет для модели 2

В численных моделях рассматривалось сферическое облако радиусом 1 пк. Температура облака варьировалась в пределах от 10^2 К (модель 1) до 10^3 К (модель 2), концентрация от 1 до 10 $см^{-3}$. Теплая фаза межзвездной среды задавалась с температурой 10^4 К и концентрацией 0.1 $см^{-3}$. Температура на фронте ударной волны составляла порядка 10^5 К.

Моделирование показало, что при взаимодействии облака с ударной волной наблюдается сильное охлаждение его внешних слоев. В результате образуется плотная оболочка, в которой эффективно происходит формирование молекулярного водорода и рекомбинация углерода. Проведенные расчеты также позволяют сделать вывод о сильной зависимости динамики взаимодействия от плотности облака. Холодное плотное облако с температурой 100 K в результате взаимодействия уменьшается в размерах, но “выживает”. Перегретое же облако (с температурой 1000 K) практически полностью прогревается за времена около $2.5 \cdot 10^5$ лет, оставляя после себя лишь филаментные структуры. Химическая эволюция также сильно различается в этих моделях: в первом случае масса углерода быстро растет и стремится к насыщению, что связано с быстрой рекомбинацией ионов в облаке, масса образовавшегося молекулярного водорода в два раза больше, чем во второй модели (см. рисунок 3 a, b).



a b

Рис. 3. Зависимость относительного содержания молекулярного водорода $m(H_2)/m_0(H_2)$ и углерода $m(C)/m_0(C)$ от времени для моделей 1 и 2 (красная и синяя кривые соответственно)

3. Основные выводы

В заключение нашей работы сформулируем основные выводы:

1. Разработан и реализован трехмерный параллельный код AstroChemHydro для численного моделирования химической и динамической эволюции межзвездной среды.
2. Разработана и реализована библиотека MPIParallel3D, позволяющая значительно сократить время для распараллеливания трехмерных вычислительных программ.
3. Результат взаимодействия облака с ударной волной существенным образом зависит от начальной плотности газа в облаке.
4. В процессе взаимодействия облака с ударной волной наблюдается сильная молекуляризация водорода во внешних слоях облака.

Литература

1. Spitzer L. Physics of fully ionized gases. // New York: Interscience, 1962.
2. Марочник Л. С., Сучков А. А. Галактика. // М.: Наука, 1984.
3. Бочкарев Н. Г. Основы физики межзвездной среды. // М: МГУ, 1991.

4. Field G.B. Thermal Instability. // The Astrophysical Journal, 1965, 142, pp. 531–567.

5. McKee C.F., Ostriker J.P. A theory of the interstellar medium – Three components regulated by supernova explosions in an inhomogeneous substrate. // The Astrophysical Journal, 1977, 218, pp. 148 – 169.

6. Зельдович Я.Б., Райзер Ю.П. Физика ударных волн и высокотемпературных гидродинамических явлений. // М., Наука, 1966.

7. Оран Э. , Борис Дж. Численное моделирование реагирующих потоков // М., Мир, 1990.

8. Brown P. N., Byrne G. D., & Hindmarsh A. C. VODE: A Variable-Coefficient ODE Solver // SIAM J. Sci. Stat. Comput., 1989, 10, pp. 1038 –1051.

9. Asplund, M., Grevesse, N., & Sauval, A. J. Cosmic Abundances as Records of Stellar Evolution and Nucleosynthesis. // 2005, in ASP Conf. Ser. ed. T. G. Barnes III & F. N. Bash (San Francisco: ASP), 336, pp. 25 – 35.

10. Galli D. & Palla F., The chemistry of the early Universe // Astronomy and Astrophysics, 1998, 335, pp. 403 – 420.

11. Shapiro P.R. & Kang H. Hydrogen molecules and the radiative cooling of pregalactic shocks. // The Astrophysical Journal, 1987, 318, pp. 32 – 65.

12. Hollenbach D., & McKee C. F. Molecule formation and infrared emission in fast interstellar shocks. I Physical processes. // The Astrophysical Journal Supplement Series, 1979, 41, pp. 555 – 592.

13. Cen R. A hydrodynamic approach to cosmology - Methodology. // The Astrophysical Journal Supplement Series, 1992, 78, pp. 341 – 364.

14. Mac Low M.-M., Shull J.M. Molecular processes and gravitational collapse in intergalactic shocks. // The Astrophysical Journal, 1986, 302, pp. 585 – 589.

15. Hollenbach D., & McKee C. F. Molecule formation and infrared emission in fast interstellar shocks. III - Results for J shocks in molecular clouds. // The Astrophysical Journal, 1989, 342, pp. 306 – 336.

16. Launay, J. M., Le Dourneuf, M., & Zeippen, C. J. The reversible $H + H^- = H_2(v, j) + e^-$ reaction – A consistent description of the associative detachment and dissociative attachment processes using the resonant scattering theory. // Astronomy and Astrophysics, 1991, 252, pp. 842 – 852.

17. Harten A. High Resolution Schemes for Hyperbolic Conservation Laws. // Journal of Computational Physics, 1983, 49, №3, pp. pp. 357 – 593.

18. van Leer B. Towards the ultimate conservative difference scheme. V. A second order sequel to Godunov's methods. // Journal of Computational Physics, 1979, 32, №1, pp. 101 – 136.

19. Toro E.F. Riemann Solvers and Numerical Methods for Fluid Dynamics. A Practical Introduction. // Springer, Berlin, 1997.

20. Антонов А.С., Параллельное программирование с использованием технологии MPI. // – М.: МГУ, 2004, 71 с.

21. R. Andrews G.R. Foundations of Multithreaded, Parallel, and Distributed Programming. // Addison-Wesley, 2000.

22. Nakamura F., McKee Ch.F., Klein R. I., Fisher R.T. On the Hydrodynamic Interaction of Shock Waves with Interstellar Clouds. II. The Effect of Smooth Cloud Boundaries on Cloud Destruction and Cloud Turbulence. // The Astrophysical Journal Supplement Series, 2006, 164, pp. 477 – 505.

23. Klein R.I., McKee Ch.F., Colella Ph. On the hydrodynamic interaction of shock waves with interstellar clouds. 1: Nonradiative shocks in small clouds. // The Astrophysical Journal, 1994, 420, pp. 213 – 236.

24. Pittard J. M., Hartquist T. W., Falle S.A.E.G. The turbulent destruction of clouds - II. Mach number dependence, mass-loss rates and tail formation. // The Monthly Notes of the Royal Astronomical Society, 2010, 405, pp. 821 – 838.

Моделирование переноса электронов в веществе на гибридных вычислительных системах

М.Е.Жуковский, С.В.Подоляко, Р.В.Усков

Институт прикладной математики им. М.В.Келдыша РАН

На основе использования данных для сечений упругих и неупругих процессов взаимодействия электронов с веществом строятся распределения характеристик электронов для моделирования их переноса в веществе. Построенная модель не использует распространенные приближения непрерывного замедления и теории многократного рассеяния для описания переноса электронов. Разработаны алгоритмы статистического моделирования переноса электронов для проведения вычислительных экспериментов на гибридных суперкомпьютерах, в частности, с использованием технологии NVIDIA©CUDA. Проведена серия расчетов для исследования взаимодействия электронов с веществом мишени рентгеновской трубки на кластере МВС-Экспресс (**http://www.kiam.ru/MVS/resourses/mvse.html**) и на новом суперкомпьютере К-100.

1. Введение

К настоящему времени разработано много моделей и численных алгоритмов для математического моделирования переноса электронов в веществе. Созданы мощные программные комплексы для исследования взаимодействия ионизирующего излучения с веществом. Среди них: MCNP (Monte Carlo N-Particle Transport Code System for Multiparticle and High Energy Applications, RSICC Computer Code, LANL, Los Alamos), GEANT (http://geant4.web.cern.ch/geant4/), PENELOPE (PENetration and Energy LOss of Positrons and Electrons, a code system for Monte Carlo simulation of electron and photon transport, NEA-OECD, Paris). Эти программные продукты нашли широкое применение в различных отраслях науки и техники. К числу их несомненных достоинств, прежде всего, относится богатое константное обеспечение. Используемые численные алгоритмы надежны и точны, особенно в части моделирования столкновений квантов с большой передачей импульса. В то же время для моделирования ряда столкновительных эффектов указанные программные комплексы используют приближенные подходы. В основном это относится к процессам с малой передачей импульса, которые наиболее вероятны при столкновениях электронов с атомами и молекулами вещества. В основе этих подходов лежат такие развитые приближения, как теория многократного рассеяния, теория Ландау с различными поправками и уточнениями, приближение непрерывного замедления и т.д. Эти приближения имеют достаточную точность, если рассматривается область с однородными на масштабе тормозного пути электрона рассеивающими свойствами.

Моделирование процессов взаимодействия излучения с веществом объектов, имеющих сложную разномасштабную структуру, включающую микроструктурные элементы, требует рассмотрения более точных моделей без использования указанных приближенных теорий. Для этого необходима современная гибридная вычислительная техника и применение соответствующих ее архитектуре технологий распараллеливания. Это, главным образом, обусловлено тем, что моделирование десятков тысяч электронных столкновений с малой, в среднем, передачей энергии требует огромного объема вычислений. Поэтому применение существующих программных комплексов для объектов, включающих изделия микроэлектроники, затруднительно.

В последнее время появились разработки, посвященные моделированию переноса излучения на графических процессорах с использованием технологии NVIDIA© CUDA. Так, например, для решения задач рентгеновской медицины создан проект MC-GPU [1] в основе которого лежит методика моделирования переноса рентгеновского излучения методом Монте-Карло.

Авторами настоящей статьи разработаны параллельные алгоритмы моделирования переноса гамма-излучения на гибридных кластерах с применением технологии NVIDIA© CUDA [2].

Ниже, в настоящей работе описаны подходы к моделированию переноса электронов методом Монте-Карло на гибридных вычислительных системах.

2. Физическая модель взаимодействия электронов с веществом

Сложный процесс прохождения частицы через вещество можно представить в виде последовательности элементарных процессов взаимодействия этой частицы с веществом. К этим процессам относятся рассеяние, торможение или гибель частицы в результате поглощения или вылета из рассматриваемой системы. Упругое рассеяние электронов описывается обычно в рамках приближенной теории многократного рассеяния Гоудсмита-Саундерсона, а неупругие взаимодействия электронов с веществом описываются в большинстве известных работ с использованием различных модификаций приближения непрерывного замедления.

Ниже описывается модель процессов взаимодействия электронов, построенная на основе использования табулированных данных для сечений указанных процессов. Основным источником этих данных является база данных Национального центра ядерных данных США [3], а именно Evaluated Nuclear Data File [4].

Основная цель построения указанной модели – получение вероятностных распределений характеристик электронов, изменяющихся в процессе их взаимодействия с веществом. К таким характеристикам относятся углы отклонения направления движения электронов в результате упругого рассеяния и потери энергии в результате неупругих взаимодействий.

Фундаментальную роль при моделировании процессов переноса частиц играют распределения характеристик частиц, изменяющихся в моделируемых процессах. Эти распределения вычисляются с помощью сечений (дифференциальных сечений) рассматриваемых процессов.

Пусть x - значения величины ξ, характеризующей состояние частицы. Если известна (нормированная на 1) плотность распределения $f(x)$ этой величины в данном физическом процессе, то распределение этой величины $F(x)$ определяется интегралом $F(x)=\int_{-\infty}^{x} f(t)dt$. Величина $F(x)$ равна вероятности того, что значение ξ будет меньше x: $F(x)=P(\xi<x)$.

Для моделирования (разыгрывания) случайной величины ξ наиболее общим методом является метод «обратной функции» [5]. Этот метод основан на теореме о том, что, если $\gamma\in(0,1)$ - равномерно распределенная случайная величина, а значения x случайной величины ξ удовлетворяют уравнению $\int_{-\infty}^{x} f(t)dt=\gamma,\quad \gamma\in(0,1)$, то ξ имеет плотность распределения $f(x)$. Следовательно, x вычисляется как значения функции, обратной к $F(x)$: $x=F^{-1}(\gamma)$.

Таким образом, для получения (путем розыгрыша случайной величины) характеристик частиц в процессе из переноса основным является уравнение:

$$\int_{-\infty}^{x} f(t)dt=\gamma,\quad \gamma\in(0,1) \qquad (1)$$

Функция распределения косинуса полярного угла *упругого рассеяния* $y=\cos\theta$ строится с использованием данных для плотности распределения этого угла:

$$f_{el}(y \mid E) = \frac{1}{\sigma_{el}^{o}} \frac{d\sigma_{el}}{dy},\ \sigma_{el}^{o} = \int_{-1}^{1} \frac{d\sigma_{el}}{dy} dy .$$

Для качественного анализа искомого распределения воспользуемся формулой Резерфорда для дифференциального сечения упругого рассеяния: $\frac{d\sigma}{dy} \sim \frac{1}{E^2 (1-y)^2}$. Проводя интегрирование $\int_{-1}^{y} \frac{d\sigma}{dt} dt$ и решая уравнение (1) получим, что $y \sim E^2 \gamma - 1$ при $\gamma \to 0$.

С учетом вышесказанного можно сделать вывод о том, что наиболее удобной с точки зрения моделирования процесса упругого рассеяния является величина $F_{el}(y) \equiv lg(1+y) = F(lg\,\gamma)$, поскольку для ее хранения можно использовать равномерные по $lg\,\gamma$ сетки, независящие от энергии электрона. С использованием табулированных значений величины $\frac{d\sigma_{el}}{dy}$ были получены соответствующие распределения $F_{el}(lg\,\gamma)$.

Дифференциальное по энергии фотона сечение тормозного излучения обратно пропорционально энергии фотона [6,7]: $\frac{d\sigma_{br}}{dE_{ph}} \sim \frac{1}{E_{ph}}$. Проводя соответствующие преобразования, получим: $lg(E_{ph}) \sim \gamma$. То есть логарифм от потерь энергии электроном на тормозное излучение (E_{ph}) ведет себя как функция γ по закону, близкому к линейному. Поэтому моделирование процесса тормозного излучения оказывается удобным проводить с помощью функции $F_{br} = lg(E_{ph})$.

Моделирование процесса *ионизации атомов* гораздо сложнее упругого рассеяния и тормозного излучения, поскольку ионизация атома может происходить путем «отрыва» электрона с разных оболочек атома, на каждой из которых электрон имеет свою энергию связи. Кроме того, дифференциальное по потерянной энергии сечение ионизации также зависит от того, с какой оболочки выбивается атомный электрон.

Когда электрон с энергией E_o налетает на атом, имеющий N_{sh} оболочек, он может потерять энергию на ионизацию («отрыв» атомного электрона с оболочки) некоторой оболочки и на передачу части своей кинетической энергии соответствующему атомному электрону. Плотность распределения потери энергии $f_n(\varepsilon \mid E_o)$ налетающим электроном с энергией E_o на «ионизацию» равна

$$f_n(\varepsilon \mid E_o) = \frac{\sigma_n(E_o)}{\sum_{n=1}^{N_{sh}} \sigma_n(E_o)} \delta(\varepsilon - E_n), \qquad (2)$$

где $\sigma_n(E_o)$ - сечение ионизации на n-ой оболочке; E_n - энергия связи; ε - потеря энергии.

Плотность распределения переданной вторичному электрону энергии

$$f_n^{sec}(\varepsilon \mid E_o) = \frac{\sigma_n}{\sum_{n=1}^{N_{sh}} \sigma_n} \frac{d\sigma_n}{d\varepsilon} \eta(E_o - E_n). \qquad (3)$$

После ионизации образуются два электрона. Вторичным называют электрон с меньшей энергией. Поэтому переданная вторичному электрону энергия может изменяться от **O** до $\left(E_o - E_n\right)/2$. Суммируя (2) и (3), и интегрируя по переданной энергии, получим:

$$F_{ion}\left(\varepsilon \mid E_o\right) = \sum_{n=1}^{M} \eta\left(E_o - E_n\right) \frac{\sigma_n\left(E_o\right)}{S(M)} \left\{ \eta\left(\frac{E_o - E_n}{2} - \varepsilon \right) \int_o^{\varepsilon} \frac{d\sigma_n}{dt} dt + 1 \right\}, \qquad (4)$$

Первое слагаемое в фигурных скобках – распределение энергии, переданной вторичному электрону, второе – распределение потерь энергии на «отрыв» этого электрона от атома. В формуле (4) $S = \sum_{n=1}^{M} \sigma_n\left(E_o\right)$, а $M = \sum_{n=1}^{N_{sh}} \eta\left(E_o - E_n\right)$, η - единичная функция.

С помощью (4) построены распределения ионизационных потерь для моделирования процесса переноса электронов.

Отметим, что дифференциальное сечение ионизации обратно пропорционально квадрату переданной вторичному электрону энергии [8]. Проведя соответствующие вычисления, получим, что уравнение (1) принимает вид: $\frac{1}{\varepsilon}$ □ γ. Отсюда видно, что для моделирования процесса ионизации удобно использовать переменные $lg\varepsilon, lg\gamma$.

Описанная модель может быть использована при построении алгоритмов решения различных задач переноса электронов, ориентированных на современную и перспективную (в том числе гибридную) вычислительную технику. Важной особенностью разработанной модели является то, что при уточнении и дополнении данных по сечениям рассматриваемых процессов построенные распределения могут быть без труда перестроены и уточнены.

3. Алгоритмы моделирования переноса электронов

Моделированию процессов трансформации проникающего излучения в материалах объектов посвящено большое число публикаций. В одних работах используются и развиваются сеточные методы решения уравнения переноса излучения [9, 10]. В других разрабатываются вычислительные алгоритмы, основанные на статистическом моделировании методом Монте-Карло процессов переноса и взаимодействия излучения с веществом [11, 12]. Преимущество метода Монте-Карло перед альтернативными методами, основанными на численном решении кинетического уравнения, определяется удобством и приспособленностью этого метода к решению сложных граничных задач в многокомпонентных средах.

Эффективность применения метода Монте-Карло определяется в настоящее время, во-первых, развитием способов уменьшения статистической погрешности результатов расчетов и, во-вторых, прогрессом в области создания быстродействующих многопроцессорных вычислительных систем, в том числе гибридных. В работе [13] построена эффективная модель переноса электронов, относящаяся к классу моделей «вложенных траекторий». Эта модель («модель утолщенных траекторий») не использует приближение теории многократного рассеяния, а энергетические потери в неупругих соударениях учитываются в приближении непрерывного замедления, причем флуктуации этих потерь рассчитываются по теории Ландау.

Как указывалось выше, в настоящей работе описывается подход к построению алгоритмов моделирования взаимодействия электронов с веществом без использования указанных приближений. Ниже в этом разделе рассмотрены основные особенности разработки таких алгоритмов для проведения моделирования на гибридных вычислительных системах.

Статистическая оценка математического ожидания искомого функционала, который соответствует измеряемой в эксперименте величине, с помощью моделирования случайных траекторий электронов методом Монте-Карло подразумевает последовательное независимое моделирование заданного числа таких траекторий и определение аддитивного вклада каждой

траектории в общий результат, причем вычислительная схема моделирования на каждом «звене» траектории одинакова.

Такие алгоритмы, имеющие большое число независимых вычислительных ветвей (в данном случае электронных траекторий), легко параллелятся и масштабируются (больше траекторий — выше точность результатов), причем на любых архитектурах, в том числе на тех, в которых присутствуют лишь минимальные способы взаимодействия между параллельными ветвями. К таковым, как было показано в [2], относится и технология nVidia© CUDA, реализуемая на аппаратных средствах, изначально предназначенных для совершения большого числа однотипных независимых операций.

3.1 Особенности реализации алгоритмов на гибридных системах

Алгоритмы моделирования траекторий частиц для конкретных задач могут иметь значительные отличия, однако при адаптации и эффективной реализации этих алгоритмов на гибридных вычислительных комплексах можно указать ряд общих особенностей.

Во-первых, при разработке параллельных алгоритмов моделирования электронных траекторий естественным является подход, при котором отдельные вычислительные потоки моделируют отдельные траектории, поскольку вычисления большей частью являются независимыми и не требуют синхронизации. Кроме того, вычислительная схема моделирования каждой траектории одинакова. Такая организация вычислений отлично подходит для параллельных архитектур типа SIMD (Single Instruction Multiple Data), характерных для современных графических ускорителей (GPU).

Во-вторых, одним из важнейших условий построения эффективной реализации алгоритма для проведения вычислений на GPU является правильная организация хранения и работы с данными в памяти видеоадаптера [14, 15].

При моделировании траекторий электронов, в общем случае, работа ведётся с тремя основными типами данных: параметрами моделируемых частиц; табличными значения коэффициентов взаимодействия излучения с веществом и вероятностных распределений характеристик процессов; параметрами объектов.

Параметрами моделируемых частиц являются координаты, направление движения, энергия, статистический вес и т.д. С точки зрения однопроцессорной реализации расчетного алгоритма естественным является расположение этих параметров в памяти друг за другом для каждой частицы (рис. 1).

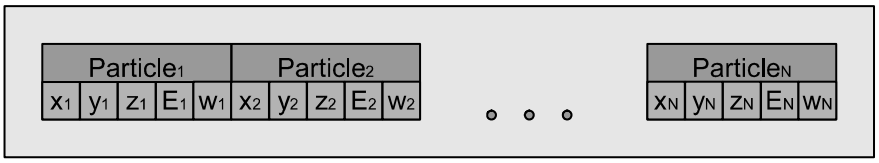


Рис. 1 - Оптимальная для выполнения на CPU структура хранения параметров электронов.

Как показано в [16], такой подход не является эффективным при реализации для GPU, так как не обеспечивает связанности запросов к памяти. Связанностью запросов к памяти называется объединение «логических» запросов к ячейкам памяти от различных вычислительных потоков в один «физический» запрос. Такое объединение значительно повышает производительность памяти и является практически необходимым условием работы с достаточно медленной глобальной памятью видеоадаптера.

Более эффективным является подход, когда одни и те же параметры частиц хранятся в памяти друг за другом для всех моделируемых частиц (рис. 2). Такой подход хорошо подходит для вычислений на GPU, однако в случае гибридного распараллеливания подобное хранение параметров частиц может замедлять вычисления на CPU. Размер фрагмента памяти, содержащего совокупность параметров частицы, которая обрабатывается центральным процессором, может превышать размер кэша CPU. Это приводит к постоянному обращению к оперативной памяти, что снижает эффективность расчетов.

Компромиссным является способ, при котором частицы группируются в блоки, число которых выбирается исходя из условия размещения всех данных блока внутри кэша CPU, а

внутренняя структура строится с учетом требования связности запросов при обращении к памяти GPU.

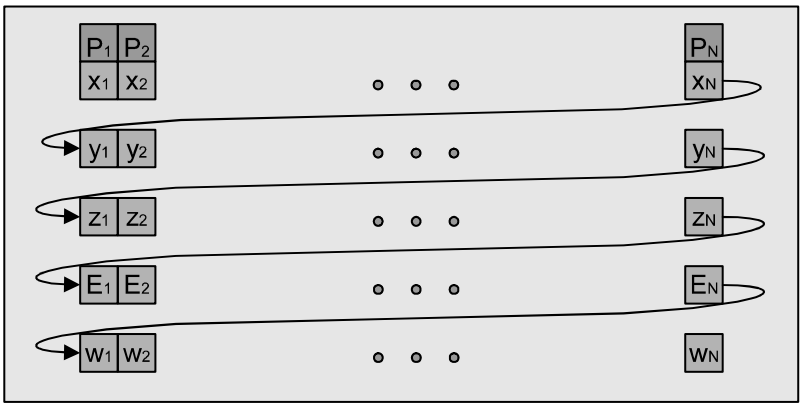


Рис. 2 - Оптимальная для выполнения на GPU структура хранения параметров электронов

Такой способ позволяет обеспечить как близость данных для частиц внутри блока для выполнения требования связанности запросов на GPU, так и попадание всех характеристик одной частицы в кэш центрального процессора для уменьшения количества обращений к оперативной памяти (рис. 3).

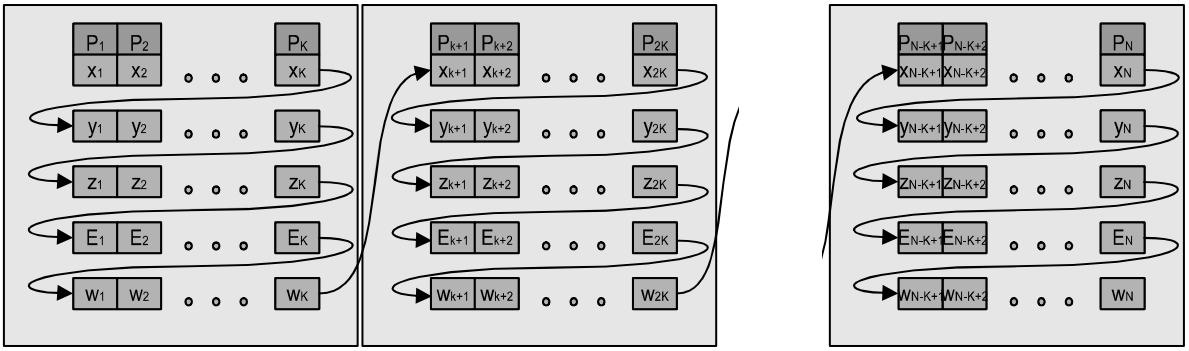


Рис. 3 - Компромиссная структура хранения характеристик электронов

Далее. В отличие от параметров частиц, распределения характеристик процессов взаимодействия электронов с веществом, являются данными со случайным доступом. Поэтому, в случае использования их на GPU, следует по возможности помещать их в быструю разделяемую память, не столь чувствительную к отсутствию возможности связывания [5]. Зачастую, однако, объем памяти необходимый для хранения этих данных значительно превышает доступный объем быстрой памяти, либо накладные расходы на загрузку данных в быструю память превышают выигрыш от её использования. В таком случае использование глобальной памяти является предпочтительным.

Для современных архитектур графических процессоров требования, выполнение которых необходимо для обеспечения «связанности» запросов, стали менее жесткими, а также появилась возможность частичного связывания (то есть объединения логических запросов от части потоков в один физический). Эти возможности позволяют увеличить производительность памяти при работе с данными со случайным доступом, путём обеспечения частичного связывания.

В-третьих, сложность реализации вычислительных алгоритмов на гибридных вычислительных комплексах связана с тем, что в силу особенностей архитектуры графического адаптера, условные переходы являются самым тяжелым видом операций для GPU. Более того, графический процессор в принципе не выполняет условные переходы. Всегда выполняются обе ветви алгоритма, результаты одной из которых затем аннулируются. Это приводит к необходимости построения максимально линейных алгоритмов или «выпрямления» уже существующих.

Стандартный метод моделирования переноса частиц подразумевает проверку попадания электрона в детектор для регистрации частицы и вычисления ее вклада в соответствующий функционал. Такой подход может оказаться неэффективным, когда на каждом звене электронной траектории нужно проверять факт попадания электрона в детектор, что может случиться не более одного раза за всю траекторию.

Гораздо более эффективным являются алгоритмы, относящиеся к классу «весовых» модификаций метода Монте-Карло [17], когда на каждом звене траектории считается, что «часть» электрона достигнет детектора (одного или нескольких), а статистический вес этой «части» приравнивается вероятности электрону попасть в данный детектор. Такой способ позволяет значительно снизить дисперсию результатов, поскольку приводит к эффективному увеличению «источников» электронов, а также уменьшает количество условных переходов в алгоритме.

3.2 Пример расчета с использованием гибридных кластеров

Рассмотрим задачу о спектральном распределении электронов в мишени, облучаемой потоком электронов, в зависимости от глубины проникновения электронов. Такие задачи актуальны, например, при исследовании свойств мишеней рентгеновских аппаратов.

В качестве детекторов будем рассматривать набор параллельных плоскостей, первая из которых является граничной поверхностью мишени. Искомой величиной для каждой такой плоскости будет энергетическое распределение электронов, пересекающих ее в «прямом» и в «обратном» направлении.

Ниже схематично рассмотрен способ построения алгоритма для моделирования переноса электронов в веществе мишени с учетом описанных выше особенностей гибридного распараллеливания с использованием технологии nVidia© CUDA.

Состояние частицы в каждой точке фазового пространства описывается набором $\{E, \boldsymbol{\Omega}, \boldsymbol{r}, w\}$, в котором указаны энергия, направление движения, координаты и статистический вес соответственно. Для всех K электронов из блока (рис.3) эти данные помещаются в глобальную память видеоадаптера непосредственно перед запуском ядра в соответствии с условием «связывания».

Вероятностные распределения величин, характеризующих процессы взаимодействия электронов с веществом (см. раздел 2), полученные путем обработки дифференциальных сечений этих процессов, табулированы и задаются в виде массивов. Объем этих массивов минимизируется путем введения неравномерных сеток по соответствующим переменным и размещается в «быстрой» памяти графического ускорителя.

Расчетный алгоритм состоит из следующих основных частей:

- Розыгрыш длины пути до очередной точки взаимодействия согласно плотности распределения $f_s = \frac{1}{\mu}\exp\left(-\int_0^s \mu(x)\,dx\right)$, и вычисление координат искомой точки;
- Розыгрыш типа взаимодействия (упругого рассеяния, возбуждения, ионизации, тормозного излучения);
- Розыгрыш характеристик электрона после соответствующего процесса (угла изменения направления движения, потерь энергии в неупругих столкновениях) согласно распределениям рассмотренной выше модели переноса электронов;
- Вычисление вклада очередного звена электронной траектории в искомую величину;
- Продолжение траектории электрона до тех пор, пока его энергия не станет ниже заранее выбранного порога.

Вклад частицы в конечный результат определяется в результате случайного события: попадание частицы в детектор. Для моделирования процесса регистрации частиц (расчета показаний детектора) был построен алгоритм, в котором вклад электрона в искомую величину рассчитывается аналитически на каждом звене траектории, причем вес регистрируемого электрона умножается на вероятность попасть в данный детектор. Тем самым повышается информационная ценность каждого звена траектории частицы, что снижает дисперсию результатов и значительно уменьшает количество условных переходов в алгоритме.

Расчетная величина $F_e(E)$ определяется следующим образом.

Пусть $\{E_k\}_{k=0}^{K} = \{E_0 = E_{\min}, E_1, ..., E_K = E_{\max}\}$ - заданная сетка по энергии, а $\{P_n\}_{n=0}^{N}$ - набор регистрирующих плоскостей. Вклад текущего электрона в *k*-тую энергетическую ячейку на *n*-ой плоскости вычисляется по формуле:

$$F_n^+(E_k) = \omega_n \eta(E - E_{k-1}) \cdot \eta(E_k - E); k = 1, ..., N .$$

В этой формуле ω_n равна вероятности достичь электрону *n*-ой плоскости без взаимодействия: $\omega_n = \exp(-\mu(E) s_n)$, где μ, s_n - полное макроскопическое сечение взаимодействия электрона с веществом мишени, и расстояние от очередной точки взаимодействия до *n*-ой плоскости соответственно. F_n^+ - вклад от электрона, движущегося в «прямом» направлении (вглубь мишени). Аналогично определяется величина F_n^- вклада от электронов, движущихся в обратном направлении. Искомое спектральное распределение $f_n^{\pm}$ определяется с помощью нормировки: $f_n^{\pm} = F_n^{\pm} / trapz(\{F_{n,k}^{\pm}\}, \{E_k\})$. Функция $trapz(\{F_{n,k}^{\pm}\}, \{E_k\})$ обозначает интегрирование по формуле трапеций на сетке $\{E_k\}_{k=0}^{K}$.

Описанный расчетный алгоритм реализован в виде параллельного кода для проведения моделирования первоначально на кластере МВС-Экспресс (http://www.kiam.ru/MVS/resourses/mvse.html), а затем адаптирован для проведения расчетов на разработанном совместно ИПМ им. М.В.Келдыша и ФГУП «Квант» гибридном суперкомпьютере К-100. Схема проведения вычислений представлена на рис. 4.

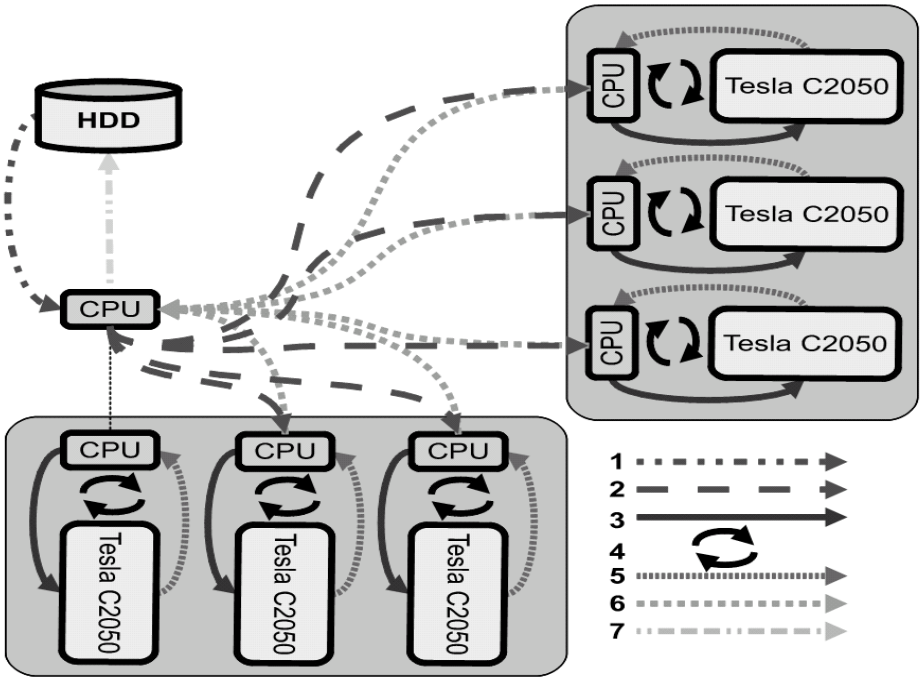


Рис. 4 – Схема проведения вычислений на К-100. 1 - Загрузка и обработка параметров расчета главным CPU; 2 - Рассылка начальных данных центральным процессорам; 3 - «Подъем» данных на GPU; 4 - Итерационный запуск «ядер» для моделирования электронных траекторий; 5 - «Спуск» результатов на CPU; 6 - Сбор результатов главным CPU; 7 - Сохранение результатов.

Программный комплекс реализован на языке C/C++ с использованием библиотек shmem, MPI, CUDA. Комплекс состоит из 3 компонентов:

- Первый компонент осуществляет загрузку исходных параметров и их обработку, а также сохраняет результаты;
- Второй компонент отвечает за организацию межпроцессорных взаимодействий (MPI, shmem);
- Третий компонент занимается непосредственной работой с данными GPU и выдачей вычислительных заданий (запуском «ядер»).

С помощью разработанного программного кода проведены исследования спектральных распределений электронов внутри мишени рентгеновского аппарата по заказу Федерального института исследования и контроля материалов (Берлин, Германия). На рис.5 изображены графики спектра электронов, пересекающих детекторные плоскости в прямом направлении, а на рис.6 – в обратном для различных глубин в мишени из алюминия. На этих рисунках видна деградация спектра электронов с увеличением расстояния от граничной поверхности мишени, на которую падает поток электронов с энергией 100 кэВ. Результаты проведенных расчетов позволяют оценить градиент деградации электронного спектра, что является полезной информацией при оценке эффективности использования исследуемой мишени рентгеновской трубки.

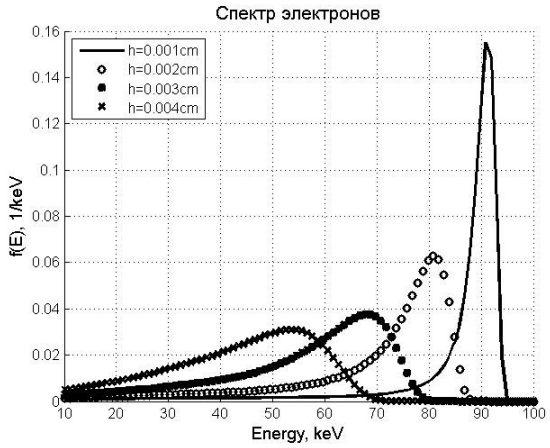


Рис. 5 – Спектр электронов, движущихся вглубь мишени

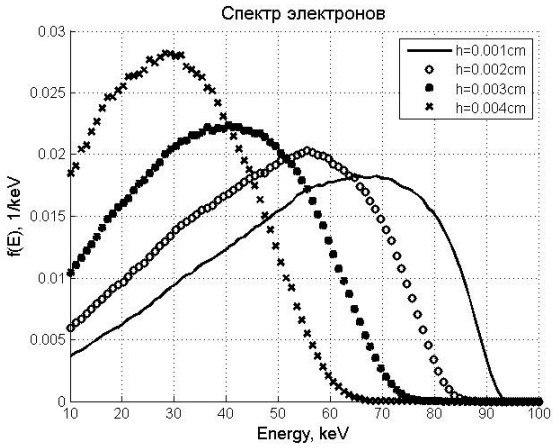


Рис. 6 - Спектр электронов, движущихся к граничной поверхности мишени

Полученные результаты сравнивались с аналогичными результатами, рассчитанными с применением пакета MCNP [13]. Сравнение показало совпадение результатов в пределах статистической погрешности. При этом эффективность моделирования на К-100 почти на два порядка превысила эффективность расчетов с использованием пакета MCNP на обычном многопроцессорном кластере.

4. Заключение

Анализ результатов моделирования переноса электронов на гибридном суперкомпьютере К-100 и прототипе МВС-Экспресс показал высокую эффективность разработанных методов и алгоритмов моделирования процессов взаимодействия излучения с веществом на компьютерах с гибридной архитектурой. Ускорение вычислений по сравнению с параллельными системами с обычной линейной архитектурой может достигать двух порядков. Это дает возможность сделать вывод о перспективности суперкомпьютеров с гибридной архитектурой для проведения математического моделирования процессов переноса электронов в веществе.

Литература

1. Monte Carlo simulation of x-ray transport in a GPU with CUDA. http://code.google.com/p/mcgpu/
2. Жуковский М.Е., Усков Р.В. О применении графических процессоров видеоускорителей в прикладных задачах (Часть II. Моделирование поглощения гамма-излучения). // Препринт ИПМ им. М.В.Келдыша РАН № 20, 2010 г.
3. National Nuclear Data Center, http://www.nndc.bnl.gov/
4. Evaluated Nuclear Data File (ENDF, http://www.nndc.bnl.gov/exfor/endf00.jsp)
5. Соболь И.М. Численные методы Монте-Карло. М., «Наука», 1973.
6. Stephen M. Seltzer, “Cross Sections for Bremsstrahlung Production and Electron Impact Ionization,” in *Monte Carlo Transport of Electrons and Photons*, edited by Theodore M. Jenkins, Walter R. Nelson, and Alessandro Rindi, (Plenum Press, New York, 1988) 81.
7. Bethe H. A. and Heitler W. “On Stopping of Fast Particles and on the Creation of Positive Electrons,” Proc. Roy. Soc. (London) **A146** (1934) 83.
8. Ландау Л.Д. О потерях энергии быстрыми частицами на ионизацию. Собр. Трудов. Т. 1. М., «Наука», 1969.
9. Шагалиев Р.М. и др. Математическое моделирование и методики решения многомерных задач переноса частиц и энергии, реализованные в комплексе САТУРН-3. // Вопросы атомной науки и техники, серия: Матем. моделирование физических процессов. 1999, вып. 4, с. 20-26.
10. Сушкевич Т.А. Математическое моделирование переноса излучения. М., Бином, 2005.
11. Аккерман А.Ф.. Моделирование траекторий заряженных частиц в веществе. М., Энергоатомиздат, 1991, 200 с.
12. Briesmeister J.F. (ed.). MCNP - A General Monte Carlo N-Particle Transport Code. LANL Report LA-13709-M, Los Alamos, 2000.
13. Жуковский М.Е., Скачков М.В.. О статистических методах моделирования переноса электронов в веществе. Вестник МГТУ им. Н.Э.Баумана, 1(32) 2009, с. 31-46.
14. NVIDIA CUDA Programming Guide, Version 2.3.1. 2009.
15. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010.
16. Жуковский М.Е., Усков Р.В. О применении графических процессоров видеоускорителей в прикладных задачах. Препринт ИПМ им. М.В.Келдыша РАН, №2, 2010.
17. Г.А.Михайлов. Весовые методы Монте-Карло. Новосибирск, изд-во Сибирского отд. РАН, 2000.

Параллельные методы декомпозиции в пространствах следов

В.П. Ильин, Д.В. Кныш

Институт вычислительной математики и математической геофизики СО РАН

1. Введение

Альтернирующий метод Шварца, возникший изначально как средство теоретического исследования сложных краевых задач математической физики, впоследствии породил большое семейство алгоритмов декомпозиции областей, ставшим основным орудием распараллеливания при решении многомерных дифференциальных уравнений. Данным проблемам посвящено огромное количество работ российских и зарубежных авторов, а также имеется специальная регулярная международная конференция, активно работающая в течение многих лет, см. [1] – [15] и цитируемую там литературу.

Общая идея методов декомпозиции заключается в следующем. Пусть в заданной расчетной области Ω с границей Γ требуется решить линейную дифференциальную краевую задачу

$$Lu = f(\vec{r}), \quad \vec{r} \in \Omega; \quad lu|_\Gamma = g, \tag{1}$$

где u, f, g суть искомая и заданные функции, а L и l – операторы, определяющие исходное уравнение и граничные условия.

Разобъем Ω на некоторое количество P пересекающихся или непересекающихся подобластей: $\Omega = \bigcup\limits_{q=1}^{P} \Omega_q$, – границу каждой из которых обозначим через Γ_q, а замыкание области – через $\bar{\Omega}_q = \Omega_q \bigcup \Gamma_q$. Если $\Gamma_{q,q'}$ при $q' \neq q$ означает часть Γ_q, принадлежащую $\bar{\Omega}_{q'}$, то можно записать $\Gamma_q = (\Gamma \bigcap \bar{\Omega}_q) \bigcup\limits_{q' \in \omega_q} \Gamma_{q,q'}$, где выражение в скобках представляет внешнюю границу подобласти Ω_q (если последняя является околограничной), а остальные $\Gamma_{q,q'}$ составляют внутреннюю границу, причем ω_q есть совокупность номеров подобластей $\Omega_{q'}$, соседних к Ω_q. В случае $\Gamma_{q',q} = \Gamma_{q,q'}$ области Ω_q и $\Omega_{q'}$ касаются (сопрягаются) без налегания, т.е. их пересечение $\Delta_{q,q'} = \Omega_q \bigcap \Omega_{q'}$ есть пустое множество. Исходной задаче (1) сопоставим совокупности вспомогательных краевых задач в подобластях:

$$Lu_q(\vec{r}\,) = f_q, \quad \vec{r} \in \Omega_q, \quad l_{q,q'}u_q|_{\Gamma_{q,q'}} = g_{q,q'} \equiv l_{q,q'}(u_{q'})\big|_{\Gamma_{q',q}}, \quad q' \in \omega_q, \quad q = 1, ..., P, \tag{2}$$

причем на внешней части границы Γ_q, если она существует, ставится исходное краевое условие из (1), а $f_q(\vec{r})$ при $r \in \Omega_q$. Типичное условие сопряжения на внутренней границе записывается в виде краевого условия 3-го рода, или Робена:

$$\alpha_q u_q + \beta_q \frac{\partial u_q}{\partial n_q}\Big|_{\Gamma_{q,q'}} = g_{q,q'} \equiv \alpha_q u_{q'} + \beta_q \frac{\partial u_{q'}}{\partial n_q}\Big|_{\Gamma_{q',q}}, \tag{3}$$

где $\alpha_q \cdot \beta_q \geq 0$, $\dfrac{\partial u_q}{\partial n_q}\Big|_{\Gamma_{q,q'}}$ есть одностороннее значение (внутреннее по отношению к Ω_q)внешней нормальной производной к границе Γ_q, а случаи $\beta_q = 0$ или $\alpha_q = 0$ соответственно означают условия Дирихле или Ньютона. Сами коэффициенты α_q, β_q являются функциями координат $\vec{r}$ и обычно выбираются кусочно-постоянными, одинаковыми для фиксированной пары q, q'.

Предполагается, что решения задач (1) и (2) существуют, являются единственными и совпадают между собой. Решение исходной задачи (1) (сложной, или "большой") ищется с помощью итерационного решения последовательности более простых вспомогательных задач (2):

$$Lu_q^n = f_q, \quad l_{q,q'}u_q^n|_{\Gamma_{q,q'}} = l_{q,q'}u_{q'}^{n-1}|_{\Gamma_{q',q}}, \quad q = 1, ..., P, \tag{4}$$

причем условия сопряжения имеют записанный вид для всех $q' = \omega_q$. Если $u_q^n \to u_q$ при $n \to \infty$, что обеспечивается выбором соответствующих $\Gamma_{q,q'}$, α_q и β_q, то мы получаем параллельный алгоритм, поскольку каждая из задач (4) для фиксированных n, q может решаться одновременно при наличии P процессоров. Данный итерационный процесс представляет собой метод одновременных смещений, или Якоби. Отметим, что если в правой части условия сопряжения (4) брать $u_{q'}^n$ вместо $u_{q'}^{n-1}$ для $q' < q$, для уже рассчитанных на данной итерации подобластей $\Omega_{q'}$, то мы приходим к методу последовательных смещений Зейделя. Последний может иметь более быструю сходимость, но для него неприменимо естественное распараллеливание.

Численные алгоритмы решения задач (1)–(4) основываются на дискретизации, т.е. построении сетки и аппроксимации методами конечных разностей, конечных объемов или конечных элементов (МКР, МКО или МКЭ, см. [16], [17]), в результате чего мы приходим к системам линейных алгебраических уравнений (СЛАУ) и алгебраическим методам декомпозиции, представляющим собой специальные блочные итерационные алгоритмы для решения больших систем с разреженными матрицами.

При этом сеточные СЛАУ в подобластях Ω_q решаются, как правило, также итерационными методами, и мы приходим к проблеме оптимизации трудоемких двойных итерационных процессов. При этом необходимо решать два основных вопроса: ускорение сходимости внешних и внутренних итераций, а также минимизация коммуникационных потерь при реализации распараллеливания на многопроцессорных вычислительных системах (МВС).

Среди многообразия возникающих здесь подходов можно выделить следующие.

- Топология декомпозиции областей: в трехмерной краевой задаче подобласти могут составлять одномерную, двумерную или трехмерную сеть (1D, 2D или 3D, см. рис. 1). Теоретически предпочтительней 3D-декомпозиция, с точек зрения как сокращения внешних итераций, так и уменьшения отношения *объем пересылок/объем вычислений*, однако реальная картина может сильно зависеть от возможностей эффективного отображения алгоритмов на архитектуру конкретной МВС.

- Наличие и величина пересечений смежных подобластей. Как и для классического альтернирующего метода Шварца, интуитивно ясно, что с увеличением перехлеста будет ускоряться внешний итерационный процесс, но зато дорожает реализация вспомогательных задач в подобластях (в предельном абсурдном случае, когда все подобласти Ω_q совпадают между собой и с Ω, требуется всего одна итерация, но вспомогательная задача совпадает с исходной). По-видимому, получение строгих теоретических оценок оптимального размера пересечения $\Delta_{q,q'}$ является в общем случае проблематичным, и для выбора практических рекомендаций для конкретных ситуаций свое слово должен сказать эксперимент.

- Тип граничных условий на внутренних границах $\Gamma_{q,q'}$, через которое идет обмен информацией между подобластями, т.е. фактически значения коэффициентов α_q, β_q в (3). Вполне возможно, что их оптимальный выбор связан с предыдущей задачей о перехлесте $\Delta_{q,q'}$.

- Очередность пересчета подобластей: последовательная или одновременная. Получаемые блочные алгоритмы оказываются типа Зейделя или Якоби и называются также

мультипликативными или аддитивными соответственно. Очевидные здесь альтернативные предпочтения: первый – с точки зрения ускорения итераций, а второй – из эффективности распараллеливания,– могут быть дополнены какими-либо компромиссными вариантами типа шахматной упорядоченностью, или черно-белой раскраской подобластей.

- Организация итерационных процессов. Наиболее эффективные современные подходы основаны на предобусловленных методах в подпространствах Крылова, реализующих вариационные и/или ортогональные принципы, см. [16], [17]. Сама декомпозиция области играет роль предобуславливателя, а оптимальный внешний итерационный процесс – это какой-то из крыловских методов. Важный момент заключается в построении двойного итерационного алгоритма. Естественно, в каждой из подобластей целесообразно применять также крыловские методы, причем на первых внешних итерациях бессмысленно решать вспомогательные задачи в подобластях с высокой точностью. В таком случае приходим к динамическому внешнему итерационному процессу с переменными предобуславливателями.

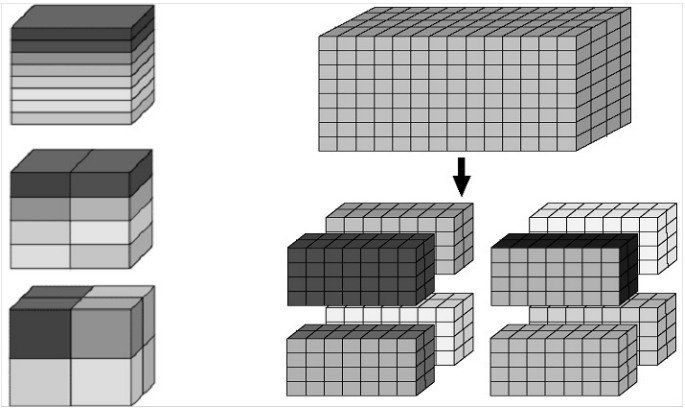


Рис. 1. Схемы 1D, 2D и 3D декомпозиции области

Отдельный важный методологический и практический вопрос — это выбор количества подобластей и соответствующего числа процессоров. Рассмотрим для простоты трехмерную сеточную расчетную область-параллелепипед, с числом узлов $M_xM_yM_z$ параллелепипедоидальной сетки, которая (область) разбита на число $P = P_x \cdot P_y \cdot P_z$ подобластей такой же конфигурации. Один из принципиальных вопросов — масштабируемость параллельного алгоритма, т.е. допускает ли он приемлемую величину ускорения вычислений с ростом числа процессоров? Например, при $M_x, M_y, M_z > 10^3$ и $P_x \to M_x, P_y \to M_y, P_z \to M_z$? Ответ здесь зависит, очевидно, не только от алгоритма, но и от архитектуры МВС. Зачастую в реальных условиях вопрос заключается в тривиальной доступности нужного объема компьютерных ресурсов, но с предстоящим появлением экзафлопных вычислителей [18] проблема массивного параллелизма перейдет и в практическую плоскость.

Данная работа посвящена экспериментальному исследованию многопараметрической одномерной декомпозиции трехмерных краевых задач для уравнения Пуассона. В п. 2 мы приводим алгебраическое представление рассматриваемых методов. Пункт 3 посвящен вопросам оценок эффективности их распараллеливания, а в последнем разделе обсуждаются результаты методических экспериментов на представительной серии тестовых задач с различными шагами сеток и варьируемыми алгоритмическими параметрами.

2. Алгебраическое представление методов декомпозиции

Рассмотрим для простоты первую краевую задачу для уравнения Пуассона в единичном кубе $\Omega = [0 \times 1]^3$:

$$-\Delta u = f, \quad u|_\Gamma = g, \tag{5}$$

которая на кубической сетке аппроксимируется семиточечной СЛАУ порядка M^3:

$$\begin{aligned}(Au^h)_{i,j,k} &= 6u^h_{i,j,k} - u^h_{i-1,j,k} - u^h_{i,j-1,k} - u^h_{i+1,j,k} - u^h_{i,j+1,k} - \\ &- u^h_{i,j,k-1} - u^h_{i,j,k+1} = f^h_{i,j,k}; \quad i,j,k = 1,...,M.\end{aligned} \tag{6}$$

Здесь предполагается, что в левой части уравнений члены со значениями индексов 0 или M, известные из граничного условия, перенесены в правую часть. Очевидно, что такая алгебраическая система имеет единственное решение при любом векторе $f^h = \{f^h_{i,j,k}\}$. Далее верхний индекс "h" будем для краткости опускать.

Разобъем расчетную сеточную область $\Omega^h = \{i,j,k\}$ вдоль оси z на P одинаковых подобластей, как это указано на рис. 2.

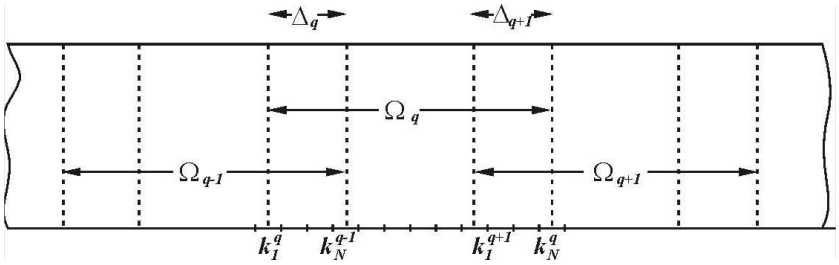


Рис. 2. Схема одномерной декомпозиции с пересечением

Здесь Δ_q означает общую часть подобластей Ω_{q-1} и Ω_q причем предполагается, что границы $\Gamma_{q,q\pm1}$ проходят посередине между сеточными узлами. В каждой из сеточных подобластей Ω^h_q обозначим через k^q_1 и k^q_N номера первой и последней z-х координатных сеточных плоскостей, а через N и m – количество таких плоскостей в Ω_q и Δ_q, так что для всех q выполняются соотношения

$$N = k^q_N - k^q_1 + 1, \quad m = k^q_N - k^{q+1}_1 + 1, \quad M = PN - (P-1)m. \tag{7}$$

Систему (6) с учетом интерфейсных условий сопряжения запишем в следующем блочном виде:

$$-A_{q,q-1}u_{q-1} + A_{q,q}u_q - A_{q,q+1}u_{q+1} = f_q, \quad q = 1,...,P, \quad A_{1,0} = A_{P,P+1} = 0, \tag{8}$$

где u_q и f_q суть векторы, а $A_{q,q}$ и $A_{q,q\pm1} = A^t_{q\pm1,q}$ – квадратные матрицы порядка M^2N, равного числу узлов в Ω_q.

Оставляя в левой части уравнения (8) только "диагональный" член с u_q, блочный метод Якоби (4) можно переписать в форме

$$A_{q,q}u^n_q = \bar{f}^{n-1}_q \equiv f_q + \hat{f}^{n-1}_q + \check{f}^{n-1}_q, \quad \hat{f}^{n-1}_q = A_{q,q-1}u^{n-1}_{q-1}, \quad \check{f}^{n-1}_q = A_{q,q+1}u^{n-1}_{q+1}. \tag{9}$$

Участвующие здесь векторы имеют блочный вид $u_q = (u_{k^q_1},...,u_{k^q_N})^t$, где $u_{k^q_1} = \{u_{i,j,k^q_1}; i,j = 1,...,M\}$, например, есть подвектор размерности M^2, соответствующий сеточной плоскости

с координатой $z = z_{k_1^q}$. Для более детального представления матриц в (9) перепишем его в блочно-компонентной форме

$$(A_{q,q}u_q^n)_k = \begin{cases} (C-\theta I)u^n_{k_1^q} - u^n_{k_1^q+1} = f_{k_1^q} + v^{n-1}_{q-1}, v^{n-1}_{q-1} = u^{n-1}_{k_1^q-1} - \theta u^{n-1}_{k_1^q}, & k = k_1^q, \\ (C-\theta I)u^n_{k_N^q} - u^n_{k_N^q-1} = f_{k_N^q} + w^{n-1}_{q+1}, w^{n-1}_{q+1} = u^{n-1}_{k_N^q+1} - \theta u^{n-1}_{k_N^q}, & k = k_N^q, \\ -u^n_{k-1} + Cu^n_k - u^n_{k+1} = f_k, & k = k_1^q+1, ..., k_N^q - 1, \end{cases} \quad (10)$$

где C – матрица порядка M^2, а выражение Cu_k соответствует первым четырем членам в левой части уравнения (6), т.е. не содержащим индексы $k \pm 1$.

Зависящие от u_k^{n-1} величины в (10) фактически берутся как подвекторы из $\Omega_{q\pm1}$, реализуя пограничный интерфейс между подобластями. При этом нижние индексы надо поменять формально в соответствии с (7): $k_1^q = k_N^{q-1} - m + 1$, $k_N^q = k_1^{q+1} + m - 1$. Параметр θ может быть выражен через коэффициенты α_q и β_q из условий сопряжения в (3), которые для простоты считаем одинаковыми во всех подобластях Ω_q (в частности, $\theta = 0$ и $\theta = 1$ соответствуют условиям Дирихле и Неймана). Векторы $A_{q,q\pm1}u_{q\pm1}^{n-1}$ содержат ненулевые подвекторы, только соответствующие сеточным плоскостям в окрестности внутренних границ $\Gamma_{q,q\pm1}$. После вычисления вектора u_q^n (из решения краевой подзадачи в Ω_q) в дальнейших итерациях фактически будут участвовать только значения правых частей граничных условий из (2), (3), определяемых алгебраически как $A_{q\pm1,q}u_q^n$ и используемых в подобластях $\Omega_{q\pm1}$.

Отсюда нетрудно видеть, что итерационный процесс (9) для векторов $u_q \in \mathcal{R}^{M^2N}$ можно переформулировать в терминах подвекторов меньшей размерности $v_q, w_q \in \mathcal{R}^{M^2}$, соответствующих внутренним граничным интерфейсам на $\Gamma_{q,q\pm1}$. Для этого можно ввести обозначения

$$v_q = C_{q,q-1}u_q, \quad w_q = C_{q,q+1}u_q, \quad A_{q,q\pm1} = Q_{q,q\pm1}C_{q,q\pm1}, \quad (11)$$

где $C_{q,q\pm1} \in \mathcal{R}^{M^2N,M^2}, Q_{q,q\pm1} \in \mathcal{R}^{M^2M^2N}$ суть прямоугольные матрицы "сужения" и "продолжения" соответствующих векторных подпространств.

Умножая обе части равенства (9) один раз на $C_{q,q-1}$, а второй – на $C_{q,q+1}$, получаем соотношения вида

$$\begin{aligned} &v_q^n = \hat{B}_{q,q-1}w_{q-1}^{n-1} + \hat{B}_{q,q+1}v_{q+1}^{n-1} + \hat{g}_q, \quad q = 2, ..., P, \\ &w_q^n = \check{B}_{q,q-1}w_{q-1}^{n-1} + \check{B}_{q,q+1}v_{q+1}^{n-1} + \check{g}_q, \quad q = 1, ..., P-1, \\ &\hat{B}_{1,0} = \hat{B}_{P,P+1} = 0, \end{aligned} \quad (12)$$

в которых введены обозначения

$$\begin{aligned} &\hat{g}_q = C_{q,q-1}A_{q,q}^{-1}f_q, \quad \check{g}_q = C_{q,q+1}A_{q,q}^{-1}f_q, \\ &\hat{B}_{q,q\pm1} = C_{q,q-1}A_{q,q}^{-1}Q_{q,q\pm1}, \quad \check{B}_{q,q\pm1} = C_{q,q+1}A_{q,q}^{-1}Q_{q\pm1}. \end{aligned} \quad (13)$$

Мы не будем вдаваться в детали представления матриц $\hat{B}_{q,q\pm1}, \check{B}_{q,q\pm1}$ и отметим, что они в алгебраическом смысле являются матричными аналогами операторов Пуанкаре–Стеклова, реализуя итерационные интерфейсы только между внутренними граничными условиями на $\Gamma_{q,q\pm1}$. При этом, естественно, на каждом шаге решаются задачи в подобластях Ω_q, но как вспомогательные (реально вычисляются частичные решения, т.е. векторы правых частей имеют "очень мало" ненулевых компонент, а в искомых векторах также представляют интерес только некоторые компоненты). После сходимости с достаточной

точностью итерационных приближений $v_q^n \to v_q, w_q^n \to w_q$ во всех подобластях могут быть восстановлены "полные" векторы u_q.

Обозначая через $s = (w_1, v_2, ..., v_{P-1}, w_{P-1}, v_P)^t$ и $g = (\check{g}_1, \hat{g}_2, \check{g}_2, ..., \hat{g}_{P-1}, \check{g}_{P-1}, \hat{g}_P)^t$ векторы порядка $2M^2(P-1)$, итерационный процесс (12) можно записать в виде

$$s^n = Ts^{n-1} + g, \quad n = 1, 2, ..., \tag{14}$$

где матрица перехода T является блочно-двудиагональной, имея нулевую главную блочную диагональ и составленные из блоков $\hat{B}_{q,q\pm1}, \check{B}_{q,q\pm1}$ соседние к ней диагонали.

Если последовательные приближения сходятся, т.е. $s^n \to s$ при $n \to \infty$, то мы приходим к решению предобусловленной системы уравнений

$$\bar{A}s \equiv (I - T)s = g, \tag{15}$$

и для ускорения итераций (14) естественно применить какой-либо из методов в подпространствах Крылова.

Например, при использовании алгоритмов сопряженных градиентов (СГ) или сопряженных невязок (СН) получаем следующие формулы, см. [17]:

$$\begin{aligned}
&r^0 = g - \bar{A}s^0 = \hat{s}^1 - s^0, \quad \hat{s}^1 = Ts^0 + g, \quad p^0 = r^0, \\
&s^{n+1} = s^n + \alpha_n^{(\nu)} p^n, \quad \alpha_n^{(\nu)} = \rho_n^{(\nu)}/\delta_n^{(\nu)}, \quad \rho_n^{(\nu)} = (\bar{A}^\nu r^n, r^n), \quad \delta_n^{(\nu)} = (\bar{A}p^n, \bar{A}^{(\nu)}p^n), \\
&r^{n+1} = r^n - \alpha_n^{(\nu)} \bar{A}p^n, \quad p^{n+1} = r^{n+1} + \beta_n^{(\nu)} p^n, \quad \beta_n^{(\nu)} = \rho_{n+1}^{(\nu)}/\rho_n^{(\nu)}.
\end{aligned} \tag{16}$$

Здесь $\nu = 0$ и $\nu = 1$ соответствуют алгоритмам СГ и СН, а индексы "ν" у векторов r^n, s^n и p^n для краткости опущены. Отметим, что реализация каждой итерации (16) связана с одним умножением матрицы T на вектор (в том числе при $\nu = 1$, в силу соотношения $\bar{A}p^{n+1} = \bar{A}r^{n+1} + \beta_n^{(\nu)}\bar{A}p^n$). А вследствие определения T, это связано с вычислением частичных решений в подобластях Ω_q (параллельно для всех $q = 1, ..., P$). Однако заметим, что скалярные произведения в (16) определяются для векторов размерности $2(P-1)M^2$, равной общему количеству узлов во всех интерфейсных сечениях Γ_q. Традиционное условие окончания итераций имеет вид $(r^n, r^n)/(g, g) \le \varepsilon_e^2 \ll 1$, при заданном значении ε_e.

Решение вспомогательных задач в подобластях Ω_q целесообразно (при достаточно большом значении M^2N) также проводить с помощью предобусловленного метода в подпространствах Крылова. В качестве предобуславливателя естественным образом выбирается какой-то из алгоритмов неполной факторизации. Если через $r_{in}^{n_q}$ обозначить величину невязки на n_q-м шаге внутреннего итерационного процесса вида (16), то критерием его окончания можно выбрать

$$(r_{in}^{n_q}, r_{in}^{n_q})/(f_q^n, f_q^n) \le (\varepsilon_{in}^{(n)})^2 \tag{17}$$

где величина $\varepsilon_{in}^{(n)}$ может выбираться динамически в зависимости от номера внешней итерации n.

Отметим, что при этом фактически для внешнего итерационного процесса используется динамическое предобуславливание (в [15] используется термин flexible, т.е. гибкое). Строго говоря, в (15) матрица зависит от n, что может интерпретироваться, как приближенное умножение на $\bar{A}$ в формулах (16).

3. Особенности распараллеливания алгоритмов

В соответствии с определением блочной структуры матрицы T и векторов, параллельная реализация крыловского метода (16) (при выбранном начальном векторе u^0) осуществляется следующим образом. Будем предполагать, что вычисления проводятся на МВС с

распределенной памятью, имеющей P процессоров, каждый из которых соответствует "своей" подобласти Ω_q.

а. До проведения итераций вычисляется вектор начальной невязки r^0 (и равный ему начальный направляющий вектор $p^0 = r^0$). Для этого в каждой подобласти Ω_q сначала находятся подвекторы начальных граничных условий, представляющие блочные компоненты s^0:

$$\begin{aligned} v_q^0 &= C_{q,q-1}u_q^0 = \{u^0_{i,j,k_N^{q-1}+1} - \theta u^0_{i,j,k_N^{q-1}}; i,j,= 1,...,M\}, \\ w_q^0 &= C_{q,q+1}u_q^0 = \{u^0_{i,j,k_1^{q+1}-1} - \theta u^0_{i,j,k_1^{q+1}}; i,j,= 1,...,M\}. \end{aligned} \tag{18}$$

Отметим, что v_q^0 представляет правое краевое условие для Ω_{q-1}, а w_q^0 – левое краевое условие для Ω_{q+1}. Затем осуществляются встречные пересылки всех компонент $v^0_{q,i,j}, w^0_{q,i,j}$ данных векторов по следующей схеме:

$$v_q^0, w_q^0 : \Omega_q \to \Omega_{q\pm1}; \quad v_{q+1}^0, w_{q-1}^0 : \Omega_{q\pm1} \to \Omega_q. \tag{19}$$

Далее формируются и решаются вспомогательные СЛАУ вида (9) в Ω_q:

$$A_{q,q}\hat{u}_q^1 = \bar{f}_q = \begin{cases} f_{k_1^q} + w_{q-1}^0, & k = k_1^q, \\ f_{k_N^q} + v_{q+1}^0, & k = k_N^q, \\ f_k, & k = k_1^q + 1, ..., k_N^q - 1. \end{cases} \tag{20}$$

По найденным решениям на интерфейсных границах $\Gamma_{q,q\pm1}$, с помощью формул (18) (при замене u^0 на $\hat{u}^1$) находятся новые краевые услвоия $\hat{v}_q^1, \hat{w}_q^1$, составляющие блочные компоненты вектора $\hat{s}^1$, из которого и находятся $r^0 = p^0 = \hat{s}^1 - s^0$.

б. Реализация итераций для каждого n включает одно умножение матрицы на вектор $\bar{A}p^n$ (или $\bar{A}r^n$), вычисление двух скалярных произведений и нахождение линейных комбинаций векторов. Первая из этих операций, вследствие очевидного соотношения

$$t^n \equiv \bar{A}p^n = p^n - q^n, \quad q^n = Tp^n, \tag{21}$$

выполняется в соответствии с формулами (18)–(20): сначала определяются аналоги краевых условий (в (18) надо только величины v_q^0, w_q^0 и u^0 поменять на $\hat{t}_q^n, \check{t}_q^n$ и p^n), затем в подобластях Ω_q решаются аналогичные (20) подзадачи –

$$A_{q,q}\varphi_q^n = \psi_q^n = \begin{cases} \check{t}_{q-1}^n, & k = k_1^q, \\ \hat{t}_{q+1}^n, & k = k_N^q, \\ 0, & k = k_1^q + 1, ..., k_N^q - 1. \end{cases} \tag{22}$$

где в правых частях отсутствуют не зависящие от n члены f_k, и наконец, по найденным значениям φ_q^n вычисляются блочные компоненты вектора $q^n = (\check{q}_1^n, \hat{q}_2^n, \check{q}_2^n, ..., \hat{q}_{P-1}^n, \check{q}_{P-1}^n, \hat{q}_P^n)$. Формально последний этап тоже заключается в нахождении краевых условий по формулам (18), надо в данном случае только заменить векторы:

$$\begin{aligned} \hat{q}_q^n &= \{\varphi^n_{i,j,k_N^{q-1}+1} - \theta\varphi^n_{i,j,k_N^{q-1}}; \quad i,j = 1,...,M\}, \\ \check{q}_q^n &= \{\varphi^n_{i,j,k_1^{q+1}-1} - \theta\varphi^n_{i,j,k_1^{q+1}}; \quad i,j = 1,...,M\}. \end{aligned} \tag{23}$$

Итоговые блочные компоненты результата операции (21) имеют очевидный вид: $\hat{t}_q^n = \hat{p}_q^n - \hat{q}_q^n, \quad \check{t}_q^n = \check{p}_q^n - \check{q}_q^n$.

Скалярные произведения векторов в (16) вычисляются путем суммирования по всем узлам интерфейсных сечений Γ_q, т.е. например,

$$(t^n, p^n) = \sum_{i,j=1}^{M} \Big(\sum_{q=2}^{p} \hat{t}_{q,i,j}^n \cdot \hat{p}_{q,i,j}^n + \sum_{q=1}^{p-1} \check{t}_{q,i,j}^n \cdot \check{p}_{q,i,j}^n \Big) = \ (\check{t}^n, \check{p}^n)_1 + \sum_{q=2}^{p-1} (t^n, p^n)_q + (\hat{t}^n, \hat{p}^n)_p,$$

$$(t^n, p^n)_q = (\hat{t}^n, \hat{p}^n)_q + (\check{t}^n, \check{p}^n)_q, \quad (\hat{t}^n, \hat{p}^n)_q = \sum_{i,j=1}^{M} \hat{t}_{q,i,j}^n \hat{p}_{q,i,j}^n. \tag{24}$$

в. После сходимости итерационных приближений v_q^n, w_q^n с достаточной точностью мы легко получаем искомое решение из вспомогательных задач вида (20): в правой части надо только поменять w_{q-1}^0 и v_{q+1}^0 на w_{q-1}^{n-1} и v_{q+1}^{n-1}, в результате чего вместо $\hat{u}_q^1$ слева будем иметь u_q^n.

Обозначая традиционно через T_P время реализации данного алгоритма на P процессорах, для коэффициентов ускорения и эффективности использования МВС имеем выражения

$$S_P = T_1/T_P, \quad E_P = S_P/P, \tag{25}$$

где полное время решения задачи складывается из времени выполнения арифметических операций T_P^a и времени обмена данными T_P^c между подобластями Ω_q:

$$T_P = T_P^a + T_P^c \approx \tau_a V_a + N_a(\tau_0 + \tau_c V_c). \tag{26}$$

Здесь τ_a обозначает среднее время исполнения одного арифметического действия, V_a – общий объем выполняемых одним процессором операций (предполагается, что все процессоры работают синхронно), N_a – количество информационных обменов, τ_0 и τ_c – время задержки (настройки) одного обмена и время передачи одного числа, а V_c – объем передаваемого за один раз массива данных, причем имеют место неравенства $\tau_0 \gg \tau_c \gg \tau_a$.

Поскольку времена реализации пп. *а,в* сравнимы со временем выполнения одной итерации в п.б, то для оценки величин S_P и E_P при $n \gg 1$ достаточно проанализировать качество распараллеливания одного итерационного шага.

Решение СЛАУ (22) в каждой q-й подобласти занимает время $T_a^{(1)}$, пропорциональное числу узлов M^2N в Ω_q и количеству внутренних итераций $n_{in}(\varepsilon_{in})$, которое для современных предобусловленных крыловских методов составляет $N^\gamma |ln\varepsilon_{in}|, \gamma \geq 1/2$:

$$T_a^{(1)} = C_1 M^2 N^{\gamma+1} |ln\varepsilon_{in}| \tau_a, \tag{27}$$

где C_1 – количество арифметических действий на выполнение одной итерации, которое для разных предобуславливателей можно оценить величиной 20 или 30.

Время вычисления "краевых условий" $\hat{q}_q$ и $\check{q}_q$ из (23) мало по сравнению с (27): $T_a^{(2)} \leq 6M^2\tau_a$. Выполнение векторных операций на одной внешней итерации (16) в Ω_q (их всего пять) занимает время

$$T_a^{(3)} \approx 20M^2 + C_2, \tag{28}$$

где последнее слагаемое относится к "довычислению" компонент векторных произведений (24), т.е. суммированию частичных произведений вида $(t^n, p^n)_q$, расположенных в разных процессорах. Если все процессоры передают их одновременно в головной процессор, где

они складываются, а результат (опять одновременно) рассылается по всем процессорам, то $C_2 \approx P$.

Времена пересылок "краевых условий" между процессорами оцениваются величиной

$$T_P^c \le C_3(\tau_0 + 2\tau_c M^2 N). \tag{29}$$

Здесь значение C_3 не зависит от M, N и P в идеальном случае, когда все пересылки типа (19) между соседними процессорами происходят одновременно.

Реализация арифметических действий в рассматриваемом алгоритме на одном процессоре занимает в P раз больше времени, и при $P \gg 1$ мы имеем практически линейное ускорение, т.е.

$$S_P = T_P^a \cdot P/(T_P^a + T_P^c) \approx P, \quad E_P \approx 1. \tag{30}$$

Необходимо иметь в виду, что общий объем вычислений в двойном итерационном процессе может оказаться больше, чем при использовании "обычного" одноуровневого предобусловленного крыловского метода для решения полной исходной СЛАУ. Это связано, во-первых, с тем, что реально $n_{in} \cdot n_e(\varepsilon) > n(\varepsilon)$, ($n_{in}$ – среднее число внутренних итераций, а $n(\varepsilon)$ – количество итераций одноуровневого алгоритма), а во-вторых – с неизбежными "накладными расходами" метода декомпозиции.

Однако принцип "разделяй и властвуй" является неизбежным при решении больших задач (M исчисляется тысячами или десятками тысяч), когда главная проблема заключается в интеграции вычислительных ресурсов, в силу тривиальной нехватки памяти одного процессора.

4. Примеры численных экспериментов

Мы проиллюстрируем скорость сходимости и эффективность распараллеливания рассмотренных алгоритмов в применении к решению симметричных систем сеточных семиточечных уравнений (6), аппроксимирующих задачу Дирихле для уравнения Пуассона (5) в единичном кубе $\Omega = [0,1]^3$ на различных кубических сетках с шагами $h = 1/(M+1)$. Функции f и g в (5) выбирались из условия, что точное решение $u(x,y,z)$ равно единице. В качестве начального приближения выбиралась функция $u^0(x,y,z) = 0$, а критерий окончания внешних и внутренних итераций полагался одинаковым и равным $\varepsilon_e = \varepsilon_{in} = 10^{-3}$, за исключением оговоренных ниже случаев. Все вычисления проводились со стандартной двойной точностью, а разделение областей осуществлялось одномерное по направлению оси z, в соответствии со схемой на рис.2. Расчеты выполнялись на кластере с процессорами Itanium-2, а внутренние итерации для решения задач в подобластях для простоты реализовывались с помощью "явного"метода сопряженных градиентов, т.е. без предобуславливания.

В табл. 1 приведены характеристики двойного итерационного процесса при декомпозиции расчетной области на две подобласти с величинами перехлеста $\Delta = 2, 4$, которые измеряем в количестве шагов сетки. При этом общее количество шагов сетки бралось равным $M = 9, 19, 29, 49, 99$, а значение итерационного параметра в (10), характеризующее тип условия внутреннего сопряжения, выбиралось $\theta = 0, 0.25, 0.5, 0.75, 1.0$.

В каждой клетке таблицы приведены 5 чисел: количества потребовавшихся внешних итераций (верхние два числа), общее число внутренних итераций (вторая строка) и время счета в секундах (для вариантов, соответствующих значениям $\Delta = 4$). В данном случае внешние итерации проводились без применения СГ и представляли собой фактически просто блочный алгоритм Якоби.

В табл. 2 приводятся аналогичные данные, с тем отличием, что внешние итерации выполнялись по формулам (16) метода сопряженных градиентов.

Табл. 3 и 4 содержат те же характеристики итерационных процессоров, но при декомпозиции расчетной области на три одинаковые подобласти, с теми же величинами пересечений

θ/M	9	19	29	49	99
0	8 4 100 73 0	15 8 338 214 5.07-02	21 11 659 399 0.3	34 18 1631 957 3.54	64 33 5542 3094 137.78
0.25	6 4 82 75 0	11 7 264 197 5.07-02	16 10 534 377 0.28	26 15 1312 838 3.13	49 29 4430 2803 124.07
0.5	4 3 77 69 0	8 5 210 159 3.9-02	11 8 402 321 0.25	18 12 976 710 2.66	33 23 3164 2325 103.7
0.75	4 3 84 71 0	4 3 151 140 3.51-02	5 5 251 252 0.2	9 7 579 477 1.84	17 14 1844 1583 72.37
1.00	∞	∞	∞	∞	∞

Таблица 1. Характеристики двойных итераций без внешнего ускорения для $P = 2, \Delta = 2, 4$

θ/M	9	19	29	49	99
0	3 2 57 48 0	5 3 176 114 3.9-02	6 4 292 214 0.17	7 5 511 400 1.55	10 7 1197 920 43.83
0.25	3 2 60 50 0	4 3 145 118 3.12-02	5 4 259 218 0.17	6 5 463 418 1.6	9 7 1157 990 46.52
0.5	2 2 46 52 0	4 3 150 129 3.12-02	4 3 220 172 0.14	6 4 495 358 1.41	8 6 1102 990 42.88
0.75	2 2 50 54 0	3 2 126 102 2.34-02	3 3 175 198 0.16	4 4 372 376 1.46	6 5 961 861 40.73
1.00	∞	∞	∞	∞	∞

Таблица 2. Характеристики двойных итераций с внешним ускорением методом СГ для $P = 2$, $\Delta = 2, 4$

Δ в два и четыре шага сетки.

θ/M	9	19	29	49	99
0	9 4 98 72 -	17 8 290 183 4.68-02	24 12 560 334 0.24	39 20 1390 780 2.25	73 38 4572 2573 95.88
0.25	6 4 77 71 -	13 7 241 172 4.68-02	18 11 449 322 0.23	30 18 1129 729 2.11	56 33 3631 2325 87.10
0.5	4 3 70 61 -	8 6 176 159 4.68-02	12 9 369 289 0.21	20 14 819 607 1.82	38 26 2653 1931 73.39
0.75	4 3 73 64 -	4 3 148 122 3.12-02	6 5 262 222 0.164	10 8 498 423 1.32	20 16 1692 1354 52.74
1.00	8 4 61 43 -	18 6 219 110 34.68-02	∞	∞	∞

Таблица 3. Характеристики итерационных процессов для $P = 3, \Delta = 2, 4$ без внешних сопряженных градиентов

θ/M	9	19	29	49	99
0	6 4 94 80 -	7 5 199 162 3.9-02	8 6 307 267 0.19	10 8 578 512 1.51	14 10 328 1061 43.56
0.25	5 4 82 80 -	6 5 181 166 4.68-02	8 6 324 274 0.19	9 7 545 469 1.40	12 9 1215 1010 41.05
0.5	5 5 85 88 -	5 5 162 170 4.29-02	6 5 271 243 0.17	7 7 465 487 1.44	10 8 1120 964 38.92
0.75	6 5 95 94 -	5 6 174 205 5.07-02	5 4 255 224 0.16	5 5 393 403 1.21	7 7 914 958 38.49
1.00	∞	∞	∞	∞	∞

Таблица 4. Характеристики итерационных процессов с внешними СГ ($P = 3, \Delta = 2, 4$)

Анализ приведенных результатов позволяет сделать следующие выводы:

- с увеличением отношения M/Δ (т.е. с относительным уменьшением размера перехлеста подобластей) количество внешних итераций и общее число внутренних итераций значительно увеличиваются, причем этот рост существенно слабее при использовании СГ;

- при изменении параметра θ от 0 до 1 характерна следующая тенденция: при использовании СГ и без него количество внешних и суммарное число внутренних итераций уменьшаются для всех M, но при $\Delta = 1$ обнаруживается, как правило, расходимость процесса (в таблицах обозначаемая символом "∞"), в результате чего обнаруживается наличие минимума (в том числе и по времени счета) для значений $\theta = 0.5 \div 0.75$;

- применение ускорения внешних итераций дает значительный эффект, особенно в "плохих"случаях, т.е. при увеличении общего числа узлов и количества подобластей (процессоров); однако при этом наблюдается парадоксальное на первый взгляд уменьшение коэффициента ускорения S_P (в сравнении без применения СГ); тем не менее, этот факт еще раз подтверждает то известное явление, что более плохие алгоритмы легче и эффективнее распараллеливаются.

Очевидно, что при увеличении количества подобластей и соответствующего числа процессоров P количество внешних итераций (для фиксивованного общего числа узлов в расчетной области) будет расти. Хотя при этом трудоемкость решения каждой вспомогательной задачи в подобласти уменьшается, общий объем вычислений в методе декомпозиции, по сравнению с решением исходной задачи на одном процессоре (если это позволяет его память), будет увеличиваться.

Безальтернативность метода декомпозции заключается в его распараллеливании и конечном росте производительности МВС. Однако вопросы оптимизации таких вычислительных процессов являются во многом открытыми и требуют серьезных исследований. Например, теоретически является естественным переход к двумерной и трехмерной декомпозиции, однако реальная эффективность такого подхода будет зависеть от качества отображения получаемых алгоритмов на архитектуру вычислительной системы (в частности, от реализации коммуникаций).

Список литературы

1. Смелов В.В., Журавлева Т.Б. Принципы итерирований по подобластям с эллиптическим уравнением.–М., ОВМ РАН, 1981, препринт N 14.

2. Лебедев В.И., Агошков В.И. Операторы Пуанкаре–Стеклова и их приложения в анализе.–М., ОВМ АН СССР, 1983.

3. Marchuk G.I., Kuznetsov Y.A., Matsokin A.M. Fictitious domain and domain decomposition method.– Sov.J. Num. Anal. Math. Mod., 1986, v. 1, N 1, 3-35.

4. Quarteroni A., Valli A. Domain Decompozition Methods for Partial Differential Equations.– Oxford Science Publications/Claredon Press., Oxford, 1999.

5. Bramble J.H., Pasciak J.E., Wang J., Xu J. Convergence estimates for product iterative methods with applications to domain decomposition.//Math., Comp., v. 57, N 195, 1991, 1-21.

6. Dryja M., Widlund O.B. Domain decomposition algorithms with small overlap.–SIAM J. Sci. Comp., 1994, 15(3), 604-620.

7. Ильин В.П. О стратегиях распараллеливания в математическом моделировании.// Программирование, N 1, 1999, 41-46.

8. Ильин В.П., Свешников В.М. Оценки эффективности распараллеливания алгоритмов декомпозиции областей.//Автометрия, N 1, 2002, 31-41.

9. Ильин В.П. Параллельные алгоритмы для больших прикладных задач: проблемы и технологии.//Автометрия, т. 43, N 2, 2007, 3-21.

10. Nepomnyashchikh S.V. Domain decomposition methods.–Radon Series Comp. Appl. Math., 2007, 1, 89-159.

11. Василевский Ю.В., Ольшанский М.А. Краткий курс по многосеточным методам и методам декомпозиции области.–М., изд. фак. ВМиК МГУ; МАКС Пресс, 2007.

12. Loisel S., Szyld D.B. On the geometric convergence of optimized Schwarz methods with application to elliptic problems.–Numer. Math., v. 114, 2010, 697-728.

13. Свешников В.М. Построение прямых и итерационных методов декомпозиции.// СибЖИМ, 2009, т.12, N 3(39), с. 99 - 109.

14. Ильин В.П. Проблемы высокопроизводительных технологий решения больших разреженных СЛАУ.–Вычислительные методы и программирование.–М., МГУ, т. 10, 2009, 141-147.

15. http://www.ddm.org/

16. Ильин В.П. Методы конечных разностей и конечных объемов для эллиптических уравнений.–Новосибирск, изд. ИВМиМГ СО РАН, 2001.

17. Ильин В.П. Методы и технологии конечных элементов.–Новосибирск, изд. ИВМиМГ СО РАН, 2007.

18. Ильин В.П. Об экзапроблемах математического моделирования.–CAD/CAM/CAE Observer, N 2 (54), 2010, 85-92.

19. Saad Y. Iterative methods for sparse linear systems. - PWS Publishing: New York, 1996

Параллельные процессы на этапах петафлопного моделирования

В.П. Ильин

Институт вычислительной математики и математической геофизики СО РАН,
Новосибирский государственный университет

1. Введение

Целью данной работы является рассмотрение современных вопросов масштабируемого распараллеливания алгоритмов при решении больших наукоемких задач математического моделирования. Эта тема тесно связана с проблемой высокопроизводительных вычислений на многопроцессорных вычислительных системах (МВС), и зачастую такая связь формулируется односторонне как отображение алгоритмов на архитектуру МВС. Однако неизбежно появление и встречного движения, которое можно сформулировать как конвергенцию алгоритмических структур и вычислительных архитектур, обсуждавшуюся в [1] и цитируемых там работах.

Появившиеся в конце "нулевых" годов петафлопные вычислители (которые были точно предсказаны еще за двадцать лет) и грядущий приход "экзафлопников" к концу первого десятилетия 21-го века переводят рассматриваемую проблематику на качественно новый уровень. В соответствии с прогнозами "дорожной карты" в [2], мировое сообщество ожидают в недалеком будущем компьютеры с десятками миллионов процессоров и миллиардами вычислительных ядер. С одной стороны, этот факт необозримо усиливает возможности вычислительных экспериментов для сложнейших явлений и процессов за реальное время, что делает математическое моделирование не просто третьим, а основным путем познания (наряду с теоретическими и натурными исследованиями). Но в то же время кардинально меняющиеся технические условия требуют обновления и модели вычислений, и технологий программирования, и концепции взаимодействия с динамически развивающимися суперкомпьютерами (хотя зачастую новое — это хорошо забытое старое).

Согласно общепринятым представлениям, архитектура МВС в ближайшее десятилетие не претерпит революционных изменений и будет эволюционировать в направлении гетерогенных многопроцессорно-многоядерных систем, использующих в вычислительных узлах ускорители типа графических процессорных элементов (GPU, которые уже успешно применяются и для арифметических задач) или/и программируемых логических интегральных схем (реконфигурируемых ПЛИСС, или FPGA, см. [3]).

Что касается конкретной аппаратной инфраструктуры, то здесь на крайних позициях находятся многоядерные "персональные суперкомпьютеры" и голиафы – "датацентры" (datacenter, или ЦОД – центр обработки данных) с петафлопной и более высокой производительностью. Промежуточное положение занимают мини- или миди-датацентры с быстродействием около десятков терафлопс, решающие частные корпоративные задачи.

Техническое оборудование (hardware, или "железо", сложнейший комплекс которого получил название IaaS – Infrastructure as a Service), естественно, неотделимо от системного программного обеспечения, включающего и операционные системы с компиляторами, и средства управления заданиями с диспетчеризацией и распараллеливанием, которые в совокупности составляют компьютерную платформу (PaaS – Platform as a Service) для пользователей – разработчиков прикладного программного обеспечения (приложений) в различных предметных областях. Относительно новой концепцией организации датацентров является формирование интеллектуального вычислительного сервиса (SaaS – Software as a Service) для конечных пользователей, заинтересованных только в результатах по решению

конкретных задач из энергетики, биологии, геофизики и т.д. В целом описываемое программное окружение ЦОДа составляет информационные технологии (ИТ или IT) высокого уровня, предоставляющие пользователям универсальный доступ через Интернет и получившего название облачные вычисления (Cloud Computing, см. [4], [5]). В определенном смысле, предлагаемые здесь решения — это осовременные подходы к проблемам вычислительных центров коллективного пользования, обсуждавшиеся еще лет двадцать назад, например, в [6].

К настоящему времени характерные типы программных наукоемких средств, предоставляемых прикладным математикам, программистам и инженерам — это многочисленные пакеты программ (ППП) или библиотеки программ (популярные примеры – ANSYS и NETLIB, по которым, как и по многим другим, имеются обширные материалы в Интернет), существующие в виде свободного доступа или коммерческих продуктов. При этом пользователь или фактически разрабатывает свое приложение, решая сопутствующие трудоемкие вопросы распараллеливания и интерфейсов, или проводит расчеты в жестких рамках существующих ППП, где, возможно, он вставляет свои программные фрагменты, согласно предоставляемым ему правилам (UDF – User Defined Functions).

В настоящей работе развивается проблематика распараллеливания алгоритмов в рамках базовой системы моделирования (БСМ), ранее рассмотренная в [7], [8], которая может явиться конкретизацией технологий облачных вычислений. В п. 2 описывается на содержательном уровне достаточно общая постановка задач математического моделирования, включая междисциплинарные и обратные, а также основные технологические этапы крупномасштабного вычислительного эксперимента. В соответствии с таким представлением в п. 3 обсуждаются принципы построения главных компонент, архитектура, вопросы распараллеливания и пользовательских интерфейсов БСМ в рамках функционирования датацентров. Подчеркнем, что базовая система моделирования рассматривается как совокупность достаточно автономных вычислительных проблемно-независимых инструментариев, согласованных по смежным структурам данных и ориентированных на гибкую сборку широкого класса приложений по принципу детского конструктора.

2. Постановки задач и этапы наукоемкого моделирования

Среди всевозможных применений компьютера мы останавливаемся только на задачах математического моделирования, классические постановки которых описываются дифференциальными и/или интегральными уравнениями или соответствующими вариационными формулировками для обобщенных решений. Традиционно такая область называется математической физикой, хотя в последние десятилетия сюда добавляются вычислительная химия, вычислительная биология и т.д.

Междисциплинарность проблемы означает необходимость одновременного расчета процессов различной природы. Например, проектирование корабля требует моделировать его гидродинамические качества, прочность корпуса, тепловые режимы, электротехническое оборудование и т.п. С математической точки зрения это означает решение не одного, а системы уравнений, как правило, нелинейной и нестационарной, в которой начально-краевые условия и коэффициенты отражают материальные свойства сред и внешние воздействия. Реальные конфигурации расчетных областей с криволинейными кусочно-гладкими границами и острыми углами привносят свои сложности при формировании и обосновании математической постановки. Важно также разделить понятия прямых и обратных задач. К первым относятся те, у которых надо найти решение при полностью заданных начально-краевых условиях. К обратным мы относим оптимизационные постановки с параметризованными данными, которые надо определить по условию минимума некоторого целевого функционала. Такие задачи идентификации параметров модели требуют многократного решения прямых задач и применения алгоритмов нелинейного программирования.

Геометрическое функциональное моделирование. Задание пользователем исходных данных задачи и указаний по методам ее решения — это первая стадия вычислительного эксперимента, которую назовем этапом геометрического и функционального моделирования. Его содержанием является описание и возможное редактирование геометрических объектов расчетной области (вершины, ребра, грани, фигуры), вместе с указанием топологических связей между ними, а также функциональных объектов, характеризующих типы решаемых уравнений, краевые и начальные условия, а также представляемых в них коэффициентах (с привязкой к соответствующим подобластям и граничным поверхностным сегментам). Для обратных задач дополнительно указываются параметризованные исходные данные, совместно с описанием ограничений на возможные вариации параметров и формулировкой целевого функционала. Для обеспечения дружественного интерфейса пользователю должны на данном этапе предоставляться интеллектуальные графические и текстовые средства формирования математической модели и вычислительного задания. А результатом работы этой стадии является геометрическая и функциональная структуры данных (ГСД и ФСД), однозначно определяющие исходную информацию, необходимую для выполнения последующих этапов моделирования. В целях придания гибкости БСМ предусматривается возможность множественность представления форматов, с их взаимной конвертизацией. В частности, это необходимо для поддержки взаимодействия с распространенными внешними приложениями, например, графическими или САПРовскими продуктами, базирующимися на общепринятых геометрических форматах, см. подробнее [9], [10].

Дискретизация математической модели. Построение сетки представляет особо сложную проблему в многомерных случаях с кусочно-гладкими криволинейными и, возможно, движущимися границами (для последних специальный класс составляют т.н. свободные границы, положение которых заранее неизвестно). Сама сетка определяется совокупностью своих объектов различной размерности: узлы, ребра, грани, конечные объемы,– а также топологическими связями между ними. Между геометрической структурой расчетной области с подобластями и сеточной структурой существует большая аналогия, и они содержат по сути однотипные наборы объектов макро- и микро-уровня.

Существует большое количество критериев качества сетки, которые в значительной степени определяют точность и экономичность численного решения. Одно из главных требований сетке – адаптивность, означающая в том или ином смысле учет особенностей конфигурации границы и свойств искомого решения, которые определяются или теоретически априори или апостериори экспериментально, т.е. на основе предварительных расчетов.

В первую очередь требуется, чтобы вершины, ребра и граничные поверхности расчетной области составлялись из соответствующих сеточных объектов, или в крайнем случае аппроксимировались ими с возможно малой погрешностью. Второй аспект связан с возникающей сингулярностью, т.е. сильным ростом производных, в окрестности углов и ребер границы, что требует специального сгущения узлов в таких подобластях. А при наличии сильно меняющихся пространственно-временных особенностях решения сетки должны быть динамическими, т.е. регулярно или периодически перестраиваемыми.

Распространенные сеточные технологии включают триангуляции Делоне, ячейки Дирихле-Вороного и различные приемы контроля вырожденных случаев. Важным вычислительным средством являются многосеточные методы, основанные на построении последовательности вложенных сеток или на их локальном сгущении.

Наиболее просто конструируемыми и одновременно обеспечивающими больший порядок точности являются равномерные сетки, которые могут иметь различные типы конечных элементов: кубы или параллелепипеды, призмы, тетраэдры и т.д. Однако в реальных задачах сетки приходится строить неравномерные и нерегулярные, или неструктурированные, у которых для каждого узла номера его ближайших соседей можно задать только перечислением. Существуют компромиссные квазиструктурированные сетки, когда расчетная сеточная область состоит из сеточных подобластей, в каждой из которых сетка строится

по своим правилам и может быть структурированной. Такие сетки могут быть несогласованными или согласованными - в последних случаях узлы на смежных границах раздела соседних сеточных подобластей являются совпадающими.

Методы построения сеток отличаются большим разнообразием и имеют обширную профессиональную литературу. Здесь используются и квазиконформные отображения, и вариационные принципы, и специальные метрические пространства, и многочисленные эмпирические приемы. Соответствующее программное обеспечение, или генераторы сеток, существует в широком ассортименте, или в свободном доступе через Интернет, или в качестве коммерческих продуктов. Зачастую процедуры построения сеток являются неотъемлемой частью проблемно-ориентированных пакетов прикладных программ (ППП), но они также существуют и в автономной форме, позволяющей их встраивать в различные приложения.

Согласно идеологии БСМ, стадия дискретизации задачи реализуется независимо от остальных, используя на входе только один из форматов ГСД, ГФСД и формируя на выходе сеточную структуру данных (ССД), также допускающую множественные форматы данных с конверторами. Подчеркнем, что методология квазиструктурированных сеток допускает использование внешних существующих сеточных генераторов, осуществляя таким образом актуальные возможности интеграции различных приложений и переиспользования программного кода.

Алгоритмы аппроксимации. Конечной целью дискретизации является переход от исходных функциональных соотношений к конечно-мерным уравнениям, неравенствам или рекурсиям. При этом фактически задача алгебраизируется, что достигается путем аппроксимации функций, производных и интегралов. Основные подходы к построению сеточных соотношений – это методы конечных разностей, конечных объемов, конечных элементов (МКР, МКО, МКЭ, см. [11], [12]), коллокаций и спектральные методы, связанные с разложениями в ряды Фурье. Не пытаясь делать обзора имеющегося огромного материала по данным вопросам, мы коротко остановимся главным образом на технологических аспектах наиболее универсальных МКО и МКЭ, позволяющих конструировать сеточные аппроксимации высоких порядков точности на различных типах конечных объемов для широкого класса задач математического моделирования.

Важным моментом этих двух подходов является поэлементная технология вычисления локальных матриц и векторов правых частей с последующей сборкой (ассемблированием) глобальных матриц и систем алгебраических уравнений (линейных или нелинейных – СЛАУ или СНАУ). Этот прием значительно упрощает программную реализацию и естественным образом обеспечивает эффективное распараллеливание данного вычислительного этапа, поскольку такие процедуры аппроксимации носят локальный характер, реализация которых использует свойства только топологически ближайших сеточных объектов, но никак не использует их дальние связи..

В нестационарных проблемах пространственная аппроксимация осуществляется на каждом временном шаге, а при наличии нелинейностей такая процедура повторяется итерационно для всех шагов. Наиболее трудоемкими оказываются задачи с динамическими сетками, если их приходится перестраивать на каждой итерации для каждого шага по времени.

Совокупность аппроксимационных алгоритмов для типовых задач математической физики может быть систематизирована и классифицирована по следующим признакам:

- по типу аппроксимируемого члена дифференциального уравнения (градиент, дивергенция и т.д.) или его механического смысла; примером могут быть матрицы жесткости и матрицы масс в МКЭ;

- по виду конечных элементов или объемов: тетраэдры, параллелепипеды, призмы и т.д.; при этом могут выделяться частные случаи, которые наиболее экономично реализуются (например, правильные фигуры);

- по характеру базисных функций, которые могут отличаться своими носителями, по-

рядками и структурными свойствами (лагранжевые и эрмитовые, скалярные и векторные, и т.п.);

- по конфигурации сеточного шаблона, т.е. совокупности узлов, участвующих в одном уравнении, получаемом в итоге формирования алгебраических систем; наиболее экономичными оказываются компактные схемы, в которых участвуют только наиболее близкие геометрически соседние узлы; как правило, повышение порядка аппроксимации ведет к растягиванию сеточного шаблона и расширению ленточной структуры итоговых матриц, так что с точки зрения общей эффективности алгоритмов приходится искать золотую середину.

Результатом выполнения этапа аппроксимации является алгебраическая структура данных (АСД), аккумулирующая на дискретном уровне всю исходную информацию о решаемой проблеме, и для которой имеются общепринятые в мире сжатые матричные форматы, вместе с соответствующими переходниками – конверторами.

Задачи вычислительной линейной алгебры. Если решаемая проблема в целом является нестационарной и нелинейной, все равно после этапов временной аппроксимации и квазилинеаризации мы приходим к задачам линейной алгебры, из которых наиболее типичны – это решение СЛАУ или проблемы собственных значений.

Характерные матрицы, возникающие в сеточных методах решения дифференциальных задач, являются разреженными, ленточными и большими. Это означает, во-первых, что порядки N достигают десятков и сотен миллионов, а ненулевые элементы сосредоточены в некоторой полосе ширины m около главной диагонали, причем величина m и число ненулевых элементов в каждой строке не зависят от N. Дискретизированные алгебраические системы, возникающие из аппроксимации интегральных уравнений (определенных на границе или в объеме расчетной области) являются, наоборот, плотными, но зачастую имеют специальные структурные свойства (например, являются теплицевыми, квази- или блочно-теплицевыми).

Вычислительная линейная алгебра – хорошо продвинутая математическая дисциплина и содержит большое количество алгоритмов для решения задач с самыми разными типами матриц: вещественными и комплексными, квадратными и прямоугольными, эрмитовыми и неэрмитовыми, положительно определенными и знаконеопределенными. Имеется также соответствующее обширное программное обеспечение в виде библиотек или прикладных пакетов, как свободно распространяемых в Интернете, так и коммерческих.

Алгебраические задачи – "узкое горлышко" математического моделирования, поскольку потребляемые вычислительные ресурсы нелинейно растут с увеличением порядка N. Поэтому здесь особенно актуально распараллеливание алгоритмов на МВС с общей, разделенной и гибридной памятью.

Существенным моментом программного обеспечения для разреженных матриц является оптимизация кода, поскольку применение сжатых форматов данных позволяет кардинально сокращать объем хранимой информации, но существенно усложняет реализацию доступа к матричным элементам, что особенно критично при многоуровневой неоднородности кэша и оперативной памяти.

Методы оптимизации и решения нелинейных уравнений. Алгоритмы оптимизации решения обратных задач, сводящихся к проблеме условной минимизации функционала, являются бурно развивающейся в последние десятилетия областью вычислительной математики. Здесь развиты модификации методов множителей Лагранжа и внутренних точек, являющихся развитием подходов со штрафными функциями, использование доверительных интервалов для регулировки последовательности шагов, а также варианты алгоритмов Ньютона и последовательного квадратичного программирования для решения возникающих на промежуточных этапах СНАУ.

В данной тематике имеется много актуальных и далеко не решенных вопросов, связанных с поиском глобального минимума и оптимального управления, применения методов

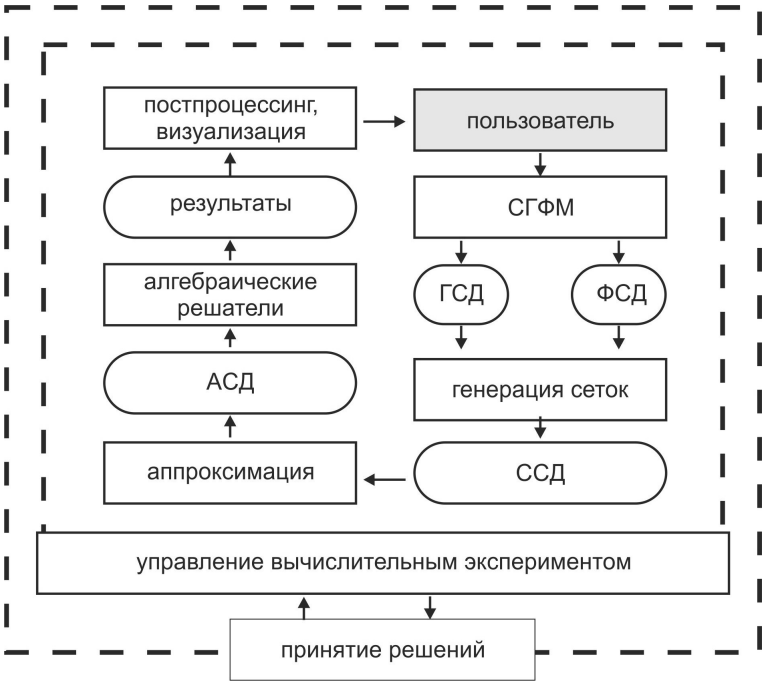


Рис. 1. Структура функциональных и информационных компонент БСМ

теории возмущений и сопряженных уравнений, нахождения градиентов функционалов или функций чувствительности к вариациям исходных данных. Зачастую постановки требуют штучного исследования устойчивости и корректности задачи для поиска соответствующего регуляризационного подхода. К этой же области можно отнести изучение нелинейных динамических систем, связанных с эффектами бифуркаций, самоорганизации и хаоса, странных аттракторов и т.д., где зачастую еще остаются открытыми методологические принципы математического моделирования.

Постобработка и визуализация результатов. Непосредственная реализация наукоемких алгоритмов в многомерных задачах может составлять отнюдь не главную долю общего вычислительного процесса, поскольку анализ получаемых результатов требует их презентабельной визуализации с предварительной обработкой сеточных функций, что требует выполнения ресурсоемких технологических операций по формированию сечений, изолиний и изоповерхностей, различных графиков с возможной анимацией многоцветных изображений. Такие технологические проблемы особенно актуальны в системах принятия решений по результатам математического моделирования.

3. Структура и принципы распараллеливания БСМ

Рассмотренные технологические этапы решения больших задач естественным образом отображаются на архитектуру и состав основных функциональных и информационных компонент БСМ, изображенных на рис. 1. Приведенная схема фактически представляет последовательность действий, осуществляемых при проведении полноформатного вычислительного эксперимента для изучения какого-то технического процесса или природного явления.

Для конечного пользователя (“модельера”) операционное окружение представляется, с одной стороны, входным интерфейсом, т.е. системой геометрического и функционального моделирования (СГФМ), включающей средства управления вычислительным процессом и принятия решений (последние аспекты представляют самостоятельную тему, выходящую за рамки данной работы). Выходной же интерфейс обеспечивается обработкой и визуали-

зацией результатов расчета (постпроцессинг), на основе анализа которых могут меняться исходные данные, математическая модель или алгоритмы и формироваться новые вычислительные сеансы. Получаемые в итоге выполнения этого этапа информационные массивы ГСД и ФСД содержат, как правило, до нескольких сот объектов, в силу чего при его параллельной реализации на МВС формируемые данные целесообразно копировать в памяти всех процессоров, во избежание излишних коммуникационных потерь.

Библиотека сеточных генераторов (условное название – DELAUNAY), говоря формальным языком, является преобразователем данных ГСД+ФСД→ССД и, как уже отмечалось, служит интегратором программных процедур от различных разработчиков, на основе гибкой системы внутренних интерфейсов. Актуальную роль играет библиотека конечно-объемных и конечно-элементных "аппроксиматоров", которая на основе ССД, ГСД и ФСД формирует алгебраическую структуру данных.

На основе АСД функционирует библиотека алгебраических решателей KRYLOV, концепция и общая структура которой описана в [13].

Решение сложной математической проблемы организуется в общем случае по многократно вложенным циклам: численное интегрирование нестационарных задач по временным шагам, проведение на каждом шаге итераций по нелинейности, если таковая присутствует, расщепление по физическим процессам или/и по пространственным переменным, варьирование исходных данных в многовариантных расчетах или в обратных задачах. На всех этих уровнях могут строиться свои тактики и стратегии распараллеливания, но в любом случае одним из наиболее критичных моментов является параллельная реализация СЛАУ. Здесь главный подход заключается в алгебраической декомпозиции, которая на дискретном представлении отражает геометрическое разделение сточной расчетной области со сбалансированным разделением подобластей по соответствующим процессорам. Данная проблема представляет собой специальную задачу на графах с высокой вычислительной сложностью, и здравый компромисс заключается в поиске ее приближенного решения.

Следует сказать, что структура АСД с обычными сжатыми матричными форматами ориентирована на экономию памяти при решении больших разреженных СЛАУ и слабо "помнит" топологические свойства исходной расчетной области, которые могут иметь значительное влияние на эффективность метода декомпозиции. Поэтому разделение областей целесообразно начинать еще на этапе построения сетки, что должно отражаться в формировании и сеточных, и алгебраических структур данных.

При решении стационарной задачи, т.е. СЛАУ – на дискретном уровне,- алгоритм декомпозиции формально представляет собой крупно-блочный итерационный метод, скорость сходимости которого тем выше, чем меньше минимальное расстояние (в топологическом смысле) между подобластями. Поэтому при возможном разделении трехмерной расчетной области в одномерные ($1D$), двумерные ($2D$) или трехмерные ($3D$) информационные структуры предпочтение, естественно, следует отдавать последним. Однако эффективность $3D$ – декомпозиции при ее реализации на конкретной МВС сильно зависит от реальных физических связей между процессорами (идеальный случай, когда они составляют трехмерную вычислительную сеть).

Оптимизация массивного распараллеливания в конкретных случаях требует анализа коэффициентов ускорения и использования вычислительной системы

$$S_p = T_1/T_p, \quad E_p = S_p/p,$$

где T_p – время решения задачи (или реализации алгоритма) на p процессорах. К сожалению, сделать это непросто в силу отсутствия адекватных моделей машинных вычислений на МВС с иерархической памятью. При оценке эффективности производительности многопользовательского вычислительного центра ситуация значительно усложняется и требует учета распределения глобальных ресурсов между потоками задач.

Поскольку главное снижение производительности многопроцессорных компьютеров происходит из-за коммуникационных потерь, с точки зрения эффективности решения рассматриваемых задач перспективными являются следующие направления развития архитектур МВС:

- вычислительные сети (ВС) различной размерности (одно-, дву- и трехмерные);
- динамическая реконфигурация ВС с реализацией как структурированных, так и неструктурированных сеток;
- быстрые ближние связи и некоторые специальные коммуникации (типа гиперкуба);
- синхронизация обменов и вычислений, совмещение во времени двусторонних обменов;
- специализированные вычислительные устройства и гетерогенные МВС (например, аппаратная поддержка постобработки и визуализации, алгоритмов линейной алгебры).

В силу принципиальных появившихся технических возможностей, благодаря прогрессу "кремниевых технологий", становится реальным переход от лозунга "отображение алгоритмов на архитектуру МВС" к двустороннему движению (конвергенции) между данными категориями.

Список литературы

1. Ильин В.П. Об экзапроблемах математического моделирования. CAD/CAM/CAE Observer, N 2(54), 2010, 85-92.
2. Dongarra J., Beckman P., et. al. - IESP: International Exascale Software Project. Road Map, 18 Nov., 2009, www.exascale.org.
3. Каляев И.А., Левин И.И. Семейство реконфигурируемых вычислительных систем с высокой производительностью.//Вычислительные методы и программирование, т. 10, N 1, 2009, 207-214.
4. Armbrust M. et al. About the Clouds: A Berkeley View of Cloud Computing.–Technical Report No. UCB/EECS – 2009-28 http://www.eecs.berkeley.edu/Pubs.
5. Колесов А. IT-область. Сгущается облачность. Суперкомпьютеры. N 3, 2010, 8-13.
6. Алексеев А.С., Гололобов В.И., Ильин В.П., Карначук В.И. Комплексный центр математического моделирования: концепция программного обеспечения.–Новосибирск, ВЦ СО АН СССР, препринт N 821, 1988.
7. Ильин В.П. Параллельные алгоритмы для больших прикладных задач: проблемы и технологии.//Автометрия, N 2, 2007, 3-21.
8. Ильин В.П. Экзапроблемы математического моделирования.//Вестник ЮУрГУ, сер. "Математическое моделирование и программирование", вып. 6, N 35(211), 2010, 28-39.
9. Ильин В.П. Геометрическое и функциональное моделирование в задачах математической физики.–Новосибирск, Вычислительные технологии, т. 6, ч. 2, 2001, 315-321.
10. Ушаков Д.М. Введение в математические основы САПР.–Новосибирск, изд. комп. Ледас, 2006.
11. Ильин В.П. Методы конечных разностей и конечных объемов для эллиптических уравнений.–Новосибирск, изд. ИВМиМГ СО РАН, 2001.

12. Ильин В.П. Методы и технологии конечных элементов.–Новосибирск, изд. ИВМиМГ СО РАН, 2007, 370с.

13. Бутюгин Д.С., Ильин В.П., Ицкович Е.А., Петухов А.В., Кныш Д.В. Krylov: библиотека высокопроизводительных алгоритмов для решения разреженных СЛАУ.–Труды 13-й Всероссийской конференции “Современные проблемы математического моделирования”. – Ростов, изд. ЮФУ, 2009, 110-128.

Параллельный алгоритм для решения трёхмерных уравнений Максвелла с разрывной диэлектрической проницаемостью на призматических сетках

Т.З. Исмагилов, А.И. Горбачёв
Новосибирский Государственный Университет

Предлагается конечно-объёмный метод для численного решения трёхмерных уравнений Максвелла с разрывной диэлектрической проницаемостью на призматических сетках. Метод позволяет проводить расчёты для разрыва диэлектрической проницаемости проходящего по произвольному гладкому цилиндру. Численный алгоритм допускает параллельную реализацию для использования на многопроцессорных ЭВМ. Приведённые результаты тестовых расчётов подтверждают второй порядок предлагаемого метода и высокую эффективность параллельной реализации.

1. Введение

Численная аппроксимация дифференциальных уравнений Максвелла с помощью конечно-разностных схем [1] используется для решения задач связанных с распространением электромагнитных волн и их взаимодействием с зарядами и токами. Начало широкому использованию конечно-разностных схем для численного решения уравнений Максвелла было положено в работе [2]. В этой работе автор предложил схему второго порядка аппроксимации по времени и пространству, основанную на введении смещённых сеток. В дальнейшем конечно-разностные алгоритмы применялись для решения различных задач [3–5].

Тем не менее, для многих задач со сложной геометрией предпочтительнее использовать конечно-объёмные схемы. Они позволяют проводить аппроксимацию на неструктурированных сетках и, таким образом, более точно представлять границы расчётной области, а также границы между подобластями с различными свойствами среды внутри расчётной области. Было предложено несколько конечно-объёмных схем для решения уравнений Максвелла. В схемах предложенных в работе [6] электрические и магнитные поля аппроксимируются на смещённых сетках как и в работе [2]. В алгоритмах рассмотренных в работах [7–10] все компоненты электромагнитного поля аппроксимируются в центрах ячеек.

Одной из основных трудностей при построении схем второго порядка для уравнений Максвелла остаётся случай разрывных свойств среды. В работе [3] рассматривались различные способы сглаживания разрывной диэлектрической проницаемости. Но ни один из них не позволял сохранять порядок аппроксимации исходной схемы. Для решения этой проблемы в работе [10] была предложена конечно-объёмная схема для решения двумерных уравнений Максвелла на треугольных сетках. Тестовые расчёты показывали близкий ко второму порядок аппроксимации для случая разрывной диэлектрической проницаемости. К сожалению эта схема может использоваться только когда разрыв диэлектрической проницаемости проходит по координатной линии.

Более универсальная схема была предложена в [11, 12]. Эта схема позволяет проводить расчёты для трёхмерных уравнений Максвелла на тетраэдральных сетках в областях с разрывной диэлектрической проницаемостью где разрыв может проходить по произвольной гладкой поверхности. Алгоритм допускет эффективную параллельную реализацию для использования на многопроцессорных ЭВМ.

Для трёхмерных задач в которых вычислительная область имеет вид правильного цилиндра, а свойства среды не меняются при сдвиге параллельно образующим цилиндра, в использовании тетраэдральной сетки нет необходимости. Для таких задач лучше подойдут

призматические сетки. Такие сетки можно легко построить с помощью сдвигов треугольной сетки для основания цилиндра. Использование призматических сеток позволяет легко проводить декомпозицию рассчётной области и добиваться равномерного распределения нагрузки между процессами при разработке параллельной реализации.

В данной работе предлагается конечно-объёмная схема для численного решения трёхмерных уравнений Максвелла на призматических сетках с разрывной диэлектрической проницаемостью. Для повышения порядка аппроксимации используется вычисление градиентов специально выбираемых компонент с помощью метода наименьших квадратов. Предлагаемая схема допускает эффективную параллельную реализацию с помощью метода геометрической декомпозиции. В статье приводятся результаты расчётов на многопроцессорных ЭВМ, которые подтверждают второй порядок точности предлагаемой схемы и высокую эффективность параллельной реализации.

2. Уравнения Максвелла

В отсутствие зарядов и токов система уравнений Максвелла в безразмерных переменных имеет следующий вид:

$$\frac{\partial \mathbf{D}}{\partial t} - \mathbf{rotH} = 0, \text{ где } \mathbf{D} = \varepsilon \mathbf{E}, \tag{1}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \mathbf{rotE} = 0, \text{ где } \mathbf{B} = \mu \mathbf{H}, \tag{2}$$

$$\mathbf{divD} = 0, \quad \mathbf{divB} = 0. \tag{3}$$

Здесь $\mathbf{E}$ — электрическое поле, $\mathbf{H}$ — магнитное поле, $\mathbf{D}$ — электрическая индукция, $\mathbf{B}$ — магнитная индукция, ε — диэлектрическая проницаемость, μ — магнитная проницаемость. Далее везде полагаем, что магнитная проницаемость $\mu = 1$. Данная система может быть представлена в векторной консервативной форме

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x_1}\mathbf{F}_1 + \frac{\partial}{\partial x_2}\mathbf{F}_2 + \frac{\partial}{\partial x_3}\mathbf{F}_3 = 0, \tag{4}$$

где $\mathbf{U}$ — вектор консервативных переменных, $\mathbf{F}_1$, $\mathbf{F}_2$ и $\mathbf{F}_3$ — векторы потоков

$$\mathbf{U} = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix}, \ \mathbf{F}_1 = \begin{pmatrix} 0 \\ H_3 \\ -H_2 \\ 0 \\ -E_3 \\ E_2 \end{pmatrix}, \ \mathbf{F}_2 = \begin{pmatrix} -H_3 \\ 0 \\ H_1 \\ E_3 \\ 0 \\ -E_1 \end{pmatrix}, \ \mathbf{F}_3 = \begin{pmatrix} H_2 \\ -H_1 \\ 0 \\ -E_2 \\ E_1 \\ 0 \end{pmatrix}. \tag{5}$$

Проинтегрировав уравнение (4) по объёму Ω с границей $\partial\Omega$ можно получить эквивалентную интегральную форму

$$\frac{\partial}{\partial t}\int\limits_{\Omega} \mathbf{U} d\Omega + \int\limits_{\partial\Omega} \left(n_1\mathbf{F}_1 + n_2\mathbf{F}_2 + n_3\mathbf{F}_3\right) dS = 0, \tag{6}$$

где $\mathbf{n} = (n_1, n_2, n_3)$ — внешняя нормаль. Систему (4) также можно записать в недивергентной форме

$$\frac{\partial}{\partial t}\mathbf{V} + A_1\frac{\partial}{\partial x_1}\mathbf{V} + A_2\frac{\partial}{\partial x_2}\mathbf{V} + A_3\frac{\partial}{\partial x_3}\mathbf{V} = 0, \tag{7}$$

где $\mathbf{V}$ вектор потоковых переменных связаный с вектором консервативных переменных матрицей перехода Θ: $\mathbf{V} = \Theta\mathbf{U}$

$$\mathbf{V} = \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ H_1 \\ H_2 \\ H_3 \end{pmatrix}, \quad \Theta = \begin{pmatrix} \frac{1}{\varepsilon} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\varepsilon} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\varepsilon} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{8}$$

а матрицы A_1, A_2, A_3 записываются как

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\varepsilon} \\ 0 & 0 & 0 & 0 & -\frac{1}{\varepsilon} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{9}$$

$$A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -\frac{1}{\varepsilon} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\varepsilon} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{\varepsilon} & 0 \\ 0 & 0 & 0 & -\frac{1}{\varepsilon} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{10}$$

3. Разностная схема

Рассмотрим расчётную область в трёхмерном пространстве. Будем считать что в ней построена сетка из призм Δ. Если две различные ячейки сетки соприкасаются, то они имеют общую грань, общее ребро или общую вершину. Для каждой ячейки Δ определим объём Ω_Δ и барицентр $\mathbf{X}_\Delta^B$ как

$$\Omega_\Delta = \int\limits_\Delta d\Omega, \quad \mathbf{X}_\Delta^B = \frac{1}{\Omega_\Delta}\int\limits_\Delta \mathbf{X} d\Omega. \tag{11}$$

Для каждой грани Γ определим площадь S_Γ и центр $\mathbf{X}_\Gamma^C$ как

$$S_\Gamma = \int\limits_\Gamma dS, \quad \mathbf{X}_\Gamma^C = \frac{1}{S_\Gamma}\int\limits_\Gamma \mathbf{X} dS. \tag{12}$$

Для приближенного решения уравнения (6) рассмотрим разностную схему

$$\Omega_{\Delta_i}\frac{\mathbf{U}_i^{n+1}-\mathbf{U}_i^n}{\tau} + \sum_{k=1}^{m}\int\limits_{\Gamma_k}\left(n_1\mathbf{F}_1 + n_2\mathbf{F}_2 + n_3\mathbf{F}_3\right)d\Gamma = 0. \tag{13}$$

где $\mathbf{U}^n$ аппроксимация значения $\mathbf{U}$ в барицентре i-ой ячейки $\mathbf{X}^B$ в момент времени $t_n = n\tau$, τ — шаг по времени, а Ω_{Δ_i} объём i-ой ячейки.

Для того чтобы в (13) найти значение $\mathbf{U}$ на новом временном слое — $\mathbf{U}_i^{n+1}$, надо вычислить интегралы по граням Γ_k ячейки Δ_i которые представляют собой потоки искомых величин через грани. Предполагаем, что в ячейке искомые функции изменяются линейно, по значениям этих функций в барицентре ячейки и по вычисленным градиентам функций находим значения функций в центре грани со стороны i-ой ячейки в момент времени $n\tau + \tau/2$. Аналогичным образом находим значения функций в центре грани со стороны ячейки, находящейся по другую сторону грани. По значениям функций по разные стороны грани, вообще говоря различным, находим потоки в центре грани в момент времени $n\tau+\tau/2$. Тогда интегралы в (13) приближенно вычисляем по формуле прямоугольников и получаем.

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \frac{\tau}{\Omega_{\Delta_i}}\sum_{k=1}^{m} s_{\Delta_i}^k \mathbf{F}_i^k, \tag{14}$$

$s_{\Delta_i}^k$ — площадь k-ой грани $\mathbf{F}_i^k$— поток через k-ую грань.

3.1. Нахождение потоков через границу ячейки

Пусть $\mathbf{n} = (n_1, n_2, n_3)$ — вектор нормали к общей грани ячеек Δ_L и Δ_R в точке $\mathbf{X}^C$. Если предположить что производные компонент электромагнитных полей по касательной к грани равны 0, то систему в недивергентной форме (7) можно переписать как

$$\frac{\partial}{\partial t}\mathbf{V} + A\frac{\partial}{\partial n}\mathbf{V} = 0, \tag{15}$$

где $A = A_1 n_1 + A_2 n_2 + A_3 n_3$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{\varepsilon}n_3 & -\frac{1}{\varepsilon}n_2 \\ 0 & 0 & 0 & -\frac{1}{\varepsilon}n_3 & 0 & \frac{1}{\varepsilon}n_1 \\ 0 & 0 & 0 & \frac{1}{\varepsilon}n_2 & -\frac{1}{\varepsilon}n_1 & 0 \\ 0 & n_3 & n_2 & 0 & 0 & 0 \\ n_3 & 0 & -n_1 & 0 & 0 & 0 \\ -n_2 & n_1 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{16}$$

Для этой системы можно рассмотреть одномерную задачу Римана где в качестве начальных значений принять аппроксимацию компонент электромагнитных полей в центре грани со стороны ячеек Δ_L и Δ_R — $\mathbf{V}_L\left(\mathbf{X}^C\right)$ и $\mathbf{V}_R\left(\mathbf{X}^C\right)$. Решение этой задачи [9, 10] будем использовать для вычисления потока через границу ячейки.

Обозначим $D = R^{-1}AR$, здесь D — диагональная матрица из собственных значений матрицы A, $D^{\pm}$ — диагональные матрицы, полученные из D заменой всех отрицательных (положительных) собственных чисел нулями, R — матрица, столбцы которой являются правыми собственными векторами матрицы A. Тогда $A = RDR^{-1} = RD^{+}R^{-1} + RD^{-}R^{-1} = A^{+} + A^{-}$. Учитывая $\mathbf{V} = \Theta\mathbf{U}$ для вычисления потоков через грань в (7) будем использовать

$$\mathbf{F} = \Theta^{-1}(A^{+}\mathbf{V}_L\left(\mathbf{X}^C\right) + A^{-}\mathbf{V}_R\left(\mathbf{X}^C\right)) \tag{17}$$

или

$$\mathbf{F} = C^{+}\mathbf{V}_L\left(\mathbf{X}^C\right) + C^{-}\mathbf{V}_R\left(\mathbf{X}^C\right). \tag{18}$$

3.2. Нахождение значений компонент электромагнитных полей на границе ячейки

Рассмотрим исходную систему уравнений в недивергентной форме (7). Пусть $\mathbf{X}^B$ — барицентр ячейки, а $\mathbf{X}^C$ — центр грани. Тогда $\mathbf{V}(\mathbf{X}^C)$ может быть найдена по следующей формуле со вторым порядком по времени и пространству:

$$\begin{aligned}\mathbf{V}(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}^B)(\mathbf{X}^C - \mathbf{X}^B) - \\ &\frac{\tau}{2}\left(A_1\frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}^B) + A_2\frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}^B) + A_3\frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}^B))\right).\end{aligned} \tag{19}$$

Таким образом значения электромагнитных полей на грани ячейки сетки в (18) выражаются следующим образом

$$\begin{aligned}\mathbf{V}_L(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}_L^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}_L^B)(\mathbf{X}^C - \mathbf{X}_L^B) - \\ &\frac{\tau}{2}\left(A_1\frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}_L^B) + A_2\frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}_L^B) + A_3\frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}_L^B)\right),\end{aligned} \tag{20}$$

$$\begin{aligned}\mathbf{V}_R(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}_R^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}_R^B)(\mathbf{X}^C - \mathbf{X}_R^B) - \\ &\frac{\tau}{2}\left(A_1\frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}_R^B) + A_2\frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}_R^B) + A_3\frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}_R^B)\right).\end{aligned} \tag{21}$$

3.3. Нахождение градиентов компонент электромагнитных полей в ячейке

Вычисление градиентов проведём с использованием вектора непрерывных переменных. На границе разрыва диэлектрической проницаемости такими переменными будут нормальная компонента вектора электрической индукции, касательные компоненты вектора электрического поля и декартовы компоненты вектора магнитного поля. В каждой ячейке сетки введём свой вектор переменных $\mathbf{W}$. Будем считать что образующие цилиндра поверхности разрыва диэлектрической проницаемости параллельны оси x_3, В ячейках имеющих общую грань с поверхностью разрыва диэлектрической проницаемости выберем

$$\mathbf{W} = \Xi(\phi,\varepsilon)\mathbf{V} = \begin{pmatrix} \varepsilon\cos(\phi) & \varepsilon\sin(\phi) & 0 & 0 & 0 & 0 \\ -\sin(\phi) & \cos(\phi) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\mathbf{V}, \tag{22}$$

где ϕ — угол вектора нормали к поверхности разрыва в полярной системе координат в плоскости x_1,x_2 , а ε диэлектрическая проницаемость в ячейке. В остальных ячейках выберем $\mathbf{W} = \mathbf{V}$. В узлах сетки введём угол ϕ. В узлах на поверхности разрыва диэлектрической проницаемости выберем его как угол вектора нормали к поверхности разрыва в полярной системе координат в плоскости x_1,x_2. В остальных узлах углы можно выбрать произвольно.

Сначала найдём градиенты $\mathbf{W}$ в ячейках с помощью метода наименьших квадратов. Для этого воспользуемся значениями $\mathbf{V}$ в барицентре самой ячейки и барицентрах соседних ячеек. Может получится так, что у ячейки отсутствуют несколько соседних ячеек. В таком случае вместо значений в их барицентрах возьмём значения в барицентрах ячеек соседних для одной из присутствующих соседних ячеек. Таким образом в методе наименьших квадратов будем использовать

$$\left\{(\mathbf{X}^{B_j}, \Xi(\phi_i, \varepsilon_j)\mathbf{V}_j)\right\}, \tag{23}$$

где j пробегает индексы ячеек которые используются для вычисления градиентов в ячейке i. После нахождения градиентов в i-ой ячейке сетки по найденным градиентам вычислим значения $\mathbf{W}_i\left(\mathbf{X}^P\right) = \mathbf{W}_i^P$ в её узлах $\mathbf{X}^P = \left\{x_l^P\right\}$

$$\begin{aligned}\mathbf{W}_i\left(\mathbf{X}^P\right) &= \Xi(\phi^P, \varepsilon_i)\Xi^{-1}(\phi_i, \varepsilon_i)\left(\mathbf{W}_i + \frac{\partial}{\partial x_1}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_1^P - x_1^{B_i}\right) + \right. \\ &\left. \frac{\partial}{\partial x_2}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_2^P - x_2^{B_i}\right) + \frac{\partial}{\partial x_3}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_3^P - x_3^{B_i}\right)\right),\end{aligned} \tag{24}$$

где ϕ^P значение углов в узле P. В одном узле $\mathbf{X}^P$ получаем несколько различных значений $\mathbf{W}_i\left(\mathbf{X}^P\right)$ по числу соседних ячеек, за итоговое значение $\mathbf{W}^P = \mathbf{W}\left(\mathbf{X}^P\right)$ принимаем их среднее арифметическое.

Теперь в каждой ячейке найдём градиенты $\mathbf{V}$ с помощью метода наименьших квадратов. Для этого будем использовать значения $\mathbf{V}$ в вершинах ячейки. Таким образом в методе наименьших квадратов для вычисления градиентов $\mathbf{V}$ в ячейке i возьмём

$$\left\{(\mathbf{X}^{P^k}, \Xi^{-1}(\phi^{P^k}, \varepsilon_i)\mathbf{W}^{P^k})\right\}, \tag{25}$$

где P^k, $k = 1, 2, ...$ номера вершин ячейки i.

4. Параллельная реализация

Параллельная реализация алгоритма основана на геометрической декомпозиции расчётной области. N слоёв призматической сетки в расчётной области делятся между s процессами так чтобы процессам с более высокими номерами соответствовали слои с более высокими номерами и разница между числами слоёв у различных процессов была не больше единицы. Если N слоёв сетки пронумерованы с 1 по N один из вариантов такой декомпозиции это когда процессу с номером r соответствуют все слои в интервале

$$[Nr/s + 1, N(r + 1)/s]. \tag{26}$$

Все процессы кроме $r = 0$ имеют дополнительный "нижний" слой фиктивных ячеек для получения данных из последнего слоя процесса с номером $r - 1$. Аналогично все процессы кроме $r = s-1$ где имеют дополнительный "верхний" слой фиктивных ячеек для получения данных из первого слоя процесса с номером $r + 1$.

Алгоритм требует обмена данных в вершинах сетки, в центрах призм и на сторонах граней. Обмен данных в вершинах сетки используется для получения суммы величин хранящихся в одной и той же вершине сетки в разных процессах. С его

помощью вычисляются средние значения компонент электромагнитных полей в вершинах. Обмен данных в призмах используется для передачи величин в фиктивные ячейки. Передаваемые величины включают компоненты электромагнитных полей и кооординаты центров призм. Обмен данных на сторонах граней используется для вычисления потоков. Таким образом, последовательность действий в параллельной программе на каждом шаге по времени следующая: 1. Передача значений электромагнитных полей в ячейках. 2. Вычисление предварительных градиентов и частичных сумм значений электромагнитных полей в вершинах. 3. Передача частичных сумм и получение окончательных значений электромагнитных полей в вершинах. 4. Вычисление окончательных градиентов. 5. Вычисление значений на гранях. 6. Передача значений на гранях. 7. Вычисление значений в ячейках на новом временном слое. Особо отметим что алгоритм не требует передачи градиентов между процессами.

5. Результаты тестовых расчётов

Для проверки свойств предложенной схемы были проведены тестовые расчёты. Призматические сетки строились с помощью сдвигов двумерных треугольных сеток [13, 14]. Точность численного алгоритма оценивалась путём сравнения с аналитическими решениями. Ошибка численного решения в момент времени $t^n = n\tau$ вычислялась по формуле

$$\frac{\left\|\mathbf{V}^n\left(\mathbf{X}^B\right) - \mathbf{V}^{\mathbf{exact}}\left(\mathbf{X}^B, t^n\right)\right\|_{L_2}}{\left\|\mathbf{V}^{\mathbf{exact}}\left(\mathbf{X}^B, t^n\right)\right\|_{L_2}} = \sqrt{\frac{\sum\limits_{i=1}^{P}\left[\sum\limits_{k=1}^{6}\left(\mathbf{V}_k^n\left(\mathbf{X}^{B_i}\right) - \mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)\right)^2\right]\cdot S_{\Delta_i}}{\sum\limits_{i=1}^{P}\left[\sum\limits_{k=1}^{6}\left(\mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)\right)^2\right]\cdot S_{\Delta_i}}} \tag{27}$$

где P — общее количество призм в вычислительной области, $\mathbf{V}_k^n\left(\mathbf{X}^{B_i}\right)$ и $\mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)$ — вычисленные и точные значения электромагнитных полей в центре ячейки i, соответственно.

5.1. Тест 1

Рассмотрим распространение электромагнитной волны в среде с постоянным коэффициентом диэлектрической проницаемости. В этом случае одним из точных решений системы уравнений Максвелла будет:

$$\begin{aligned}
E_1 &= 3\cos\left(\pi x_1\right)\cos\left(\pi x_2\right)\cos\left(\pi x_3 - \sqrt{3}\pi t\right),\\
E_2 &= \sin\left(\pi x_1\right)\sin\left(\pi x_2\right)\cos\left(\pi x_3 - \sqrt{3}\pi t\right),\\
E_3 &= 2\sin\left(\pi x_1\right)\cos\left(\pi x_2\right)\sin\left(\pi x_3 - \sqrt{3}\pi t\right),\\
H_1 &= \tfrac{1}{\sqrt{3}}\cos\left(\pi x_1\right)\cos\left(\pi x_2\right)\cos\left(\pi x_3 - \sqrt{3}\pi t\right),\\
H_2 &= -\tfrac{4}{\sqrt{3}}\cos\left(\pi x_1\right)\cos\left(\pi x_2\right)\cos\left(\pi x_3 - \sqrt{3}\pi t\right),\\
H_3 &= -\tfrac{5}{\sqrt{3}}\sin\left(\pi x_1\right)\cos\left(\pi x_2\right)\sin\left(\pi x_3 - \sqrt{3}\pi t\right).
\end{aligned} \tag{28}$$

Область в которой проводились расчёты представляла из себя единичный куб. Расчёты проводились на последовательности сеток состоящих из 2400, 6552, 18960, 52136 и 151360 призм. Шаги по времени в расчётах брались пропорционально линейным размерам призм. На **Рис. 1** показан пример сетки состоящей из 3784 треуголников которая использовалась для построения сетки из 151360 призм. На **Рис. 2** показано вычисленное распределение

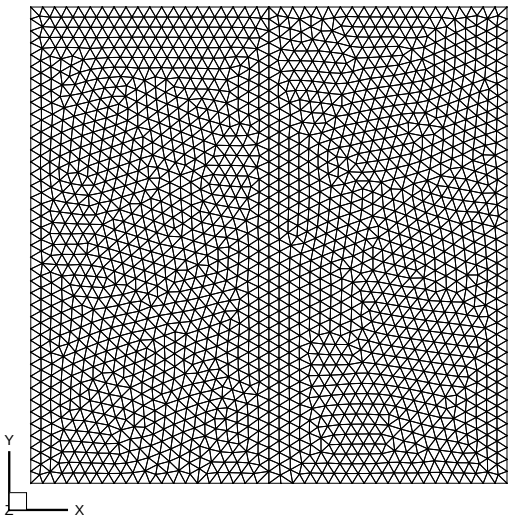


Рис. 1. Двумерная сетка

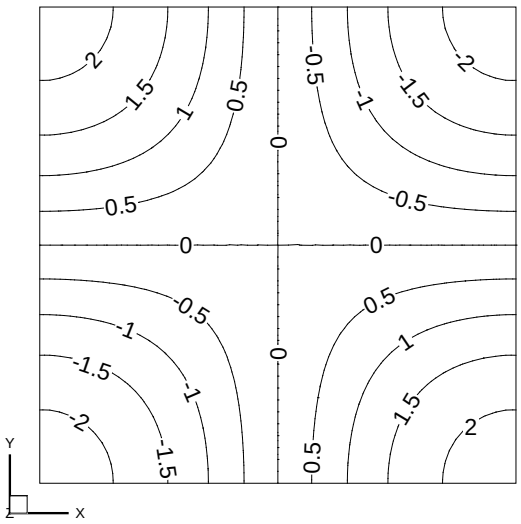


Рис. 2. Распределение E_1

E_1 в момент времени $T = 4.77$ в сечении $x_3 = 0.5$ полученное на сетке из 151360 призм. Эволюция ошибки δ_2 в норме L_2 для различных сеток показана на **Рис. 3**. В таблице **1** приводятся максимальные значения ошибки в норме L_2 для последовательности сеток. Поведение ошибки соответствует второму порядку точности.

Таблица 1. Максимальная ошибка и порядок аппроксимации

Число призм	δ_2	Порядок аппроксимации
2400	0.013659	
6552	0.006670	2.14
18960	0.003100	2.15
52136	0.001497	2.15
151360	0.000709	2.14

5.2. Тест 2

В качестве второго теста рассмотрим распространение гибридной электромагнитной волны в световоде со ступенчатым профилем диэлектрической проницаемости. Разрыв диэлектрической проницаемости ε проходит по криволинейной поверхности $r \equiv \sqrt{x_1^2 + x_2^2} = a$

$$\varepsilon = \varepsilon(r) = \begin{cases} \varepsilon_1 = n_1^2, \text{ где } 0 \le r \le a, \\ \varepsilon_2 = n_2^2, \text{ где } r > a. \end{cases} \tag{29}$$

В этом случае система уравнений Максвелла имеет аналитическое решение [15] которое в цилиндрических координатах записывается в виде

$$
\begin{aligned}
0 \le r \le a: \quad & E_r = \beta \tfrac{a}{u} \left[\tfrac{1-s}{2} J_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s}{2} J_2\left(\tfrac{u}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& E_\theta = -\beta \tfrac{a}{u} \left[\tfrac{1-s}{2} J_0\left(\tfrac{u}{a} r\right) + \tfrac{1+s}{2} J_2\left(\tfrac{u}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z), \\
& E_z = J_1\left(\tfrac{u}{a} r\right) \cdot \cos(\theta) \cos(kt - \beta z), \\
& H_r = k n_1^2 \tfrac{a}{u} \left[\tfrac{1-s_1}{2} J_0\left(\tfrac{u}{a} r\right) + \tfrac{1+s_1}{2} J_2\left(\tfrac{u}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z) \\
& H_\theta = k n_1^2 \tfrac{a}{u} \left[\tfrac{1-s_1}{2} J_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s_1}{2} J_2\left(\tfrac{u}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& H_z = -\tfrac{\beta}{k} s J_1\left(\tfrac{u}{a} r\right) \cdot \sin(\theta) \cos(kt - \beta z),
\end{aligned}
\tag{30}
$$

$$
\begin{aligned}
r > a: \quad & E_r = \beta \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s}{2} K_0\left(\tfrac{w}{a} r\right) - \tfrac{1+s}{2} K_2\left(\tfrac{w}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& E_\theta = -\beta \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s}{2} K_0\left(\tfrac{w}{a} r\right) + \tfrac{1+s}{2} K_2\left(\tfrac{w}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z), \\
& E_z = \tfrac{J_1(u)}{K_1(w)} K_1\left(\tfrac{w}{a} r\right) \cdot \cos(\theta) \cos(kt - \beta z), \\
& H_r = k n_0^2 \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s_0}{2} K_0\left(\tfrac{w}{a} r\right) + \tfrac{1+s_0}{2} K_2\left(\tfrac{w}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z) \\
& H_\theta = k n_0^2 \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s_0}{2} K_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s_0}{2} K_2\left(\tfrac{w}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& H_z = -\tfrac{\beta}{k} s \tfrac{J_1(u)}{K_1(w)} K_1\left(\tfrac{w}{a} r\right) \cdot \sin(\theta) \cos(kt - \beta z),
\end{aligned}
\tag{31}
$$

где J_0 и J_1 функции Бесселя первого рода, K_0 и K_1 функции Бесселя второго рода, $s_0 = s\beta^2/k^2 n_0^2$, $s_1 = s\beta^2/k^2 n_1^2$, $u = a\sqrt{k^2 n_1^2 - \beta^2}$, $w = a\sqrt{k^2 n_2^2 - \beta^2}$,

$$
s = 2\left(\frac{1}{u^2} + \frac{1}{w^2}\right) \left[\frac{J_0(u) - J_2(u)}{u J_1(u)} - \frac{K_0(w) - K_2(w)}{w K_1(w)} \right]^{-1}, \tag{32}
$$

а β находится из дисперсионного соотношения. Тестовые константы $\varepsilon_1 = 2.25$, $\varepsilon_2 = 1.0$, $k = 6.0$, $a = 0.64$, $\beta = 8.402440923258$, $u = 2.063837416842$, $w = 3.764648073438$.

В качестве расчётной области был выбран цилиндр радиуса 1.28 и высотой 0.6. Расчёты проводились на последовательности сеток состоящих из 9352, 27080, 75964, 217000 и 609000 призм. Сетка строилась таким образом чтобы разрыв диэлектрической проницаемости проходил по граням призм. Шаг по времени брался пропорциональным линейным размерам призм. На **Рис. 4** показан пример двумерной треугольной сетки состоящей из 2708 треугольников, которая использовалась для построения сетки из 27080 призм. На **Рис. 5** показано распределение третьей компоненты магнитного поля H_3 в момент времени $T = 4.73$ в сечении $x_3 = 0.3$ полученные на сетке из 27080 призм. На **Рис. 6** показана эволюция ошибки δ_2 в норме L_2 на последовательности из пяти сеток. В таблице **2** приводятся максимальные значения ошибки в норме L_2. Поведение ошибки соответствует второму порядку точности.

5.3. Масштабируемость

Для проверки свойств масштабируемости параллельной реализации были проведены расчёты с использованием последовательной и параллельной реализаций предложенного алгоритма. Расчёты проводились на кластере Новосибирского Государственного Университета. Кластер построен на базе двойных блэйд-серверов HP BL2x220c, имеющих

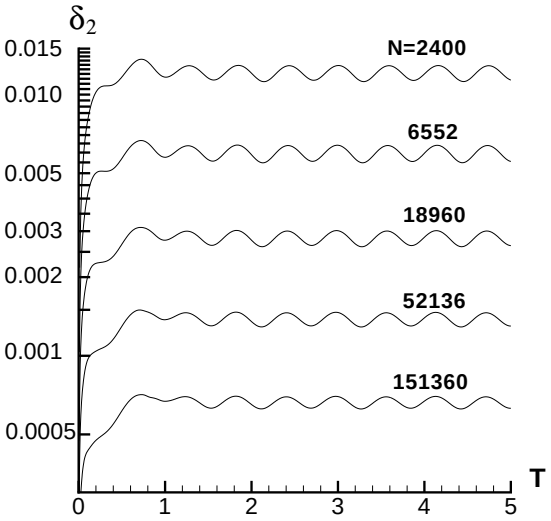


Рис. 3. Поведение ошибки

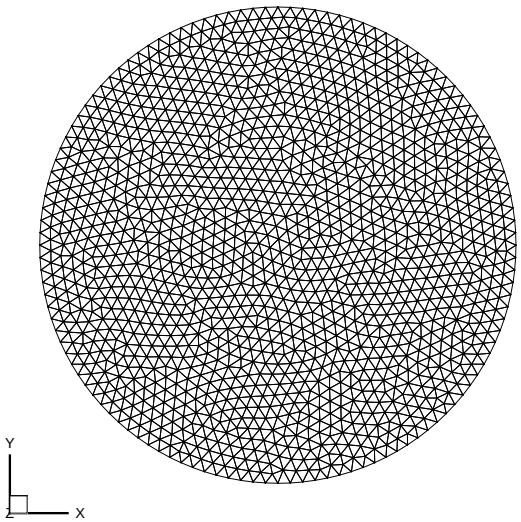


Рис. 4. Двумерная сетка

Таблица 2. Максимальная ошибка и порядок аппроксимации

Число призм	δ_2	Порядок аппроксимации
9352	0.136843	
27080	0.055602	2.54
75964	0.027663	2.29
217000	0.013417	2.21
609000	0.007118	2.12

по 32 ГБ оперативной памяти и по четыре 4х-ядерных процессора Xeon E5540 (2.53 ГГц). В качестве коммуникационной среды использовался InfiniBand. Результаты всех расчётов с использованием параллельной версии и расчёта последовательной версии совпадали. Время счёта было различным. В таблице **3** приводятся затраты времени на проведения расчёта в зависимости от числа процессов для сетки состоящей из 3472000 призм (320 слоёв по 10850 призм). Видно что до момента когда процесс проводит расчёт лишь на 10 слоях достигается ускорение близкое к линейному, что говорит о высокой эффективности и хорошей масштабируемости параллельной реализации.

6. Заключение

В статье был предложен метод конечных объёмов для решения нестационарных уравнений Максвелла с разрывной диэлектрической проницаемостью на призматических сетках. Для использования предложенного метода на высокопроизводительных ЭВМ с распределённой памятью была предложена параллельная реализация вычислительного алгоритма с помощью интерфейса передачи сообщений MPI. Для проверки свойств метода и параллельной реализации были проведены тестовые расчёты. Поведение ошибки для задачи о распространении гибридной электромагнитной волны в световоде со ступенчатым

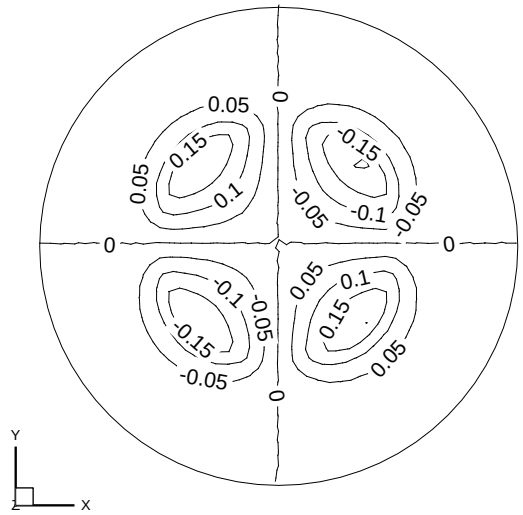


Рис. 5. Распределение H_3

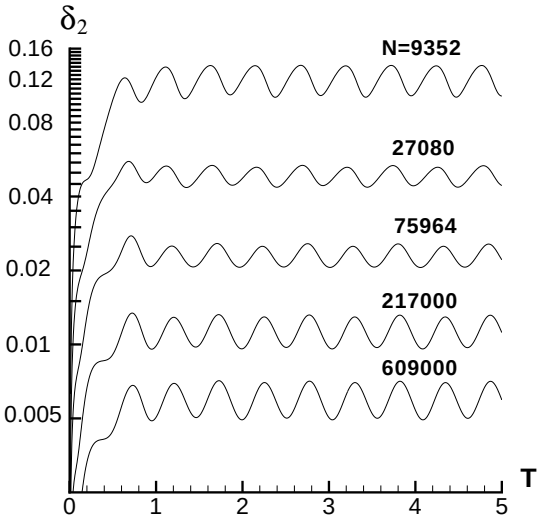


Рис. 6. Поведение ошибки

Таблица 3. Масштабируемость параллельной программы

Число процессов	Время счёта сек.
1	29871
2	14602
4	7379
8	3836
16	1912
32	973
64	518
128	325

профилем диэлектрической проницаемости соответствует второму порядку аппроксимации. Время расчёта демонстрирует практически обратную зависимость для широкого диапазона изменения числа процессов, что свидетельствует о высокой эффективности и хорошей масштабируемости параллельной реализации.

Литература

1. Годунов С.К., Рябенький В.С. Разностные схемы. Введение в теорию. // Москва, Наука, 1977

2. Yee K.S. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media // IEEE Trans. Antennas Propagat. 1966. -Vol. 14. -P. 585-589.

3. Taflove A. Advances in Computational Electrodynamics: the Finite-Difference Time-Domain Method // Boston, Artech House, 1998.

4. Taflove A. and Hagness S.C. Computational Electrodynamics: the Finite-Difference Time-Domain Method // Boston, Artech House, 2000.

5. Sullivan D.M. Electromagnetic Simulation Using the Finite-Difference Time-Domain Method // New York, IEEE, 2000.

6. Hermeline F. Two coupled particle-finite volume methods using Dalaunay-Voronoi meshes for approximation of Vlasov—Poisson and Vlasov — Maxwell equations // J. Comput. Phys. 1993. -Vol. 106. -P. 1-18.

7. Cioni J.-P., Fezoui L., Issautier D. Higher order upwind schemes for solving time domain Maxwell equations // La Recherche Aĕrospatiale 1994. N. 5. -P. 319-328.

8. Cioni J.-P., Fezoui L., H. Steve A parallel time-domain Maxwell solver using upwind schemes and triangular meshes // IMPACT Comput. Sci. Eng Academic Press, Orlando, FL, USA 1994. -Vol. 5. -P. 215-247.

9. Лебедев А.С., Федорук М.П., Штырина О.В. Решение нестационарных уравнений Максвелла для сред с неоднородными свойствами методом конечных объёмов // Вычисл. технологии. 2005. -Том 10, N 2. -С. 60-73.

10. Лебедев А.С., Федорук М.П., Штырина О.В. Конечно-объёмный алгоритм решения нестационарных уравнений Максвелла на неструктурированной сетке // ЖВМ и МФ. 2006. -Том 47, N. 7. -С. 1286-1301.

11. Исмагилов Т.З. Параллельный алгоритм для решения трёхмерных уравнений Максвелла с разрывной диэлектрической проницаемостью // Труды конференции, ПаВТ 2010, Уфа, 2010.

12. Исмагилов Т.З. Параллельный алгоритм для решения трёхмерных уравнений Максвелла с разрывной диэлектрической проницаемостью на тетраэдральных сетках // Вестник УГАТУ, 2010. -Том 14, N 4. -С. 152-159.

13. Geuzaine C. and Remacle J.-F. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities// Int. J. Numer. Meth. Engng. 2009. -Vol. 3. -P. 1-24.

14. J. Schoberl. NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. // Comput.Visual.Sci. 1997. -Vol. 1. -P. 41-52.

15. Okamoto K. Fundamentals of Optical Waveguides // London, Academic Press, 2000.

Реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex-6

И.А. Каляев[1], И.И. Левин[1], Е.А. Семерников[2], А.И. Дордопуло[2]

[1]НИИ многопроцессорных вычислительных систем имени академика А.В. Каляева Южного федерального университета
[2]Южный научный центр Российской академии наук

Реконфигурируемые вычислительные системы на основе ПЛИС обладают высокой реальной производительностью и близким к линейному ростом производительности при увеличении аппаратного ресурса системы. В статье рассматриваются конструктивные особенности, технические характеристики и достигаемые значения реальной производительности для вычислительных модулей реконфигурируемых вычислительных систем на основе ПЛИС семейства Virtex-6. Описан программный комплекс средств разработки параллельных прикладных программ для РВС.

1. Введение

В последние годы наметилась устойчивая тенденция использования новых архитектурных решений для достижения пиковых значений производительности сверхвысокопроизводительных систем. Одним из наиболее распространенных решений является использование программируемых логических интегральных схем (ПЛИС) для выполнения вычислений. На второй строчке списка TOP-500 за ноябрь 2010 года значится суперЭВМ Jaguar - Cray XT5-HE, произведенная фирмой Cray Inc., с пиковой производительностью 2331.00 Тфлопс (в предыдущем списке TOP-500 за июнь 2010 года эта суперЭВМ находилась на первом месте), в составе которой в качестве сопроцессоров используются ПЛИС большой интеграции. В большинстве содержащих ПЛИС вычислительных систем, так же как и в Jaguar - Cray XT5-HE, кристаллы ПЛИС используются как дополнение к микропроцессорам, выполняющее трудно- или неэффективно реализуемые на универсальных микропроцессорах фрагменты вычислений.

Однако, как это показано в [1,2,3], ПЛИС обладают значительно большим вычислительным потенциалом, который в полной мере может быть реализован в реконфигурируемых вычислительных системах (РВС), содержащих множество кристаллов ПЛИС, используемых как основной вычислительный элемент. Успешно развивающаяся более 20 лет в НИИ многопроцессорных вычислительных систем Южного федерального университета (г. Таганрог) концепция построения многопроцессорных вычислительных систем с программируемой архитектурой позволила создать целый ряд РВС 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 используются ПЛИС Xilinx семейства Virtex большой интеграции, соединенные в единый вычислительный ресурс быстрыми каналами передачи данных – LVDS и Rocket GTX.

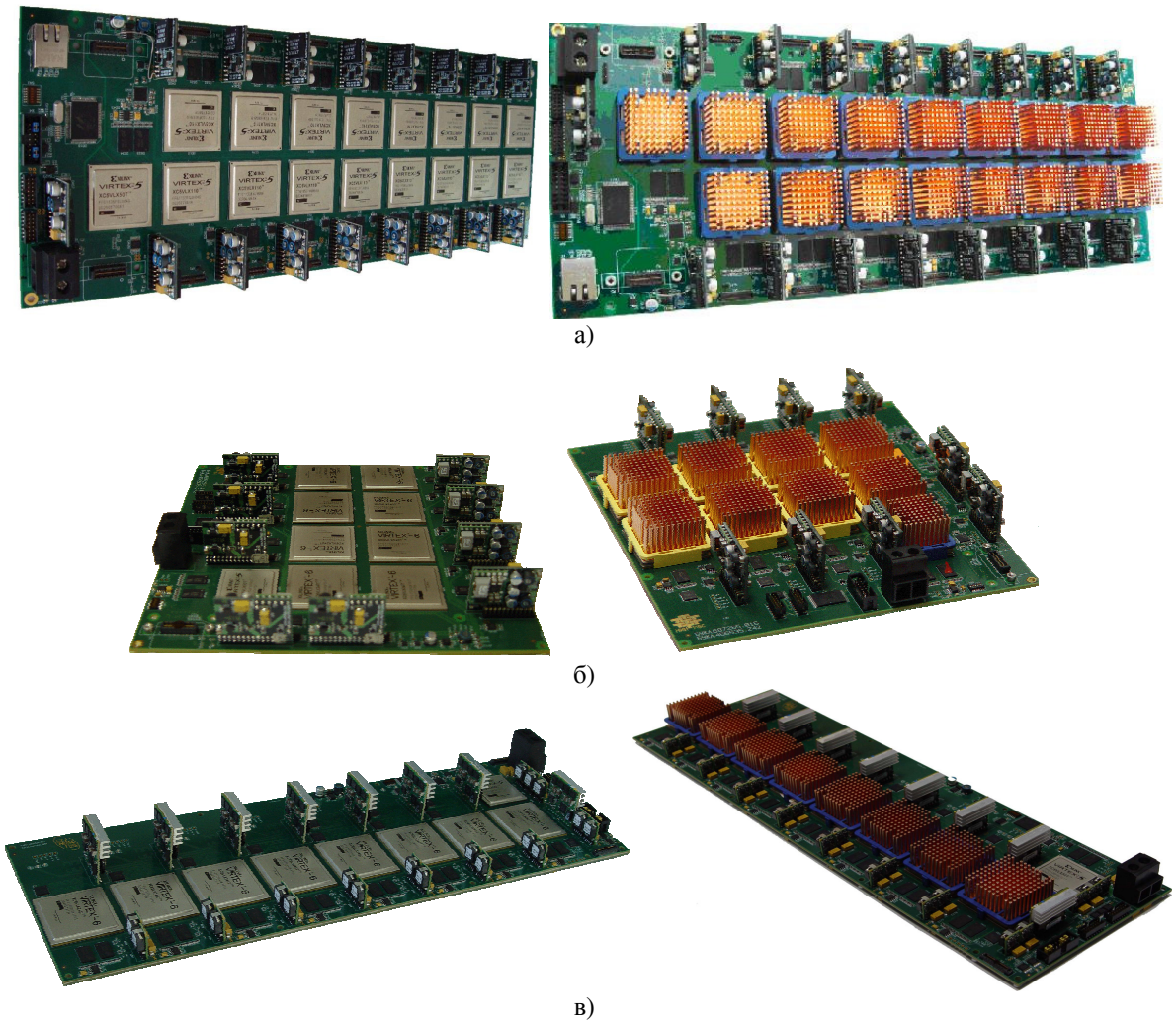
В НИИ многопроцессорных вычислительных систем Южного федерального университета серийно выпускались реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex 5, описанные в [3,4], разработанные по государственному контракту №02.524.12.4002 «Создание семейства высокопроизводительных многопроцессорных вычислительных систем с динамически перестраиваемой архитектурой на основе реконфигурируемой элементной базы и их математического обеспечения для решения вычислительно трудоемких задач», выполняемого по заданию Федерального агентства по науке и инновациям в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2012 годы».

Переход к принципам открытой масштабируемой архитектуры [1] в области разработки РВС положил начало новому семейству вычислительных систем под названием «Орион» и привел к созданию в 2010 году платы модифицированного вычислительного модуля с новой

компоновкой и конструктивными решениями на основе ПЛИС семейства Virtex 5, принципы построения и технические характеристики которого описаны в [5].

2.Вычислительные модули РВС на основе открытой масштабируемой архитектуры

В настоящее время коллектив разработчиков НИИ многопроцессорных вычислительных систем Южного федерального университета приступил к выпуску РВС нового поколения на основе разработанных вычислительных модулей с использованием ПЛИС семейства Virtex-6. Разработаны и созданы платы нового поколения на основе ПЛИС семейства Virtex-6, построенные на основе открытой масштабируемой архитектуры [1] для вычислительных модулей двух перспективных конструктивных исполнений – «Саиф» и «Ригель», названных именами звезд из астрономического созвездия «Орион». Фотографии платы модифицированного вычислительного модуля на основе ПЛИС семейства Virtex 5 и плат нового поколения представлены на рисунке 1а-в.



а) плата вычислительного модуля на основе ПЛИС семейства Virtex 5; б) плата вычислительного модуля «Саиф»; в) плата вычислительного модуля «Ригель»

Рис. 1. Платы вычислительных модулей

В таблице 1 приведены технические характеристики рассматриваемых плат вычислительных модулей. Вычислительные модули на основе этих плат «Орион-5», «Саиф» и «Ригель» имеют высоту 1U, 6U и 1U соответственно и предназначены для установки в стандартную 19"

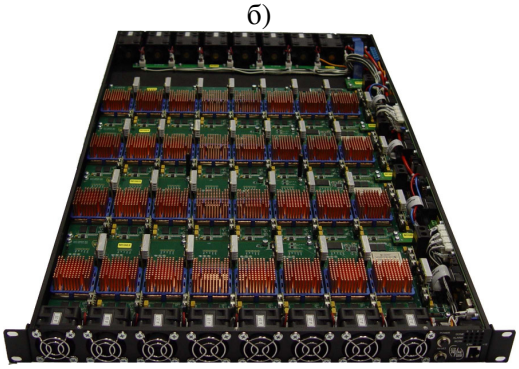
вычислительную стойку, которая является базовым компонентом для создания сверхвысокопроизводительных комплексов на основе ПЛИС. Фотографии вычислительных модулей «Орион-5», «Саиф» и «Ригель» представлены на рисунке 2.

Таблица 1. 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
«Орион-5»	16	Virtex 5	11	LVDS, 1,2	250
«Саиф»	8	Virtex 6	24	Gigabit Ethernet, 1	300
«Ригель»	8	Virtex 6	24	Gigabit Ethernet, 1	300

а)

б)



в)

а) вычислительный модуль «Орион-5»; б) вычислительный модуль «Саиф»; в) вычислительный модуль «Ригель»

Рис. 2. Вычислительные модули нового поколения

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 для вычислительного модуля «Орион-5». 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.

В таблице 2 представлены пиковые производительности рассматриваемых вычислительных модулей и вычислительных стоек на их основе. 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 2.

Таблица 2. 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)
«Орион-5»	1000/340	24	24/8,1
«Саиф»	1600/500	6	9/3
«Ригель»	1600/500	24-36	34,5 – 51,8

В таблице 3 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 3. Производительность вычислительных модулей

Вычислительный модуль	Символьная обработка данных (Топ/с)	Математическая физика, арифметика с плавающей запятой (Тфлопс)
«Орион-5»	116	1/0,34
«Саиф»	199,6	1,6/0,5
«Ригель»	199,6	1,6/0,5

В таблице 4 приведены суммарные скорости передачи данных между кристаллами ПЛИС и блоками распределенной памяти, между ПЛИС в пределах одного вычислительного модуля и других вычислительных модулей.

Таблица 4. Скорость передачи данных

Вычислительный модуль	С блоками распределенной памяти (Гбит/с)	Между ПЛИС вычислительного поля (Тбит/с)	С другими вычислительными модулями (Тбит/с)
«Орион-5»	12,8	1,2	1,2
«Саиф»	12,8	1,0	1,0
«Ригель»	12,8	1,0	1,0

Таким образом, вычислительные модули нового поколения «Саиф» и «Ригель» на основе ПЛИС семейства Virtex 6 открывают перспективы для построения вычислительных систем более высокой производительности при сохранении стоимости системы по сравнению с РВС на основе вычислительного модуля «Орион-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.

3. Программное обеспечение РВС на основе открытой масштабируемой архитектуры

Для вычислительных модулей нового поколения «Саиф» и «Ригель» сохраняется преемственность принципов программирования РВС. Программирование всех рассмотренных вычислительных модулей и систем на их основе осуществляется с помощью единого комплекса системного программного обеспечения, поддерживающего структурно-процедурные методы орга-

низации вычислений. Программирование РВС отличается от программирования суперЭВМ традиционной архитектуры, поскольку включает организацию не только параллельных процессов и потоков данных, но и программирование структуры вычислительной системы в поле логических ячеек ПЛИС. Комплекс программного обеспечения вычислительных модулей предоставляет прикладному программисту следующие возможности:

- программирование как структурной, так и процедурной составляющих на языке высокого уровня без участия высококвалифицированного схемотехника;
- реконфигурация прикладных программ при перераспределении вычислительного ресурса РВС;
- обеспечение совместимости и переносимости проектов между РВС разных архитектур;
- масштабирование прикладной задачи при увеличении ресурса;
- удаленное использование вычислительных ресурсов РВС.

Созданный комплекс программного обеспечения [3] по функциональному назначению разделяется на комплекс средств разработки прикладных программ и комплекс средств управления и администрирования ресурсов РВС.

Средства разработки прикладных программ содержат: транслятор языка ассемблера; транслятор языка программирования РВС высокого уровня COLAMO; интегрированную среду разработки прикладных задач IDE, поддерживающую языки ассемблера и COLAMO; синтезатор масштабируемых параллельно-конвейерных решений, оперирующий библиотекой IP-ядер и интерфейсов.

Язык программирования высокого уровня COLAMO [3,4,5] обеспечивает поддержку создания как структурной, так и процедурной составляющих прикладной программы, реконфигурацию прикладных задач без участия высококвалифицированного схемотехника за счет неявного описания параллелизма и переносимость прикладных задач между РВС разных архитектур за счет использования файла описания архитектуры РВС и элементов библиотеки масштабируемых IP-ядер. Транслятор COLAMO v.2.0 осуществляет трансляцию процедурной составляющей программы, организующей потоки данных, в язык ассемблера Argus v.3.0 и создание структурной составляющей в объектном представлении, которая автоматически передается в среду разработки масштабируемых параллельно-конвейерных процедур Fire!Constructor для синтеза конфигурационных файлов ПЛИС на языке VHDL.

Фундаментальным типом вычислительной структуры в языке COLAMO является конструкция "кадр". Кадром является программно-неделимый компонент, представляющий собой совокупность арифметико-логических команд, выполняемых на различных элементарных процессорах, обладающих распределенной памятью и соединенных между собой в соответствии с информационной структурой алгоритма таким образом, что вычисления производятся с максимально возможными параллелизмом и асинхронностью.

Кадр фактически определяет вычислительную структуру и потоки данных в РВС в данный момент времени. При этом все операции в теле кадра выполняются асинхронно с максимальным параллелизмом, а последовательность смены кадров однозначно определяется программистом.

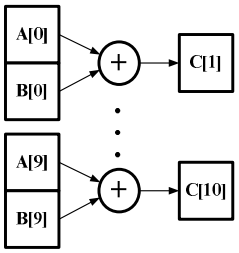
В языке отсутствуют явные формы описания параллелизма. Распараллеливание достигается с помощью объявления типов доступа к переменным и индексации элементов массивов. Для исключения конфликтов одновременного чтения и записи ячеек памяти в пределах текущего кадра используется широко распространенное в языках потока данных правило единственной подстановки: переменная, хранящаяся в памяти, может получить значение в кадре только один раз.

Для обращения к данным используются два основных метода доступа: параллельный доступ (задаваемый типом Vector) и последовательный доступ (задаваемый типом Stream). На рисунке 3 представлены программы, являющиеся граничными примерами извлечения параллелизма, и графы синтезируемых вычислительных структур.

Тип доступа Stream указывает на последовательную обработку элементов одномерного массива, а тип Vector позволяет обрабатывать элементы одномерного массива одновременно.

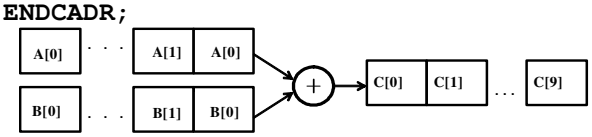
Многомерные массивы состоят из множества измерений, каждое из которых может иметь последовательный или параллельный тип доступа, задаваемый ключевым словом Stream или Vector соответственно.

```
VAR A,B,C: Integer [10 : Vector]
Mem;
VAR I : Number;
CADR SummaVector;
     For I := 0 to 9 do
         C[I] :=A[I]+B[I];
ENDCADR;
```



а)

```
VAR A,B,C : Integer [10 : Stream]
Mem;
VAR I : Number;
CADR SummaStream;
     For I := 0 to 9 do
         C[I] :=A[I]+B[I];
ENDCADR;
```



б)

а) тип доступа Vector; б) тип доступа Stream

Рис. 3. Параллельное и последовательное сложение массивов

Применение неявного описания параллелизма за счет задания типа доступа позволяет достаточно просто управлять степенью распараллеливания программы на уровне описания структур данных и дает возможность программисту максимально просто описывать различные виды параллелизма в достаточно сжатом виде.

Трансляция программы на языке высокого уровня COLAMO состоит в создании схемотехнической конфигурации вычислительной системы (структурной составляющей) и параллельной программы, управляющей потоками данных (потоковой и процедурной составляющих).

Операторы и функции языка (сумматоры, умножители, функции сравнения, тригонометрические функции и др.), используемые в тексте параллельной программы, имеют готовые схемотехнические решения. Данные решения разрабатываются специалистами-схемотехниками в интегрированной среде разработки цифровых устройств ISE фирмы XILINX или с ней совместимых и включаются в библиотеку транслятора языка COLAMO и библиотеку стандартных примитивов среды Fire!Constructor.

В процессе работы транслятора языка COLAMO формируется информационный граф прикладной задачи из текста параллельной программы, где операторы и функции языка по определённым правилам заменяются соответствующими блоками или группами блоков из библиотеки стандартных примитивов.

Синтезированный вычислительный граф задачи передается в среду разработки вычислительных структур Fire!Constructor для укладки на множество ПЛИС РВС и обеспечения синхронизации между ПЛИС [5]. Одной из задач среды является формирование разбиения информационного графа прикладной задачи на непересекающиеся подграфы, каждый из которых будет структурно реализован в кристаллах ПЛИС выбранной РВС.

Процесс синтеза результата разбиения информационных графов прикладных задач состоит из следующих этапов:

- решения задачи разбиения (компоновки) узлов информационного графа прикладной задачи на непересекающиеся подграфы, каждый из которых будет размещён в соответствующем БМ;

- решения задачи размещения и трассировки для узлов информационного графа в каждом БМ в отдельности и задачи трассировки связей между БМ;

- синтеза файлов VHDL-описаний и файлов временных и топологических ограничений для каждой ПЛИС, каждого БМ выбранной РВС.

Среда Fire!Constructor упрощает создание масштабируемых структурных решений и сокращает время разработки за счет автоматизированного выполнения следующих трудоемких процедур:

- согласования входов и выходов совместно работающих ПЛИС (ucf-файлов);

- автоматической синхронизации информационных потоков при размещении функциональных устройств в едином вычислительном контуре, расположенном в различных кристаллах ПЛИС;

- автоматического обеспечения сбалансированного размещения функциональных устройств по различным ПЛИС.

Технология создания прикладных программ для РВС и общая взаимосвязь транслятора языка COLAMO, среды Fire!Constructor и синтезатора конфигурации ПЛИС в рамках комплекса системного программного обеспечения при создании многокристального схемотехнического решения для РВС представлена на рисунке 4.

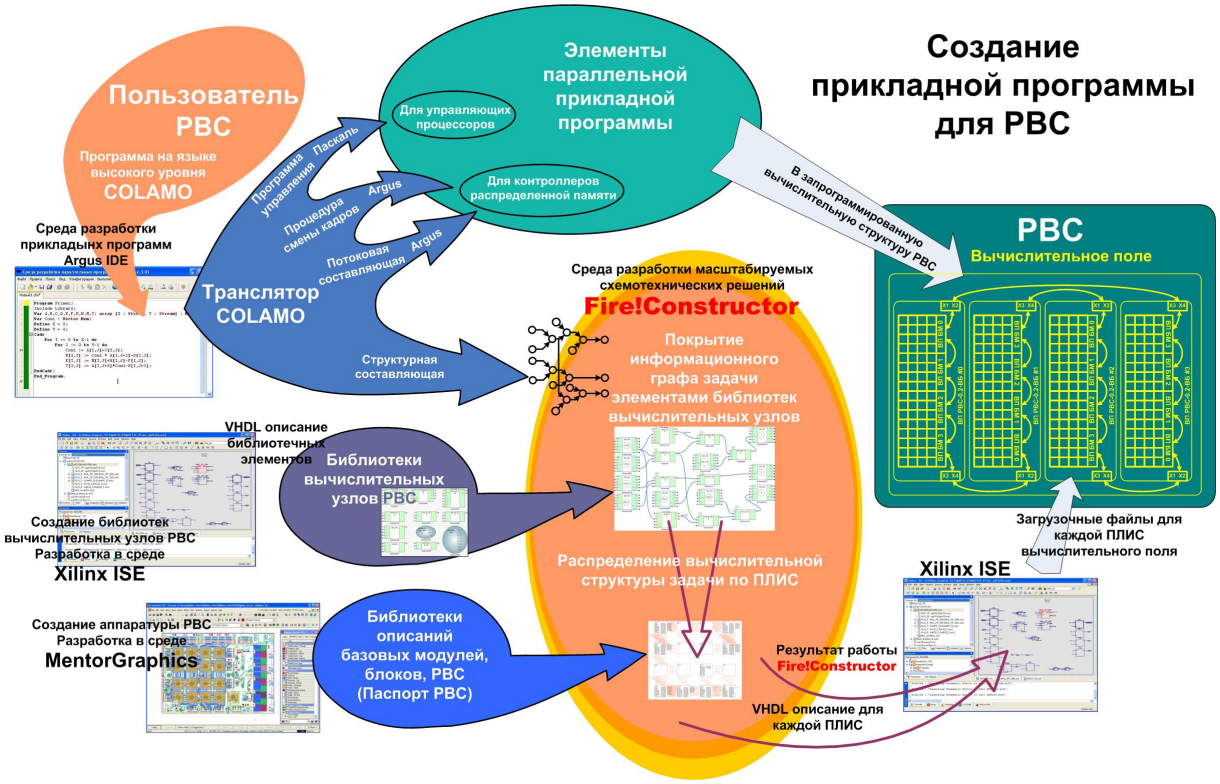


Рис. 4. Технология создания прикладных программ для РВС

Такой подход к программированию реконфигурируемых вычислительных систем позволяет освободить программиста от построения графа задачи в виде функциональных библиотек в среде Fire!Constructor и синхронизации потоков данных в РВС, сократив время создания параллельных программ для РВС в 3-10 раз, и исключить участие специалиста-схемотехника при разработке параллельных прикладных программ.

Язык структурно-процедурного программирования Argus представляет собой низкоуровневый язык (ассемблер), предназначенный для описания процедурной составляющей прикладной параллельной программы РВС [3,4]. Программа на языке Argus организует потоки данных на уровне команд контроллеров распределенной памяти, обеспечивая их синхронизацию.

Интегрированная среда разработки Argus IDE предназначена для интерактивной разработки параллельных программ на языках высокого уровня COLAMO и языке ассемблера Argus в едином языковом пространстве. Среда Argus IDE, объединяя в своем составе трансляторы языков COLAMO и Argus, обеспечивает эффективную разработку масштабируемых параллельных программ для РВС.

Созданное параллельное решение прикладной задачи в виде загрузочного модуля РВС с помощью драйвера загружается в вычислительный модуль РВС. Драйвер вычислительных модулей обеспечивает программную поддержку функций непосредственного доступа к высокоскоростному аппаратному интерфейсу, поддерживающему пакетные режимы работы и обеспечивающему механизмы прямого доступа к физической памяти управляющего компьютера.

Для удаленного доступа и управления вычислительными ресурсами РВС разработана система удаленного доступа, которая состоит из сервера, обрабатывающего удаленные заявки на использование вычислительных ресурсов и поддерживающего очередь заявок, и клиента, формирующего заявки на основе команд пользователя. К функциям системы удаленного доступа относятся функции включения, выключения, остановки и запуска как отдельных вычислительных модулей, так и стоек и РВС в целом.

Созданный комплекс программного обеспечения позволяет создавать эффективные прикладные программы для РВС при решении задач различных предметных областей, обеспечивает удобство программирования и сокращает время разработки прикладного решения в 3-5 раз, обеспечивая при этом автоматизированный перенос структурного решения с одной архитектуры РВС на другую.

4. Заключение

РВС являются перспективным направлением развития высокопроизводительной вычислительной техники, которое, в отличие от кластерных суперЭВМ, предоставляет пользователю возможность создавать в базовой архитектуре виртуальные специализированные вычислители, структура которых адекватна структуре решаемой задачи. Это, в свою очередь, обеспечивает высокую эффективность вычислений и близкий к линейному рост производительности при наращивании вычислительного ресурса.

Следует отметить, что переход на новую компоновку модулей, позволившую сосредоточить в пределах вычислительных модулей «Орион-5» и «Ригель» высотой 1U мощный вычислительный ресурс на основе ПЛИС, обеспечивает удельную производительность РВС на уровне лучших мировых показателей для суперЭВМ с кластерной архитектурой.

Анализ тенденции развития ПЛИС и построения реконфигурируемых вычислительных систем на их основе показывает, что РВС обладают высокой удельной производительностью при решении задач различных классов и могут служить основным средством для создания суперЭВМ нового поколения.

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.

Автоматическое распараллеливание последовательных программ для гибридной системы с ускорителем на основе потока данных

Арк.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский

Институт проблем проектирования в микроэлектронике РАН

Рассматривается модель вычислений на основе потока данных и ее реализация в виде многоядерного сопроцессора - ускорителя вычислений. Предлагается метод автоматической трансляции последовательных программ для их выполнения на гибридной системе. Метод основан на построении полного графа алгоритма и эффективен для программ линейного класса.

1. Введение

Проблема автоматического распараллеливания программ хорошо известна. Но редко обращают внимание на то, что для различных целевых архитектур приходится решать ее поразному. Некоторых результатов удается достичь там, где целевая модель вычислений мало отличается от обычной модели исходной последовательной программы. Это, прежде всего, OpenMP, где параллелизм выражается в виде дополнений к основному последовательному коду в виде директив компилятору. Аналогичным свойством обладает модель DVM [1], для которой также есть некоторая перспектива автоматизации распараллеливания. Но уже для модели распределенных вычислений SPMD (Single Program применяется к Multiple Data), к которой относятся системы, основанные на MPI, а также PGAS-языки: UPC, CAF и т.д., разработчики распараллеливающих компиляторов сталкиваются со значительными трудностями. Похожая ситуация складывается и для модели вычислений, основанной на графических ускорителях GPU, на которых удается достичь высокой производительности. Так называемые «распараллеливающие» компиляторы для CUDA от Portland Group Inc. [2] – не что иное, как попытка прикрыть сложности и замысловатости гибридной модели программирования для GPU языком директив над обычным Фортраном или Си. Некоторый оптимизм вселяет недавняя работа [3], где для ускорителей на базе GPU автоматически распараллеливаются последовательные программы того же класса, что и в нашей работе.

Основная причина трудностей автоматизации параллельного программирования для названных моделей вычислений, на наш взгляд, коренится в самих этих моделях. Они изначально требуют *полного контроля над параллелизмом со стороны программиста.* Это, конечно, создает дополнительные возможности для повышения эффективности, но приходится платить высокую цену - увеличение трудоемкости. Причем приходится тратить силы не столько на алгоритмы или математику, сколько на оптимизацию «раскладки» уже существующих алгоритмов на имеющиеся аппаратные ресурсы: перегруппировку вычислений, агрегацию данных и на оптимальное распределение этих групп и агрегаций по пространству и времени.

Нами предлагается другой путь, когда основной контроль над параллелизмом отдается аппаратуре. А программист-математик занимается только логическими взаимосвязями внутри алгоритма, определяя, что чему логически предшествует или что без чего не может быть вычислено. Он разбивает программу вычислений на небольшие фрагменты (будем называть их узлами), выражая некоторым образом связи по данным между ними. Каждый узел принимает какие-то данные от других узлов и формирует новые данные для других. При этом инициатива взаимодействия между получателем и отправителем принадлежит отправителю. Иначе говоря, программа каждого узла должна уметь «вычислять адреса» своих получателей. Сам узел-получатель становится активным лишь тогда, когда к нему поступает вся нужная информация. Ниже модель вычислений, основанная на этих принципах, излагается более подробно.

Трудности для программиста оборачиваются и трудностями для компилятора, точнее для разработчика компилятора. Поэтому модель вычислений, облегчающая ручное параллельное программирование, открывает больше перспектив и для автоматизации распараллеливания.

Говоря о простоте программирования, мы имеем в виду абстрактную простоту решаемой задачи. Программирование на языке DFL, к сожалению, пока нельзя назвать простым: во-первых, оно слишком отличается от традиционного и потому требует переучивания, а во-вторых, недостаточно развиты инструменты, облегчающую написание и отладку программ. Но для задачи автоматизации распараллеливания эти проблемы как раз не важны, и это делает задачу автоматической трансляции последовательных программ в DFL особенно актуальной.

Различие в моделях целевых архитектур порождает и существенные различия в способах анализа и компиляции программ. Существующие компиляторы распараллеливают циклы, пытаясь выяснить про каждый цикл, может ли он выполняться как параллельный. Для этого проверяется, нет ли зависимостей между операциями чтения и записи, мешающих параллельному выполнению цикла. Цикл не может выполняться как параллельный, если на разных его итерациях производится доступ к одному и тому же элементу массива, и хотя бы один из этих доступов является записью. При этом важно обнаружить сам факт наличия или отсутствия зависимости, а какие конкретно операции пишут и читают, и в какие элементы – не важно. Поэтому, такой анализ имеет право быть неполным [4, гл.11]. Более развитые компиляторы пытаются преобразовать циклы так, чтобы хотя бы один (желательно, внешний) цикл стал параллельным.

Наш подход основан на переводе программы в модель вычислений, в которой неявный параллелизм извлекается динамически по принципу готовности данных. Для перевода в такую модель не требуется дополнительной информации, кроме той, которую хороший анализатор способен извлечь из исходной программы – точное описание графа потоковых (истинных) зависимостей между экземплярами операций с памятью. Построение этого описания – наиболее сложная часть процесса трансляции в DFL, который, в сущности, является одной из форм представления этого графа. Выполнение этого описания требует известных накладных расходов, на минимизацию которых и нацелена архитектура предлагаемого вычислителя.

Статья имеет следующую структуру. В разделе 2 приводится описание гибридной модели вычислений, в которой обычная последовательная программа взаимодействует с программой на языке потока данных. Потоковая часть во многом подобна классической модели dataflow [5], но имеет и важное отличие, состоящее в наличие контекста, который полностью контролируется (задается) программистом. Описываются основные черты языка программирования DFL, соответствующего этой модели вычислений. В разделе 3 описывается архитектура вычислителя, способного выполнять программу на DFL. В разделе 4 обсуждается трансляция последовательных программ в эту модель вычислений. При этом накладываются определенные ограничения на допустимые исходные программы, и вводится понятие графа алгоритма, описание которого может быть автоматически построено для любой допустимой программы. По описанию графа уже легко строится программа на DFL. В разделе 5 работа компилятора демонстрируется на примере программы решения двумерной задачи Пуассона методом Якоби.

2. Модель вычислений

Вычисление, рассматриваемое как целое, разбивается на *вычислительные элементы*, каждый из которых является экземпляром одного из *типовых узлов*. Между элементами возникают связи по данным: одни элементы вычисляют некоторые данные, другие эти данные используют. Интерфейс и поведение типового узла задаются в его описании.

Конечный набор описаний типовых узлов и является программой на языке DFL. Описание типового узла состоит из *заголовка* и *программы узла*. В заголовке указываются *входы* (имя и тип каждого) и *контекст*, задаваемый как список имен полей целого типа. В программах узлов для записи вычислений используется подмножество языка Паскаль. В качестве аргументов могут использоваться только значения входов, полей контекста, а также *констант*, а результатом работы может быть только посылка некоторого количества данных на входы других узлов (или хост-процессор).

Данные между узлами передаются в виде *токенов*. Оператор посылки токена имеет вид:

$$v \rightarrow N.p\{e_1,\ldots,e_k\}$$

где v – посылаемое значение, N – имя узла, $\underline{p}$ – имя входа, e_i – целые выражения, в совокупности задающие контекст целевого виртуального узла. Читается: послать значение v на вход p виртуального узла $N\{e_1,\ldots,e_k\}$. Или короче: послать v на $N.p\{e_1,\ldots,e_k\}$. Виртуальным узлом мы называем экземпляр типового узла с конкретным набором значений полей контекста. Несколько токенов, направленных на разные входы одного и того же виртуального узла, должны встретиться, и когда соберутся токены для всех входов, образуется пакет – задание на выполнение программы узла. Созданные пакеты друг от друга не зависят и могут выполняться в произвольном порядке или в параллель. Для каждого пакета выполняется программа соответствующего типового узла, в результате чего могут появиться новые токены.

Для иллюстрации модели вычислений приведем простой пример: вычисление массива частичных сумм произведений двух массивов. На Рис. 1 слева приводится спецификация алгоритма в виде фрагмента на Фортране, а справа – соответствующий фрагмент на DFL, состоящий из одного трехвходового узла P.

Здесь параллелизма нет: узел P{i+1} не может работать, пока узел P{i} не отправил ему свою частичную сумму s. Но можно делать умножения параллельно, если выделить его в отдельный узел

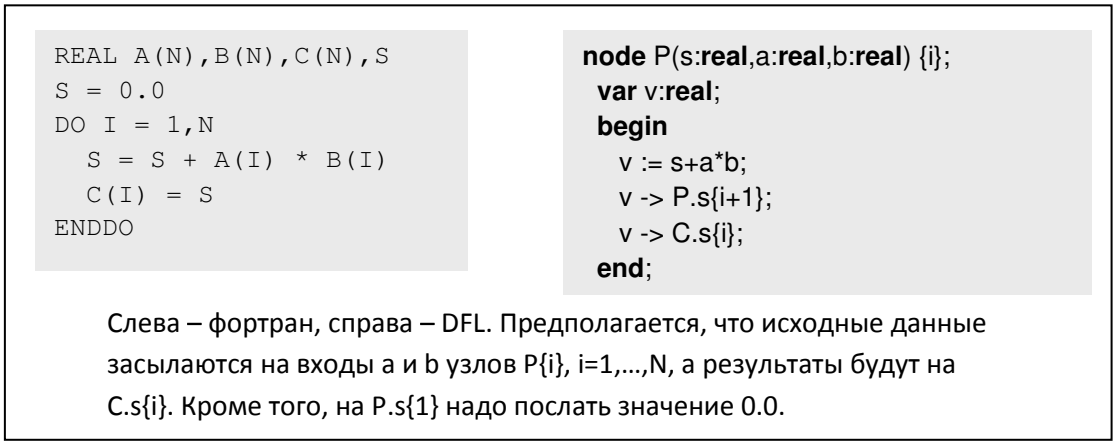


Рис. 1. Фрагмент программы вычисления частичных сумм произведений двух векторов

Параллельная система работает в паре с обычным процессором, будем называть его хост-процессором, или просто хостом. На нем в рамках обычной ОС запускается последовательная программа, в которой некоторые сложные счетные подзадачи решаются на параллельной подсистеме. Хост-программа посылает исходные данные для подзадачи в виде токенов на параллельную подсистему, дожидается приема результирующих токенов и продолжает последовательную работу. Во время ожидания хост-процессор может заниматься счетом других подзадач, если они есть. Параллельная программа в подсистему должна быть загружена заранее.

На уровне DFL прием токенов из хоста не отличается от приема любых других токенов: в токене должен быть задан код принимающего узла, контекст и значение. Токены-результаты, поступающие на хост-процессор, немного отличаются от стандартных: их целевой узел имеет имя с суффиксом _out, единственный вход и пустую программу.

В хост-программе посылка входных токенов выражается специальным оператором типа

SEND X,Y

у которого аргумент – список имен массивов (или переменных). Для каждого элемента заданного массива X формируется токен со значением в качестве данного, именем узла X_in и индексами элемента в качестве контекста. Результаты принимаются оператором типа

RECEIVE A(N),B(N+1),N

в котором тоже задан список массивов или простых переменных, но при имени массива обязательно должно быть указано в скобках выражение – количество принимаемых токенов. В данном случае это количество N также принимается отдельным токеном. Для каждого элемента списка X(m) в параллельной подсистеме должны быть созданы m токенов, направленных на узел X_out с некоторым контекстом {i,j,…}. Каждый такой токен передается на хост, где его значение записывается в элемент X(i,j,…) и подсчитывается их количество. Число размерно-

стей должно совпадать с числом полей контекста. Массив X считается принятым после поступления ровно *m* таких токенов. Токены-результаты для разных массивов и простых переменных могут приходить в любом порядке. Выполнение оператора RECEIVE завершается, когда все указанные в нем массивы и переменные будут приняты. Описанные правила взаимодействия хост-процессора с параллельной потоковой подсистемой образуют основу гибридной модели вычислений.

3. Архитектура сопроцессора

Архитектура вычислителя для данной модели вычислений была предложена в [6] и получила свое дальнейшее развитие в [7]. Здесь мы применяем его как сопроцессор для ускорения счетных подзадач. Вычислитель состоит из некоторого числа ядер K, соединенных коммутатором (Рис. 2). К этому же коммутатору подключен блок ввода-вывода. Ядро состоит из модуля ассоциативной памяти (МАП) и исполнительного устройства (ИУ). Токены через коммуникационную сеть приходят в МАП и могут там находиться в ожидании других токенов. МАП состоит из ассоциативной памяти (АП) ключей, где хранятся ключи со ссылками на наборы токенов, и памяти для наборов токенов (ПТ). Ключ токена состоит из имени (кода) узла и контекста. В аппаратной реализации он упаковывается в одно 64-битное слово. Входящий токен сравнивается в АП по ключу со всеми присутствующими ключами, и если обнаруживается совпадение, то его данное вносится в соответствующий хранимый набор на место указанного в нем входа. Если набор оказывается полным, выполняется процедура образования пакета. Если совпадения ключа входящего токена с хранимым ключом не обнаружено, в АП заносится новый ключ и в ПТ отводится место для нового набора, куда помещается сам токен. Если узел одновходовой, то сразу выполняется процедура образования пакета.

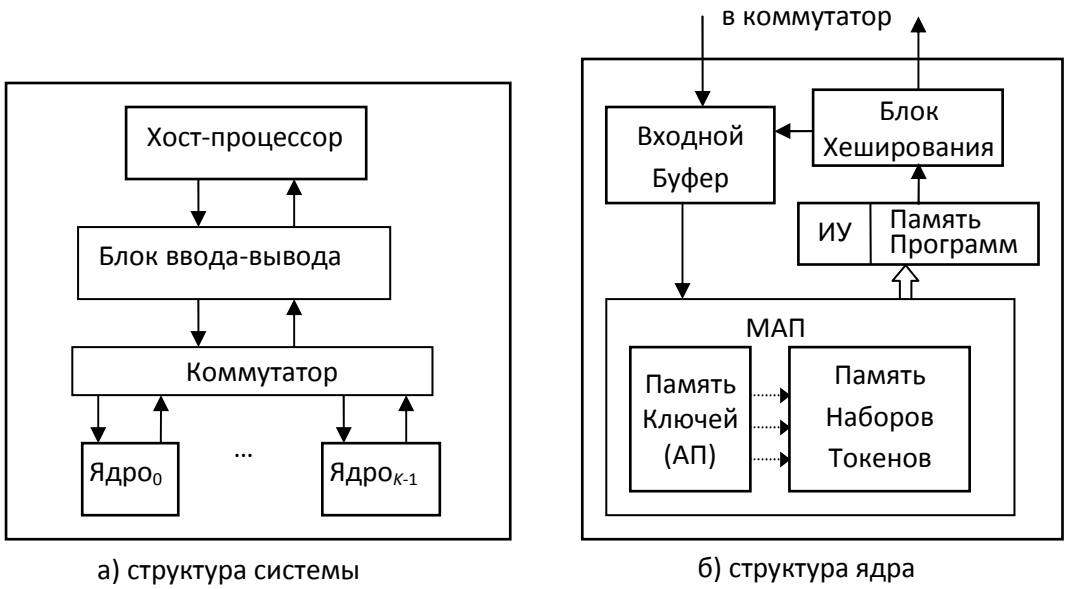


Рис. 2. Архитектура потокового вычислителя

Токены с общим ключом должны оказаться в одном и том же МАП. Для этого номер целевого МАП формируется при создании токена посредством вычисления определенной функции распределения, зависящей только от ключа. Это принципиальный момент – этим обеспечивается возможность эффективной и распределенной реализации ассоциативного поиска. Стандартная функция распределения обеспечивает достаточно хорошее перемешивание, но разрушает потенциальную локальность. Пользователь может указать или задать свою функцию распределения. Если при хорошем балансе нагрузки удается обеспечить локальность (чтобы токены чаще посылались в свое ядро), то сокращается нагрузка на коммутатор и, как следствие, повышается общая производительность системы. Перед выходом на коммутатор номер вычисленного

МАП сравнивается с номером данного ядра и в случае совпадения токен направляется во входной буфер, минуя коммутатор.

Сформированный пакет, состоящий из токенов набора и их общего ключа, поступает через буфер пакетов в ИУ. Обычно на каждый вход приходит по одному токену. После формирования пакета набор токенов и ключ удаляются из МАП. Пока набор токенов не полон, он продолжает храниться в МАП в памяти наборов токенов, а общий ключ токенов – в памяти ключей.

Для каждого поступающего пакета в ИУ выполняется программа соответствующего узла. При этом могут образоваться новые токены, которые через буфер и блок хеширования (БХ) поступают на вход коммуникационной сети.

В памяти программ каждого ИУ хранятся копии программ и констант. Они загружаются туда из хост-процессора перед началом работы (на схеме соответствующий канал не показан).

4. Трансляция последовательных программ в DFL

В языке DFL программа узла, вычисляющая значение, должна уметь разослать его всем узлам, которые в нем нуждаются. Поэтому для перевода заданной последовательной программы P в язык DFL необходимо для каждой операции записи (то есть присваивания элементу массива или простой переменной) определить все операции чтения (в правых частях операторов присваивания), в которых читается записываемое здесь значение. Фактически нужно построить описание графа алгоритма [8, гл.6], которое в параметрическом виде описывает любой граф вычисления программы, определяемый по прогону программы при конкретных входных данных. Граф вычисления состоит из вершин двух типов. Вершины первого типа (записи) соответствуют исполнениям операций записи в память, а второго типа (чтения) операциям чтения из памяти. Из вершины-записи идет дуга в вершину-чтение, если эта операция чтения считывает из памяти именно то значение, которое было записано данной операцией записи.

Отдельную операцию записи или чтения можно идентифицировать парой $(M,\boldsymbol{I})$, где M – место в программе, где эта операция находится, а $\boldsymbol{I}$ – набор значений параметров внешних (для этого места) циклов. Используя эти пары как идентификаторы вершин, определим отображение

$$\boldsymbol{F}: (R, \boldsymbol{I}_R) \rightarrow (W, \boldsymbol{J}_W) \qquad (1)$$

которое по идентификатору чтения $(R, \boldsymbol{I}_R)$ выдает идентификатор записи $(W, \boldsymbol{J}_W)$, записавшей читаемое значение (или знак, обозначающий, что записей не было, а читается исходное значение).

Но для перевода в DFL нам требуется обратное: для каждой операции записи найти все (их может быть несколько или ни одной) операции чтения, которые именно это значение читают, то есть нужно многозначное отображение

$$\boldsymbol{G}: (W, \boldsymbol{J}_W) \text{ -> } \{ (R, \boldsymbol{I}_R) \} \qquad (2)$$

которая по идентификатору записи $(W,\boldsymbol{J}_W)$ выдает множество идентификаторов чтений $\{(R,\boldsymbol{I}_R)\}$, читающих записываемое значение.

К сожалению, выразить в явной форме эти отображения удается далеко не всегда. Но есть хорошо определенный класс программ, для которых это возможно: это *линейные* программы [8, гл.6]. Они могут содержать следующие конструкции:

- операторы присваивания с линейными индексными выражениями во всех обращениях к массивам;
- правильно вложенные циклы do-enddo с единичным шагом, линейными выражениями для верхней и нижней границы и без преждевременных выходов из цикла;
- правильно вложенные условные операторы if-then-else-endif с условиями вида e=0 или e>0, где e – линейное выражение.

Выражения называются *линейными*, если они являются линейными формами с целыми коэффициентами, где в качестве переменных используются лишь только параметры внешних циклов и некоторые фиксированные параметры, которые определяются вне участка, подлежащего анализу и преобразованию, и в нем не изменяются.

Пока мы рассматриваем только строго линейные программы. Более того, счетный участок целиком должен быть представлен подпрограммой, принимающей входные данные и возвращающей результаты через параметры. Впоследствии некоторые из ограничений могут быть

сняты. Предполагаемая общая схема обработки исходной программы такая: транслятор сканирует программу, ища участки, удовлетворяющие условиям допустимости, и переводит их в DFL. А на их место вставляется новый код, который передает данные на входные узлы и принимает результаты с выходных. По желанию программист может отметить сам нужные счетные участки директивами или ограничить область поиска таких участков. Сама внешняя программа выполняется на хост-процессоре, а ее преобразованная параллельная часть – на параллельном сопроцессоре. Таким образом, в худшем случае, наш транслятор ничего не сделает, и программа останется последовательной. Но на практике вычислительные ядра очень многих реальных программ удовлетворяют условиям допустимости.

В работе [9] введено понятие дерева выбора, с помощью которого возможно описать граф потоковых зависимостей любой линейной программы. Там же представлен алгоритм получения такого графа. Имея граф, нетрудно произвести трансляцию в DFL. Каждый оператор присваивания переходит в отдельный узел. Элементы массивов в правой части превращаются во входы. Контекстом узла становится список переменных всех внешних циклов данного оператора. В начало узла помещается сам оператор присваивания, где элементы массивов правой части заменены именами соответствующих входов, а левая часть – локальной переменной. Дерево выбора, описывающее множество получателей данной операции записи, превращается в составной оператор, который в зависимости от заданных условий выполняет рассылку вычисленного значения на входы других узлов.

5. Пример трансляции с Фортрана в DFL

На Рис. 3 вверху слева показан пример исходной линейной подпрограммы на Фортране, которая решает двумерную задачу Пуассона простым итерационным методом Якоби. Начальное приближение поступает на вход задачи как двумерный массив A(0:L,0:L). В нем же выполняются итерации с использованием вспомогательного массива B(0:L,0:L) и возвращается результат. Крайние элементы не пересчитываются и используются как постоянные граничные условия. Остальные фрагменты (серые) – результат трансляции. Слева в центре – вариант подпрограммы, которым подменяется исходная. Здесь язык Фортран расширен операторами LOADCONST, которыми заносятся постоянные значения L и M в память констант всех ядер, а также SEND и RECEIVE, которые, соответственно, посылают исходные данные в параллельную подсистему и принимают ее результаты.

На Рис. 3 внизу и справа показан перевод тела этой подпрограммы в DFL. В подпрограмме два оператора присваивания, назовем их B1 и A1, каждый вложен в тройной цикл. Соответственно, узлы B1 и A1 имеет контекст из трех полей. Тело каждого узла содержит образ самого оператора присваивания, за которым следуют операторы рассылки вычисленного значения. Узел A_in принимает входные элементы и рассылает их куда требуется, а через узел A_out происходит возврат результатов.

```
SUBROUTINE JAC(A,L,M)
REAL A(0:L,0:L),B(0:L,0:L),X,Y
IF M.GE.1 THEN
IF L.GE.2 THEN
DO IT = 1,M
 DO J = 1,L-1
  DO I = 1,L-1
   B(I,J)=(A(I-1,J)+A(I,J-1)+
          A(I+1,J)+A(I,J+1))/4
  ENDDO
 ENDDO
 DO  J = 1, L-1
  DO I = 1, L-1
   A(I, J) =  B(I,J)
  ENDDO
 ENDDO
ENDDO
ENDIF
ENDIF
END
```

```
SUBROUTINE JAC(A,L,M)
REAL(8) A(N,N)
LOADCONST L,M
SEND A
RECEIVE A((N+1)*(N+1))
END
```

```
vconst int L,M;

node B_1(A_1:real,A_2:real,A_3:real,
                A_4:real) {IT,J,I};
   (((A_1+A_2)+A_3)+A_4)/4 → A_1.B_1{IT,J,I};

node A_1(B_1:real) F3{IT,J,I};
 if IT=M then
   B_1 → A_out{I,J}
 else begin
   if J<>(L-1) then
    B_1 → B_1.A_2{IT+1,J+1,I};
   if I>=2 then
    B_1 → B_1.A_3{IT+1,J,I-1};
   if J>=2 then
    B_1 → B_1.A_4{IT+1,J-1,I};
   if I<>(L-1) then
    B_1 → B_1.A_1{IT+1,J,I+1};
 end;
```

```
node A_in(A:real32) {I_0,I_1};
 var IT:int;
 begin
  if I_1=L then begin
   A → A_out{I_0,L};
   if I_0>=1 then
    if I_0<>L then
     for IT := 1 to M do
      A → B_1.A_4{IT,L-1,I_0};
  end
  else
   if L=1 then A → A_out{I_0,I_1}
   else
    if M>=1 then begin
     if I_1=0 then A → A_out{I_0,0};
     if I_0=L then
      if I_1>=1 then begin
       for IT := 1 to M do
        A → B_1.A_3{IT,I_1,L-1};
       A → A_out{L,I_1};
      end else begin
       if I_0<>0 then
        if I_1<>L-1 then
         A → B_1.A_2{1,I_1+1,I_0};
       if I_1=0 then
        for IT := 2 to M do
         if I_0>=1 then
          A → B_1.A_2{IT,1,I_0}
        else begin
         if I_0=0 then begin
          for IT := 2 to M do
           A → B_1.A_1{IT,I_1,1};
          A → A_out{0,I_1};
         end else begin
          if I_1>=2 then
           A → B_1.A_4{1,I_1-1,I_0};
          if I_0>=2 then
           A → B_1.A_3{1,I_1,I_0-1};
         end;
         if I_0<>L-1 then
          A → B_1.A_1{1,I_1,I_0+1};
        end;
       end
      end
     else
      A → A_out{I_0,I_1};
 end;

node A_out(A:real) {I_0,I_1};
```

Рис. 3. Пример трансляции подпрограммы метода Якоби

Следует отметить, что в основной части полученной программы нет циклов. Есть несколько циклов во входном узле A_in, каждый из которых рассылает одно значение сразу на много узлов. Но исходные циклы исчезли. Вместо них в хост-процессоре создается поток элементов входных массивов, которым активируется сразу множество узлов. Одни узлы через данные передают активность другим узлам; этим порождается тот же объем вычислений, который в исходной программе порождался циклами.

При выполнении на системе с достаточно большим числом ядер в нашем примере достигается ускорение порядка L^2, если не учитывать время на пересылки. При этом совмещение вычислений с обменами между ядрами происходит автоматически, благодаря отсутствию барьерных синхронизаций.

6. Выводы

Наш компилятор выявляет потоковые зависимости алгоритма и переводит программу в такую форму, в которой зависимости выражены явно. При этом никаких явных указаний на параллелизм в результирующем коде нет. Программа всякого узла активируется по готовности входных данных независимо от других узлов. Таким образом, автоматическое распараллеливание достигается совместными усилиями компилятора и исполняющей системы. Компилятор лишь выявляет полный граф потоковых зависимостей и выражает его на языке DFL. Это отличает наш подход от традиционных распараллеливающих компиляторов, которые стремятся выявить параллелизм на стадии компиляции, используя модели вычислений (Open MP, MPI, CUDA и др.), требующие явно выраженного параллелизма. И это позволяет нам, в конечном счете, извлекать из программ больше параллелизма, чем это делают традиционные распараллеливающие компиляторы.

Список литературы

1. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В.. Автоматическое распараллеливание последовательных программ для многоядерных кластеров. //Труды Международной научной конференции "Научный сервис в сети Интернет: суперкомпьютерные центры и задачи", сентябрь 2010 г., г. Новороссийск. - М.: Изд-во МГУ, 2010, с.12-15.
2. The Portland Group. PGI Accelerator Programming Model for Fortran & C. November 2010. URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (дата обращения: 15.02.2011).
3. Baskaran, M.M., Ramanujam, J., Sadayappan, P. Automatic C-to-CUDA Code Generation for Affine Programs. // In: Gupta, R. (ed.) CC 2010. LNCS, vol. 6011, Springer, Heidelberg, 2010, pp. 244-263.
4. Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: пер. с англ. М: «И.Д. Вильямс», 2008.
5. Arvind, S. The evolution of dataflow architectures: from static dataflow to P-RISC. // International Journal of High Speed Computing. 1993. V. 5, no. 2, pp. 125-153.
6. Бурцев В.С. Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ // В сб. Бурцев В.С. Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ, ИВВС РАН, М.:1997, с.41-78.
7. Стемпковский А.Л., Левченко Н.Н., Окунев А.С, Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // "Информационные технологии" №10, 2008, с.2–7.
8. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб: БХВ-Петербург, 2004.
9. Климов Арк.В. Использование деревьев выбора для описания состояний в распараллеливающем компиляторе. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность. Труды Всероссийской суперкомпьютерной конференции, М.: Изд-во МГУ, 2009, с.238-240.

Высокоэффективный низкоуровневый интерфейс передачи сообщений SkifCh*

Ю.А. Климов[1], А.Ю. Орлов[2], А.Б. Шворин[2]

Институт прикладной математики им. М.В. Келдыша РАН[1], Институт программных систем им. А.К. Айламазяна РАН[2]

В работе описывается SkifCh — низкоуровневый интерфейс передачи сообщений. Данный интерфейс представляет абстракцию надежной передачи пакета между процессами, исполняемыми на суперкомпьютере. Важно, что операция по передаче пакета может эффективно поддерживаться на уровне сетевого оборудования, которое, в свою очередь, может быть реализовано в ПЛИС (как сделано в суперкомпьютере СКИФ-Аврора) или в специализированных микросхемах.
Интерфейс SkifCh может быть использован для высокоэффективных сетевых обменов непосредственно из прикладных программ, а также для реализации коммуникационных библиотек более высокого уровня. На данный момент поверх интерфейса SkifCh реализованы системы MPI, SHMEM, GASNet и ARMCI. В работе также приведено сравнение эффективности использования SkifCh и MPI на суперкомпьютере СКИФ-Аврора.

Введение

На сегодняшний день интерфейс передачи сообщений MPI [4] является стандартом дефакто при написании программ для работы на машинах с распределенной памятью. Он отлично задокументирован, имеет ряд замечательных реализаций, и, главное, является достаточно выразительным средством как для написания множества прикладных программ, так и для создания более высокоуровневых библиотек.

Однако, как это обычно бывает, платой за универсальность является потеря эффективности там, где подошли бы более узкоспециализированные решения. В случае MPI, благодаря механизму переупорядочивания сообщений при приеме их из сети, программист получает определенное удобство структуризации своей программы. Но, независимо от того, пользуется ли преимуществами этого механизма, он в любом случае вынужден расплачиваться некоторой потерей эффективности.

В данной работе мы предлагаем интерфейс передачи сообщений SkifCh, в котором переупорядочивание сообщений на приемном конце значительно упрощено и может быть частично или полностью реализовано в аппаратуре. SkifCh является интерфейсом более низкого уровня чем MPI в том смысле, что возможности и ограничения аппаратуры выражены в нем более явно. Тем не менее, он допускает эффективную реализацию как полного стандарта MPI, так и других распространенных библиотек обмена данными. На момент написания этой статьи имеются (помимо MPI-2) реализации SHMEM [6], GASNet [7] и ARMCI [8] поверх SkifCh. Также планируется поддержка Charm++ [9] и других систем.

За счет упрощения и перехода на более низкий уровень SkifCh позволяет более эффективно использовать возможности коммуникационной сети. В первую очередь речь идет о темпе выдачи сообщений [2], и это будет продемонстрировано ниже. Заметим, что дисциплины коммуникаций на основе односторонних обменов данными, такие как SHMEM [6], могут быть реализованы на SkifCh без существенной потери эффективности. В связи с этим мы призываем авторов библиотек для параллельного программирования рассматривать SkifCh как возможную основу коммуникационной части.

*Работы выполняются по научно-технической программе Союзного государства «СКИФ-ГРИД» [5], а также при поддержке РФФИ по проекту № 09-07-13598-офи_ц.

1. Особенности MPI

Интерфейс MPI предоставляет пользователю возможность посылать сообщения в одном порядке, а принимать — в другом.

```
int MPI_Recv (void* buf,
              int count,
              MPI_Datatype datatype,
              int source,
              int tag,
              MPI_Comm comm,
              MPI_Status *status);
```

Рис. 1. Прототип функции `MPI_Recv()`

Для этого при вызове функции `MPI_Recv()` (см. рис. 1) пользователь указывает коммуникатор comm, ранг отправителя source (либо константу `MPI_ANY_SOURCE`) и тег сообщения tag (либо константу `MPI_ANY_TAG`). Если по сети пришло несколько сообщений, то библиотека MPI выберет из них первое, которое удовлетворяет указанным параметрам.

Таким образом, реализация библиотеки MPI должна принимать сообщения в том порядке, в котором они приходят, а выдавать их пользователю в том порядке, в котором он просит. Это означает, что приходящие сообщения необходимо хранить и переупорядочивать.

Переупорядочивание вносит ощутимый накладной расход (что будет показано в разделе 4), даже если пользователь не использует переупорядочивание явно (то есть вызывает `MPI_Recv()` с параметрами `MPI_ANY_SOURCE` и `MPI_ANY_TAG`). Дело в том, что библиотека не может заранее знать, потребуется ли хранение входящего сообщения, и поэтому стек функций, обеспечивающих механизм переупорядочивания, в любом случае должен отработать. «Схлопнув» этот стек (убрав его совсем или поместив в аппаратуру), возможно добиться существенного повышения темпа обработки сообщений.

Отказ от накладных расходов на переупорядочивание входящих сообщений предполагает возложение контроля за переносом сообщений из сети в рабочие процессы на пользователя интерфейса. Таким образом, для эффективного использования коммуникационной среды необходимо сформулировать интерфейс более низкого, чем MPI, уровня.

2. Интерфейс SkifCh

2.1. Краткое описание интерфейса SkifCh

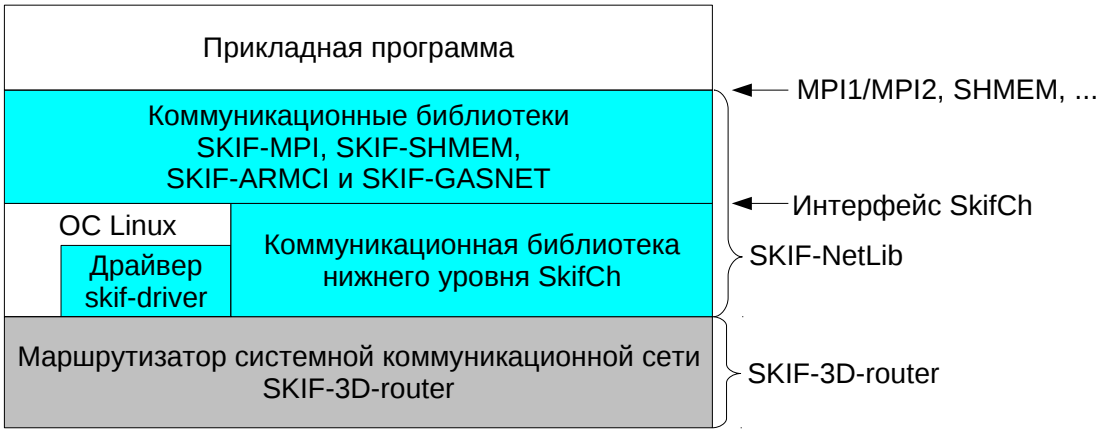


Рис. 2. Стек протоколов

Для сети топологии 3D-тор суперкомпьютера СКИФ-Аврора [1] был разработан низко-

уровневый интерфейс SkifCh. Его положение в стеке коммуникационных протоколов показано на рис. 2.

```
// struct iovec from sys/uio.h:
struct iovec {
    void * iov_base;
    size_t iov_len;
};

#define SKIFCH_CONT_SIZE 3
typedef uint32_t netaddr_t;
struct SkifCh;

ssize_t SkifCh_Send (SkifCh * skifch,
                     netaddr_t dst_netaddr,
                     const struct iovec * iov,
                     int iov_count);

ssize_t SkifCh_Recv (SkifCh * skifch,
                     struct iovec cont[SKIFCH_CONT_SIZE],
                     int * cont_count);

int SkifCh_RecvComp (SkifCh * skifch);
```

Рис. 3. Описание ядра интерфейса SkifCh

Ядро интерфейса SkifCh представлено на рис. 3. Остановимся на нем более подробно, чтобы сравнить с интерфейсом MPI.

2.1.1. Функция `SkifCh_Send()`

Функция `SkifCh_Send()` похожа на функцию `MPI_Send()`, а также на библиотечную функцию `writev()` из `sys/uio.h` (см., например, [13]). Она имеет следующий прототип:

- `skifch` — указатель на структуру канала SkifCh;
- `dst_netaddr` — адрес получателя (аналог ранга в MPI);
- `iov` и `iov_count` — задают данные в формате аналогичном тому, что используется в функциях `readv()` и `writev()` [13];
- возвращаемое значение — количество успешно отправленных байтов, то есть размер отправленного сообщения.

Параметры `iov` и `iov_count` задают массив указателей на данные пользователя. Разбиение на части при передаче не учитывается: можно считать, что посылается один сплошной массив данных. Разбиение используется лишь для того, чтобы избежать дополнительного копирования при отправке данных.

Функция `SkifCh_Send()` формирует сообщение из данных пользователя и отправляет его в сеть, причем размер сообщения определяется динамически в зависимости от состояния сети. Таким образом `SkifCh_Send()` имеет право отправить не все данные пользователя, а только их часть, что аналогично поведению функции `writev()`.

Чтобы отправить оставшиеся данные, необходимо соответствующим образом изменить параметры `iov` и `iov_count` (чтобы они описывали оставшиеся данные) и, возможно, изменить сами данные, а затем снова вызвать функцию посылки `SkifCh_Send()`. Данную процедуру, возможно, придется повторять многократно.

2.1.2. Функция `SkifCh_Recv()`

Функция `SkifCh_Recv()` значительно отличается и от `MPI_Recv()`, и от `readv()`. Она имеет следующие параметры:

- `skifch` — указатель на структуру канала SkifCh;
- `cont` — указатель на массив, размера не менее, чем `SKIFCH_CONT_SIZE`;

• `cont_count` — указатель на место, где будет размещен размер заполненной части массива `cont` после завершения функции;

• возвращаемое значение — количество успешно полученных байтов, размер полученного сообщения.

Массив `cont` размера `SKIFCH_CONT_SIZE` должен быть заведен пользователем перед вызовом, например, на стеке. Элементы массива не должны быть проинициализированы.

После вызова `SkifCh_Recv()` в возвращаемом значении пользователю будет выдан размер принятого сообщения в байтах. Массив `cont` будет заполнен указателями на части сообщения, которые храняться во внутренней памяти реализации SkifCh. Реальное количество заполненных элементов массива `cont` будет указано в `cont_count`. Разбиение сообщения на части несущественно, можно считать, что сообщение — это один сплошлой массив.

За один вызов `SkifCh_Recv()` будет получено одно сообщение, которое было отправлено одним вызовом функции `SkifCh_Send()`. Как и при отправке, чтобы получить все необходимые данные, отправленные несколькими вызовами `SkifCh_Send()`, потребуется такое же количество вызовов функции `SkifCh_Recv()`.

SkifCh гарантирует порядок сообщений между отправителем и получателем. Однако при приеме сообщения заранее неизвестно, от какого отправителя будет получено сообщение. Поэтому, даже если данные были отправлены несколькими подряд идущими вызовами функции `SkifCh_Send()`, то не обязательно они будут приняты подряд идущими вызовами функции `SkifCh_Recv()`. Между получением сообщений от данного отправителя могут быть получены сообщения от других отправителей.

После обработки сообщения пользователь обязан вызвать `SkifCh_RecvComp()`, которая уведомляет реализацию SkifCh о том, что сообщение прочитано, и соответствующее место в служебном буфере можно использовать повторно.

Такой механизм, в частности, позволяет избежать обязательного копирования на приеме. Если пользователь может обработать полученное сообщение на месте, то интерфейс позволяет ему это сделать.

2.2. Каналы SkifCh

Следует отметить, что теги и/или коммуникаторы в MPI используются для разделения пространства сообщений между различными подсистемами — например, между библиотекой и программой. Совсем выбросить такой полезный механизм было бы опрометчиво, поэтому разделение пространства сообщений возможно и с использованием интерфейса SkifCh. Для этого каждая часть программы (например, библиотека), которая должна использовать сеть независимо от других систем, получает свой канал SkifCh и свой сетевой адрес `dst_netaddr`. Это обеспечивает получение лишь тех сообщений, которые предназначены именно ей. В некотором смысле каналы SkifCh аналогичны портам протокола TCP/IP.

При помощи каналов интерфейс SkifCh абстрагирует для пользователя разделение сообщений между получателями. Это может обеспечиваться непосредственно аппаратурой, либо программной прослойкой. В случае сети 3D-тор суперкомпьютера СКИФ-Аврора сетевые адаптеры реализованы в ПЛИС, и разделение пространства сообщений происходит именно там. Использование ПЛИС позволяет гибко описывать, какая часть сортировки и переупорядочивания сообщений будет реализована в аппаратуре (при ее ограниченных возможностях), а какая будет выполняться на уровне библиотеки (если это реально нужно в приложении), например, в библиотеке MPI.

2.3. Особенности SkifCh и его место в разработке ПО

SkifCh позиционируется авторами в первую очередь как низкоуровневое средство, на котором следует реализовывать коммуникационные библиотеки более высокого уровня, такие как MPI, SHMEM, Charm++ и т. п. Однако это не умаляет возможности использования

SkifCh непосредственно при разработке прикладных программ.

В чем же преимущество использования низкоуровневого интерфейса по сравнению, например, с MPI?

Прежде всего — в повышении эффективности за счет снижения накладных расходов. Например, функция `SkifCh_Recv()` (в отличие от `MPI_Recv()` и `readv()`) *не копирует* сообщение в буфер пользователя, а оставляет его в буферах библиотеки SkifCh и показывает, где именно оно расположено. Пользователь волен скопировать сообщение в свой буфер либо обработать его на месте. Основная идея такого подхода заключается в том, чтобы иметь возможность избежать лишнего копирования.

Интерфейс SkifCh подразумевает, что внутри библиотеки не происходит дополнительной буферизации. Например, если сообщение не может быть отправлено немедленно, то вызов `SkifCh_Send()` завершится неуспехом, и программист должен повторить вызов через некоторое время.

При посылке слишком большого сообщения может быть передана только его часть. Эта особенность вызвана ограничением аппаратных ресурсов сети и отсутствием буферизации в SkifCh. Поэтому, если программисту необходимо передать большое сообщение, то при отправлении оно будет нарезано на части. Это автоматически означает, что программисту необходимо озаботиться собиранием больших сообщений из нескольких кусочков на приемном конце.

Заметим, что из-за отсутствия буферизации и переупорядочивания, `SkifCh_Recv()` получает сообщения от разных отправителей в произвольном порядке. Однако, как и в случае с интерфейсом MPI, SkifCh гарантирует (и накладывает соответствующие ограничения на нижележащую реализацию сети), что сообщения от одного отправителя приходят в том же порядке, в котором были посланы.

Заботясь об эффективности, не стоит забывать и о продуктивности — трудоемкости создания программного обеспечения с точки зрения разработчика. Как правило, чем ниже уровень, на который опирается разработчик, чем ближе к аппаратуре, тем продуктивность ниже. (Этим фактом, в частности, оправдывается существование высокоуровневых средств программирования.) Описанные выше ограничения SkifCh, с одной стороны, действительно усложняют его использование. С другой стороны, усложнения касаются в основном обмена большими сообщениями. Во многих случаях (например, для библиотеки SHMEM) эти ограничения несущественны. И именно благодаря этим ограничениям удается не потерять эффективность сети.

3. Примеры использования SkifCh

В рамках работ по созданию сети 3D-тор суперкомпьютера СКИФ-Аврора авторами были реализованы библиотеки MPI и SHMEM [6], основанные на описанном выше интерфейсе SkifCh.

Другими коллективами были разработаны системы GASNet [7] и ARMCI [8], что показывает достаточную функциональную наполненность и жизнеспособность интерфейса SkifCh.

SkifCh изначально разрабатывался для сети 3D-тор суперкомпьютера СКИФ-Аврора. Как уже было сказано, реализация коммуникационной среды в этой машине выполнена в ПЛИС, что дает достаточно большую свободу при выборе границы и методов разделения работы между программной и аппаратной частями. Впоследствии SkifCh был перенесен на сеть МВС-Экспресс [3], на основе которой в ИПМ РАН совместно с НИИ «Квант» были созданы суперкомпьютеры МВС-Экспресс и К-100. Эта сеть основана на прямой коммутации PCI-Express при помощи стандартных микросхем фирмы PLX.

Последнее является свидетельством того, что интерфейс SkifCh обладает достаточной гибкостью и может быть адаптирован к разным типам коммуникационного оборудования.

4. Эффективность SkifCh

В работе [2] была продемонстрирована важность такого показателя качества коммуникационной сети как темп выдачи сообщений.

Для сравнения эффективности работы с использованием интерфейсов SkifCh и MPI будем использовать стандартный тест MsgRate[1] [15]. Ядро его реализации на MPI приведено на рисунке 4. С использованием SkifCh тест записывается аналогично.

```
MPI_Barrier(MPI_COMM_WORLD);
for (i = 0; i < N /*количество итераций*/; i++)
  if (my_rank == 0)
    MPI_Send(buf, len, MPI_BYTE, 1 /*dst*/, 0, MPI_COMM_WORLD);
  else if (my_rank == 1)
    MPI_Recv(buf, len, MPI_BYTE, 0 /*src*/, 0, MPI_COMM_WORLD, &sts);
MPI_Barrier(MPI_COMM_WORLD);
```

Рис. 4. Тест MsgRate

Здесь один процесс (с номером 0) только отправляет сообщения, а другой (с номером 1) только принимает сообщения. Измеряется общее время выполнение теста. Количество итераций деленное на полученное значение — это темп выдачи сообщений.

На графике на рисунке 5 приведены результаты теста в зависимости от размера сообщения для реализаций на MPI и на SkifCh. Хорошо видно, что для коротких сообщений использование SkifCh дает громадное преимущество по сравнению с MPI на сети 3D-тор, реализованным поверх SkifCh.

В данном тесте никак не используются преимущества переупорядочивания сообщений на приемном конце. Есть сообщения только одного вида, и все они должны быть получены в порядке их появления. Таким образом, разница в скорости работы теста на SkifCh и на MPI полностью обусловлена теми накладным расходами на переупорядочивание сообщений, которые присутствуют всегда — независимо от того, используется ли данный механизм или нет.

Для сравнения приведены также значения для того же самого кода на MPI, при использовании реализации Open MPI [10] на сети InfiniBand QDR. Все замеры (и на сети 3D-тор, и на сети InfiniBand) производились на узлах суперкомпьютера СКИФ-Аврора.

5. Заключение

В статье предложен низкоуровневый интерфейс передачи сообщений SkifCh. Были показаны преимущества и недостатки его использования по сравнению с интерфейсом MPI. Продемонстрировано, что SkifCh позволяет эффективно использовать возможности коммуникационной сети, достигая высокого темпа выдачи сообщений за счет простоты интерфейса и переноса части работы в аппаратуру.

Следует отметить, что близкие по положению в стеке сетевых протоколов высокоэффективные интерфейсы естественным образом возникают у многих разработчиков сетевой аппаратуры и коммуникационных библиотек.

Например, близким аналогом SkifCh в библиотеке MPICH2 [11] является внутренний интерфейс CH3, не предназначенный, однако, для стороннего использования.

В библиотеке Intel-MPI [12] таким интерфейсом является TMI, который дает возможность разработчиками эффективно использовать различные коммуникационные среды, например, QLogic PSM и Myrinet MX.

Большой интерес вызывает QLogic PSM [16] — низкоуровневый высокоэффективный

[1]Этот тест входит в поставку ScaliMPI под именем «Ping-Ping», что может ввести в заблуждение, поскольку в наборе IMB [14] есть более известный тест с тем же именем Ping-Ping, который устроен совершенно иначе. Во избежание путаницы мы решили дать свое название тесту.

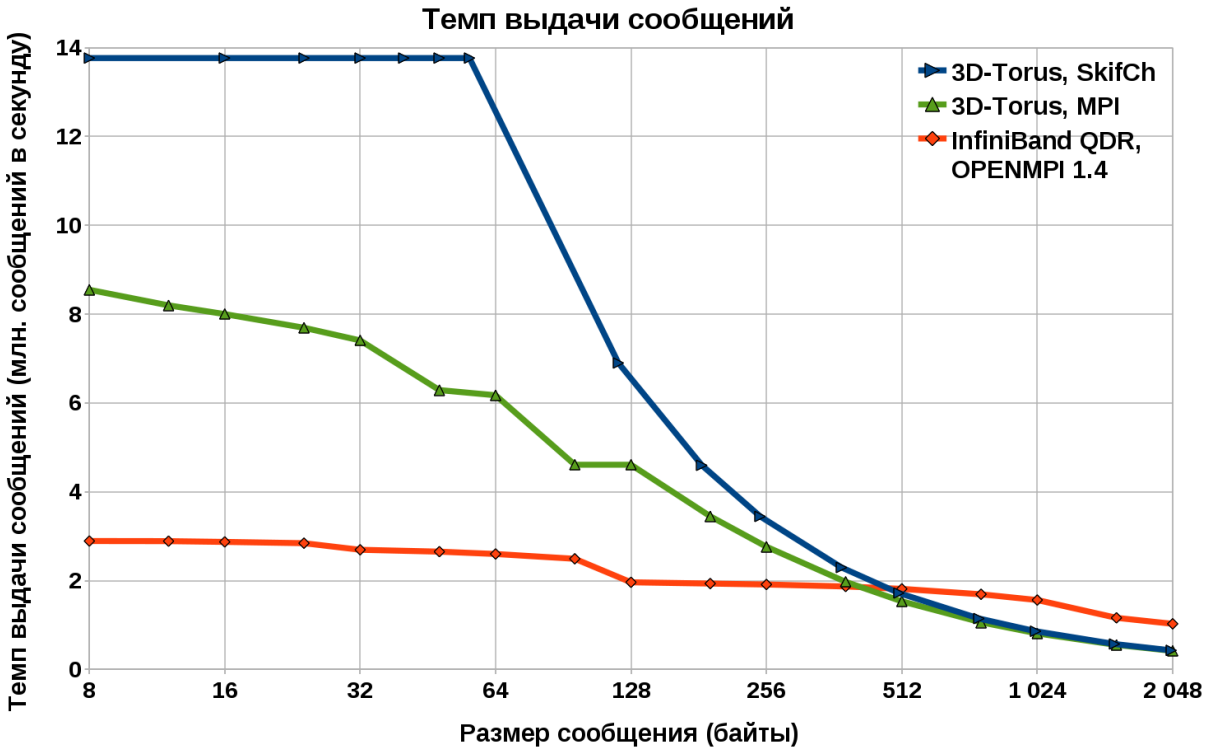


Рис. 5. Темп выдачи сообщений (больше — лучше)

интерфейс, разработанный фирмой QLogic для своей сетевой аппаратуры. В отличие от CH3 и TMI этот интерфейс предназначен для реализации не только MPI, но различных коммуникационных библиотек — имеются реализации SHMEM и других. PSM является наиболее близким из известных авторам данной работы аналогов SkifCh. Интересно, что фирма QLogic заявляет о эффективности PSM именно в связи с высоким темпом выдачи сообщений. Однако до сих пор нам не представилась возможность увидеть заявленные цифры в реальности.

Основными преимуществами интерфейса SkifCh по сравнению с аналогами могут быть названы следующие:

- ориентация на создание различных библиотек с высокоэффективной коммуникационной частью;
- достаточная гибкость, позволяющая не потерять эффективность сети на разных классах сетевого оборудования;
- доступность на современных суперкомпьютерах СКИФ-Аврора и К-100, оба из которых являются гибридными.

Заметим напоследок, что использование гибридных машин предполагает обычно мелкозернистые обмены. Это обстоятельство значительно повышает требования к качеству коммуникационной сети в целом, и, в частности, достижение высокой эффективности использования машины становится невозможным без высокого темпа выдачи сообщений.

Литература

1. Адамович И.А., Климов А.В., Климов Ю.А., Орлов А.Ю., Шворин А.Б. Опыт разработки коммуникационной сети суперкомпьютера «СКИФ-Аврора» // Программные системы: теория и приложения: электронный научный журнал. — 2010. — № 3 (3). — С. 107-123. — URL: `http://psta.psiras.ru/read/psta2010_3_107-123.pdf` (дата обращения:

15.12.2010).

2. Климов Ю.А., Орлов А.Ю., Шворин А.Б. Темп выдачи сообщений как мера качества коммуникационной сети // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (20–25 сентября 2010 г., г. Новороссийск). —М.: Изд-во МГУ, 2010. — С. 414-417.

3. Лацис А. О. Вычислительная система МВС-Экспресс // URL: http://www.kiam.ru/MVS/research/mvs_express.html (дата обращения: 15.12.2010).

4. Message Passing Interface (MPI) // URL: http://www.mpi-forum.org/ (дата обращения: 15.12.2010).

5. Суперкомпьютерная программа Союзного государства «СКИФ-ГРИД» (2007-2010 гг.) // URL: http://skif-grid.botik.ru/ (дата обращения: 15.12.2010).

6. SHMEM application programming interface // URL: http://www.shmem.org/ (дата обращения: 15.12.2010).

7. GASNet communication system // URL: http://gasnet.cs.berkeley.edu/ (дата обращения: 15.12.2010).

8. Aggregate Remote Memory Copy (ARMCI) library // URL: http://www.emsl.pnl.gov/docs/parsoft/armci/ (дата обращения: 15.12.2010).

9. Charm++ programming language // URL: http://charm.cs.uiuc.edu/ (дата обращения: 15.12.2010).

10. Open MPI: Open Source High Performance Computing // URL: http://www.open-mpi.org/ (дата обращения: 15.12.2010).

11. MPICH2: High-performance and Widely Portable MPI // URL: http://www.mcs.anl.gov/research/projects/mpich2/ (дата обращения: 15.12.2010).

12. Intel MPI library // URL: http://software.intel.com/en-us/articles/intel-mpi-library/ (дата обращения: 15.12.2010).

13. Linux man-pages project // URL: http://www.kernel.org/doc/man-pages/online/pages/man2/readv.2.html (дата обращения: 15.12.2010).

14. Набор тестов Intel MPI Benchmarks (IMB) // URL: http://software.intel.com/en-us/articles/intel-mpi-benchmarks/ (дата обращения: 15.12.2010).

15. Тест Bandwidth // URL: http://botik.ru/~klimov/bandwidth.tgz (дата обращения: 15.12.2010).

16. Scaling IB Fabrics to Meet the Needs of a PetaFlop World // URL: http://www.cse.scitech.ac.uk/disco/mew20/presentations/QLogic.pdf (дата обращения: 15.12.2010).

Об особенностях решения больших систем линейных алгебраических уравнений на многопроцессорных вычислительных системах с различной архитектурой

Б.И. Краснопольский

НИИ механики МГУ

В работе рассматриваются особенности использования разработанного набора итерационных методов Крыловского подпространства (CGS, BiCGStab) с алгебраическим многосеточным предобуславливателем для решения больших сильно разреженных систем линейных алгебраических уравнений на многопроцессорных вычислительных системах. Приводятся результаты исследования характеристик масштабируемости методов для MPI и гибридной реализаций. Обсуждаются особенности использования реализованного набора методов на вычислительных системах, построенных на базе процессоров различных типов (Intel Harpertown, Intel Nehalem, AMD Magny-Cours).

1. Введение

Решение больших систем линейных алгебраических уравнений вида

$$Ax = b$$

является одной из традиционных и широко распространенных задач для многопроцессорных вычислительных систем. Например, такие матрицы возникают при разностной аппроксимации дифференциальных уравнений в частных производных на больших расчетных сетках. Зачастую шаблон аппроксимации, а как следствие и количество ненулевых элементов в строке матрицы, не превышает нескольких десятков элементов, тогда как размер матрицы может достигать 10^8 и более неизвестных. Это приводит к необходимости использования специализированных форматов хранения данных (например, CSR, DMSR и пр.) и выбора соответствующих численных методов, которые могут быть реализованы для таких форматов данных. В общем случае, наиболее распространенными численными методами для таких задач являются итерационные методы Крыловского подпространства с частичным LU [1] или многосеточным [2] предобуславливателями, которые могут успешно применяться на вычислительных системах с распределенной памятью.

Данная статья является продолжением работ [3, 4] по разработке библиотеки эффективных методов решения больших систем линейных алгебраических уравнений. В работе [4] основное внимание было уделено исследованию результатов масштабируемости MPI и MPI+ShM реализаций переупорядоченного метода BiCGStab [3] с алгебраическим многосеточным предобуславливателем на вычислительных платформах СКИФ МГУ “Чебышёв” и “Ломоносов”. Здесь основной акцент уделяется обсуждению особенностей использования таких систем (библиотеки MPI, схемы привязки вычислительных процессов к физическим ядрам и пр.) на трех вычислительных системах, построенных на базе различных типов процессоров (Intel Harpertown, Intel Nehalem, AMD Magny-Cours).

2. Тестовые задачи

В качестве тестовой задачи в работе использовалась матрица размером 8 млн. неизвестных, полученная при разностной аппроксимации уравнения Пуассона в кубической области на равномерной расчетной сетке (7-точечный шаблон). Правая часть исходной системы уравнений задавалась постоянной. Для решения этой системы уравнений использовался итераци-

онный метод BiCGStab с алгебраическим многосеточным предобуславливателем. Оптимальное разбиение между вычислительными процессами и переупорядочивание строк матрицы строилось с помощью свободно распространяемой библиотеки PT-Scotch [5].

3. Тестовые платформы

Основные характеристики использованных в работе аппаратных платформ, список реализации библиотек MPI и компиляторов, приведены в табл. 1–2.

Таблица 1. Аппаратные характеристики тестовых платформ

Название	**“Чебышёв”**	**“Ломоносов”**	**“Зилант”**
Тип процессора	Intel Harpertown	Intel Nehalem	AMD Magny-Cours
Модель процессора	E5472	X5570	Opteron 6174
Количество процессоров	2 (8 ядер/узел)	2 (8 ядер/узел)	2 (24 ядра/узел)
Тактовая частота	3.0 Ггц	2.93 ГГц	2.2 ГГц
Размер кэш-памяти	L2 12 Мб	L3 8 Мб	L3 12 Мб
Оперативная память	8 Гб	12 Гб	64 Гб
Частота и тип памяти	DDR2 1333 МГц	DDR3 1333 МГц	DDR3 1333 МГц
Пропускная способность	12.8 Гб/сек	32 Гб/сек на проц.	42.7 Гб/сек на проц.
Внутриузловой интерконнект	-	Quick Path	HyperTransport 3
Межузловой интерконнект	IB DDR	IB QDR	IB QDR
Принадлежность	Суперкомпьютерный комплекс МГУ		ЗАО “Т-Сервисы”

Таблица 2. Библиотеки MPI и компиляторы

Вычислительная система	**Библиотека MPI**	**Компилятор**
“Чебышёв”	MVAPICH 1.1	Intel Compiler 11.1
	Open MPI 1.4	GNU Compiler Collection 4.1.2
“Ломоносов”	Open MPI 1.4.3	Intel Compiler 12.0
“Зилант”	MVAPICH 1.2	Intel Compiler 12.0
	Open MPI 1.4.1	Intel Compiler 12.0
	Platform MPI 8.0	Intel Compiler 12.0

4. Результаты

4.1. Масштабируемость внутри одного узла

В первом параграфе этой статьи приведены результаты масштабируемости приложения внутри одного вычислительного узла[1].

4.1.1. MPI модель

Для MPI модели реализации программы были исследованы два случая распределения процессов по физическим ядрам процессоров: режим “1 CPU”, когда все процессы размещаются на физических ядрах только одного процессора, и режим “2 CPU’s”, когда процессы равномерно распределяются между процессорами.

Масштабируемость внутри одного процессора на вычислительной системе “Чебышёв” оказывается достаточно слабой (рис. 1.а): переход от 1 к 4 ядрам сопровождается незначительным монотонным ростом, и ускорение в итоге не превышает 1.4 раза. Использование 2 ядер с 2 процессоров дает ускорение в 1.8 раза, однако дальнейшее наращивание количества вычислительных процессов на узле сопровождается столь же слабым приростом ускорения: для 8 ядер оно составляет всего 2.5 раза. В целом, столь низкие результаты масштабируемости для этой платформы представляются ожидаемыми, ввиду наличия общего контроллера памяти для обоих процессоров и низкой скорости доступа к памяти. Выбранное же тестовое приложение реализует существенно неравномерный доступ к памяти (используемые форматы хранения разреженных матриц порождают двойную индексацию при обращении к данным, причем эти данные могут обладать достаточно слабой временной и пространственной локальностью), что приводит к большим задержкам при обращении к памяти.

Существенно лучшими оказываются результаты масштабируемости на “Ломоносове”[1] (рис. 1.б). Ускорение внутри одного процессора составляет 3.3 раза для 4 ядер, что в сравнении с процессором E5472 демонстрирует почти 3-кратный прирост. Перераспределение процессов на вычислительных ядрах, расположенных физически на разных процессорах, также оказывается более эффективным, однако, если для процессоров на “Чебышёве” такое перераспределение давало ощутимый прирост производительности, то здесь выигрыш оказывается в пределах 1-2%. Монотонность роста, хотя и с некоторым замедлением, наблюдается вплоть до 8 ядер, и достигает 4.8 раза, что почти вдвое лучше результатов, полученных на “Чебышёве”.

Схожая с “Ломоносовым” картина масштабируемости наблюдается и на одном узле системы “Зилант” (рис. 1.в). Ускорение на 1 процессоре для 12 ядер составляет 6.5 раз, а для 24 ядер и двух процессоров - 11.6 раза.

Если говорить об абсолютных величинах времени расчета тестовой задачи, то время на одном ядре “Чебышёва” и одном ядре “Зиланта” оказываются одинаковыми с точностью до 1-2% и составляют 156 секунд, несмотря на разницу тактовой частоты процессоров в 800 МГц. Что интересно, полученные на “Чебышёве” времена решения задачи оказались практически одинаковыми как для компилятора Intel, так и для GNU-компилятора. Время расчета на 1 ядре “Ломоносова” оказалось заметно лучше и составило 77 секунд.

С точки зрения абсолютных времен, при использовании одного вычислительного узла

[1]Под временем решения задачи здесь и далее понимается время вычисления 20 итераций метода. Такое ограничение принято во избежание незначительных флуктуаций необходимого количества итераций для достижения критерия сходимости итерационного процесса.

[1]Данные тесты были проведены на вычислительном узле с оперативной памятью размером 24 Гб. На вычислительных узлах с 12 Гб оперативной памяти время работы последовательной программы оказывалось в среднем на 15-20% больше. Это обусловлено тем, что в первом случае тестовая задача полностью умещается в память одного NUMA-node, в то время как во втором случае задействуется и вторая область памяти, доступ к которой реализуется через QPI и контроллер памяти второго процессора.

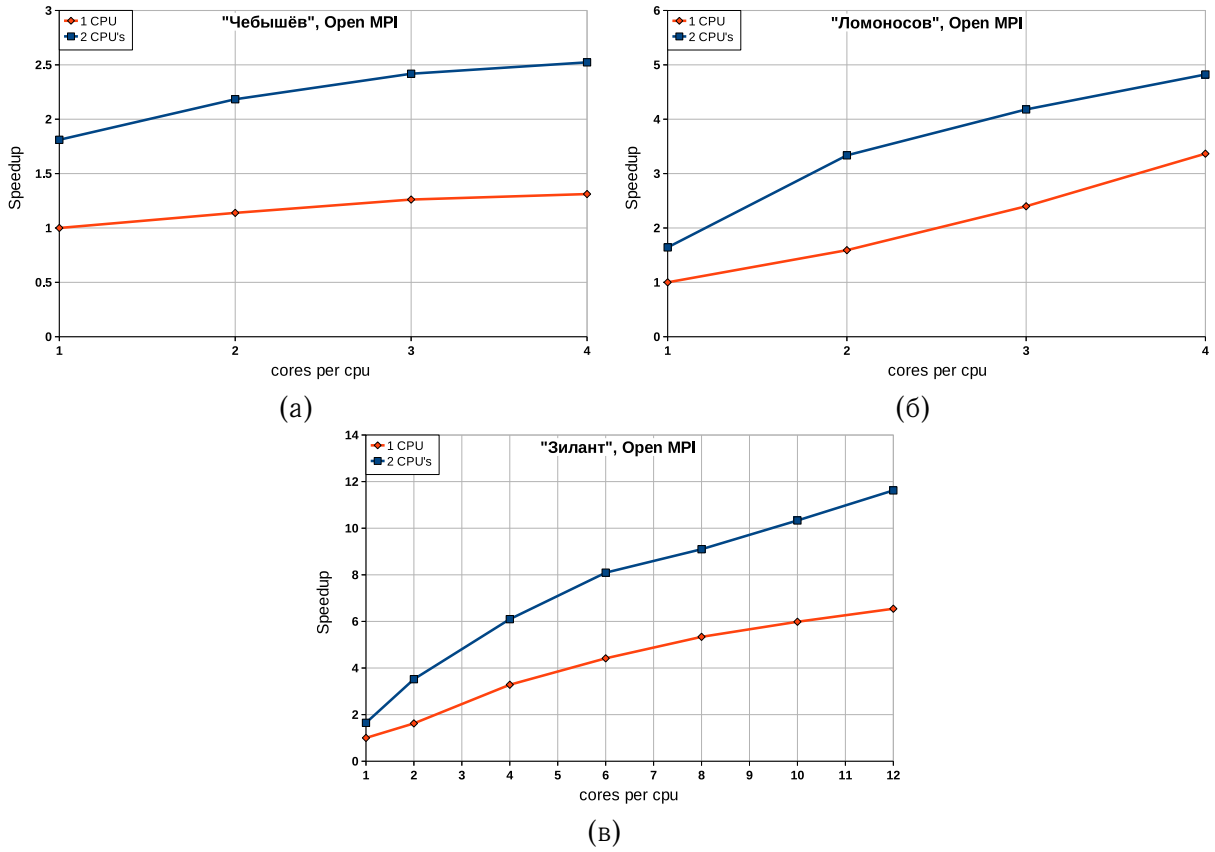


Рис. 1. Графики зависимости масштабируемости MPI версии приложения от количества используемых ядер внутри одного вычислительного узла: (а) СКИФ МГУ “Чебышёв”, (б) “Ломоносов”, (в) “Зилант”.

наиболее эффективным на данном тесте оказывается узел на базе процессоров AMD Magny-Cours, для которого время расчета задачи составляет 14.1 секунды. Близкие результаты наблюдаются и для “Ломоносова”: 16.1 секунды. В свою очередь вычислительный узел на базе процессоров E5472 демонстрирует куда худшие результаты, и время расчета задачи оказывается в 4 раза больше – 62 секунды.

4.1.2. Гибридная модель

На следующем этапе работы было проведено исследование эффективности гибридной версии программы, где обмены между процессами внутри одного узла были реализованы через общую память вычислительного узла. В данном случае на узле использовались все доступные физические ядра. В тестах было рассмотрено несколько случаев, когда варьировалось количество процессов, которые объединялись через общую память.

Как и можно было ожидать, для одного узла “Чебышёва” увеличение количества процессов, взаимодействующих через общую память, приводит к незначительному, но улучшению масштабируемости (табл. 3). Полностью противоположная ситуация наблюдается для одного узла “Ломоносова”: оптимальным оказывается использование 8 независимых MPI-процессов. В свою очередь, на “Зиланте” минимальные времена достигаются для блоков размером 3-6 процессов, с соответствующей схемой привязки этих процессов к физическим ядрам одного процессора, и, как следствие, к одному NUMA-node, а при дальнейшем увеличении размера блоков наблюдается резкое ухудшение результатов.

Для пояснения полученных результатов масштабируемости необходимо сказать несколько слов об особенностях реализации гибридной модели. С целью экономии оперативной

Таблица 3. Время расчета задачи при использованни гибридной версии кода

Вычислительная система	**Версия MPI**	**Размер блоков**							
		1	2	3	4	6	8	12	24
“Чебышёв”	Open MPI	62	61.74	-	61.25	-	60.43	-	-
“Ломоносов”	Open MPI	16.13	17.04	-	22.6	-	21.1	-	-
“Зилант”	Open MPI	14.13	12.92	13.06	-	13.89	-	23.88	49.19
	MVAPICH	14.53	-	12.87	-	12.77	-	24.3	48.86
	Platform MPI	14.06	-	12.75	-	13.52	-	24.74	48.97

памяти на узлах, в этой версии программы имеются достаточно большие фрагменты данных, расположенные в общей для всех процессов памяти (локальные фрагменты матриц на вычислительных узлах хранятся в общей памяти). Изначально протестированный на “Чебышёве”, такой подход на UMA-архитектуре не выявил существенных недостатков с точки зрения скорости доступа к памяти, при этом потребовав относительно небольших затрат на реализацию. Однако, как показали проведенные тесты, для NUMA-архитектуры такой подход неудачен. Это вызвано тем, что несколько процессов с разных процессоров вынуждены обращаться в общую память, которая распределена между областями памяти обоих NUMA-node. Это порождает большое количество обращений к памяти чужого NUMA-node через межпроцессорный интерконнект (QPI или HyperTransport), что и приводит к сильному падению производительности. Наиболее наглядно этот эффект наблюдается на вычислительной системе “Зилант” для блоков по 24 процесса.

В целом достаточно хорошие результаты масштабируемости для MPI-версии кода в сравнении с гибридной моделью, можно объяснить еще двумя соображениями. Во-первых, увеличение количества MPI-процессов приводит к разбиению данных на блоки, что несколько улучшает локализацию данных, и как следствие, эффективность доступа к памяти. Во-вторых, большинство реализаций библиотеки MPI на системном уровне также реализует обмен сообщениями внутри узла через общую память, что в данном случае нивелирует выигрыш от применения гибридной модели.

4.2. Межузловая масштабируемость

4.2.1. MPI модель

На следующем этапе работ была исследована масштабируемость MPI-версии кода в зависимости от количества вычислительных узлов и задействованных ядер.

Графики зависимости ускорения от количества вычислительных узлов “Чебышёве” приведены на рис. 2.а. В данном случае использование всех 8 ядер на узле оказывается бесполезным даже для малого количества узлов. Крайне незначительный выигрыш по отношению к 4 ядрам, сравнимый с погрешностью измерений, имеется только для 1-4 узлов. Пиковые значения ускорения в 74 раза достигаются для 1 и 2 ядер на 128 узлах. При этом, для 2 вычислительных процессов прирост ускорения к этому моменту существенно замедляется и кривая уже близка к насыщению, тогда как график для 1 вычислительного процесса на узел демонстрирует почти линейный рост. Минимальное время расчета задачи составляет 2.12 секунды.

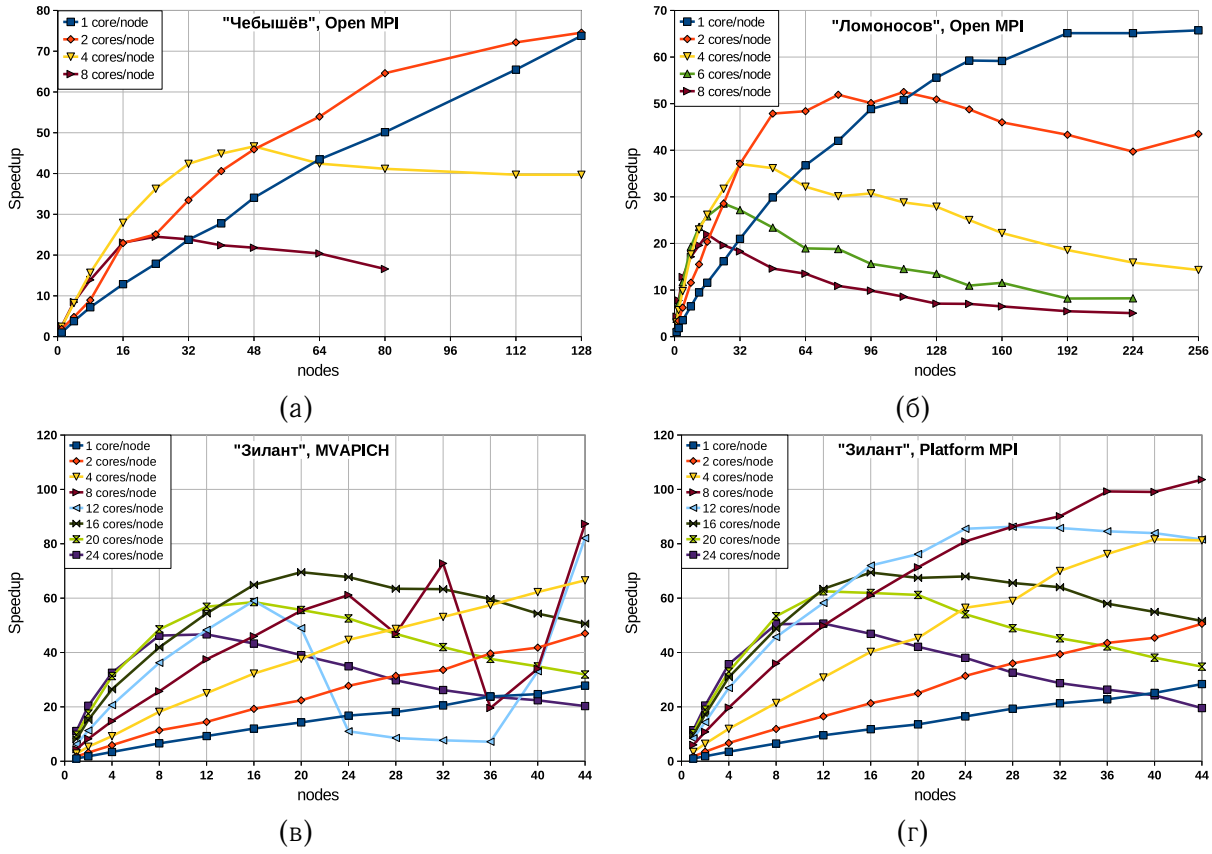


Рис. 2. Графики зависимости масштабируемости MPI-версии приложения от количества вычислительных узлов: (а) “Чебышёв”, (б) “Ломоносов”, (в) “Зилант”: MVAPICH, (г) “Зилант”: Platform MPI.

Расчеты на “Ломоносове” показали, что максимальные ускорения порядка 65 раз достигаются при использовании 192 узлов и только одного вычислительного процесса на узле (рис. 2.б). Для MPI-версии кода это представляется вполне ожидаемым, поскольку в таком режиме каждый процесс “монопольно” использует коммуникационную сеть узла. Однако, при ограниченном количестве ресурсов ситуация оказывается не столь однозначной: в зависимости от количества доступных узлов, наиболее эффективным может оказаться использование и большего количества ядер. Так, для 1-4 узлов оптимально использовать 8 ядер; для 8-12 узлов – 6 ядер; для 16-32 узлов – 4 ядра; для 48-112 узлов – 2 ядра, и только для задач 128 и более узлов эффективным оказывается использование лишь одного ядра. Минимальное время расчета задачи в этом случае составляет 1.18 секунды.

Относительные результаты масштабируемости на “Ломоносове”, несмотря на более быструю коммуникационную сеть, оказываются даже несколько хуже, чем на “Чебышёве”. Однако, этот факт объясняется тем, что непосредственно сами вычисления на процессорах X5570 проходят почти вдвое быстрее (если судить по времени расчета последовательных запусков), тем самым оставляя существенно меньше времени для эффективного выполнения MPI-коммуникаций между узлами.

Графики масштабируемости тестовой задачи на вычислительной системе “Зилант” для библиотек MPI MVAPICH и Platform MPI приведены на рис. 2.в и 2.г соответственно. Для библиотеки MVAPICH лучшие результаты достигаются при использовании 16 ядер на узлах. Пиковое ускорение в этом случае достигает 70 раз. Близкие значения также получены и для 4 MPI-процессов, приходящихся на один узел. При этом ускорение в 67 раз достигается для максимального количества доступных узлов, а до этого момента зависимость ускорения от количества узлов оказывается близкой к линейной. Это позволяет предполагать, что при

дальнейшем увеличении узлов текущее пиковое значение может быть существенно превышено. В целом, согласно приведенным данным, наблюдается достаточно типичная картина поведения графиков масштабируемости. Существенно выбиваются из этой картины лишь две кривых для 8 и 12 процессов. Начиная с 20 узлов графики ускорения демонстрируют значительные колебания, причем эти колебания носят регулярный характер.

Наличие таких осцилляций времени расчета задачи для библиотеки MVAPICH подтолкнуло к исследованию других реализаций библиотеки обмена сообщениями. Графики зависимости ускорения от количества узлов, полученные при использовании библиотеки Platform MPI, приведены на рис. 2.г. В этом случае графики масштабируемости оказываются более предсказуемыми, а результаты – лучше. Так, пиковое значение ускорения было получено для 8 MPI-процессов на узле и составило 103 раза. Соответствующее время расчета задачи в этом случае оказалось равным 1.55 секунды.

Таким образом, несмотря на существенно лучшие результаты масштабируемости, полученные на AMD-платформе, абсолютное время расчета задачи на "Ломоносове" оказалось меньше на 30%. Вместе с тем, судя по поведению кривых масштабируемости для "Зиланта", можно предполагать, что на большем количестве узлов показатели масштабируемости могут быть значительно улучшены, а время расчета задачи будет сравнимым или лучшим по сравнению со временем расчета на "Ломоносове". Отдельно следует отметить, что на результаты масштабируемости оказывает очень существенное влияние выбор схемы привязки вычислительных процессов к физическим ядрам процессоров: в некоторых случаях различия в результатах могут составлять до одного порядка. Наиболее удачной оказалась следующая схема привязки процессов:

```
CPU_MAP:0,12,6,18,3,15,9,21,1,13,7,19,4,16,10,22,2,14,8,20,5,17,11,23
```

В этом случае происходит равномерное распределение процессов как между двумя 12-ядерными процессорами так и внутри этих процессоров между двумя "склеенными" 6-ядерными процессорами.

4.2.2. Гибридная модель

Достаточно интересными оказались и результаты исследования масштабируемости для гибридной версии кода (рис. 3). В этих тестах в расчетах также использовались все доступные физические ядра (8 для Intel-платформ и 24 для AMD-платформы)[1].

На вычислительной системе "Ломоносов" объединение 8 процессов в один блок позволяет заметно улучшить результаты масштабируемости. Пиковое значение достигается на 224 узлах и составляет 96 раз, в сравнении с ускорением в 65 раз для MPI-версии кода. Абсолютное время расчета тестовой задачи в этом случае уменьшается в полтора раза с 1.18 до 0.81 секунды.

Для "Зиланта" исследование было проведено также для двух библиотек MPI: MVAPICH и Platform MPI. Причем, если в случае MPI-приложения заметное преимущество было у библиотеки Platform MPI, то здесь наблюдаемая картина оказывается в точности противоположной. Так, для MVAPICH при использовании не более, чем 40 узлов, наиболее удачной конфигурацией оказывается использование на каждом узле 4 блоков по 6 процессов с привязкой этих процессов к одной из половин 12-ядерного процессора. Пиковые значения оказываются на уровне ускорения в 156 раз, что эквивалентно времени расчета тестовой задачи за 1.03 секунды. В районе 40 узлов наблюдается тенденция к сближению кривых для блоков из 6 и 12 процессов, что говорит о том, что для количества узлов порядка 60 оптимальным будет использование уже блоков из 12 процессов. При использовании Platform MPI наибо-

[1]Под параметром "Block-size" в легенде графика понимается количество MPI-процессов, объединенных в единый блок, обмен данными внутри которого происходит через общую память, и в котором используется только один внешний MPI-процесс; случай "Block-size=1" эквивалентен обычной MPI-версии приложения.

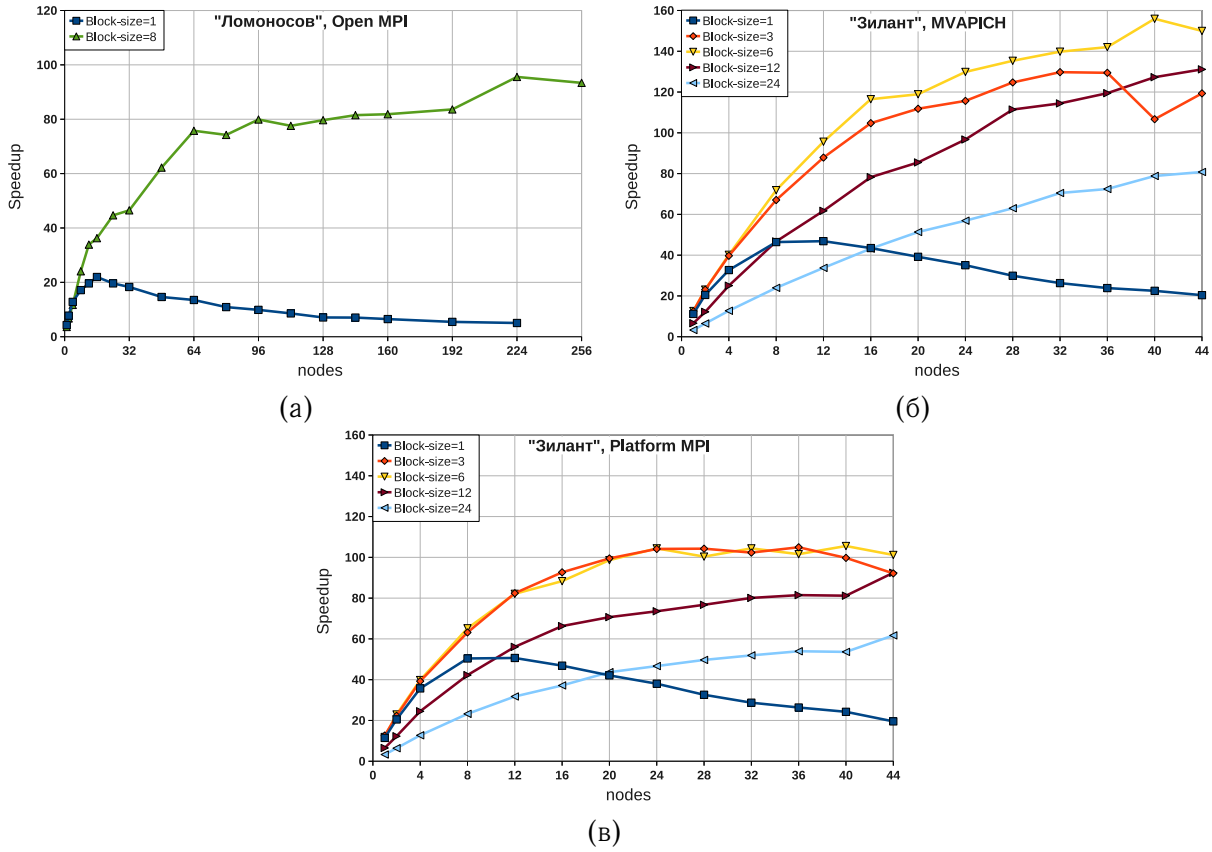


Рис. 3. Графики зависимости масштабируемости гибридной версии приложения от количества вычислительных узлов: (а) “Ломоносов”, (б) “Зилант”: MVAPICH, (в) “Зилант”: Platform MPI.

лее удачным оказывается использование блоков по 3 или 6 процессов, с соответствующей схемой привязки процессов к ядрам. Однако, даже в этом случае получаемое ускорение чуть более 100 раз сравнимо с параметрами масштабируемости для MPI приложения и заметно ниже значений для библиотеки MVAPICH.

Неудовлетворительными оказываются результаты для блоков из 24 процессов. Несмотря на то, что в этом случае на узле остается всего один коммуникационный MPI-процесс, что заметно снижает нагрузку на сеть, возникает большой трафик к областям общей памяти, расположенной в обеих NUMA-node, что существенно ухудшает результаты масштабируемости. Полученные данные наглядно свидетельствуют о необходимости доработки гибридной версии кода с учетом особенностей NUMA-архитектуры.

5. Заключение

В работе представлены результаты исследования масштабируемости тестового приложения решения системы линейных алгебраических уравнений с сильно-разреженной матрицей на трех различных аппаратных платформах. Тесты показали, что для эффективного использования многоядерных процессоров необходимо использовать привязку вычислительных процессов к физическим ядрам процессоров, причем при выборе схемы привязки следует исходить из архитектуры используемой аппаратной платформы. Выбор библиотеки MPI может существенным образом сказаться на масштабируемости приложения. При этом для MPI и для гибридной версий приложения наиболее удачный выбор библиотеки MPI может оказаться различным. При использовании только одного вычислительного узла MPI-приложения могут оказаться столь же эффективными, как и приложения, работающие через общую

память. При реализации гибридных моделей на NUMA-архитектуре необходимо минимизировать размер общих для всех процессов областей памяти, чтобы сократить количество обращений в память другого NUMA-node через межпроцессорный интерфейс.

Автор благодарит Вл.В. Воеводина за предоставленную возможность проведения тестирования на вычислительных системах СКИФ МГУ “Чебышёв” и “Ломоносов” суперкомпьютерного комплекса Московского университета, а так же компанию “Т-Сервисы” за предоставленный доступ к вычислительной системе “Зилант”.

Литература

1. Saad, Y. Iterative methods for sparse linear systems, 2nd edition. SIAM, 2003.
2. Trottenberg U., Oosterlee C.W., Schuller A. Multigrid. New York: Academic Press, 2001.
3. Krasnopolsky, B. The reordered BiCGStab method for distributed memory computer systems // Procedia Computer Science, 2010. Vol. 1. P. 213–218.
4. Краснопольский Б.И. Сравнение результатов исследования эффективности переупорядоченного метода BiCGStab на вычислительных системах «Чебышёв» и «Ломоносов» // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г. Новороссийск) – М.: Изд-во МГУ, 2010. – 694 с. С. 401-407.
5. Pellegrini F., Chevalier, C. SCOTCH library: software package and libraries for graph, mesh and hypergraph partitioning, static mapping, and parallel and sequential sparse matrix block ordering. URL: http://gforge.inria.fr/projects/scotch/

Разработка и анализ высокопроизводительных параллельных алгоритмов решения кооперативных игр*

М.Ю. Нестеренко, А.С. Кириллов

Государственное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет»

В работе рассматривается подход к решению вычислительно-сложной задачи – построение оптимальной коалиции и распределение выигрыша в кооперативной игре заданной множеством биматричных игр для *n* игроков с помощью решения стратегических игр и использованием параллельных вычислительных технологий.

1. Актуальность

Возросшие запросы производства и экономики, развитие информационных систем и технологий предъявляют все большие требования к процессу принятия решений и управления в различных областях человеческой деятельности. При создании систем поддержки принятия решений активно используется аппарат теории игр. При этом очень актуальна проблема разработки эффективных алгоритмов для поиска оптимальных стратегий в игровых моделях. На практике требуется, чтобы алгоритмы были способны обрабатывать большое количество информации в режиме реального времени.

Среди направлений теории игр особое место занимают кооперативные игры, например, в разработке оптимальных стратегий производства, безопасности систем, эффективной ценовой политики и др. В кооперативных играх игроки имеют возможность принимать совместные решения и, в некоторых случаях, перераспределять выигрыши между собой. В кооперативной игре задаются возможности и предпочтения различных групп игроков (коалиций), для которых строятся оптимальные стратегии и задаются распределения между ними суммарных выигрышей. Результатом решения кооперативной игры являются оптимальная коалиция, в которую следует вступить игрокам, их оптимальные смешанные стратегии и выигрыш коалиции. Особого внимания заслуживают задачи игрового моделирования. Примером подобной задачи может быть моделирование конкурентного рынка в различных отраслях, где игроки имеют возможность объединяться в коалиции, конкурировать за соответствующие ниши рынка.

Разработка параллельных алгоритмов для решения задач теории игр – новое направление в алгоритмической теории игр. На сегодняшний день имеются лишь единичные публикации в этой области, например работы Уиджера и Гросу для некооперативных игр [1,2,3]. Для решения кооперативных игр параллельные алгоритмы пока не разработаны.

Существуют различные способы решения кооперативных игр, например: решение на основе функции полезности Неймана-Моргенштерна, решение игры в виде С-ядра и в стратегиях угроз, однако эти подходы не определяют оптимальных стратегий игроков внутри коалиций. А решение игры в виде С-ядра дает целое множество допустимых решений, не определяя оптимального решения. Кроме того, эти подходы рассматривают кооперативные игры на основе характеристических функций и опускают способы получения значений характеристической функции, что само по себе является отдельной и трудоемкой задачей.

Как видно, решение задач кооперативных игр процесс трудоемкий, и, с увеличением числа игроков, трудоемкость решения задачи только повышается [9].

* Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг.

2. Понятие кооперативной игры

Кооперативной игрой [5,6,7,9,12,13] n лиц называется игра вида $\Gamma = (N, v)$, где $N = \{1, 2, ..., n\}$ – множество игроков, которым разрешено вести переговоры и объединяться в коалиции, а $v : 2^N \to R$ – характеристическая функция, определяющая наибольший уверенно получаемый выигрыш $v(K)$ для каждой возможной коалиции $K \subseteq N$ [7].

3. Алгоритм решения кооперативной игры, заданной биматричными играми

Обозначим через *N* множество всех игроков, *N={1, 2, ..., n}*, а через *K* – любое его подмножество. Пусть игроки из *K* договариваются между собой о совместных действиях и, таким образом, образуют одну коалицию. Образовав коалицию, множество игроков *K* действует как один игрок против остальных игроков, и выигрыш этой коалиции зависит от применяемых стратегий каждым из *n* игроков.

Функция *v*, ставящая в соответствие каждой коалиции *K* наибольший, уверенно получаемый ею выигрыш *v(K)*, называется характеристической функцией игры [9]. Всего значений характеристической функции может быть 2^n, по количеству коалиций, причем коалиция с номером 0 является пустой коалицией, т.е. состоящей из пустого множества игроков, и ее выигрыш по условию всегда равен нулю.

В большинстве работ рассматриваются только классические кооперативные игры, при этом в них не приводится способ получения значений характеристических функций. При этом считается, что значения характеристических функции заранее известны, либо их можно легко получить на основании какой-либо информации об игроках.

Но, в общем случае, это не так. В практике принятия решений заранее неизвестны значения характеристических функций, имеется только предполагаемое множество стратегий игроков, т.е. их возможные действия, которые приводят к получению прибыли или приводят к убыткам. И в таких случаях получение значений характеристических функций становится сложной и трудоемкой задачей.

Одним из способов решения этой задачи является использование биматричных игр. При этом кооперативная игра задается множеством биматричных игр между каждой парой игроков. Данный подход позволяет вычислить значения характеристической функции.

Назовем систему $\Gamma_b = \{N, B\}$, состоящую из множества игроков *N* и набора биматричных игр $B = \{B_{xy}, x \in N, y \in N, x \neq y\}$, кооперативной игрой на основе биматричных игр. Для решения такой игры необходимо иметь или составить исходный набор биматричных игр.

Легко увидеть, набор биматричных игр будет состоять из $n \cdot (n-1)$ матриц, как набор размещений из *n* по 2.

Оговоримся сразу, что количество стратегий всех игроков одинаково и равно *k*, в противном случае можно добавить пустые «фиктивные» стратегии.

Набор биматричных игр *B* строится исходя из исходных данных задачи для каждой пары игроков $x, y \in N$ в виде матрицы размера $k \times k$:

$$B_{xy} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{k1} & b_{k2} & \dots & b_{kk} \end{pmatrix},$$

где элементы матрицы $b_{ij}(i, j = \overline{1,k})$ – выигрыши игрока *x,* при выборе игроком *x* стратегии *i,* в игре с игроком *y,* при выборе игроком *y* стратегии *j*.

Согласно [9,10,12], характеристическая функция для коалиции S определяет максимальный уверенно получаемый ею выигрыш. Следовательно, для вычисления этого выигрыша целесообразно считать оставшуюся часть игроков, не входящих в S, в первом приближении антагонистически настроенными и действующими совместно. В результате чего получаем матричную игру A_{SP} коалиции $S \subseteq N$ и антикоалиции $P = N - S = \{p \in N, p \notin S\}$, которая строится на основании биматричных игр.

При этом выигрыш коалиции *S* в этой матричной игре будет равен значению характеристической функции для данной коалиции S. При решении игры в смешанных стратегиях $\underline{v} = \overline{v} = v(S) = | A_{SP} |$, где $| A_{SP} |$ – решение матричной игры A_{SP} в смешанных стратегиях.

Матрицы матричной игры A_{SP} коалиции *S* в игре с остальными участниками будет иметь размер:

$$k^m \times k^{n-m},$$

где *k* – количество стратегий одного игрока, а *m* – количество игроков в текущей коалиции. Такой размер матриц обусловлен тем, что матрицы A_{SP} строятся на основании комбинированных стратегий коалиций *S*.

Матрицы матричной игры A_{SP} будут иметь следующий вид:

$$A_{SP} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1k^{n-m}} \\ a_{21} & a_{22} & \dots & a_{2k^{n-m}} \\ \dots & \dots & \dots & \dots \\ a_{k^m 1} & a_{k^m 2} & \dots & a_{k^m k^{n-m}} \end{pmatrix}, \text{ при } 0 < m \leq n,$$

где a_{ij} - выигрыш игроков коалиции *S* в игре с антикоалицией *P*, причем *i* и *j* – комбинированные стратегии коалиций.

Элемент a_{ij} вычисляется по следующему алгоритму:

1) из номера комбинированной стратегии коалиции *S,* равного *i,* отнимают 1 и переводят из десятичной системы счисления в систему счисления по основанию *k* (количество стратегий каждого игрока) записанную в обратной последовательности, обозначим такую систему счисления как система счисления по основанию *k'*, получившееся число имеет вид:

$$(i-1)_{10} = (c_m c_{m-1} \dots c_1)_{k'}$$

где $c_v + 1 \ (v = \overline{1,m})$ – стратегия *v*-го игрока коалиции *S*.

2) из номера комбинированной стратегии антикоалиции *P* равного *j* отнимают 1 и переводят из десятичной системы счисления в систему счисления по основанию *k*, получившееся число имеет вид:

$$(j-1)_{10} = (c_{n-m} c_{n-m-1} \dots c_1)_{k'}$$

где $c_w + 1 \ (w = \overline{1,n-m})$ – стратегия *w*-го игрока антикоалиции *P*.

3) на основании найденных стратегий каждого игрока для игровой ситуации (*i,j*) находят

значение выигрыша коалиции *S* при игре против антикоалиции *P*:

$$a_{ij} = \sum_{b_{c_x c_y} \in B_{xy}, x, y \in S, x \neq y} b_{c_x c_y} + \sum_{b_{c_x c_y} \in B_{xy}, x \in S, y \in P} b_{c_x c_y}$$

Для решения матричной игры, в основном, используют задачу линейного программирования [9,12].

В результате решения матричной игры получаем значение *v*, которое будет равно значению характеристической функции *v(S)* для коалиции *S*. После получения характеристической функции необходимо проверить игру на существенность, для чего необходимо проверить, удовлетворяет ли полученная характеристическая функция свойству супераддитивности: $\forall K, S \subseteq N \;\; K \cap S = \varnothing \Rightarrow \; v(K \cup S) > v(K) + v(S)$

Если характеристическая функция удовлетворяет этому свойству, то такая игра является существенной кооперативной игрой, и, соответственно, экономически устойчивой, и для нее имеет смысл в дальнейшем производить поиск дележа.

Существует несколько способов решения кооперативных игр и получения дележа: C-ядро, решение Неймана-Моргенштерна и вектор Шепли.

C-ядро [4,5] представляет собой множество недоминируемых дележей кооперативной игры. Оно устойчиво к отклонениям любой коалиции игроков.

Однако множество может быть пустым, тогда соответствующая кооперативная игра не будет иметь решения в рамках понятия C-ядра.

C-ядро в кооперативной теории представляет собой основной множественный принцип оптимальности. Однако остается проблема выбора подходящего дележа в каждом конкретным случае. Также C-ядро на практике часто оказывается пустым [9], поэтому были предложены другие методы решения кооперативной игры.

НМ-решение [5,6]. НМ-решение кооперативной игры – результат работы ученых Дж. фон Неймана и О. Моргенштерна [10]. Они предложили в качестве решения рассматривать множество подмножеств дележей, обладающих определенными свойствами, каждое из них носит название НМ-решения.

НМ-решение обладает внутренней и внешней устойчивостью. Внутренняя устойчивость обеспечивается равноправностью его дележей, т.е. невозможностью найти пару дележей, чтобы один из них доминировал над другим. Смысл внешней устойчивости в том, что произвольный дележ, не принадлежащий НМ-решению, доминируется дележом из НМ-решения.

Заметим, что если игра сбалансирована, то НМ-решение содержит в себе в качестве подмножества *C*-ядро [5].

Существуют игры, не имеющие НМ-решения, но чаще всего кооперативные игры имеют огромное количество НМ-решений. Трудоемкость нахождения ограничивает их практическое использование.

Также имеет место проблема совместного выбора участниками одного из нескольких НМ-решений, в рамках которого будет определяться вектор распределения выигрыша. Это ограничивает возможность применения данного подхода.

Вектор Шепли [5,7,9] – принцип оптимальности распределения выигрыша между игроками в задачах теории кооперативных игр. Представляет собой вектор дележа $\varphi[v]$, в котором выигрыш каждого игрока равен его среднему вкладу в благосостояние коалиции при определенном механизме ее формирования.

Одним из недостатков вектора Шепли является то, что он не является селектором *C*-ядра в общем случае, хотя обеспечивает справедливое (в смысле эффективного вклада участников) распределение выигрыша.

Для нахождения дележа используется принцип оптимальности распределения выигрыша между игроками в задачах теории кооперативных игр, называемый вектором Шепли.

4. Параллельный алгоритм решения кооперативной игры заданной биматричными играми

На основании вышесказанного нами разработан параллельный алгоритм решения кооперативной игры, заданной набором биматричных игр, с использованием технологии MPI.

Суть разработанного параллельного алгоритма состоит в следующем.

Каждым процессом выполняется считывание матриц, задающих биматричные игры, причем биматричные игры задаются в виде набора текстовых файлов, при этом нумерация файлов идет от 1 до *n*(n-1):* номер 1 соответствует набору игроков (1,2), номер 2 соответствует (1,3) и т.д. до *n*(n-1),* который соответствует набору (*n*,*n-1*).

После прочтения исходных данных каждый процесс вычисляет свой диапазон коалиций, с которыми он будет в дальнейшем работать. Для каждой коалиции из своего диапазона процесс формирует матричную игру по алгоритму, описанному выше, и решает полученную матричную игру с помощью симплекс-метода. Номера игроков, входящих в коалицию, определяются по номерам позиций единичных элементов двоичного представления порядкового номера коалиции.

Как только все процессы закончат расчет на своих диапазонах, результаты отсылаются нулевому процессу, который затем проверяет игру на существенность, и, если игра существенна, то вычисляется дележ по Шепли. Затем все результаты записывается в выходной файл.

Схема работы параллельного алгоритма представлена на рис. 1.

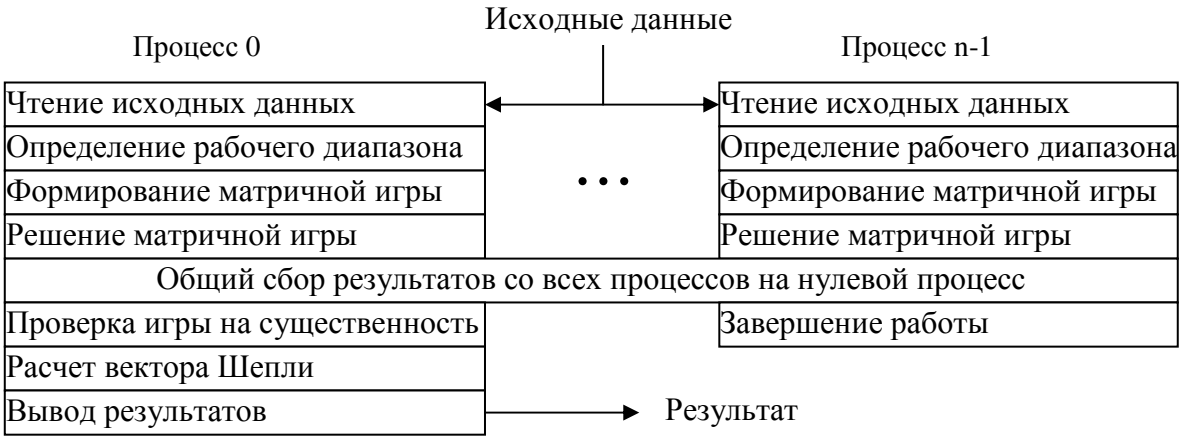


Рис. 1. Схема работы параллельного алгоритма

Тестирование разработанного параллельного алгоритма производилось на вычислительном кластере Оренбургского Государственного Университета, состоящего из следующих компонентов:

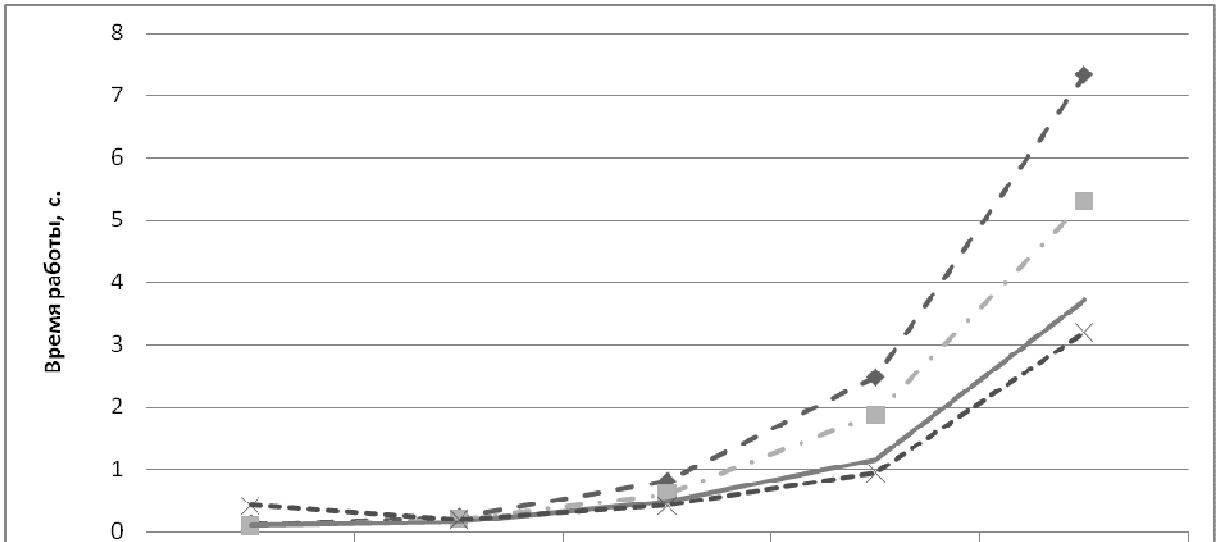
- головного узла (мастера) на базе двух процессоров Intel Xeon 3.2 ГГц, оперативной памятью 2 Гб и дисковым пространством 750 Гбайт;
- шести двухпроцессорных вычислительных узлов на базе Intel Xeon 3.2 ГГц, оперативной памятью 2 Гб и дисковым пространством 80 Гбайт (всего 12 вычислительных ядер);
- четырех двухпроцессорных вычислительных узлов на базе четырехъядерных процессоров Intel Xeon 5440, оперативной памятью 16 Гб и дисковым пространством 320 Гбайт (всего 32 вычислительных ядра).

Все узлы объединены вычислительной сетью Infiniband со скоростью 10 Гбит/с и управляющей сетью Ethernet со скоростью 1 Гбит/с. Суммарная пиковая производительность кластера составляет 451,6 GFLOP/s.

На кластере установлено следующее системное программное обеспечение:

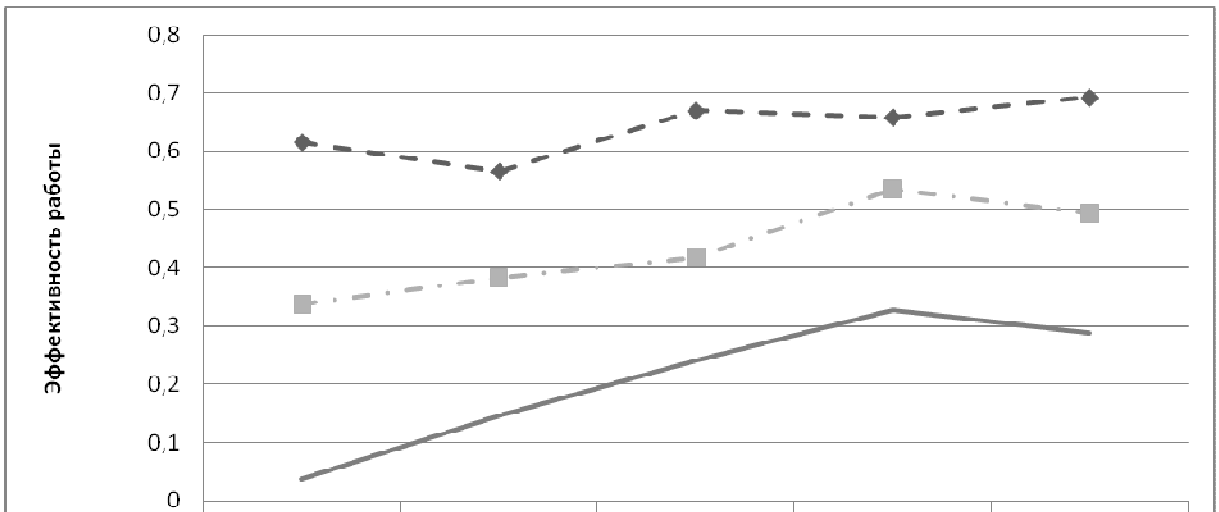
- операционная система SUSE Linux 11;
- дистрибутивы MPI: OpenMPI 1.3.2, Mvapich2-0.9.8p3;
- компиляторы: GNU GCC 4.3.2, Intel C/C++/Fortran 11.0;
- менеджер ресурсов Torque-2.3.6.

Для запуска тестовых задач использовалось четыре двухпроцессорных вычислительных узла на базе процессоров Intel Xeon 5440, на каждом узле использовалось только два ядра. Такая методика тестирования была связана с тем, что решаемая задача кооперативных игр очень требовательна к объему оперативной памяти, и для каждого экземпляра задачи выделялось до 6 Гб памяти.



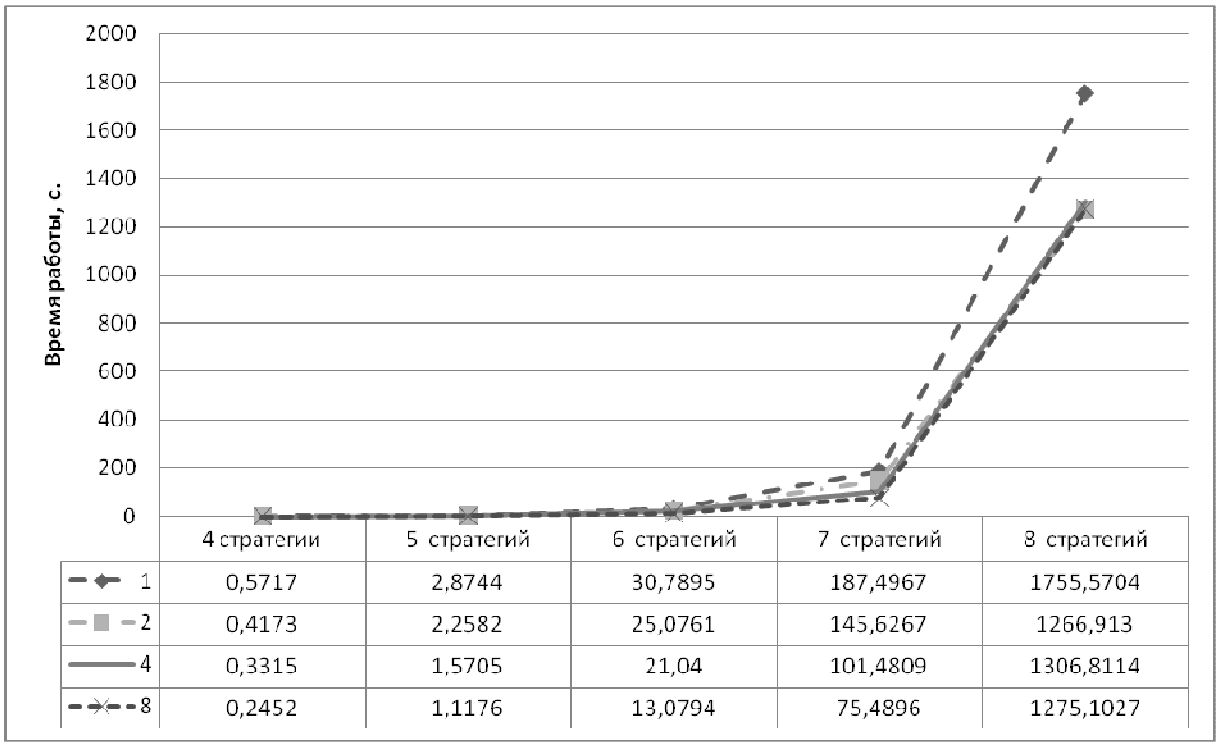
	4 стратегии	5 стратегий	6 стратегий	7 стратегий	8 стратегий
1	0,1228	0,2254	0,8023	2,4732	7,3355
2	0,0997	0,1993	0,599	1,8774	5,2963
4	0,0911	0,1474	0,4805	1,1566	3,7116
8	0,4156	0,1932	0,4165	0,9472	3,19

Рис. 2. График зависимости времени работы программы от количества процессоров и количества стратегий для пяти игроков



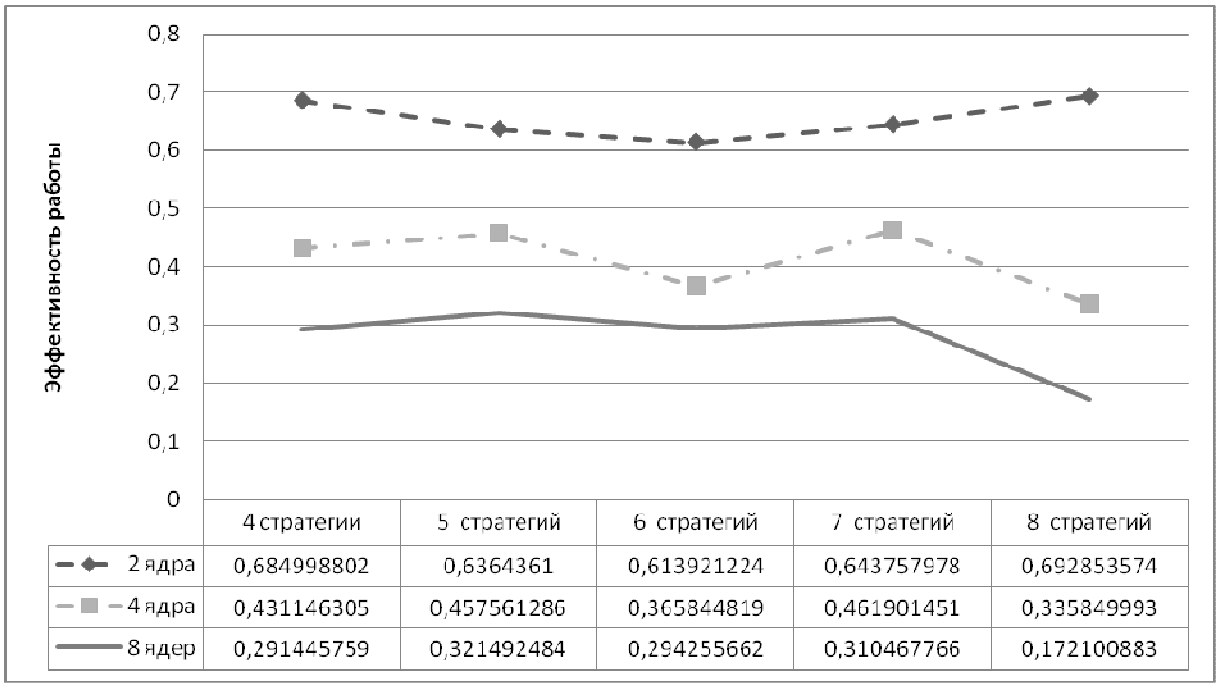
	4 стратегии	5 стратегий	6 стратегий	7 стратегий	8 стратегий
2 ядра	0,615847543	0,565479177	0,669699499	0,658676894	0,692511753
4 ядра	0,336992316	0,38229308	0,417429761	0,534584126	0,494092844
8 ядер	0,036934552	0,145833333	0,240786315	0,326383024	0,287441223

Рис. 3. График эффективности работы параллельной программы в зависимости от количества процессоров и количества стратегий для пяти игроков



	4 стратегии	5 стратегий	6 стратегий	7 стратегий	8 стратегий
1	0,5717	2,8744	30,7895	187,4967	1755,5704
2	0,4173	2,2582	25,0761	145,6267	1266,913
4	0,3315	1,5705	21,04	101,4809	1306,8114
8	0,2452	1,1176	13,0794	75,4896	1275,1027

Рис. 4. График зависимости времени работы программы от количества процессоров и количества стратегий для шести игроков



	4 стратегии	5 стратегий	6 стратегий	7 стратегий	8 стратегий
2 ядра	0,684998802	0,6364361	0,613921224	0,643757978	0,692853574
4 ядра	0,431146305	0,457561286	0,365844819	0,461901451	0,335849993
8 ядер	0,291445759	0,321492484	0,294255662	0,310467766	0,172100883

Рис. 5. График эффективности работы параллельной программы в зависимости от количества процессоров и количества стратегий для шести игроков

Тестирование разработанного алгоритма производилось последовательно для 4, 5, 6, 7 и 8 игроков, при этом каждый игрок имел 4, 5, 6, 7 и 8 стратегий. Для каждой задачи формировался произвольный набор исходных данных. Полученный набор исходных данных затем решался последовательно на 1, 2, 4 и 8 процессорах, при этом замерялось время выполнения каждой задачи.

Как видно из графиков, представленных на рисунках 2, 3, 4 и 5, не удалось добиться приемлемого масштабирования разработанного параллельного алгоритма.

Это, прежде всего, связано с тем, что формирующиеся матричные игры требуют различного времени решения и нет сбалансированного метода распределения матричных игр по процессам, для равномерного распределения нагрузки на процессы. Причем, существенно на размер матричной игры и, соответственно, на время решения задачи влияет количество игроков в исследуемой коалиции. Так же при реализации данного параллельного алгоритма использовалась недостаточно эффективная реализация алгоритма решения задачи линейного программирования, что так же могло привести к низкому показателю эффективности.

Однако, в среднем, эффективность разработанного параллельного алгоритма (отношение времени работы последовательного алгоритма к произведению времени работы параллельного алгоритма и количества процессов) составляет порядка 40%.

В дальнейшей работе планируется использовать биматричную игру двух игроков *S* и *P* для уточнения значения характеристической функции для коалиции S. Биматричная игра позволяет учитывать интересы остальных игроков, не входящих в *S*.

Литература

1. Widger, J. and Grosu, D. Computing Equilibria in Bimatrix Games by Parallel Support Enumeration. In Proceedings of the 2008 international Symposium on Parallel and Distributed Computing (July 01 - 05, 2008). ISPDC. IEEE Computer Society, Washington, DC, pp. 250-256.
2. Widger, J. and Grosu, D. Computing Equilibria in Bimatrix Games by Parallel Vertex Enumeration – Parallel Processing, 2009. ICPP '09. International Conference (22-25 Sept. 2009) pp. 116 - 123.
3. Widger, J. and Grosu, D. Parallel Computation of Nash Equilibria in N-Player Games, Computational Science and Engineering, IEEE International Conference 2009 pp. 209-215.
4. Бондарева О.Н. Некоторые применения методов линейного программирования к теории кооперативных игр // Проблемы кибернетики. – 1963. – Т. 10. – С. 119 – 140.
5. Губко М.В. Управление организационными системами с коалиционным взаимодействием участников. М.: ИПУ РАН (научное издание), 2003. – 140 с.
6. Губко М.В., Новиков Д.А. Теория игр в управлении организационными системами. Издание 2. – М.: 2005. – 138 с.
7. Данилов В.И. Лекции по теории игр. – М.: Российская экономическая школа, 2002. – 140 с.
8. Ермаков В.И., Кривенцова Н.Н., Барбаумов В.Е. Справочник по математике для экономистов: Учеб пособие / Под ред. проф. В.И. Ермаков. – М.: ИНФРА-М, 2007. – 464 с.
9. Крушевский А.В. Теория игр. – Киев.: Вища Школа, 1977. – 216 с.
10. Нейман Д., Моргенштерн О. Теория игр и экономическое поведение. – М.: Наука, 1970.
11. Нестеренко М.Ю., Леонов Д.В., Болгова Е.В., Кириллов А.С. Разработка программного обеспечения для моделирования конкурентного рынка на кластерных системах «Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики» март–апрель 2008 № 54 Технологии высокопроизводительных вычислений и компьютерного моделирования. – Санкт-Петербург, ГОУ ВПО «СПбГУ ИТМО», 2008. – с. 156 – 161.
12. Оуэн Г. Теория игр. – М.: Мир, 1971. – 229 с.
13. Шеллинг Т. Стратегия конфликта. – М.: ИРИСЭН, 2007. – 366 с.

Инструментальные средства организации параллельных вычислений в пакетах прикладных программ*

А.П. Новопашин, И.А. Сидоров, С.А. Горский

Учреждение Российской академии наук
Институт динамики систем и теории управления Сибирского отделения РАН

В работе представлена базовая модель планирования параллельных крупноблочных вычислений, в качестве основного формализма которой используется аппарат булевых уравнений. На основе этой модели разработаны и программно реализованы инструментальные средства, предназначенные для создания и применения параллельных и распределенных пакетов прикладных программ.

1. Введение

Происходящее в последнее десятилетие широкомасштабное внедрение в практику расчетных работ параллельной вычислительной техники (в особенности кластерных архитектур) способствует возрождению интереса к пакетной проблематике, что влечет, в свою очередь, необходимость реконструкции старых и создания новых средств и методов организации параллельных вычислений с использованием пакетов прикладных программ (ППП).

В качестве самостоятельного научного направления пакетная проблематика сложилась в 70-80-х годах прошлого столетия. Если первые ППП представляли собой простые тематические подборки программ для решения отдельных задач в той или иной предметной области, то современные ППП являются сложными программными комплексами, включающими:

1) прикладное программное обеспечение (функциональное наполнение) пакета в виде библиотеки прикладных программных модулей;

2) системное программное обеспечение для организации решения пользовательских задач с использованием функционального наполнения, включающее средства автоматизации планирования вычислений (в том числе параллельных);

3) специализированные языковые средства описания предметной области и постановки пользовательских задач.

Изначальная ориентация на широкий класс задач предметной области, в которой работает прикладной специалист, является одной из ключевых особенностей ППП. Сочетание в пакете разнообразных моделей и алгоритмов достигается путем использования принципа модульной организации функционального наполнения пакета. Входящий в состав ППП модуль представляется, как правило, в виде автономной программной единицы, записанной на традиционном языке программирования и обеспечивающей решение некоторой самостоятельной задачи. Предполагается, что взаимодействие модулей возможно только на уровне их входных и выходных параметров. Использование принципа модульности позволяет заменить написание программы (в традиционном понимании) ее конструированием из готовых программных блоков крупного размера. Расширение класса задач, решаемых пакетом, может достигаться за счет подключения к ППП вновь создаваемых модулей.

В настоящее время серьезное внимание уделяется проблеме создания, так называемых, интеллектуальных ППП, позволяющих конечному пользователю формулировать свою задачу в содержательных терминах без указания алгоритма ее решения. Синтез схемы решения и сборка целевой программы производятся автоматически, при этом детали вычислений остаются скрытыми от пользователя. Такой способ организации вычислений, берущий свое начало в работах Э.Х. Тыугу [1], С.С. Лаврова [2], Г.А. Опарина [3] и других, известен как концептуальное (структурное, сборочное) программирование. Он предполагает наличие вычислительной модели (схемы) предметной области, позволяющей описывать вычислительные возможности ППП

* Работа выполнена при частичной финансовой поддержке программы фундаментальных исследований Президиума РАН №13 (проект СО РАН №3).

для решения задач определенного класса. Такую вычислительную модель можно определить как совокупность значимых величин (параметров) предметной области и функциональных отношений между ними. Таким образом, вычислительная модель, по сути, определяет правила применения и сочетания модулей в процессе решения задачи и позволяет автоматически осуществлять планирование вычислений по непроцедурной постановке задачи вида: "по заданным значениям параметров $x_1, x_2, \ldots, x_k$ вычислить значения параметров $y_1, y_2, \ldots, y_r$".

Большое разнообразие разработанного прикладного программного обеспечения требует создания универсальных (изначально не привязанных к конкретной предметной области) инструментальных средств, позволяющих с наименьшими трудозатратами объединять имеющийся программный код в рамках одного ППП. Кроме этого, такой инструментарий должен обеспечивать корректность и отказоустойчивость вычислений в рамках ППП, эффективность использования функционального наполнения пакета, высокий уровень интеллектуализации с учетом современных требований, включая автоматизацию процессов выявления внутреннего параллелизма вычислительной модели, синтеза параллельных планов решения задач и генерации на основе этих планов параллельных прикладных программ. В настоящее время наблюдается дефицит инструментальных средств, обладающих перечисленными характеристиками.

Следующие разделы статьи посвящены описанию базовой вычислительной модели планирования параллельных вычислений и использующих эту модель инструментальных средств разработки параллельных и распределенных пакетов прикладных программ.

2. Базовая модель планирования вычислений

В качестве базы знаний планировщика используется структура $KB=(F, Z, T, Y, In, Out, Com)$, где:

$F = \{F_1,\ldots,F_n\}$ – множество имен программных модулей, реализующих операции предметной области;

$Z = \{Z_1,\ldots,Z_m\}$ – множество имен параметров (значимых величин) предметной области, являющихся входными или выходными параметрами модулей из F; с каждым модулем F_i связано два множества параметров A_i, $B_i \subset Z$, называемых соответственно входом и выходом;

$T = \{T_1,\ldots,T_r\}$ – множество типов параметров;

$Y = \{Y_1,\ldots,Y_p\}$ – множество узлов вычислительной системы (ВС);

$In \subset F\times Z$, $Out \subset F\times Z$ – отношения, отражающие взаимосвязь модулей с данными соответственно по входу и выходу;

$Com \subset F\times Y$ – отношение, определяющее статические связи между программными модулями и узлами ВС.

Предполагается, что база знаний KB обладает высоким уровнем внутреннего параллелизма и является избыточной в том смысле, что для решения задачи вычисления требуемого набора целевых параметров B_0 по заданному множеству параметров - исходных данных A_0 используется только часть модулей из F , и/или эта задача имеет несколько альтернативных планов решения.

Отношения In, Out и Com представлены в виде трех булевых матриц A, B и C размерности $n\times m$, $n\times m$ и $n\times p$ соответственно. Элементы матриц формируются следующим образом: $a_{ij}=1$ ($b_{ij}=1$), если параметр Z_j является входным (выходным) для модуля F_i; $c_{ij}=1$, если модуль F_i установлен в узле Y_j. Строки и столбцы матриц A, B и C являются двоичным представлением соответствующих подмножеств множеств F , Z и Y , связанных отношениями In, Out и Com.

План вычислений представляется в виде матрицы X $k\times n$ булевых переменных x_{tj}. Значение x_{jj} равное единице означает, что программный модуль F_j выполняется на t-м шаге плана X, где $t\in N=\{1,2,...\}$ играет роль дискретного времени. Общая длина плана равна k, его строка задает множество параллельно исполняемых модулей, а столбцы соответствуют множеству модулей F.

Информационно-логические связи между параметрами и модулями, входящими в план, можно представить в виде двудольного ориентированного графа, включающего два непересекающихся множества вершин (**Рис. 1**): множество вершин-параметров и множество вершин-модулей. Вершины-параметры, как и вершины-модули, являются попарно несмежными.

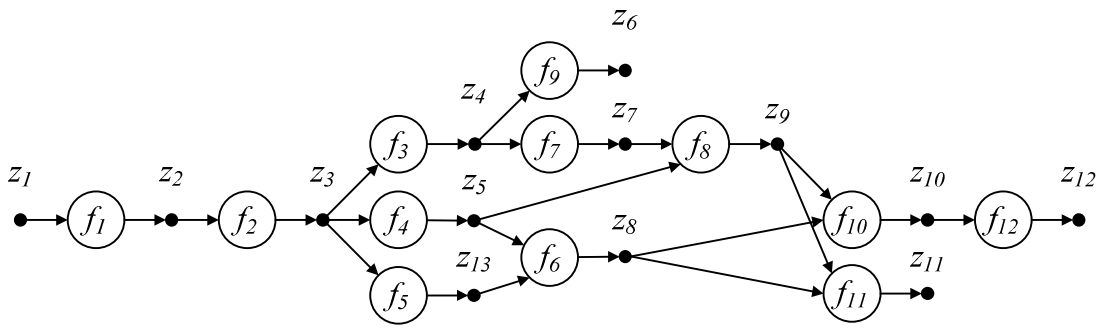


Рис. 1. Пример двудольного ориентированного графа, отражающего информационно-логические зависимости между параметрами и модулями ППП

К значениям элементов матрицы X предъявляются ограничения, которые записываются в форме булевых уравнений и представляют собой требования к искомому параллельному плану, синтезируемому в процессе решения системы таких уравнений. К числу обязательных требований относятся следующие:

1) допустимость (модули в плане должны быть упорядочены таким образом, чтобы каждый из них к моменту своего запуска был обеспечен необходимыми входными данными);

2) бесповторность (каждый модуль может включаться в план не более одного раза);

3) неизбыточность (исключение любого модуля из плана приводит к недопустимому плану);

4) оптимальность (длина плана должна быть равна заданной величине k).

Кроме обязательных требований, при необходимости могут быть сформулированы и добавлены в общую систему ограничений дополнительные условия, позволяющие при построении плана учитывать временные задержки, связанные с исполнением программных модулей, ресурсные ограничения ВС (число узлов или процессоров), привязку модулей к узлам ВС, требование синхронности запуска модулей и другие ограничения. В [4] для большинства из перечисленных ограничений получены формулировки в виде булевых уравнений.

Преимущества такого подхода к планированию крупноблочных вычислений, при котором условия задачи формулируются в виде системы булевых уравнений (ограничений), а искомый план исполнения модулей является ее решением, состоит в том, что он позволяет, во-первых, получать параллельные планы требуемой длины, во-вторых, учитывать разнообразные ограничения, предъявляемые к плану, и, наконец, в-третьих, использовать существующие эффективные решатели булевых уравнений (или SAT-решатели), которые в ряде случаев обгоняют по быстродействию специализированные алгоритмы планирования.

С практической точки зрения можно выделить два типовых подхода к организации функционального наполнения пакета и распределения модулей по узлам ВС:

1) Все вычислительные модули содержатся в единой библиотеке, установленной на рабочей станции разработчика пакета. По непроцедурной постановке задачи производится автоматическое планирование параллельной схемы вычислений (с соблюдением установленных ограничений). В соответствии с полученной схемой осуществляется синтез целевой параллельной программы, ее компиляция и запуск на заданном числе узлов однородной ВС.

2) Вычислительные модули размещаются в узлах ВС, отличающихся программно-аппаратными характеристиками, что накладывает дополнительные требования к организации функционального наполнения ППП. При этом часть модулей может быть неотчуждаема от владельцев узлов, допускается множественность установки модулей в узлах ВС. Постановка задач осуществляется в процедурном виде, а исполнение построенных схем решения выполняется в режиме интерпретации.

Далее в статье рассматриваются инструментальные средства, реализующие описанные подходы к организации функционального наполнения параллельных и распределенных пакетов прикладных программ.

3. Инструментальный комплекс ORLANDO

Данный инструментарий ориентирован на создание ППП для однородных UNIX-кластеров с возможностью автоматической генерации параллельных программ для решения вычислительных задач в виде композиции готовых функциональных блоков. На **Рис. 2** представлена архитектура инструментального комплекса (ИК) ORLANDO, которая включает следующие основные компоненты: многооконный текстовый редактор, синтаксический анализатор, планировщик, генератор, подсистему компиляции, подсистему запуска, базу расчетных данных.

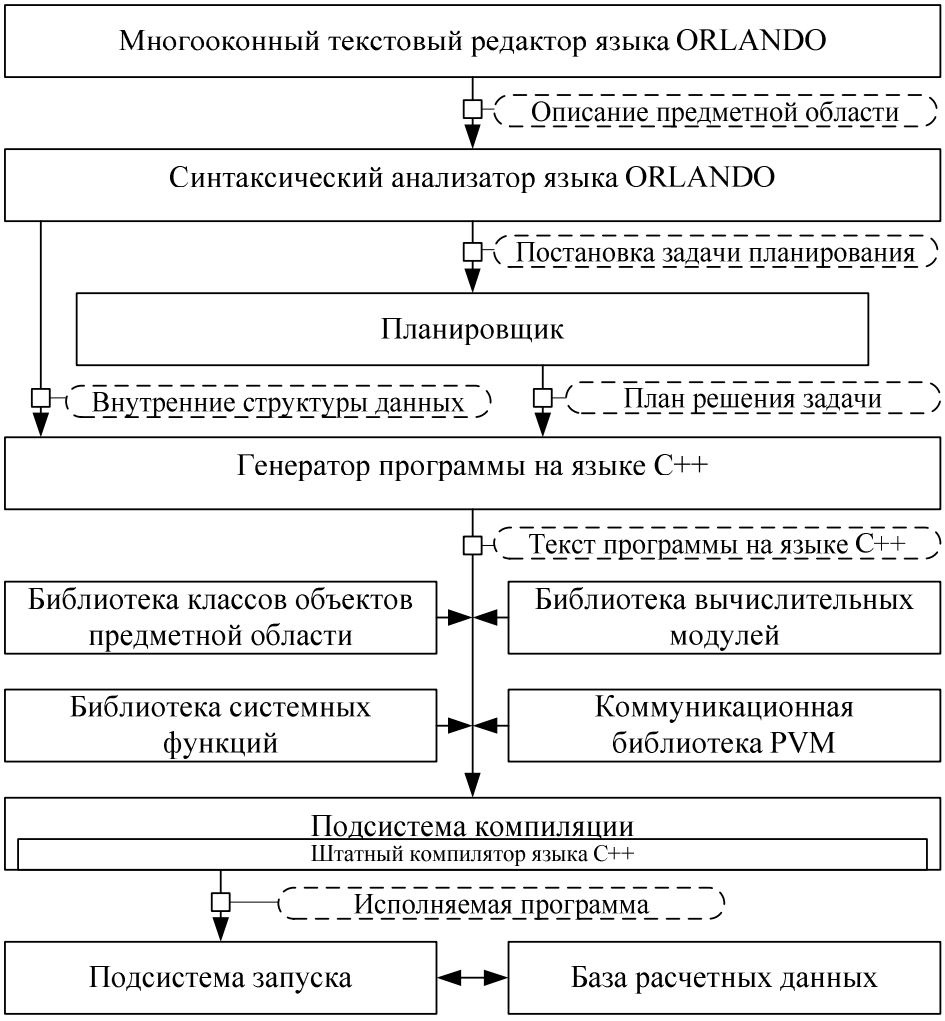


Рис. 2. Архитектура ИК ORLANDO

Высокоуровневый язык ORLANDO [5] предоставляет средства для описания объектов предметной области (параметров и операций), отношений между ними и обеспечивает на основе этих описаний постановку вычислительных задач в непроцедурном виде. Конструкции языка носят исключительно описательный (декларативный) характер и не предполагают указания порядка исполнения модулей, участвующих в решении задачи. Все информационно-логические связи между модулями выявляются на этапе трансляции описания предметной области.

Текстовый редактор предоставляет пользователю набор функций, необходимых для быстрого и удобного формирования описания предметной области и постановок задач с использованием языковых конструкций ORLANDO, а также функций для взаимодействия с другими компонентами инструментального комплекса – транслятором, подсистемами компиляции и запуска, базой расчетных данных.

Синтаксический анализатор осуществляет разбор описания предметной области на языке ORLANDO, проверку корректности и целостности такого описания и его перевод во внутреннее представление системы планирования.

Планировщик, реализованный в соответствии с рассмотренной выше базовой моделью планирования, строит по непроцедурной постановке задачи с учетом имеющихся ограничений параллельный (в общем случае) план ее решения в виде частично-упорядоченного набора имен вычислительных модулей из функционального наполнения пакета.

Полученная таким образом абстрактная программа с использованием генератора преобразуется в программу на языке C++, состоящую из двух частей – управляющей и вычислительной. Управляющая часть программы оформляется в виде управляющего (головного) модуля, отвечающего за исполнение построенного параллельного плана решения задачи. Управляющий модуль реализует следующие функции: ввод (вывод) значений входных (выходных) параметров, хранение значений параметров в процессе вычислений, хранение информации о ходе вычислений, запуск отдельных вычислительных модулей на исполнение с использованием возможностей коммуникационной библиотеки PVM [6]. Вычислительная часть программы формируется из оберточных модулей. Для каждого вычислительного модуля, входящего в план решения задачи, создаются специальные программы-обертки, отвечающие за прием значений входных параметров от управляющего модуля, вызов соответствующей подпрограммы из функционального наполнения ППП, отсылку полученных значений выходных параметров управляющему модулю.

Подсистема компиляции параллельных программ выполняет следующие основные функции: обеспечивает подключение пользователя к вычислительному кластеру, копирует исходные файлы программ на языке C++ в пользовательский директорий на управляющем узле кластера, производит компиляцию главной программы и вспомогательных модулей с использованием штатных средств компилятора языка C++ (в случае возникновения ошибок выводятся соответствующие сообщения). Для компиляции используются следующие библиотеки: библиотека классов языка C++ для программной реализации объектов предметной области; библиотека системных функций управляющего модуля; библиотека вычислительных модулей, представляющая собой функциональное наполнение ППП; коммуникационная библиотека PVM.

Подсистема запуска параллельных программ копирует файлы с наборами исходных данных, осуществляет запуск программы на целевом кластере, производит мониторинг работы программы пользователя, по завершении вычислений копирует файлы, содержащие значения целевых параметров в базу расчетных данных. Главной особенностью такой базы является возможность хранения результатов по вариантам вычислительных экспериментов.

Порядок работы ИК ORLANDO предполагает, что на начальном этапе пользователь формирует описание предметной области, используя выразительные средства языка ORLANDO. Далее он приступает к решению задачи, формулируя ее в непроцедурной форме “исходные параметры => цель расчета” в виде указания планировщику. Построенный планировщиком параллельный план решения задачи на этапе трансляции преобразуется в параллельную программу на языке C++, после чего готовая параллельная программа перемещается на целевой вычислительный кластер, где компилируется штатными средствами. С помощью подсистемы запуска из базы расчетных данных на кластер копируются необходимые исходные данные, и осуществляется запуск программы на счет.

4. Инструментальный комплекс DISCOMP

Инструментальный комплекс DISCOMP предназначен для поддержки основных этапов разработки и применения распределенных пакетов прикладных программ (РППП), ориентированных на работу в гетерогенной (многоплатформенной) распределенной вычислительной среде, которая может включать вычислительные кластеры различных конфигураций. Вычислительные модули, составляющие функциональное наполнение РППП, представляют собой исполняемые программы, которые могут быть реализованы на различных языках программирования (например, C, Fortran, Pascal и др.) и быть платформо-зависимыми. Допускается включение в состав функционального наполнения РППП нетиражируемых программных комплексов, размещенных в строго определенных узлах ВС, а также унаследованного программного обеспечения, переставшего соответствовать современным требованиям, но до сих пор эксплуатируемого в связи с трудоемкостью его модификации или замены. Удаленный запуск модулей, обмен дан-

ными между модулями через файлы и мониторинг узлов ВС реализуются средствами системной части РППП.

Для ИК DISCOMP в приведенной выше базовой модели планирования множество T включает следующие типы параметров: 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*); тип *fileconst*, введенный с целью сокращения объемов передачи данных в ВС (значение параметра типа *fileconst* один раз передается узлу ВС и затем может неоднократно использоваться при запуске модулей, размещенных в данном узле). Содержательно модуль $f_i \in F$ реализует возможность вычисления множества целевых (выходных) параметров модуля по множеству заданных (входных) параметров модуля. Поэлементная обработка параметра z_j типа *filelist* модулем f_i выполняется таким образом, что каждый элемент параметра z_{jd} обрабатывается соответствующим экземпляром модуля m_{id}.

Постановка содержательной задачи для структуры *KB* задается пользователем пакета в процедурном виде и в общем случае формулируется следующим образом: “зная *KB* выполнить Q”, где Q представляет собой частично упорядоченную последовательность модулей $Q_i \subset F$. В процессе установления частичного порядка множество Q разбивается на k непустых подмножеств. Упорядочение подмножеств осуществляется по степени готовности модулей к исполнению. В рамках каждого подмножества входящие в него модули могут выполняться независимо друг от друга в любой последовательности или параллельно.

Под схемой решения задачи (СРЗ) в ИК DISCOMP понимается модель крупноблочной программы, отражающая информационно-логическую структуру вычислений в терминах предметной области. СРЗ строится автоматически в параллельно-ярусной форме из элементов следующих множеств: множества параметров Z; множества модулей F; множества событий E, возникающих в процессе выполнения СРЗ; множества операций H, предназначенных для управления процессом выполнения СРЗ; множества специальных операторов O = {*START, STOP, READ* <список параметров>, *WRITE* <список параметров>, *CALL* <имя модуля>, *FORK, JOIN, TERMINATE* <список модулей>}.

Построение СРЗ осуществляется в два этапа:

I. Разработчик РППП формирует ярусы СРЗ, размещает на этих ярусах модули из F и определяет события из E, при возникновении которых необходимо выполнить те или иные управляющие функции из H.

II. Транслятор постановки задачи на основе информационно-логических связей между параметрами и модулями структуры *KB* определяет множества входных параметров A_0 и целевых параметров B_0 СРЗ, выполняет проверку необходимых ограничений (допустимость, бесповторность, неизбыточность) и дополняет постановку задачи специальными операторами из O.

Выполнение СРЗ в режиме интерпретации представляет собой процесс параллельно-последовательного выполнения ее операторов в соответствии с порядком их вхождения в управляющий граф (**Рис. 3а**), вершинами которого являются операторы вызова модулей (или их экземпляров) из F, а также специальные операторы. Передача данных (значений параметров) между модулями выполняется в соответствии с информационным графом (**Рис. 3б**).

На **Рис. 4** приведен пример процедурной постановки задачи в формате, разработанном на основе расширяемого метаязыка XML. Данная постановка соответствует приведенным выше информационному и управляющему графам СРЗ и включает четыре яруса. На первом ярусе выполняется модуль декомпозиции, осуществляющий разбиение входного параметра z_1 на множество блоков (элементов параметра списка z_2). На следующем ярусе производится поэлементная обработка параметра-списка z_2 экземплярами модуля *solver*, которые могут выполняться параллельно на доступных узлах ВС. На третьем ярусе полученные результаты (элементы параметра-списка z_3) передаются на вход модулю *analyse*, результатом выполнения которого является параметр z_4. И на завершающем ярусе параллельно запускаются модули *plot_graph* и *statistics*, которые получают на вход значения параметров, вычисленных на предыдущих ярусах, и производят формирование значений параметров z_5 и z_6.

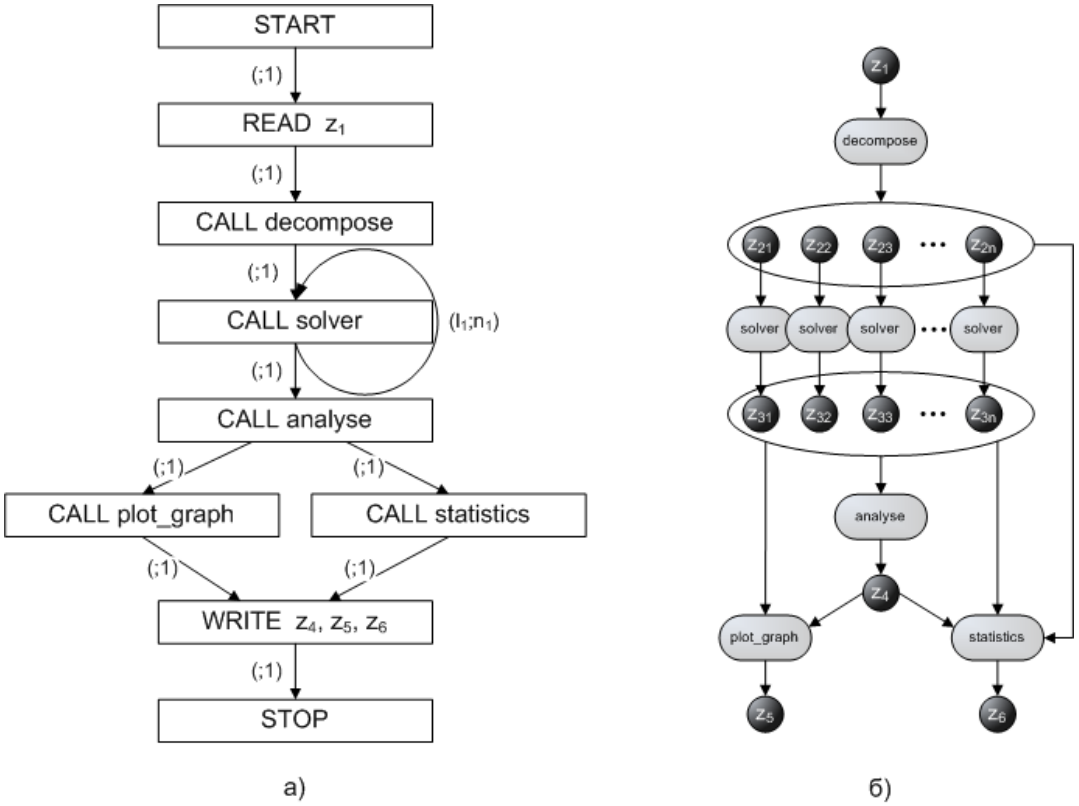


Рис. 3. Примеры а) управляющего и б) информационного графов СРЗ

```
<?xml version="1.0"?>
<scheme>
   <stage>
       <module name='decompose'/>
   </stage>
   <stage>
      <module_list name='solver'/>
   </stage>
   <stage>
      <module name='analyse'/>
   </stage>
   <stage>
      <module name='plot_graph'/>
      <module name='statistics'/>
   </stage>
</scheme>
```

Рис. 4. Пример процедурной постановки задачи

К основным составляющим программно-аппаратной архитектуры инструментального комплекса DISCOMP относятся: система управления ВС, набор вычислительных клиентов ВС, система хранения данных и средства доступа пользователей к РППП. Схема взаимодействия основных компонентов представлена на **Рис. 5**.

Система управления ВС (серверная часть ИК DISCOMP) поддерживает взаимодействие с подсистемами хранения данных и доступа пользователей к пакету, а также обеспечивает централизованное управление узлами ВС. Серверная часть ИК DISCOMP включает системное ядро, менеджеры вычислительных процессов и ресурсов, диспетчер очереди задач, подсистему журнализации и исполнительную подсистему.

Вычислительный клиент реализует процесс выполнения модуля в узле ВС и осуществляет следующие функции: организацию соответствующей среды для работы модуля (создание временных директорий, файлов входных и выходных параметров модулей, задание значений пере-

менных окружения, перенаправление ввода/вывода и т.д.); получение значений входных параметров модуля из управляющего узла; запуск модуля; контроль процесса его выполнения; отсылку значений выходных параметров в управляющий узел после завершения работы модуля.

Система хранения данных используется для структурированного размещения описаний предметной области, постановок задач в формате XML и расчетных данных в виде файлов, а также для предоставления доступа к ним из различных подсистем ИК DISCOMP. Синхронизация процессов чтения/записи данных осуществляется посредством файловых блокировок.

Средства доступа пользователей к РППП обеспечивают взаимодействие пользователя с пакетом, управление вычислительными процессами, ввод данных, получение результатов вычислений. ИК DISCOMP предоставляет два способа взаимодействия пользователя с РППП: с помощью набора утилит командной строки и посредством интерактивного пользовательского web-интерфейса.

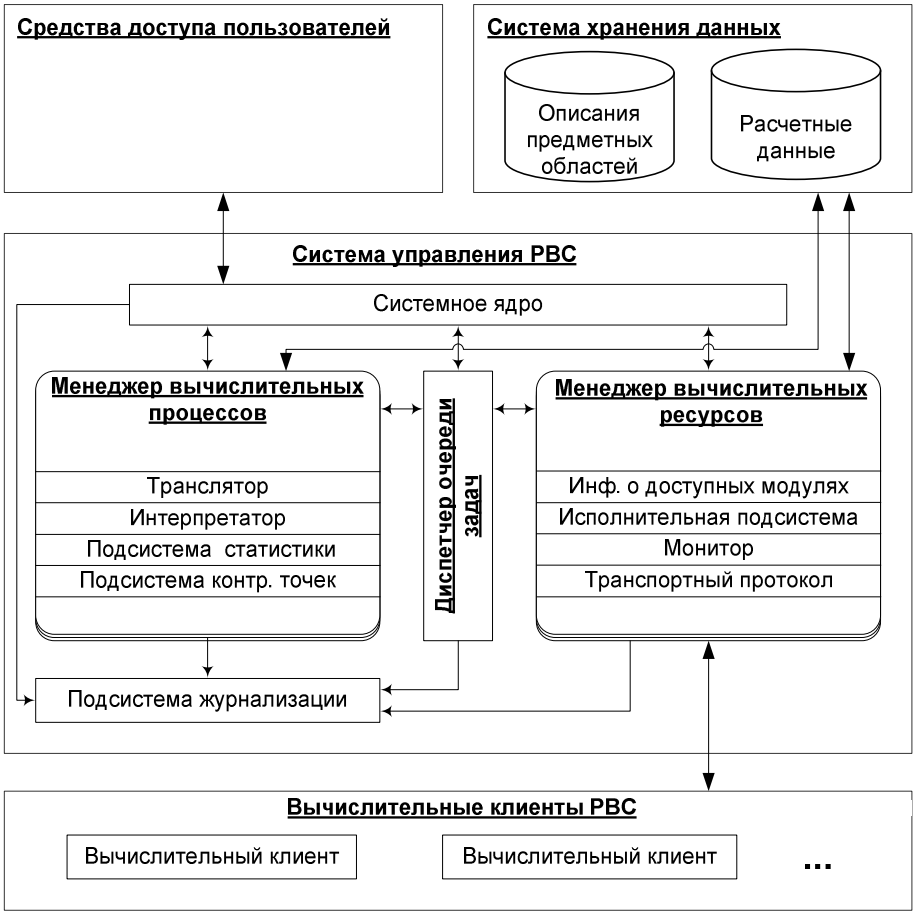


Рис. 5. Схема взаимодействия основных компонентов ИК DISCOMP

Связь распределенных компонентов ИК DISCOMP между собой реализована на основе специально разработанного протокола сетевого взаимодействия, поддерживающего обмен сообщениями, удаленный вызов процедур и передачу больших объемов бинарных данных. Протокол сетевого взаимодействия обеспечивает высокопроизводительный обмен данными между компонентами ИК DISCOMP за счет установки между ними непрерывного соединения по TCP-каналу, использования многопоточного режима работы серверной части ИК DISCOMP и минимизации объемов управляющих данных в теле передаваемого сообщения.

Эффективность решения задач в РППП обеспечивается следующими особенностями работы ИК DISCOMP: возможностью максимально эффективного использования разнородных ресурсов ВС; поддержкой высокопроизводительного взаимодействия между распределенными компонентами ИК DISCOMP; гибкой диспетчеризацией очередей задач; наличием средств оптимизации объемов данных, передаваемых между модулями в процессе решения задач в ВС; возможностью динамического управления процессами решения задач.

Динамическое управление вычислительным процессом в ИК DISCOMP базируется на механизмах обработки событий, возникающих при выполнении СРЗ. Алгоритм процесса управления включает две основные фазы: 1) применение операций, предназначенных для обработки события и выбора необходимого воздействия на процесс вычислений; 2) проверку корректности выбранного воздействия системными функциями РППП (контроль выполнения множества ограничений для интерпретатора СРЗ). Управляющие операции задаются разработчиком РППП при описании предметной области, а их вызовы реализуются в виде функций на языке JavaScript и включаются в постановку задачи. Динамическое управление вычислительным процессом позволяет сократить общее время решения задачи за счет устранения вычислительной избыточности на основе анализа текущих результатов счета в узлах ВС. Более подробно данные механизмы динамического управления вычислительным процессом рассмотрены в [7].

5. Вычислительный эксперимент

Представленные в данной работе инструментальные средства получили широкое применение в Суперкомпьютерном центре коллективного пользования ИДСТУ СО РАН при решении прикладных задач из различных предметных областей: исследование биоресурсов озера Байкал; моделирование логистических складских систем; решение задач оптимального управления, решение систем булевых уравнений и других.

С помощью ИК ORLANDO был разработан пакет ГРАДИЕНТ, предназначенный для поиска глобального минимума многоэкстремальной функции высокой размерности с помощью метода мультистарта. Пакет позволяет варьировать управляющие параметры, влияющие на точность нахождения глобального минимума и время решения задачи. Возможности пакета испытывались на ряде многоэкстремальных функций (Катковника, Растригина, Griewank), относящихся к классу сложных задач оптимизации. Результаты вычислительного эксперимента приведены в таблице 1.

Таблица 1. Время решения задачи поиска глобального минимума

Функция	**Время решения задачи на 2-х ядрах**	**Время решения задачи на 8-ми ядрах**	**Время решения задачи на 32-х ядрах**
Катковника	8 ч	115 – 125 мин	30 – 35 мин
Растригина	7 ч	100 – 115 мин	30 – 35 мин
Griewank	10 ч	150 – 170 мин	40 – 45 мин

Одним из показательных примеров применения ИК DISCOMP являются задачи обращения дискретных функций. Разработанный средствами ИК DISCOMP пакет D-SAT [8] реализует технологию крупноблочного распараллеливания SAT-задач. С применением этого пакета был успешно проведен распределенный криптоанализ генераторов двоичных шифров: Гиффорда, суммирующего и порогового. Вычислительные эксперименты проводились в распределенной ВС, включающей 20 двухпроцессорных узлов (всего 160 процессорных ядер). В таблице 2 приведены сравнительные результаты криптоанализа, выполненного на одном ядре и в распределенной ВС как с применением механизмов устранения вычислительной избыточности, так и без них.

Таблица 2. Время решения SAT-задач

SAT-задача криптоанализа	**Время решения SAT-задачи на одном вычислительном ядре**	**Время решения SAT-задачи в ВС из 160 ядер**	**Время решения SAT-задачи в ВС из 160 ядер с устранением вычислительной избыточности**
Генератор Гиффорда	14 – 30 ч	1 – 2 ч	30 – 60 мин
Суммирующий генератор	20 мин – 2 ч	10 – 30 мин	2 – 6 мин
Пороговый генератор	$\geq$ 3 суток	30 – 120 мин	6 – 10 мин

6. Заключение

Представленный в данной работе подход к организации параллельных и распределенных пакетов прикладных программ допускает естественное развитие и обобщение для базовых технологий параллельных и распределенных вычислений и новых классов фундаментальных и прикладных исследовательских задач. Архитектура и принципы работы рассмотренных инструментальных средств обеспечивают широкий спектр функциональных возможностей по разработке и применению таких пакетов для вычислительных систем различного типа.

Литература

1. Тыугу Э.Х., Харф М.Я. Алгоритмы структурного синтеза программ. – Программирование, 1980, №4. – С. 3-13.
2. Бабаев И.О., Лавров С.С., Нецветаева Г.А., Новиков Ф.А., Шувалов Г.М. СПОРА – система программирования с автоматическим синтезом программ // Применение методов математической логики. – Таллин: ИК АН ЭССР, 1983. – С. 29-41.
3. Опарин Г.А. САТУРН – метасистема для построения пакетов прикладных программ // Разработка пакетов прикладных программ. – Новосибирск, Наука, 1982. – С. 130-160.
4. Опарин Г.А., Новопашин А.П. Булевы модели синтеза параллельных планов решения вычислительных задач // Вестник НГУ. Серия: Информационные технологии. – 2008. – Т. 6. – Вып. 1. – С. 53-59.
5. Опарин Г.А., Феоктистов А.Г., Горский С.А. Язык описания модели предметной области в пакетах прикладных программ // Программные продукты и системы. – 2011. – № 1.
6. Geist G., Beguelin A., Dongarra J., Jiang W., Manchek R. and Sunderam V. PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing // MIT Press, Cambridge, MA, USA, 1995. – 273 p.
7. Опарин Г.А., Феоктистов А.Г., Сидоров И.А. Технология организации распределенных вычислений в инструментальном комплексе DISCOMP // Современные технологии. Системный анализ. Моделирование. – 2009. – № 2. – С. 175-180.
8. Заикин О.С., Семенов А.А. Технология крупноблочного параллелизма в SAT-задачах. – Проблемы управления. – 2008. – № 1. – С. 43-50.

Планирование задач для вычислительного кластера с учетом сети и многопроцессорности узлов*

П.Н. Полежаев

Оренбургский государственный университет

В данной работе приводятся результаты исследования предложенных и существующих алгоритмов планирования задач для вычислительного кластера с учетом сети и многопроцессорности узлов. Исследование проводилось с помощью разработанного симулятора вычислительного кластера и его управляющей системы, в основу которого легла предложенная модель вычислительной системы, модель управляющей системы, имитационная схема работы кластера, а также модель загрузки системы потоком задач.

1. Введение

Алгоритмы планирования задач, используемые в существующих управляющих системах вычислительных кластеров, демонстрируют неплохую эффективность работы. Тем не менее, имеется возможность дальнейшего увеличения их производительности за счет учета топологии вычислительной системы и многопроцессорности вычислительных узлов.

Действительно, если алгоритм планирования будет назначать процессы параллельной задачи на топологически близкие вычислительные ядра, то это приведет к снижению времени ее исполнения в силу сокращения коммуникационных задержек при передаче данных между ее процессами. Что, в свою очередь, увеличивает производительность всей вычислительной системы.

Учет многопроцессорности вычислительных узлов при планировании также положительно сказывается на показателях работы кластерной системы, т.к. время выполнения коммуникационных операций между процессами параллельной программы, исполняющимися на соседних процессорах (или ядрах) одного узла, гораздо меньше, чем между процессами, исполняющимися на разных узлах.

Кроме того, при назначении параллельных программ на свободные вычислительные ядра необходимо учитывать сетевую конкуренцию между процессами одновременно исполняющихся задач. Ее снижение приводит к уменьшению времени выполнения сетевых коммуникаций, а это ведет к росту производительности кластерной системы.

Целью настоящего исследования является разработка эффективных алгоритмов планирования задач для высокопроизводительных кластерных систем, учитывающих топологию, сетевую конкуренцию и многопроцессорность вычислительных узлов.

2. Модель вычислительного кластера

Произвольная кластерная вычислительная система может быть описана с помощью ориентированного графа:

$$G_T = (P, K, E, b, c, m, d, s),$$

где $P = \{p_1, p_2, ..., p_n\}$ – множество вычислительных узлов, $K = \{k_1, k_2, ..., k_z\}$ – множество коммутаторов, E – множество направленных сетевых связей между ними, $b : E \to Z_+ \cup \{0\}$ – отображение, характеризующее пропускную способность каждой сетевой связи в байтах в секунду. Функции $c, m, d : P \to Z_+$ определяют для каждого вычислительного узла p_i соответ-

* Исследования выполнены при поддержке Министерства образования и науки Российской Федерации в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг. (государственные контракты №14.740.11.0287, №14.740.11.0689).

ственно количество его вычислительных ядер $c(p_i) = C_i$, объемы оперативной $m(p_i) = M_i$ и дисковой памяти $d(p_i) = D_i$ в килобайтах. Отображение $s : P \to R_+$ для узла p_i задает относительную производительность $s(p_i) = S_i$ каждого его вычислительного ядра, которая определяет, во сколько раз ядра данного узла работают быстрее вычислительных ядер самого непроизводительного узла кластера.

Также в вычислительном кластере имеется выделенный узел p_0, на котором работает его управляющая система. Он не входит во множество P, но связан со всеми вычислительными узлами кластера с помощью выделенной управляющей сети.

Значения C_i, M_i, D_i и S_i являются статическими параметрами i-го узла вычислительного кластера. К числу его динамических характеристик относятся величины $u_i(t)$, $m_i(t)$ и $d_i(t)$, которые соответственно определяют загруженность его вычислительных ядер, объем доступной оперативной и дисковой памяти в момент времени $t \in [0;+\infty)$.

Пусть $\mathrm{X}_i = \{\chi_{i,1}, \chi_{i,2}, ..., \chi_{i,C_i}\}$ – множество вычислительных ядер узла p_i, $\mathrm{X} = \bigcup_{i=1}^{n} \mathrm{X}_i$ – множество вычислительных ядер всех узлов кластера, тогда обозначим в качестве $id(\chi, t)$ номер задачи, исполняющейся на ядре $\chi \in \mathrm{X}$ в момент времени t. Если же ядро χ в момент времени t было свободно, то $id(\chi, t) = 0$. Значения динамических параметров i-го узла в момент времени t могут быть вычислены по следующим формулам:

$$u_i(t) = \frac{\left|\{id(\chi, t) > 0 : \chi \in \mathrm{X}_i\}\right|}{C_i},$$

$$m_i(t) = M_i - \sum_{\substack{\chi \in X_i, \\ j = id(\chi, t) > 0}} m_j,$$

$$d_i(t) = D_i - \sum_{\substack{\chi \in X_i, \\ j = d(\chi, t) > 0}} d_j,$$

где m_j и d_j – соответственно объемы оперативной и дисковой памяти, необходимой каждому процессу j-й параллельной задачи. Данные формулы выведены, исходя из предположений, что мы не рассматриваем алгоритмы планирования с разделением времени, и вычислительные узлы не имеют локальную загрузку, характерную для кластеров рабочих станций.

Структура типичного вычислительного SMP-узла изображена на рисунке 1.

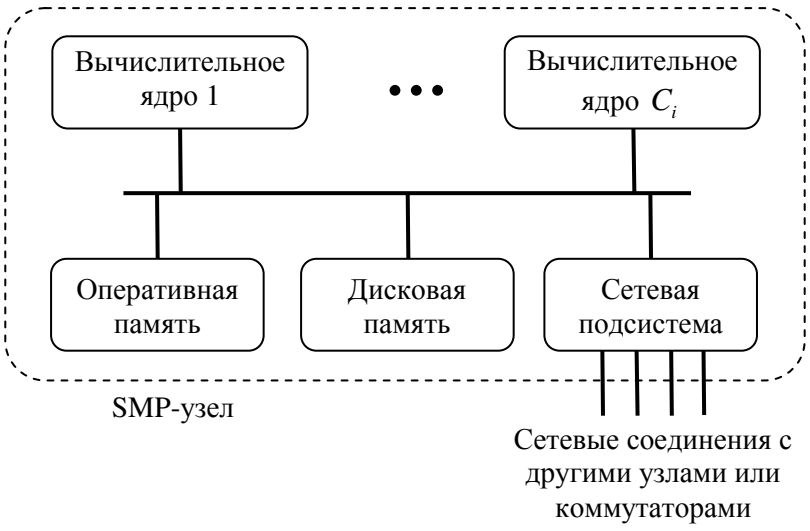


Рис. 1. Структура SMP-узла

Вычислительные ядра могут относиться как к отдельным процессорам (по одному ядру на каждом), так и к нескольким многоядерным процессорам. С целью упрощения модели они рассматриваются, как единое множество X_i. Процессы, выполняющиеся на вычислительных ядрах, могут напрямую считывать и записывать информацию в оперативную и дисковую память данного узла, взаимодействуя с ними через системную шину или коммутатор. При необходимости передачи сообщения другому вычислительному узлу задействуется сетевая подсистема, осуществляющая разбиение сообщения на пакеты и передачу их по высокопроизводительной коммуникационной сети. Сетевая подсистема каждого узла также содержит коммутатор, выполняющий функции маршрутизации собственных и транзитных пакетов.

Обозначим в качестве $N = P \cup K$ множество всех сетевых устройств системы. Сетевые связи множества $E \subseteq N \times N$ вместе с N образуют высокопроизводительную коммуникационную сеть, по которой процессы параллельных программ обмениваются сообщениями. Передача данных между управляющим узлом p_0 и вычислительными узлами совершается по отдельной управляющей сети и не мешает информационному обмену между процессами исполняющихся параллельных программ. Поэтому в рамках данной модели управляющая сеть не учитывается явно.

С помощью данной модели можно описать вычислительную систему произвольной топологии, в частности, в рамках данной работы она используется для задания вычислительных кластеров топологии толстого дерева, двумерного тора и звезды с однородными и неоднородными узлами.

3. Модель управляющей системы вычислительного кластера

В рамках настоящего исследования рассматривается классическая архитектура управляющей системы вычислительного кластера (УСВК), предполагающая наличие следующих основных компонент (см. рисунок 2):

1. Очередь представляет собой накопитель задач пользователей, отправленных на запуск.
2. Планировщик на основе информации о задачах в очереди и сведений о состоянии узлов принимает решение относительно выбора очередной задачи для ее назначения на свободные вычислительные ядра узлов, удовлетворяющих требованиям.
3. Менеджер ресурсов собирает информацию о состоянии вычислительных узлов (их доступность, загруженность и пр.).

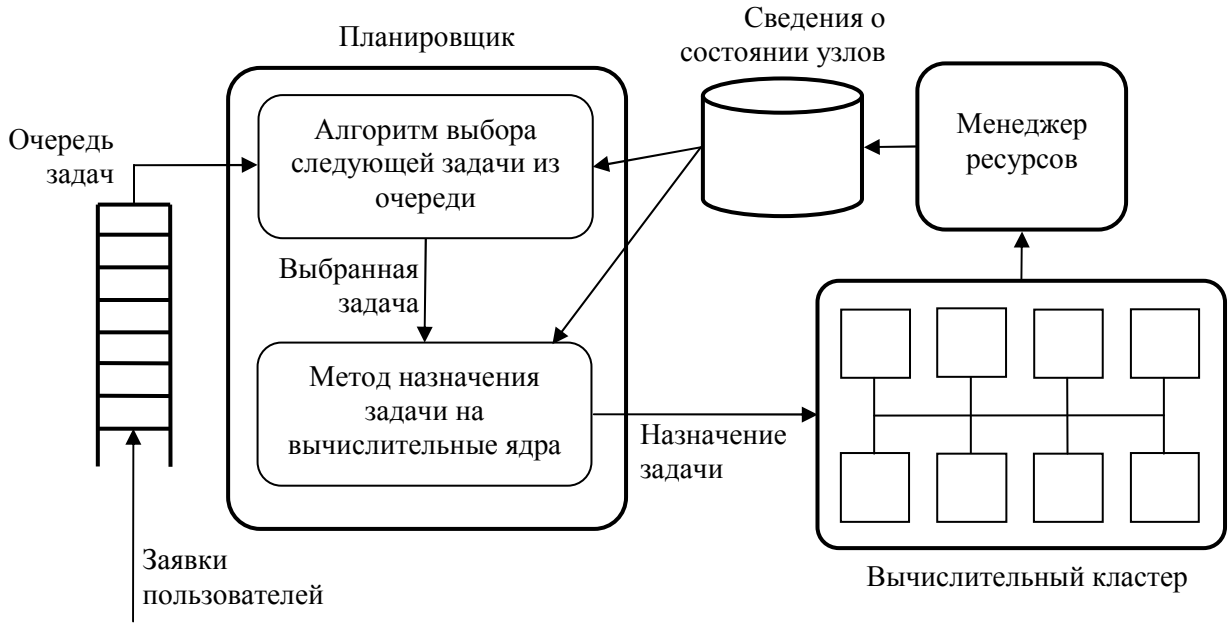


Рис. 2. Структура управляющей системы вычислительного кластера

4. Агенты на вычислительных узлах выполняют три основные функции: запуск задач на вычислительных ядрах, контроль их выполнения и сбор информации о доступности ресурсов.

Основным процессом УСВК является диспетчеризация – автоматическая обработка множества заданий. Она включает: планирование (составление расписания для задач), доставку необходимых файлов на исполнительные узлы, мониторинг процесса выполнения и сбор его результатов.

Планировщик в процессе своей работы составляет расписание запуска параллельных задач из очереди в соответствии с заложенным в него онлайновым алгоритмом планирования. В его структуре обычно выделяют: алгоритм выбора следующей задачи из очереди и метод ее назначения на свободные ядра подходящих по ресурсам вычислительных узлов.

Цикл планирования представляет собой единичный акт планирования, в течение которого происходит выбор ожидающих задач и назначение им ресурсов в соответствии с алгоритмом. Он запускается каждый раз при наступлении одного из следующих событий: поступление в очередь новой задачи, завершение выполняющейся программы или запуск зарезервированной задачи.

На рисунке 3 представлена обобщенная схема работы всех рассматриваемых в настоящем исследовании алгоритмов планирования.

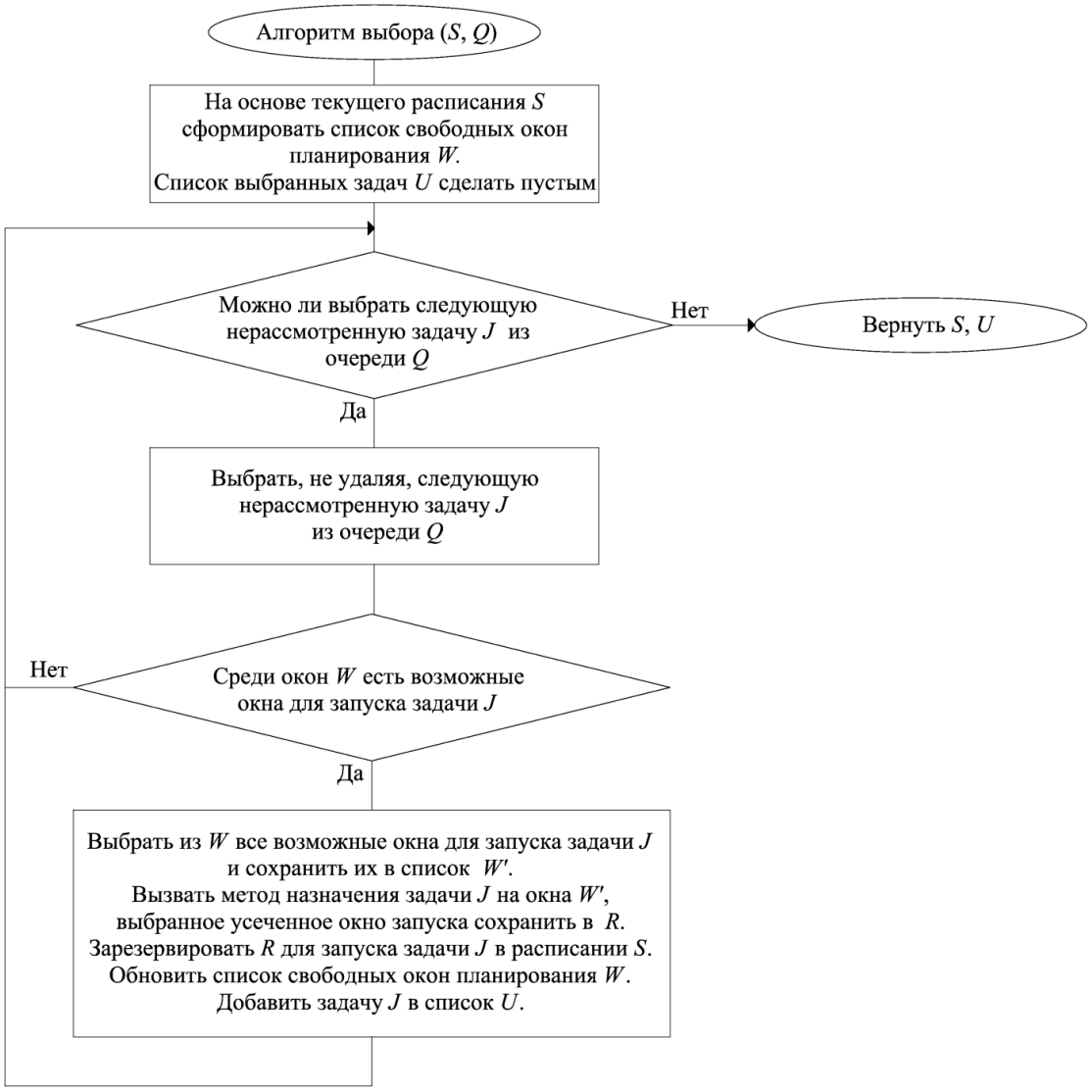


Рис. 3. Обобщенная блок-схема работы алгоритма планирования

При каждом запуске цикла планирования алгоритм выбора следующей задачи из очереди на основе текущего расписания $Schedule$ и очереди ожидающих задач Q динамически формирует на момент времени t список окон планирования WL. Затем выполняется цикл, на каждой итерации которого по некоторому принципу из Q выбирается очередная задача J, для которой на основе списка окон планирования WL составляется список подходящих окон WL', который затем передается методу назначения. В процессе своей работы последний формирует для J окно запуска R, на основе которого обновляются WL и $Schedule$. Итогом работы алгоритма является обновленное расписание $Schedule$ и список выбранных задач U.

Итоговое расписание после запуска всех задач из очереди Q может быть представлено в виде множества упорядоченных пар $Schedule = \{(J_j, W_j)\}_{j=\overline{1,m}}$, где W_j – окно запуска работы J_j.

4. Исследуемые алгоритмы планирования задач на вычислительном кластере

В рамках проводимого исследования были рассмотрены алгоритмы планирования задач, представляющие собой сочетания алгоритма выбора следующей задачи из очереди Most Processors First Served Scan (MPFS Scan) или алгоритма обратного заполнения Backfill с методами ее назначения на вычислительные ядра, учитывающими топологию вычислительной системы [1, 2]: для топологии толстого дерева – Sorting Nodes by Performance (SNP), Fat Tree Sorting Commutators by Performance (FTSCP), Fat Tree Sorting Commutators by Cores (FTSC); для топологии двумерного тора – модифицированные варианты алгоритмов Minimizing message-passing Contention 1x1 (MC1x1) и Minimizing message-passing Contention 1x1 Incremental (MC1x1+Inc); для топологии звезды – Sorting Nodes by Performance (SNP) и Sorting Nodes by Speed (SNS).

Также рассматривались сочетания алгоритмов MPFS Scan и Backfill с методами назначения, не принимающими топологию во внимание [2, 3]: First Fit (FF), Best Fit (BF), Fastest Node Fist (FNF) и Random First (RF).

В ходе исследования были предложены собственные алгоритмы планирования, представляющие собой сочетания алгоритма MPFS Scan или Backfill с разработанными методами назначения Summed Distance Minimization (SDM) и Maximum Distance Minimization (MDM). Данные методы учитывают топологию вычислительной системы, сетевую конкуренцию между одновременно исполняющимися задачами и многопроцессорность вычислительных узлов.

Пусть методу назначения передается выбранная из очереди задача J_j, требующая для своего исполнения n_j вычислительных ядер. Далее опишем принцип работы алгоритмов SDM и MDM.

Метод назначения SDM для каждого допустимого окна планирования $W \in WL'$ перебирает все сетевые устройства d кластера (коммутаторы и узлы) и определяет для каждого из них n_j близжайших вычислительных ядер окна W, подходящих по конфигурации для задачи J_j. Эти вычислительные ядра формируют возможное окно запуска $R_{W,d}$ задачи J_j. Для назначения в качестве результата R выбирается окно $R_{W,d}$ с минимальным суммарным попарным расстоянием, если таких несколько, то выбирается окно с большей суммарной производительностью вычислительных ядер.

Данный метод является обобщением алгоритма Manhattan Median для случая произвольных топологий. За счет компактного размещения процессов параллельной задачи алгоритм SDM пытается минимизировать суммарное попарное расстояние между выделенными ей вычислительными ядрами, это позволяет снизить сетевую конкуренцию между одновременно выполняющимися в системе задачами и уменьшить степень распределения процессов параллельных задач по вычислительной системе.

Метод назначения MDM для каждого допустимого окна планирования $W \in WL'$ перебирает все сетевые устройства d кластера и для каждого из них запускает алгоритм поиска в ширину. В процессе его работы формируется множество достигнутых вычислительных ядер узлов окна W, которые подходят по конфигурации для задачи J_j, до тех пор, пока не будет получено n_j ядер. При этом среди ядер, находящихся на одинаковом расстоянии от начального сетевого устройства в первую очередь выбираются те, которые имеют большую скорость. Результатом работы поиска в ширину является сформированное возможное окно запуска $R_{W,d}$. Для назначения задаче J_j в качестве результата R выбирается окно $R_{W,d}$ с минимальным максимальным расстоянием от первоначального сетевого устройства d.

Данный метод является аналогом алгоритма MC1x1 для произвольных топологий. За счет компактного размещения процессов параллельной задачи алгоритм MDM пытается минимизировать максимальное расстояние от выбранного центрального сетевого устройства до назначенных задаче вычислительных ядер.

5. Симулятор вычислительного кластера и его управляющей системы

Для экспериментального сравнительного исследования эффективности работы предложенных методов назначения с существующими вариантами использовался созданный в рамках настоящей работы симулятор вычислительного кластера и его управляющей системы TopSimity. Он был реализован на языке программирования C++ с использованием стандартной библиотеки шаблонов STL. Получено свидетельство о его регистрации в ФГУ ФИПС [4].

Имитационная схема работы симулятора представлена на рисунке 4. Источник I генерирует поток параллельных задач, отправляемых пользователями в очередь Q управляющего узла p_0. Канал S представляет собой планировщик, который в соответствии с заложенным в него алгоритмом осуществляет извлечение задач из Q и назначение их на свободные вычислительные ядра подходящих по конфигурации узлов.

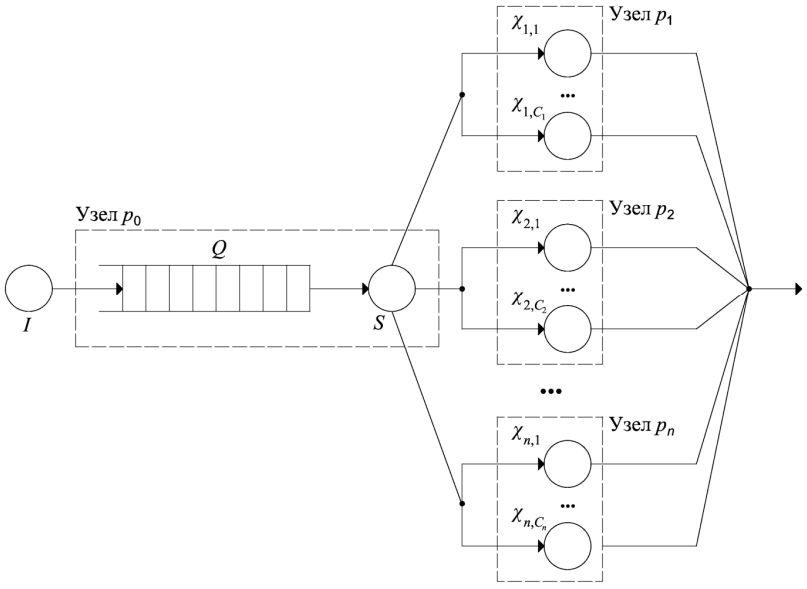


Рис. 4. Имитационная схема управляющей системы вычислительного кластера

Схема работы вычислительного узла p_i приведена на рисунке 5 p_i соединен с другими узлами и коммутаторами вычислительной сети с помощью r_i дуплексных связей. Все входящие пакеты, а также пакеты сообщений, генерируемых процессами, выполняющимися на

локальных вычислительных ядрах, сначала поступают в очередь $Q_{inp.i}$, а затем маршрутизируются каналом обслуживания R_i. Если соответствующий пакет предназначен для локального вычислительного ядра, то он передается ему непосредственно, иначе – помещается в одну из очередей $Q_{out.i,1}, Q_{out.i,2}, \ldots, Q_{out.i,r_i}$, соответствующую выбранной алгоритмом маршрутизации исходящей связи. При выполнении маршрутизации каналом R_i моделируется временная задержка величиной $l_{routing}$. Имитационная схема коммутатора представляет собой урезанный вариант имитационной схемы узла. Отличие только в том, что коммутатор не может выполнять вычисления, и поэтому у него отсутствуют вычислительные ядра.

На рисунке 5 приведена схема работы дуплексной связи $L_{ij} = \{E_{ij}, E_{ji}\}$, соединяющей сетевые устройства $d_i \in D$ и $d_j \in D$. Каналы обслуживания E_{ij} и E_{ji} добавляют к времени передачи пакета задержку величиной $b(E_{ij}) = b(E_{ji})$.

В качестве алгоритма работы симулятора используется алгоритм моделирования по принципу особых состояний (событий).

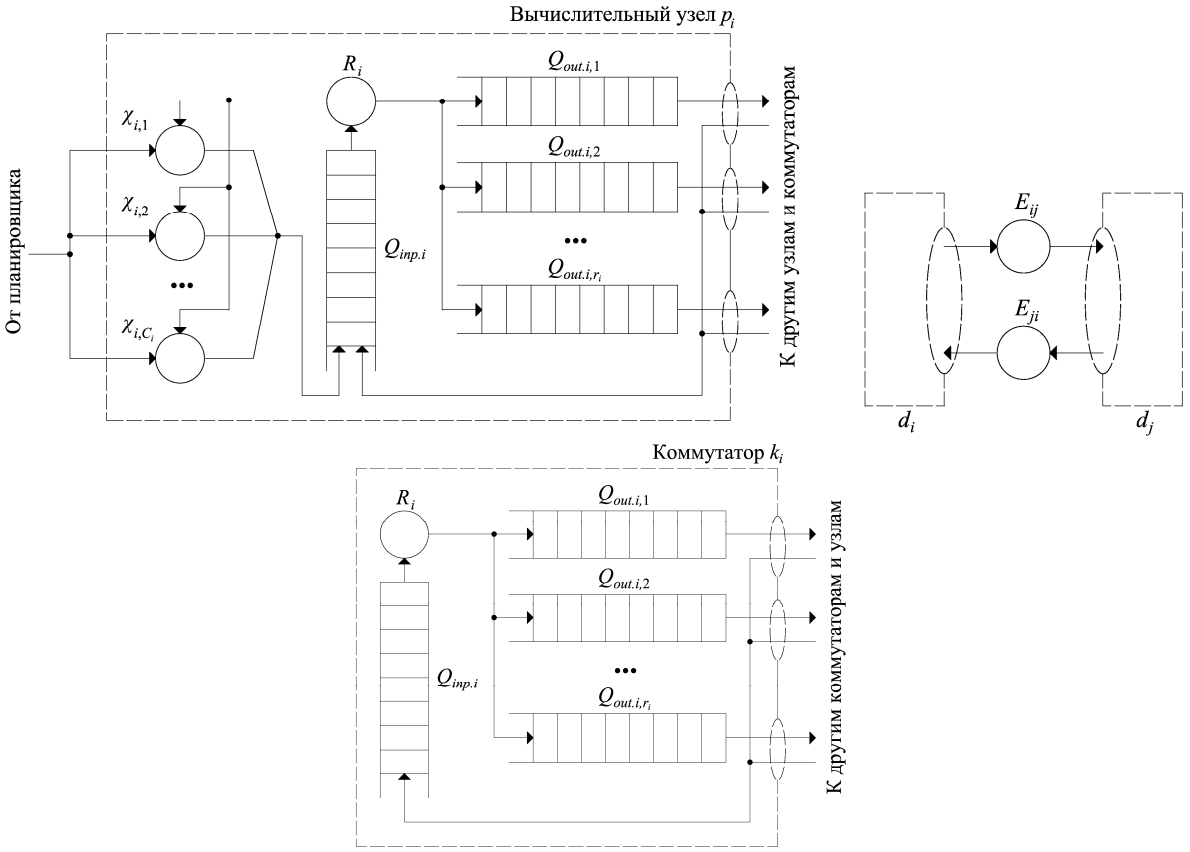


Рис. 5. Имитационная схема узла p_i, коммутатора k_i и дуплексной сетевой связи $L_{ij} = \{E_{ij}, E_{ji}\}$

Вычислительная загрузка кластера моделируется потоком задач $J = (J_1, ..., J_r)$, помещаемых пользователями в очередь его управляющей системы. Каждая задача представляет собой параллельную неинтерактивную программу, способную работать в пакетном режиме. Ее процессы запускаются планировщиком одновременно на всех выделенных вычислительных ядрах, во время работы они обмениваются сообщениями между собой. Ресурсы, выделенные задаче, освобождаются при завершении всех ее процессов.

Пользователь для задачи J_j указывает следующие требования к ресурсам кластера: количество необходимых вычислительных ядер n_j, объем оперативной mr_j и объем дисковой па-

мяти dr_j в килобайтах, необходимой для исполнения каждого процесса на узлах, оценку $\tilde{t}_j$ в секундах времени выполнения задачи на узлах с единичной относительной производительностью. Следующая формула позволяет вычислить t_j – оценку времени выполнения задачи с учетом производительности выделенных ей планировщиком вычислительных узлов:

$$t_j = \frac{\tilde{t}_j}{\min\limits_{i \in I_j} S_i},$$

где I_j – множество номеров узлов, ядра которых были отданы задаче.

Кроме описанных выше характеристик, каждая задача также имеет следующие параметры, необходимые для целей симуляции: a_j – время ее поступления в очередь, $\tau_j \in (0; \tilde{t}_j]$ – время, затрачиваемое задачей только на вычисления (без сетевых коммуникаций) при условии единичной относительной производительности всех назначенных ей вычислительных узлов.

В рамках данной модели предполагается, что пользователь, делая оценку времени выполнения задачи, возможно, допускает ошибку, переоценивая ее по сравнению с реальным временем выполнения. Противоположный случай ошибок не рассматривается, т.к. управляющая система будет принудительно завершать задачи, превысившие лимит выделенного им времени.

Структура программы каждой параллельной задачи J_j может быть представлена следующей SPMD-моделью:

Process *u*:
for *i* = 1 to q_j do
{
  Communication(*i,j*);
  Computation(*i,j*);
}

Выполнение каждого процесса параллельной задачи представляет собой чередование фаз коммуникации и вычислений.

Коммуникационная часть каждой итерации может быть описана с помощью коммуникационного паттерна – ориентированного взвешенного графа следующего вида:

$$CP_{ij} = (\Pi_j, p_{ij}),$$

где $\Pi_j = \{\pi_1, \pi_2, ..., \pi_{n_j}\}$ – множество процессов задачи, $p_{ij} : \Pi_j \times \Pi_j \to Z_+ \cup \{0\}$ – функция, определяющая вес каждой дуги, равный размеру в пакетах передаваемого сообщения. $p_{ij}(u, v)$ равно количеству пакетов сообщения, передаваемого от процесса u к процессу v на i-й итерации SPMD.

Пусть $pred_{ij}(u) = \{v \in \Pi_j : p_{ij}(v, u) > 0\}$ – множество предшественников процесса u в орграфе (6), $succ_{ij}(u) = \{v \in \Pi_j : p_{ij}(u, v) > 0\}$ – множество его последователей. Выполнение фазы Communication(*i,j*) для каждого процесса u заключается в том, что сначала происходит неблокируемая рассылка сообщений всем процессам множества $succ_{ij}(u)$, а затем выполняется блокируемый прием всех сообщений от процессов $pred_{ij}(u)$. Заметим, что остальные случаи исключены из рассмотрения, т.к. сочетания блокируемых рассылок с неблокируемыми или блокируемыми приемами сообщений могут приводить к взаимоблокировкам, а случай неблокируемых рассылок и неблокируемых приемов не представляется интересным.

Обозначим $CP_j = \{CP_{1j}, CP_{2j}, ..., CP_{q_j j}\}$ – множество всех коммуникационных паттернов задачи J_j. Можно выделить два основных способа его задания: случайная генерация и моделирование реальных программ, например, эталонных тестов, реализации быстрого преобразования Фурье, алгоритма параллельного умножения матриц и др.

Типичные коммуникационные паттерны, встречаемые в научной литературе [5–7]:

1. Random. Каждый процесс пересылает сообщение одному другому случайно выбранному процессу.
2. Pairs. Процессы случайно разбиваются на пары и обмениваются сообщениями между собой.
3. Ring. Процессы объединяются в кольцо. Каждый процесс посылает сообщения следующему и принимает данные от предыдущего.
4. One-to-all. Один случайно выбранный процесс рассылает сообщения всем остальным.
5. All-to-all. Каждый процесс отправляет сообщения остальным процессам.

В рамках проводимого исследования применял коммуникационный паттерн All-to-all, т.к. он обеспечивает максимальную загрузку вычислительной сети и сетевую конкуренцию. В дальнейшем планируется рассмотреть другие возможные варианты.

Заметим, что в рамках настоящей модели рассматриваются только коммуникации вида «точка-точка». Моделировать коллективные операции достаточно сложно, т.к. способ их выполнения в виде совокупности точечных операций зависит от нескольких факторов: используемых алгоритмов, характеристик вычислительной системы и передаваемых сообщений. В частности, коллективные операции MPI по-разному реализованы в библиотеках разных производителей, а также в разных версиях библиотеки одного производителя. В будущем планируется расширение данной модели за счет учета коллективных операций.

Пусть функция $\gamma_j : \Pi_j \to P$ позволяет определить для каждого процесса задачи J_j вычислительный узел, на ядро которого он был назначен планировщиком. Для простоты положим, что каждый процесс тратит одинаковое время на выполнение вычислительной части каждой итерации SPMD-модели задачи. Тогда время выполнения вычислительной фазы Computation(*i,j*) процессом u можно вычислить по формуле:

$$T_{вычисл.ij}(u) = \frac{\tau_j}{q_j s(\gamma(u))}.$$

Пусть $T_{комм.ij}(u)$ – время выполнения коммуникационной фазы Communication(*i,j*) процессом u задачи J_j, тогда реальное время выполнения данной задачи T_j может быть вычислено по формуле:

$$T_j = \max_{u \in \Pi_j} \sum_{i=1}^{q_j} (T_{комм.ij}(u) + T_{вычисл.ij}(u)) = \max_{u \in \Pi_j} \{ \sum_{i=1}^{q_j} T_{комм.ij}(u) + \frac{\tau_j}{s(\gamma(u))} \}.$$

Величина $T_{комм.ij}(u)$ зависит от сетевой конкуренции и от физического расположения процессов на вычислительных узлах, ее значение может быть вычислено в процессе симуляции работы вычислительного кластера и его управляющей системы. Заметим, что в рамках данной модели мы, в силу незначительности, пренебрегаем временем, которое тратится на разбиение сообщения на пакеты и его сборку из них.

Таблица 1. Законы распределений параметров модели

Параметр	Используемый закон распределения
$\log n_j$	двухэтапное равномерное распределение с выделением классов последовательных и 2^k-задач
$\log \tau_j$	гипер-гамма распределение, параметр p которого линейно зависит от n_j
$\tilde{t}_j$	равномерное распределение на отрезке $[\tau_j; 2\tau_j]$
$a_{j+1} - a_j$	экспоненциальное распределение
mr_j, dr_j	равномерное распределение
$p_{ij}(u,v)$	экспоненциальное распределение для всех дуг, существующих в CP_{ij}

Симуляция работы вычислительного кластера и его управляющей системы предполагает формирование потока задач. Все описанные выше количественные параметры генерируются на

основе определенных законов распределений случайных величин, подобранных в результате анализа реальных трасс, собираемых на кластерных вычислительных системах (см. таблицу 1). Более подробную информацию можно найти в статьях [3, 5].

Данная модель вычислительной загрузки позволяет генерировать типичные потоки задач, поступающие в очереди управляющих систем большинства современных вычислительных кластеров.

6. Результаты экспериментального исследования

Исследование проводилось для двух групп сценариев: вычислительный кластер с однородными узлами и однородной высокопроизводительной сетью и вычислительный кластер с неоднородными узлами и однородной сетью.

В первом случае моделировался кластер из 12 узлов, каждый из которых имеет 2 вычислительных ядра, 2 Гб оперативной памяти, 40 Гб локальной дисковой памяти, единичную относительную вычислительную скорость. Во втором – использовалась конфигурация из 12 узлов с суммарным количеством вычислительных ядер 24, разными объемами оперативной и дисковой памяти, разной относительной вычислительной скоростью.

Каждая группа сценариев рассматривалась для трех топологий: толстое дерево (два 6-портовых корневых коммутатора и три 8-портовых листовых коммутатора (4 восходящих портов и 4 узловых портов на каждом)), звезда (один 12-портовый коммутатор), двумерный тор размера 3 x 4 узла.

Оценка эффективности работы алгоритмов планирования осуществлялась с помощью системы количественных критериев и метрик, которая охватывает все аспекты состояния вычислительной системы: производительность, сбалансированность загруженности вычислительной системы, используемость памяти вычислительных узлов, гарантированность обслуживания задач, честность по отношению к задачам, характер распределения процессов параллельных задач по вычислительной системе и сетевую конкуренции между ними.

В частности критерий производительности включает следующие метрики: средняя загруженность вычислительных ядер узлов кластера U_{cores}, потеря производительности кластера CL, максимальное время завершения задачи C_{max}, среднее время ожидания задач в очереди $\overline{T}_{wait}$ и среднее ограниченное замедление задач $\overline{s}_{lim}$.

Для получения достоверных результатов симуляция проводилась в каждом сценарии 100 раз для различных потоков из 500 случайно сгенерированных задач. Все вычисляемые значения метрик усреднялись по всем потокам.

Сравнение алгоритмов планирования осуществлялось путем построения графиков зависимости метрик от величины системной загрузки L. Например, на рисунке 6 изображены графики зависимости метрик производительности U_{cores} и $C_{\max}$ от L для сценария кластера топологии толстого дерева с неоднородными вычислительными узлами.

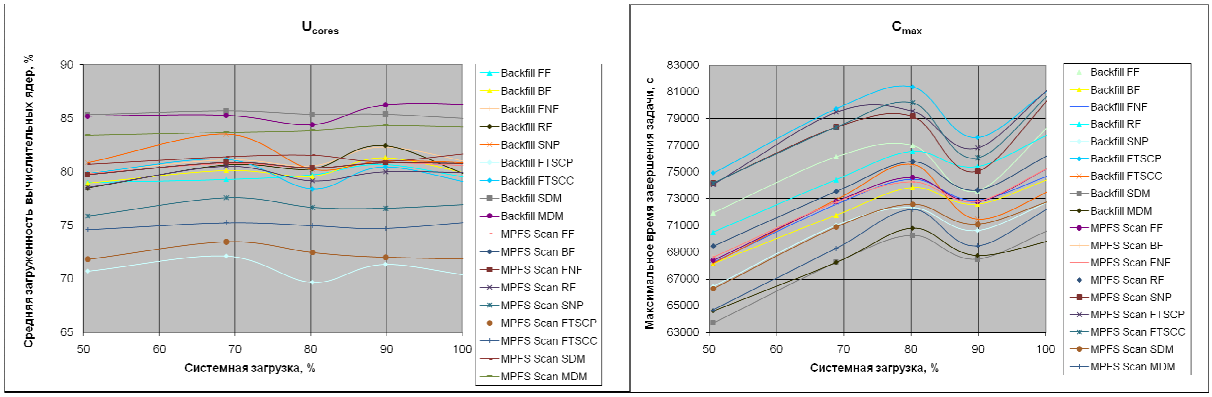


Рис. 6. Графики зависимости значений метрик производительности U_{cores} и $C_{\max}$ от величины системной загрузки L

Заметим, что алгоритмы планирования в первую очередь сравнивались по критерию производительности, остальные критерии носили вторичный характер.

Полученные результаты экспериментального исследования сведены в таблицу 2. Для каждого сценарии в зависимости от топологии системы и величины системной загрузки L может быть рекомендован свой алгоритм планирования, представляющий собой сочетания алгоритма выбора следующей задачи из очереди Backfill с предложенными методами ее назначения на вычислительные ядра MDM или SDM.

Таблица 2. Алгоритмы планирования, показавшие лучшие результаты в каждом сценарии

Группа сценариев	Топология	Наилучшие алгоритмы
Неоднородные вычислительные узлы и однородная сеть	Толстое дерево	Backfill SDM (при $L < 85$-93%), Backfill MDM (при $L \geq 85$-93%)
	Двумерный Тор	Backfill SDM (при $L < 67$-71%), Backfill MDM (при $L \geq 67$-71%)
	Звезда	Backfill SDM, Backfill MDM
Однородные вычислительные узлы и однородная сеть	Толстое дерево	Backfill SDM (при $L < 75$-79%), Backfill MDM (при $L \geq 75$-79%)
	Двумерный Тор	Backfill SDM (при $L < 71$-73%), Backfill MDM (при $L \geq 71$-73%)
	Звезда	Backfill SDM, Backfill MDM, Backfill SNP

7. Заключение

В рамках данной работы была предложена имитационная схема кластера, разработана модель вычислительной системы, модель его управляющей системы, модель вычислительной загрузки потоком задач. Все они легли в основу симулятора вычислительного кластера и его управляющей системы – основного инструмента настоящего исследования. Он позволяет проводить сравнительное экспериментальное исследование алгоритмов планирования с учетом сети и многопроцессорности вычислительных узлов.

Проведенное исследование показало эффективность работы алгоритмов планирования, представляющих собой сочетания алгоритма Backfill с предложенными методами назначения MDM и SDM.

В дальнейшем планируется реализация данных алгоритмов в рамках УСВК Torque и их апробация на реальных потоках разнообразных вычислительных задач, характерных для современных кластерных систем.

Литература

1. Топорков В.В. Модели распределенных вычислений. М.: ФИЗМАТЛИТ, 2004. – 320 с.
2. Гергель, В.П. Теоретические основы экспериментального исследования алгоритмов планирования задач для вычислительного кластера с помощью симулятора / В.П. Гергель, П.Н. Полежаев // Журнал "Вестник Оренбургского государственного университета", №9(115), 2010. - С. 114-120.
3. Полежаев П.Н. Исследование алгоритмов планирования параллельных задач для кластерных вычислительных систем с помощью симулятора // Параллельные вычислительные технологии (ПАВТ'2010): Труды международной конференции. – Челябинск: Изд. ЮУрГУ, 2010 г. стр. 287 – 298.
4. Симулятор высокопроизводительного кластера TopSimity с учетом топологии системы и многопроцессорности узлов / Полежаев П.Н. Свидетельство Федеральной службы по ин-

теллектуальной собственности, патентам и товарным знакам №2010617255 от 29 октября 2010 г.

5. Полежаев П.Н. Симулятор вычислительного кластера и его управляющей системы, используемый для исследования алгоритмов планирования задач. // Журнал «Вестник ЮУрГУ», №35(211). Серия «Математическое моделирование и программирование», вып. 6, 2010. стр. 79 – 90.
6. Moore S.Q., Lionel M.N. The Effects of Network Contention on Processor Allocation Strategies // Proceedings of the 10th International Parallel Processing Symposium. – Washington, DC: EEE Computer Society, 1996. – P. 268-273.
7. Bani-Mohammad S., Ould-Khaoua M., Abaneh, I. An efficient processor allocation strategy that maintains a high degree of contiguity among processors in 2D mesh connected multicomputers // Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, AICCSA. – 2007. – P. 934-941.

Применение многопроцессорной вычислительной техники для решения задач внутренней баллистики*

И.В. Семенов[1], П.С. Уткин[1], И.Ф. Ахмедьянов[1], И.С. Меньшов[2]

Институт автоматизации проектирования РАН[1],
Институт прикладной математики им. М.В. Келдыша РАН[2]

В работе представлены математическая модель, вычислительный алгоритм и методика его распараллеливания для решения задач внутренней баллистики в квазиодномерном приближении на многопроцессорных ЭВМ. Проведено сравнение разработанного программного комплекса с наиболее известными зарубежными пакетами прикладных программ для решения аналогичного класса задач на примере тестовой задачи AGARD. Разработанный программный комплекс может быть использован для решения широкого спектра задач, связанных как с гражданскими приложениями (функционирование пороховых зарядов для срабатывания систем тушения пожаров и пр.), так и с приложениями специального назначения для проектного предсказательного моделирования внутрибаллистического процесса в ствольных системах.

1. Введение

Задача исследования внутрибаллистического процесса [1] встречается в настоящее время в целом ряде приложений, среди которых можно отметить пороховые заряды для срабатывания систем тушения пожаров [2], пиропатроны в подушках безопасности автомобилей и т.д. «Нульмерный» термодинамический подход, развиваемый еще с начала прошлого века как в нашей стране [1], так и за рубежом (см. обзор в [3]), позволяет определить основные интегральные характеристики внутрибаллистического процесса – максимальное давление и скорость метаемого тела, но не способен описать волновые процессы (так называемые, волны Вьеля), возникающие при определенных условиях закладки пороха и воспламенении. Для исследования подобных волновых процессов используются квазиодномерные газодинамические модели [4 – 6].

Актуальность работы по совершенствованию математических моделей и вычислительных алгоритмов для решения задач внутренней баллистики диктуется необходимостью разработки новых образцов артиллерийской техники, к которой предъявляются такие требования, как автоматизация процесса заряжания и выстрела, а также повышенная скорострельность, что в свою очередь влечет за собой переход на модульную систему заряжания, а значит и возможную локализацию пороха в каморе (части орудия, в которой помещается пороховой заряд) и возникновение волновых процессов, существенно влияющих на характер внутрибаллистического процесса. Решение подобных задач с помощью классических теоретических подходов без привлечения средств вычислительного эксперимента представляется затруднительным.

Основная вычислительная сложность при решении задач внутренней баллистики в квазиодномерном приближении обусловлена наличием широкого класса моделей межфазного взаимодействия, которые необходимо внедрять для корректного описания рассматриваемых процессов. Например, в рамках подхода, используемого в данной работе, для расчета прогрева пороховых элементов в каждой ячейке расчетной области необходимо решать одномерное уравнение теплопроводности, что фактически повышает размерность задачи. Другим фактором, делающим важным использование многопроцессорной вычислительной техники, является необходимость проведения многопараметрических расчетов на широком поле входных данных, поскольку, как будет показано ниже, расчет внутрибаллистического процесса характеризуется массой определяющих параметров, значения которых не всегда известны с удовлетворительной точностью.

* Работа выполнена при поддержке РФФИ (грант № 09-01-12073-офи_м).

Таким образом, данная работа посвящена разработке математической модели, вычислительного алгоритма и методики распараллеливания расчетного алгоритма для решения задач внутренней баллистики в квазиодномерном приближении на многопроцессорных ЭВМ, а также верификации разработанного программного комплекса на примере тестовой задачи AGARD.

2. Математическая модель внутрибаллистического процесса

2.1 Основные положения модели и определяющая система уравнений

Объектом рассматриваемой модели является движущаяся гетерогенная смесь, которая представляется двухфазным континуумом, состоящим из дисперсной фазы пороховых элементов и газовой фазы продуктов горения [4 – 7]. Газовая смесь состоит из $N + 1$ компонентов, которые представляют собой продукты горения пороховых элементов, то есть пороховые газы (нумеруются от 1 до N), и один компонент нейтральной газовой среды – воздух (индекс $N + 1$). Дисперсная фаза состоит из N сортов пороха, которые могут отличаться друг от друга законом горения, характером механического и теплового взаимодействия с газовой фазой и другими параметрами. Отметим, что в силу специфики задач внутренней баллистики, связанной с пространственной стесненностью физико-химических процессов и их скоротечностью, газовая фаза представляется смесью, каждый компонент которой занимает только определенную часть объема, а давление p и скорость u газовых компонентов считаются одинаковыми. Движение дисперсной фазы также описывается одним давлением σ и одной скоростью v.

Для определения мгновенного состояния j-го компонента дисперсной фазы вводится его объемная доля β_j. Материал пороха предполагается жестким, недеформируемым, и характеризуется постоянной плотностью δ_j. Обозначим через β суммарную объемную долю дисперсной фазы, тогда объемная доля газовой фазы или пористость среды определяется как $\varphi = 1 - \beta$. Компонент газовой смеси с индексом j характеризуется средней по пористости плотностью $\rho_j = \varphi_j \cdot \rho_{0j} / \varphi$, где ρ_{0j} его истинная плотность, а φ_j – объемная доля, причем $\varphi_1 + ... + \varphi_{N+1} = \varphi$.

Связь термодинамических параметров состояния газовой фазы описывается уравнениями типа Дюпре [4] с приведенными параметрами – показателем адиабаты γ, коволюмом b и молекулярным весом M:

$p(1-b\rho) = \rho RT/M$ – термическое уравнение состояния,

$e = p(1-b\rho)/\left[(\gamma-1)\rho\right]$ – калорическое уравнение состояния,

$$\rho = \sum_{j=1}^{N+1}\rho_j,\ \frac{1}{M} = \frac{1}{\rho}\sum_{j=1}^{N+1}\frac{\rho_j}{M_j},\ b = \frac{1}{\rho}\sum_{j=1}^{N+1}\rho_j b_j,\ \gamma = 1 + \left[\sum_{j=1}^{N+1}\rho_j/M_j\right]\Bigg/\left[\sum_{j=1}^{N+1}\rho_j/\left[M_j(\gamma_j-1)\right]\right],$$

где T – температура газа, R – универсальная газовая постоянная, e – удельная внутренняя энергия газовой фазы.

Система уравнений, описывающих квазиодномерное течение газопороховой смеси в канале переменной площади сечения $S(x)$, выражает в дифференциальной форме фундаментальные законы сохранения массы, импульса и энергии для газовой фазы:

$$\begin{cases} \dfrac{\partial(S\varphi\rho_j)}{\partial t} + \dfrac{\partial(S\varphi\rho_j u)}{\partial x} = Sm_j H\left[t - t_j\right],\ j = 1,...,N,\ \dfrac{\partial(S\varphi\rho_{N+1})}{\partial t} + \dfrac{\partial(S\varphi\rho_{N+1}u)}{\partial x} = 0, \\ \dfrac{\partial(S\varphi\rho u)}{\partial t} + \dfrac{\partial\left(S\varphi\left[\rho u^2 + p\right]\right)}{\partial x} = p\dfrac{\partial(S\varphi)}{\partial x} + S\sum_{j=1}^{N}\Pi_j, \\ \dfrac{\partial(S\varphi\rho E)}{\partial t} + \dfrac{\partial(S\varphi\rho J u)}{\partial x} = -\dfrac{\partial(\beta S p v)}{\partial x} + S\sum_{j=1}^{N}Q_j - SQ_w, \end{cases} \tag{1}$$

Здесь $E = 0.5u^2 + e$ – удельная полная энергия, $J = E + p / \rho$ – удельная полная энтальпия газовой фазы, m_j, Π_j и Q_j, $j = 1, ... , N$, обозначают, соответственно, скорости изменения массы, импульса и энергии в единице объема газовой фазы в результате горения пороховых элементов и межфазного взаимодействия (см. раздел 2.2), $t_j(x)$ – время до воспламенения j-го сорта пороха,

Q_w – тепловой поток в стенки каморы и ствол, а $H[.]$ – функция Хевисайда, равная 0 в случае отрицательности аргумента и 1 в противоположном случае.

Уравнения сохранения массы и импульса для дисперсной фазы имеют следующий вид:

$$\begin{cases} \dfrac{\partial\left(S\beta_j\delta_j\right)}{\partial t}+\dfrac{\partial\left(S\beta_j\delta_j v\right)}{\partial x}=-Sm_jH\left[t-t_j\right], j=1,...,N, \\ \dfrac{\partial\left(S\beta_j\delta_j v\right)}{\partial t}+\dfrac{\partial\left(S\beta_j\delta_j v^2+S\beta_j\sigma\right)}{\partial x}=-S\beta_j\dfrac{\partial p}{\partial x}-S\Pi_j, j=1,...,N, \end{cases} \tag{2}$$

где σ – межгранулярное давление, которое вводится для препятствования чрезмерному уплотнению дисперсной фазы. Этот механизм включается, когда объемная доля дисперсной фазы достигает критического значения плотной упаковки. Межгранулярное давление зависит от объемной доли β и аппроксимируется следующей зависимостью:

$$\sigma(\beta)=H\left[\beta-\beta_0\right]B\left[\left(1-\beta_0\right)^k/\left(1-\beta\right)^k-1\right], \tag{3}$$

где β_0 – объемная доля, соответствующая плотной упаковке, а параметры B и k – эмпирические константы, определяемые на основе экспериментальных данных.

Ввиду использования Эйлерова подхода, в модель требуется ввести уравнение для изменения скалярной переменной – относительной толщины сгоревшего свода порохового элемента z, которая характеризует процесс горения:

$$\frac{\partial\left(S\beta_j\delta_j z_j\right)}{\partial t}+\frac{\partial\left(S\beta_j\delta_j v z_j\right)}{\partial x}=\left(-Sm_jz_j+S\beta_j\delta_j\omega_j\right)H\left[t-t_j\right], j=1,...,N,$$

где ω_j – скорость изменения относительной толщины сгоревшего свода (см. раздел 2.2).

2.2 Модели межфазного взаимодействия

Величины Π_j и Q_j в (1), которые определяют скорость приращения импульса и энергии в единице объема газовой фазы за счет горения пороховых элементов и межфазного взаимодействия, имеют следующий вид:

$$\begin{cases} \Pi_j=m_jvH\left[t-t_j\right]-\tau_j, j=1,...,N, \\ Q_j=m_j\left(f_j/\left[\gamma_j-1\right]+0.5v^2\right)H\left[t-t_j\right]+v\left(\beta_j\,\partial p/\partial x-\tau_j\right)-q_jH\left[t_j-t\right], j=1,...,N, \end{cases}$$

где f_j – сила j-го пороха, q_j – тепловой поток между газовой и дисперсной фазами, τ_j – сила межфазного трения.

Межфазное тепловое взаимодействие. Как видно из (2), уравнение энергии явно не входит в определяющую систему для дисперсной фазы. Учет межфазного теплообмена происходит в результате решения задачи о прогреве порохового элемента на основе одномерного уравнения теплопроводности:

$$\frac{\partial T_j}{\partial t}=\frac{\lambda_j}{c_j\delta_j}\frac{\partial^2T_j}{\partial y^2},\ \left.\frac{\partial T_j}{\partial y}\right|_{y=e_j}=\alpha_j\left(T-T_j\right);\ \left.T_j\right|_{y=0}=T_{0j};\ T_j\left(y,0\right)=T_{0j},$$

где T_j, λ_j, c_j – температура, коэффициент теплопроводности и теплоемкость j-го сорта пороха, T_{0j} – его начальная температура, e_j – половина толщины свода горения порохового элемента, y – координата, которая отсчитывается от центра порохового свода к его поверхности. Воспламенение порохового элемента происходит в результате его прогрева через время t_j, когда температура на его поверхности достигает значения, соответствующего априорно заданной температуре воспламенения. Отметим, что после того, как температура поверхности порохового элемента достигает температуры воспламенения, температурный профиль внутри свода горения «замораживается» и может учитываться в дальнейшем для коррекции скорости горения. Коэффициент теплоотдачи пороховых газов определяется как:

$$\alpha_j=\left(\lambda/\lambda_j\right)\mathrm{Nu}_j\left(s_j/6\right),$$

где λ – коэффициент теплопроводности пороховых газов, $s_j = \chi_j / e_j$ – площадь межфазной поверхности единицы объема дисперсной фазы для зерненых порохов, χ_j – безразмерный коэффициент формы порохового элемента. Что касается числа Нуссельта Nu_j, то в литературе [4, 6, 7] можно встретить его различные представления применительно к задачам внутренней баллистики. В рамках данной работы используется зависимость, полученная в [8] экспериментально для описания межфазного стационарного теплообмена в насыпных слоях:

$$\mathrm{Nu}_j = \begin{cases} 2.0 + 0.106\varphi \mathrm{Re}_j \mathrm{Pr}^{1/3} \ \text{ при } \ \mathrm{Re}_j \le 200.0, \\ 2.27 + 0.6\left(\varphi \mathrm{Re}_j\right)^{2/3} \mathrm{Pr}^{1/3} \ \text{ при } \ \mathrm{Re}_j > 200.0. \end{cases}$$

Число Рейнольдса Re_j для зерненого пороха и число Прандтля Pr определяются по формулам:

$$\mathrm{Re}_j = 6 e_j \rho |u - v| \big/ \left(\mu \chi_j\right), \ \ \mathrm{Pr} = \mu c_p / \lambda,$$

где $\mu(T)$ – динамическая вязкость пороховых газов, а c_p – теплоемкость пороховых газов при постоянном давлении.

Ввиду использования Эйлерова подхода, в модель вводится уравнение для переноса температурного профиля $T_j(y)$ внутри порохового элемента:

$$\frac{\partial\left(S\beta_j\delta_j T_j\right)}{\partial t} + \frac{\partial\left(S\beta_j\delta_j v T_j\right)}{\partial x} = -Sm_j T_j H\left[t - t_j\right] + S\beta_j \frac{\lambda_j}{c_j}\frac{\partial^2 T_j}{\partial y^2} H\left[t_j - t\right], \ j = 1,...,N.$$

Тепловой поток между газовой и дисперсной фазами, который рассчитывается после решения уравнения теплопроводности, вычисляется в соответствии с законом Ньютона:

$$q_j = \beta_j s_j^2 \lambda \mathrm{Nu}_j \left(T - T_{sj}\right) \big/ 6,$$

где T_{sj} – температура поверхности порохового элемента, а T – температура окружающего газа.

Подобным образом рассчитывается тепловой поток в стенки каморы и в ствол:

$$Q_w = K_m \lambda \mathrm{Nu}\left(T - T_w\right) \big/ r^2,$$

где T_w – температура ствола, которая считается постоянной, r – радиус ствола, K_m – параметр согласования, изменяющийся от 0 до 1. Введение параметра согласования связано с отсутствием учета прогрева ствола, который приводит к тому, что тепловой поток в процессе выстрела снижается из-за уменьшения разности температуры газа и стенки. В качестве числа Нуссельта используется зависимость для случая теплоотдачи при турбулентном течении жидкости в прямой круглой трубе [9, 10]:

$$\mathrm{Nu} = 0.023 \mathrm{Re}^{0.8} \mathrm{Pr}^{0.4} \left(\mathrm{Pr}/\mathrm{Pr}_w\right)^{0.25}, \ \ \mathrm{Re} = 2r\rho|u|/\mu, \ \ \mathrm{Pr}_w = \mu\left(T_w\right) c_p / \lambda.$$

Межфазное силовое взаимодействие. Сила сопротивления τ_j, действующая на газ со стороны j-го компонента дисперсной фазы, определяется формулой:

$$\tau_j = \beta_j s_j C_j \rho |u - v| \left(u - v\right) / 8, \ j = 1,...,N,$$

где C_j – коэффициент сопротивления. Для зерненого пороха выражение для силы сопротивления с учетом изменения формы порохового зерна в результате горения имеет следующий вид:

$$\tau_j^{зерн} = \frac{1}{8}\frac{\beta_j}{e_j\left(1 - \psi_j\right)}\frac{d\psi_j}{dz_j} C_j^{зерн} \rho |u - v| \left(u - v\right), \ C_j^{зерн} = 2.33 + 200\frac{1 - \varphi}{\varphi \mathrm{Re}_j},$$

где $\psi(z) = \chi z(1 + \lambda z + \mu z^2)$ – относительная масса сгоревшего пороха, χ, λ и μ – безразмерные коэффициенты формы порохового элемента, а приведенное выражение для коэффициента сопротивления – формула Эргана [11] – отвечает диапазону пористости от 0.4 до 0.75. Для других диапазонов используются зависимости из [4].

Горение пороховых элементов. Газоприход m_j, который начинается по окончании воспламенительного периода, определяется следующим образом [4]:

$$m_j = \left(1 - \xi_j\right)\frac{\beta_j \delta_j \omega_j}{1 - \psi_j}\frac{d\psi_j}{dz_j}, \ \ \omega_j = \frac{U_j}{e_j} p^{\nu_j}, \ j = 1,...,N,$$

где ξ – массовая доля дисперсной фазы в продуктах сгорания, U – множитель, зависящий от начальной температуры пороха, параметров обдувающего потока и состава пороха, а ν – постоянный показатель степени в законе горения.

Молярная масса пороховых газов определяется по температуре горения T_Γ и силе пороха f как $M = RT_\Gamma / f$, где R – универсальная газовая постоянная.

3. Вычислительный алгоритм

Разработанный вычислительный алгоритм решения полученной системы дифференциальных уравнений в частных производных основан на методе расщепления по физическим процессам, явной схеме интегрирования по времени и методе конечных объемов [12] для дискретизации по пространственной переменной. В вычислениях используется подвижная расчетная сетка, один из концов которой связан с движущимся телом.

На первом этапе рассматриваемая система решается в предположении, что межфазное взаимодействие отсутствует, т.е. правые части равны нулю. Таким образом, учитываются изменения параметров среды за счет конвективного переноса. На втором этапе решается система обыкновенных дифференциальных уравнений, описывающая изменение состояния среды за счет межфазного взаимодействия в отсутствие процессов конвективного переноса.

Первый этап, в свою очередь, также расщепляется на решение определяющих систем уравнений для газовой (1) и дисперсной (2) фаз. Суммируя уравнения неразрывности для всех компонентов газовой фазы, перепишем (1) в следующей векторной форме:

$$\partial \mathbf{q}/\partial t + \partial \mathbf{f}/\partial x = \mathbf{g},$$

$\mathbf{q} = \begin{bmatrix} S\varphi\rho & S\varphi\rho u & S\varphi\rho E \end{bmatrix}^T$ – вектор консервативных переменных,

$\mathbf{f} = \begin{bmatrix} S\varphi\rho u & S\varphi\left(\rho u^2 + p\right) & S\varphi\rho u J \end{bmatrix}^T$ – консервативная составляющая вектора потока,

$\mathbf{g} = \begin{bmatrix} 0 & p\,\partial\left(S\varphi\right)/\partial x & 0 \end{bmatrix}^T$ – неконсервативная составляющая вектора потока,

Пусть теперь имеется пространственная дискретизация расчетной области подвижной сеткой $\{x_i\}$. Середины расчетных ячеек будем обозначать через $x_{i+1/2} = 0.5(x_i + x_{i+1})$, а их длину – через $h_{i+1/2} = x_{i+1} - x_i$. Без ограничения общности система дискретных уравнений на неподвижной сетке имеет следующий вид:

$$\mathbf{q}_{i+1/2}^{n+1} = \mathbf{q}_{i+1/2}^{n} - \tau\left(\mathbf{F}_{i+1} - \mathbf{F}_i\right)/h_{i+1/2} + \tau\mathbf{G}_{i+1/2}.$$

Здесь τ – шаг по времени, выбираемый из условия устойчивости Куранта-Фридрихса-Леви, $\mathbf{F}_{i+1}$ и $\mathbf{F}_i$ – потоки через ребра i+1 и i. В случае, если на ребре произведение пористости на площадь поперечного сечения канала не претерпевает существенного скачка, то поток вычисляется методом С.К. Годунова [13]. В противном случае используется линеаризация Роу [14].

Аналогичным образом интегрируется система уравнений для дисперсной фазы. Потоки также вычисляются с использованием схемы Годунова или более диссипативной схемы Русанова.

На втором этапе учитываются источниковые члены в правых частях уравнений, которые определяют межфазное взаимодействие. Полученная система обыкновенных дифференциальных уравнений интегрируется с помощью формул дифференцирования назад.

4. Распараллеливание расчетного алгоритма

Распараллеливание осуществляется методом декомпозиции расчетной области. Было реализовано и апробировано три варианта алгоритма распараллеливания, начиная с самого простого – статической декомпозиции расчетной области, до адаптивного алгоритма с учетом вычислительной загрузки процессорных ядер и оценок времени передачи данных.

Алгоритм 1. В первом варианте параллельного алгоритма было реализовано статическое и равномерное распределение ячеек расчетной области между процессорными ядрами. Подобный подход оказывается оправданным только в случае отсутствия факторов, вызывающих сущест-

венную разбалансировку вычислительной нагрузки в процессе расчета. Данным фактором в рассматриваемых задачах является расчет прогрева и горения пороховых элементов по причине того, что пороховые заряды в большинстве представляющих интерес случаев локализованы в какой-то части каморы.

Таким образом, для части ячеек расчетной области вычислительная нагрузка возрастает в разы по сравнению с другими, и ситуация усугубляется тем, что пороховой заряд может перемещаться по каморе. Вычислительные эксперименты показали, что разбалансировка вычислительной нагрузки сильно снижает эффективность параллельного расчета.

Для повышения эффективности был реализован алгоритм адаптивного распараллеливания, когда множества ячеек, делегированных различным процессорным ядрам, меняются во время расчета, отражая реальную вычислительную нагрузку. Была разработана следующая схема адаптации. Периодически осуществляется замер времени, необходимого для расчета параметров в каждой ячейке на одном шаге интегрирования по времени. В случае принятия решения о необходимости балансировки производится расчет нового распределения ячеек по процессорным ядрам таким образом, чтобы нагрузка была максимально равномерной. Данную тактику иллюстрирует следующий пример. Пусть при расчете 110 ячеек, первые 10 ячеек на предыдущем шаге потребовали 1 секунду процессорного времени каждая, а оставшиеся 100 ячеек – по 0.1 секунды. Тогда в двухъядерной конфигурации распределение ячеек по процессорным ядрам будет следующим – 10 на первом, 100 на втором.

Таким образом, основными операциями адаптивного алгоритма являются следующие:

1. Решение о балансировке, основанное на: (а) сборе времен расчета каждой ячейки на одном процессорном ядре, (б) расчете нового распределения ячеек по собранным во время предыдущей подоперации данным, (в) принятии решения о необходимости балансировки и рассылке этого решения всем процессорам, участвующим в расчете.
2. Перераспределение ячеек в соответствии с результатом первой операции.

Ключевыми вопросами при этом являются:

1. Когда производить проверку на разбалансировку? Поскольку данная проверка требует некоторых ресурсов для сбора временной статистики вычислений на предыдущем шаге на одном процессорном ядре, то проводить данную проверку на каждом шаге нецелесообразно.
2. Проводить ли балансировку после проверки на разбалансировку? В силу того, что балансировка ячеек может требовать передач значительных объемов информации между вычислительными устройствами, несвоевременное устранение незначительной разбалансировки может в итоге даже замедлить расчет.

Были опробованы следующие 2 схемы.

Алгоритм 2. Первая операция проводится каждые N шагов. Если новое распределение ячеек, рассчитанное на основе статистики за предыдущий шаг, оказывается отличным от текущего распределения, то проводится балансировка. Данная схема позволила отладить базовые механизмы работы адаптивного алгоритма. Как и следовало ожидать, она работает ощутимо лучше статического распараллеливания, когда каждая ячейка привязывается к процессорному ядру один раз в начале расчета. Расчеты с данной схемой распараллеливания показали следующее. Проверка на разбалансировку и расчет нового распределения ячеек занимает малое время по сравнению со временем всего расчета – менее 2% для задачи с сеткой в 436 ячейки при расчете на 16 ядрах. При этом действительное перераспределение ячеек занимает большое время, которое сопоставимо с расчетом одного шага по времени.

Отсюда можно сделать следующие выводы:

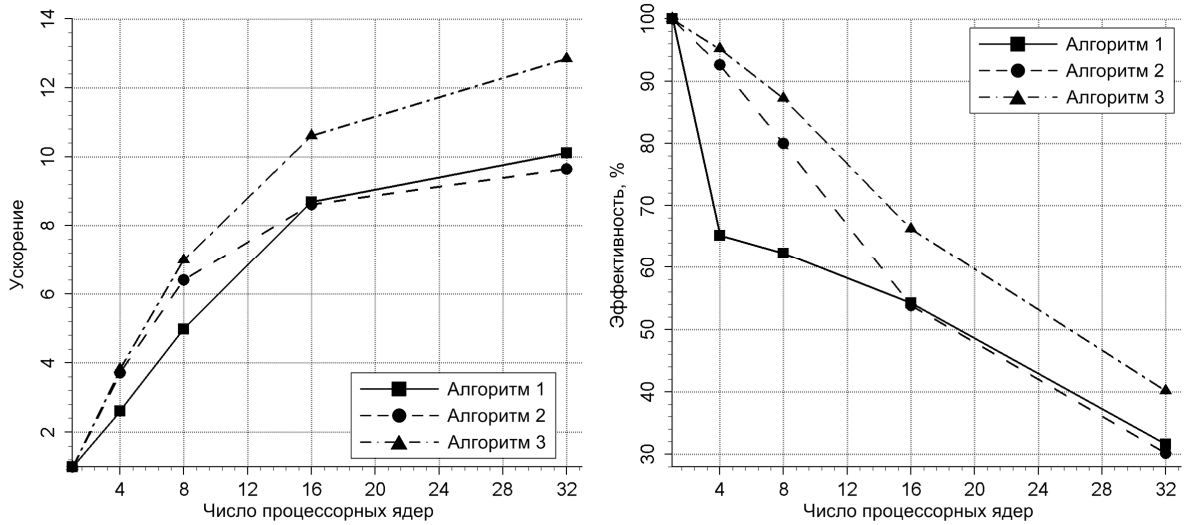
1. Не имеет смысла разрабатывать сложные алгоритмы для определения момента проверки на разбалансировку, так как, с одной стороны, сама проверка является достаточно дешевой операцией, а с другой стороны, для оптимизации проверок на разбалансировку необходимо предсказывать, как быстро в будущем будет идти разбалансировка. Таким образом, для оптимизации проверок на разбалансировку необходимо разработать сложный алгоритм без гарантии успеха, а получить минимальный выигрыш по времени расчета.
2. Необходимо свести к минимуму количество перераспределений ячеек, т.е. не всегда проводить балансировку, если нагрузка на процессорные ядра стала разбалансированной.

Алгоритм 3. Первая операция, состоящая в проверке на разбалансировку и расчете нового распределения, проводится, как и в простой схеме, каждые N шагов. Введем в рассмотрение

функцию Q, возвращающую время параллельного расчета предыдущего шага по времени с распределением ячеек D, параметрами которой являются времена расчета ячеек на предыдущем временном шаге S и распределение ячеек D. Функция Q от текущего распределения ячеек D_{curr} даст время расчета предыдущего шага. Во время проведения первой операции адаптивного алгоритма возможно вычислить идеальное распределение D_{ideal}, на котором расчет предыдущего шага был бы максимально сбалансированным. Функция Q на таком распределении дала бы минимальное время расчета для предыдущего шага. Таким образом, разница $T_{lost} = Q(S, D_{curr}) - Q(S, D_{ideal})$ определяет время, которое было потеряно на предыдущем шаге по времени вследствие разбалансировки задачи. Сумму всех T_{lost}, которые были рассчитаны во время предыдущих проверок на разбалансированность после последнего перераспределения ячеек, обозначим как A_{lost}. Таким образом, A_{lost} характеризует количество потерянного машинного времени с последнего перераспределения ячеек.

Проведем замер времени, которое требуется на перераспределение ячеек, выбирая среднее среди последних k перераспределений, и обозначим его как T_{move}. И будем перераспределять ячейки, т.е. запускать вторую операцию адаптивного алгоритма, если $A_{lost} \cdot N \cdot \alpha > T_{move}$, где α – эмпирическая константа. Было проведено несколько тестовых серий расчетов и выбраны константы $N = 10$ и $\alpha = 0.5$. Выбор констант затрудняется тем, что маловероятно существование оптимального варианта для всех типов задач и тем, что проведение серий тестовых расчетов для каждого набора констант требует значительного машинного времени. Тестирование проводилось на задаче с расчетной сеткой 436 ячеек, что в статическом распараллеливании дает не более 14 ячеек в 32-ух ядерной конфигурации. Расчеты проводятся на СК МВС-100k Межведомственного суперкомпьютерного центра РАН и СК СКИФ МГУ «Чебышев». Расчеты реальных конфигураций крупнокалиберных баллистических установок занимают порядка 10 минут при использовании 32-ух процессорных ядер.

Как видно из Рис. 1, Алгоритм распараллеливания номер 3 демонстрирует существенное преимущество в ускорении и эффективности по сравнению с Алгоритмами 1 и 2.



(а) (б)

Рис. 1. Зависимость (а) ускорения и (б) эффективности распараллеливания вычислительного алгоритма для трех различных подходов к распараллеливанию.

5. Расчет выстрела из модельной пушки AGARD

Для верификации разработанного программного комплекса был выбран тест AGARD (Advisory Group for Aerospace Research and Development – Консультативная группа аэрокосмических исследований и разработок), разработанный рядом ведущих западных компаний, занимающихся созданием пакетов прикладных программ для решения задач внутренней баллистики. Тест AGARD не соответствует никакому натурному опыту, а служит для сравнения различных пакетов программ «код в код» [15 – 16]. Для сравнения были использованы результаты,

полученные с помощью кодов, некоторые характеристики которых представлены в Таб. 1. Во всех этих кодах реализованы двухфазные модели движения газопороховой смеси.

Постановка задачи с геометрическими характеристиками изображена на Рис. 2. Заряд, заполняющий всю камору, состоит из 7-ми канального зерненого пироксилинового пороха. Путь снаряда составляет 4318 мм, давление форсирования – 13.79 МПа. В начальный момент времени температура воздуха и пороха составляет 294 К, начальное давление – 1 атм. Молярная масса воздуха – 29 г/моль, показатель адиабаты – 1.4. Тепловые потери в стенки каморы и ствол считаются пренебрежимо малыми. Свойства пороха содержатся в Таб. 2. Характеристики воспламенителя следующие: сила пороха 1.5702 МДж/кг, показатель адиабаты продуктов горения 1.25, температура горения 1706 К. Начальная объемная доля пороха в каморе равна 0.5796.

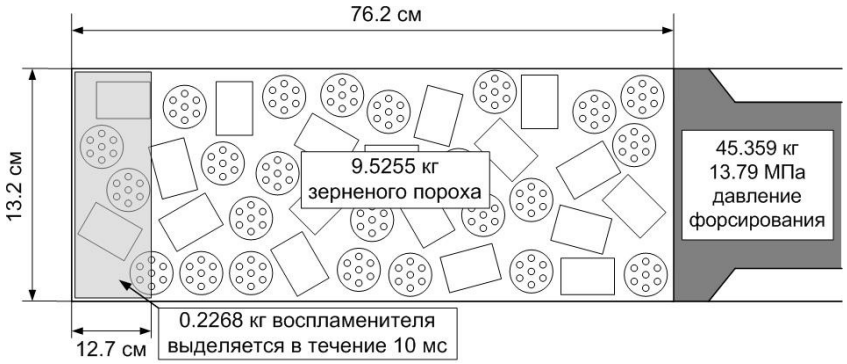


Рис. 2. Схема выстрела из модельной пушки AGARD.

Среди постановочных данных, приведенных в [15], отсутствует коэффициент теплопроводности пороховых газов, который в расчетах принимался постоянным и равным 0.1 Вт/м/К. Стоит отметить, что поиск в открытой печати специфических экспериментальных данных по температурным зависимостям коэффициентов теплопроводности и вязкости пороховых газов в условиях, соответствующих артиллерийскому выстрелу, представляет собой существенную проблему. Более того, имеющиеся данные по величинам этих коэффициентов нередко разнятся на порядки, что подчеркивает необходимость дальнейших экспериментальных исследований внутрибаллистических процессов для улучшения предсказательной способности разрабатываемых математических моделей и соответствующих комплексов программ.

В зависимости (3) для межгранулярного давления используются следующие значения параметров: $B = 1.5 \cdot 10^3$ атм, $k = 0.57$, $\beta_0 = 0.56$. Значение β_0, несколько меньшее, чем объемная доля начальной засыпки, соответствует тому, что изначально порох находится в каморе в спрессованном состоянии. Параметры для межгранулярного давления подобраны таким образом, чтобы обеспечивать нужную по постановке задачи скорость волны компактирования.

По заданным размерам порохового зерна были рассчитаны коэффициенты формы, отвечающие прогрессивной ($\chi_1 = 0.7185$, $\lambda_1 = 0.2049$, $\mu_1 = -0.0217$) и дегрессивной ($\chi_2 = 0.5386$, $\lambda_2 = -0.8977$, $\mu_2 = 0.0$) стадиям горения зерна [1].

Таблица 1. Некоторые характеристики распространенных западных внутрибаллистических кодов для решения задач внутренней баллистики.

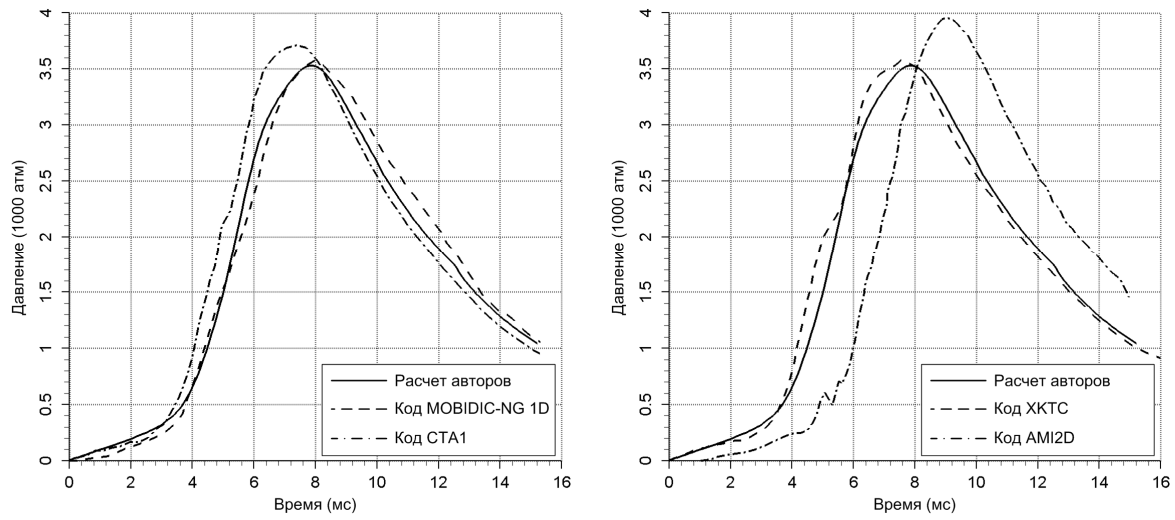
Код	Организация	Размерность	Особенности
MOBIDIC-NG 1D	ISL/ETBS (Франция)	Одномерный	Расчет прогрева пороховых элементов, см. [17]
CTA1	QinetiQ (Великобритания)	Одномерный	Нет расчета прогрева пороховых элементов, учет сил межфазного трения, нет межгранулярного напряжения
XKTC	Aberdeen Proving Ground (США)	Одномерный	Расчет прогрева пороховых элементов, учет межгранулярного напряжения, учет тепловых потерь в ствол
AMI2D	ISL/ETBS (Франция)	Двумерный	Расчет прогрева пороховых элементов

Таблица 2. Характеристики пороха в тесте AGARD.

Характеристика	Величина
Плотность δ, г/см^3	1.578
Длина зерна / диаметр зерна / диаметр канала зерна, мм	25.4 / 11.43 / 1.143
Коэффициент U в законе горения, см/с/МПа$^{0.9}$	0.078385
Показатель ν в законе горения, безразмерный	0.9
Температура горения $T_Г$, К	2585
Температура воспламенения, К	444
Коэффициент теплопроводности λ, Вт/м/К	0.2218
Теплоемкость c, Дж/кг/К	1620
Показатель адиабаты продуктов γ, безразмерный	1.27
Сила пороха f, МДж/кг	1.009
Коволюм b, см^3/г	1.0838

Рассмотрим динамику процесса выстрела. На Рис. 3 представлено сравнение рассчитанных различными внутрибаллистическими кодами кривых давления на дно каморы. Основной вывод, который можно сделать из сравнения кривых, заключается в том, что результаты, полученные разработанным программным комплексом хорошо соотносятся с результатами, полученными другими внутрибаллистическими кодами. На Рис. 4а более детально показана начальная стадия внутрибаллистического процесса до момента воспламенения всего заряда.

Сравнение разности давлений на датчиках, расположенных на расстоянии 10 мм и 750 мм от дна каморы, представленное на Рис. 4б, показывает, что фазы волн давления, полученные разработанным программным комплексом, отличаются от таковых для других кодов.

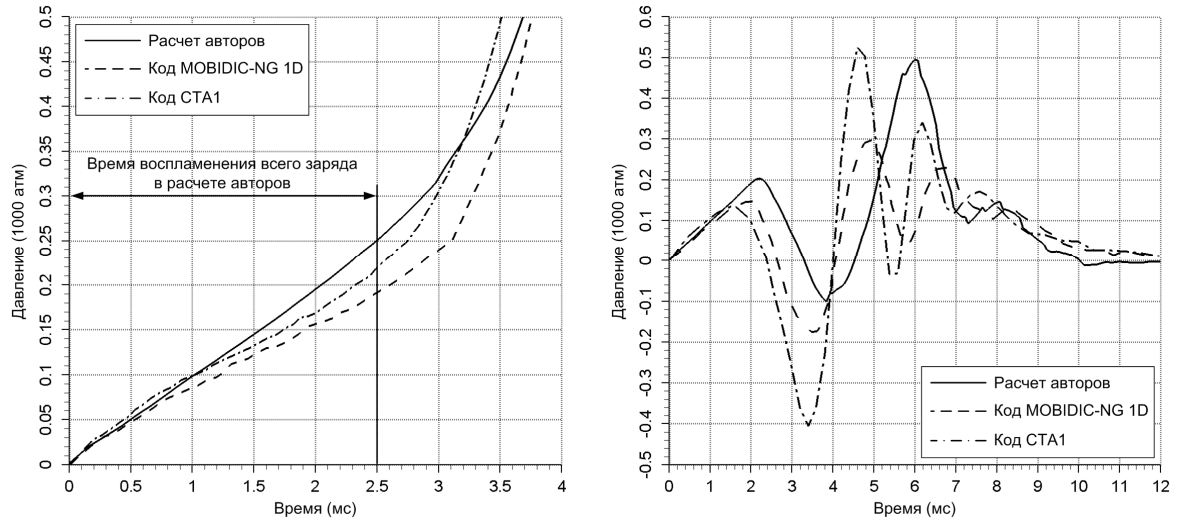


(а) (б)

Рис. 3. Сравнение кривых давления на дно каморы в процессе выстрела, полученных с помощью разработанного программного комплекса и других кодов для решения задач внутренней баллистики.

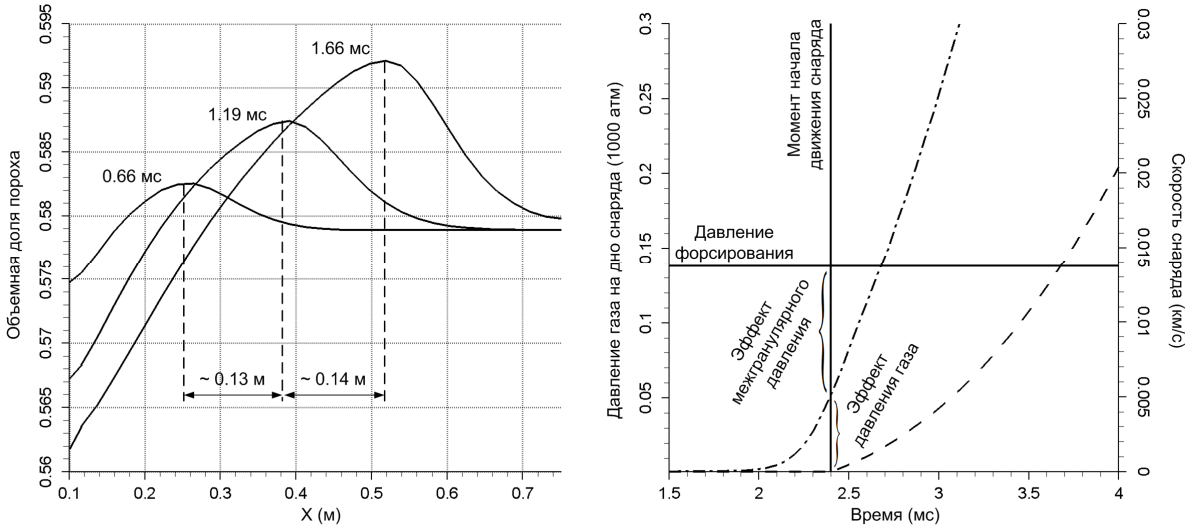
Одна из особенностей постановки рассматриваемой задачи заключается в том, что камора в начальный момент времени целиком заполнена порохом. Таким образом, становится возможным оценить эффекты, связанные с межгранулярным давлением. При указанных выше в данном разделе параметрах межгранулярного давления скорость волны компактирования составляет от 250 до 300 м/с (см. Рис. 5а), что удовлетворительно соотносится с требуемой скоростью 254 м/с [15]. Распространение по пороху волны компактирования приводит к тому, что метаемое тело начинает свое движение не только за счет давления газа, но и за счет межгранулярного давления. На Рис. 5б видно, что в момент начала движения снаряда, когда суммарное давление на его дно равняется давлению форсирования, давления газа составляет всего около 35% от давления форсирования. Остальной вклад обеспечивается межгранулярным давлением. Что касается количественных характеристик, то в расчете разработанным программным комплексом снаряд начинает движение в момент времени 2.4 мс, давление газа на его дно при этом со-

ставляет около 50 атм. В расчете кодом ХКТС (см. Таб. 1) аналогичные характеристики – 2.2 мс и 75 атм [15].



(а) (б)

Рис. 4. (а) Сравнение кривых давления на дано каморы на начальном этапе внутрибаллистического процесса, (б) Разность давлений на датчиках, расположенных на расстоянии 10 мм и 750 мм от дна каморы.



(а) (б)

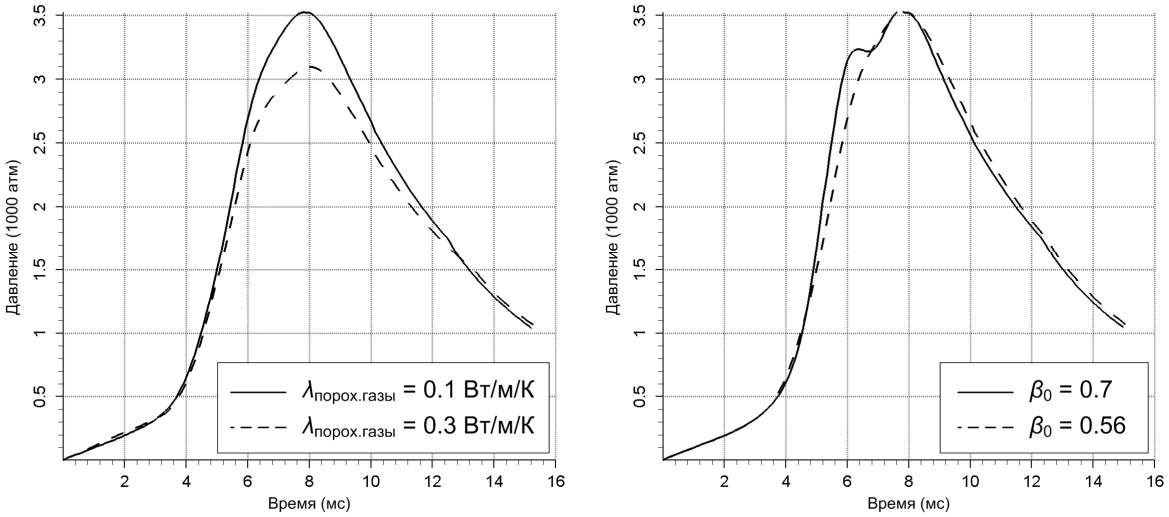
Рис. 5. (а) Распределения объемной доли пороха внутри каморы в последовательные моменты времени; (б) Влияние межгранулярного давления на начало движения снаряда. Пунктирная кривая – зависимость скорости снаряда от времени, штрихпунктирная – давления на дно снаряда от времени.

Полученные результаты хорошо соотносятся с данными, полученными с помощью других кодов, и по основным интегральным характеристикам – величинам максимального давления на дно каморы ($p_{кам}$), на дно снаряда ($p_{сн}$), по дульной скорости ($v_д$) и времени выстрела ($\tau_{выс}$) (см. Таб. 3).

Стоит, тем не менее, отметить, что расчеты внутрибаллистических процессов могут быть сильно чувствительными к изменениям таких параметров математической модели, как коэффициенты теплопроводности пороха и пороховых газов или коэффициенты в зависимости для межгранулярного давления. Отличие может быть не только количественным, в величинах максимального давления в каморе (см. Рис. 6а) или скорости вылета снаряда, но и качественным (см. Рис. 6б). Возможность получения кривых давления на дно каморы с несколькими пиками при расчете теста AGARD отмечается в [16].

Таблица 3. Сравнение интегральных характеристик выстрела из пушки AGARD.

	$p_{кам}$, МПа	$p_{сн}$, МПа	$v_д$, м/с	$\tau_{выс}$, мс
Программный комплекс авторов	353	318	695	15.20
Код CTA1	373	343	681	14.94
Код MOBIDIC-NG 1D	355	325	685	15.46
Код XKTC	350	324	683	16.58
Код AMI2D	395	367	710	16.47



(а) (б)

Рис. 6. Рассчитанные кривые давления на дно каморы в процессе выстрела для (а) различных коэффициентов теплопроводности пороховых газов и (б) для различных зависимостей для межгранулярного давления.

6. Заключение

В работе представлены математическая модель и вычислительный алгоритм для исследования внутрибаллистического процесса в квазиодномерном приближении с использованием многопроцессорной вычислительной техники. Описана методика адаптивного распараллеливания расчетного алгоритма, а также проведен анализ эффективности распараллеливания. Разработанный программный комплекс верифицирован с помощью тестовой задачи о выстреле из пушки AGARD, результаты расчета которой известны для большого количества зарубежных внутрибаллистических кодов. Представленные в работе математическая модель и вычислительный алгоритм являются основой для создания баллистического кода для решения многомерных задач внутренней баллистики.

Разработанный «Программный комплекс БАРС-1МП» подан на регистрацию в Федеральные органы по интеллектуальной собственности и используется для моделирования внутрибаллистического процесса в крупнокалиберных ствольных установках [18].

Авторы выражают глубокую признательность академикам РАН О.М. Белоцерковскому (ИАП РАН) и В.Б. Бетелину (НИИСИ РАН) за всестороннюю поддержку проводимых исследований, С.М. Фролову, Б.С. Ермолаеву, В.С. Посвянскому (ИХФ РАН) за помощь в работе, а также В.В. Чернову (ОАО «ЦНИИ «Буревестник») за экспериментальную поддержку проводимых теоретических исследований.

Литература

1. Серебряков М.Е. Внутренняя баллистика ствольных систем и пороховых ракет. – М.: Оборонгиз, 1962.

2. Klemens R., Gieras M., Kaluzny M. Dynamics of dust explosions suppression by means of extinguishing powder in various industrial conditions // Journal of Loss Prevention in the Process Industries. – 2007. – Vol. 20, № 4 – 6. – P. 664 – 674.
3. Porterie B., Loraud J.C. An investigation of interior ballistics ignition phase // Shock Waves. – 1994. – Vol. 4. – P. 81 – 93.
4. Хоменко Ю.П., Ищенко А.Н., Касимов В.З. Математическое моделирование внутрибаллистических процессов в ствольных системах. – Новосибирск: Изд-во СО РАН, 1999.
5. Русяк И.Г., Ушаков В.М. Внутрикамерные гетерогенные процессы в ствольных системах. – Екатеринбург: УрО РАН, 2001.
6. Nusca M.J., Controy P.J. Multiphase CFD Simulations of Solid Propellant Combustion in Gun Systems // Proceedings of DoD High Performance Computing Modernization Program 2001 Users Group Conference, Biloxi, MS, 18 – 21 June 2001.
7. Нигматулин Р.И. Динамика многофазных сред, Т. 1. – М.: Наука, 1987.
8. Чудновский А.Ф. Теплообмен в дисперсных средах. – М.: Гостехиздат, 1954.
9. Захаренков В.Ф. Полуэмпирический метод расчета теплообмена в гладких и шероховатых трубах при течении горячих газовых потоков // Труды Международной научно-практической конференции «Третьи Окуневские чтения», Санкт-Петербург, 24 – 29 июня 2002 г. – Т. 2. – С. 176 – 185.
10. Кутателадзе С.С., Боришанский В.М. Справочник по теплопередаче. – М.: Госэнергоиздат, 1958.
11. Ergun S. Fluid Flow through Packed Columns // Chemical Engineering Progress. – 1952. – Vol. 48, № 2. – P. 89 – 94.
12. Barth T., Ohlberger M. Finite Volume Methods: Foundation and Analysis // Encyclopedia of Computational Mechanics. – 2004. – Vol. 1. – P. 439 – 470.
13. Годунов С.К, Забродин А.В., Иванов М.Я., Крайко А.Н., Прокопов Г.П. Численное решение многомерных задач газовой динамики. – М.: Наука, 1976.
14. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. – М.: Физматлит, 2001.
15. Woodley C., Carriere A., Franco P., Groger T., Hensel D., Nussbaum J., Kelzenberg S., Longuet B. Comparisons of Internal Ballistics Simulations of the AGARD Gun // Proceedings of 22nd International Symposium on Ballistics. Vancouver, Canada, November 2005. – P. 338 – 346.
16. Gollan R.J., Johnston I.A., O'Flaherty B.T., Jacobs P.A. Development of Casbar: a Two-phase Flow Code for the Interior Ballistics Problem // Proceedings of 16th Australasian Fluid Mechanics. Crown Plaza, Gold Coast, Australia, 2 – 7 December 2007. – P. 295 – 302.
17. Longuet B., Pieta P.D., Franco P., Legeret G., Papy A., Boisson D., Reynaud C., Millet P., Taiana E., Carrere A. Mobidic-NG: A 1D/2D CFD Code Suitable for Interior Ballistics and Vulnerability Modelling // Proceedings of 22nd International Symposium on Ballistics. Vancouver, Canada, November 2005. – P. 362 – 371.
18. Закаменных Г.И., Чернов В.В., Абдуллин А.К., Семенов И.В., Уткин П.С., Лебедева А.Ю., Ахмедьянов И.Ф. Экспериментальное и численное исследование влияния положения модульного заряда на характеристики выстрела // Сборник материалов Всероссийской научно-технической конференции «Фундаментальные основы баллистического проектирования». Санкт-Петербург, 28 июня – 2 июля 2010 г. Том 1 / Под ред. д.т.н. проф. Кэрта Б.Э. – СПб.: Балт. гос. техн. ун-т, 2010. – С. 119 – 122.

О перспективах достижения экзафлопс-производительности в расчетах на основе метода частиц-в-ячейках*

А.В. Снытников

Проведенные физические оценки показали, что для адекватного воспроизведения спектра плазменной турбулентности с помощью метода частиц в ячейках необходимо проведение трехмерных расчетов с размером сетки не менее 2000 узлов по каждому направлению (и не менее 1000 частиц в ячейке). В перспективе это потребует использования экзафлопс-компьютеров. В докладе представлены два варианта реализации алгоритма для моделирования взаимодействия электронного пучка с плазмой, которые позволят достигать эффективно использовать экзафлопс–вычислители, когда они будут созданы. Первый вариант — это трехмерная декомпозиция расчетной области, с помощью которой уже сейчас достигается высокая масштабируемость. При этом для расчета в отдельной подобласти используется группа процессоров (или ядер), что позволяет не ограничивать снизу объем подобласти. Второй вариант предполагает использование графического ускорителя вместо группы процессоров для расчета отдельной подобласти.

1. Введение

Актуальность данной работы связана с необходимостью точного расчета характеристик неустойчивостей, возбуждаемых релятивистским электронным пучком в субтермоядерной плазме на установке ГОЛ-3 (ИЯФ СО РАН), а также выяснения причин возникновения аномальной электронной теплопроводности [2]. В силу того, что плазма в данном случае является существенно неравновесной, использование упрощенных (например, магнитогидродинамических) методов невозможно, расчет необходимо производить в рамках кинетического подхода, то есть с помощью метода частиц в ячейках, что требует больших вычислительных ресурсов. Конкретной физической задачей в данном случае является расчет спектра плазменной турбулентности. Для ее решения необходимо проведение трехмерных расчетов с высоким разрешением по всем трем координатам в силу того, что плазменные колебания в данном случае являются существенно анизотропными.

Также в последнее время ведутся интенсивные дискуссии о проблемах создания программного обеспечения для компьютеров с производительностью порядка 10^{18} операций с плавающей точкой в секунду (экзафлопс–компьютеров) [1]. В статье делается попытка оценить пригодность метода частиц–в–ячейках для достижения экзафлопс–производительности. Применение метода частиц принципиально необходимо в тех случаях, когда физическая ситуация является неясной, и нет возможности сделать упрощающие предположения, однако этот метод очень ресурсоемкий, что ограничивает его использование. Поэтому возможность реализации метода частиц–в–ячейках на экзафлопс–компьютерах, с одной стороны, решит проблему низкой скорости работы метода, с другой стороны, позволит достичь экзафлопс–производительности быстрее, чем на других физических приложениях, таких как гидро/газо–динамика, теория упругости и так далее (такое возможно в силу большей локальности данных в методе частиц по сравнению с решением СЛАУ, методами конечных элементов или конечных объемов).

*Работа выполнена при поддержке Российского Фонда Фундаментальных Исследований, гранты 11-01-00249, 11-01-00178 и 11-01-00459, а также интеграционных проектов СО РАН № 113 и № 26.

2. Описание алгоритма для моделирования взаимодействия электронного пучка с плазмой

2.1. Описание модели

Численная модель, используемая для решения задачи о релаксации пучка, состоит из уравнений Власова для электронной и ионной компонент плазмы и системы уравнений Максвелла. В общепринятых обозначениях эта система имеет следующий вид:

$$\frac{\partial f_{i,e}}{\partial t} + \vec{v}\frac{\partial f_{i,e}}{\partial \vec{r}} + \vec{F}_{i,e}\frac{\partial f_{i,e}}{\partial \vec{p}} = 0, \qquad \vec{F}_{i,e} = \frac{q_{i,e}}{m_{i,e}}\left(\vec{E} + \frac{1}{c}[\vec{v}, \vec{B}]\right)$$

$$rot\vec{B} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}$$

$$rot\vec{E} = -\,-\frac{1}{c}\frac{\partial \vec{B}}{\partial t} \tag{1}$$

$$div\vec{E} = 4\pi\rho$$

$$div\vec{B} = 0$$

В данной работе используется алгоритм решения системы уравнений (1), описанный в работе [3]. Далее все уравнения буду приводится в безразмерном виде. Для обезразмеривания используются следующие базовые величины:

- характерная скорость $\tilde{v}$– скорость света $\tilde{v} = c = 3 \times 10^{10}$ см/с
- характерная плотность плазмы $\tilde{n} = 10^{14}$ см$^{-3}$
- характерное время $\tilde{t}$– плазменный период (величина, обратная к электронной плазменной частоте) $\tilde{t} = \omega_p^{-1} = \left(\frac{4\pi n_0 e^2}{m_e}\right)^{-0.5} = 5.3 \times 10^{-12}$ с

Уравнения Власова решаются методом частиц в ячейках (PIC). В рамках этого метода решаются уравнения движения модельных частиц, которые являются уравнениями характеристик для уравнения Власова. Величины с индексом i соответствуют ионам, с индексом e – электронам.

$$\frac{\partial \vec{p}_e}{\partial t} = -\left(\vec{E} + [\vec{v}_e, \vec{B}]\right)$$

$$\frac{\partial \vec{p}_i}{\partial t} = \kappa\left(\vec{E} + [\vec{v}_i, \vec{B}]\right)$$

$$\frac{\partial \vec{r}_{i,e}}{\partial t} = \vec{v}_{i,e}, \qquad \kappa = \frac{m_e}{m_i}, \qquad \vec{p}_{i,e} = \gamma\vec{v}_{i,e}, \gamma^{-1} = \sqrt{1 - v^2}$$

Для решения уравнений движения используется схема с перешагиванием:

$$\frac{\vec{p}_{i,e}^{\,m+1/2} - \vec{p}_{i,e}^{\,m-1/2}}{\tau} = q_i\left(\vec{E}^m + \left[\frac{\vec{v}_{i,e}^{\,m+1/2} - \vec{v}_{i,e}^{\,m-1/2}}{2}, \vec{B}^m\right]\right)$$

$$\frac{\vec{r}_{i,e}^{\,m+1} - \vec{r}_{i,e}^{\,m}}{\tau} = \vec{v}_{i,e}^{\,m+1/2},$$

здесь τ – временной шаг. Для нахождения электрических и магнитных полей используется схема, в которой поля определяются из разностных аналогов законов Фарадея и Ампера [3].Эта схема имеет второй порядок аппроксимации по пространству и по времени.

2.2. Постановка задачи

Рассмотрим следующую постановку задачи. В начальный момент в трехмерной области решения (размер области L), которая имеет форму прямоугольного параллелепипеда:

$$0 \le x \le L_X, \qquad 0 \le y \le L_Y, \qquad 0 \le z \le L_Z$$

находится плазма, состоящая из электронов и ионов. Модельные частицы распределены по области равномерно. Задается плотность плазмы и температура электронов, температура ионов считается нулевой. Дополнительно в области присутствуют электроны пучка, которые также распределены по области равномерно (предполагается, что пучок уже вошел в расчетную область). Электроны пучка отличаются от электронов плазмы тем, что они имеют кинетическую энергию направленного движения $\varepsilon = 1$ МэВ, а их температура равна нулю. Модельные частицы, соответствующие электронам пучка, имеют меньшую массу, нежели модельные частицы, соответствующие электронам плазмы (отношение их масс равно отношению плотности плазмы и плотности пучка). Итак, исходными параметрами задачи являются: плотность и температура электронов плазмы, отношение плотности электронов плазмы к плотности электронов пучка, энергия электронов пучка:

- плотность электронов плазмы $n_0 = 10^{17}$ см^3. Плотность плазмы намеренно завышена по сравнению с реальной плазмой на установке ГОЛ-3 для того, чтобы эффекты релаксации пучка проявлялись в течение более короткого времени
- температура электронов плазмы $T_0 = 1$ КэВ
- отношение плотности электронов пучка к плотности электронов плазмы $\alpha = 10^{-3}$
- энергия электронов пучка $\varepsilon = 1$ МэВ

2.3. Параллельная реализация

Распараллеливание выполнено методом декомпозиции расчетной области по направлению, перпендикулярному направлению движения электронного пучка. Используется смешанная эйлерово–лагранжева декомпозиция. Сетка, на которой решаются уравнения Максвелла, разделена на одинаковые подобласти по одной из координат. С каждой подобластью связана группа процессоров (в том случае, когда вычисления производятся на многоядерных процессорах, процессором для единообразия будет именоваться отдельное ядро). Далее, модельные частицы каждой из подобластей разделяются между процессорами связанной с этой подобластью группы равномерно, вне зависимости от координаты.

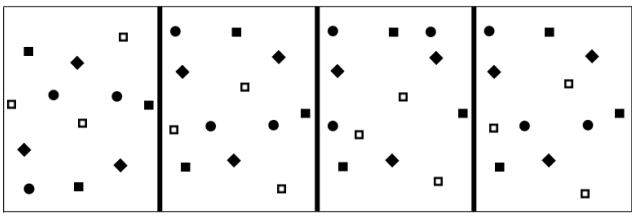


Рис. 1. Одномерная декомпозиция области. Область решения разбита вдоль координаты Y на 4 подобласти, частицы каждой подобласти равномерно распределены между четырьмя процессорами независимо от координаты. Различные символы, обозначающие частицы: круг, квадрат, ромб, полый квадрат означают принадлежность частиц к разным процессорам.

Каждый из процессоров группы решает уравнения Максвелла во всей подобласти. Далее решаются уравнения процессоров группы производит обмен граничными значениями тока и полей с соседними подобластями, и затем рассылает полученные граничные значения всем процессорам своей группы.

2.4. Эффективность распараллеливания

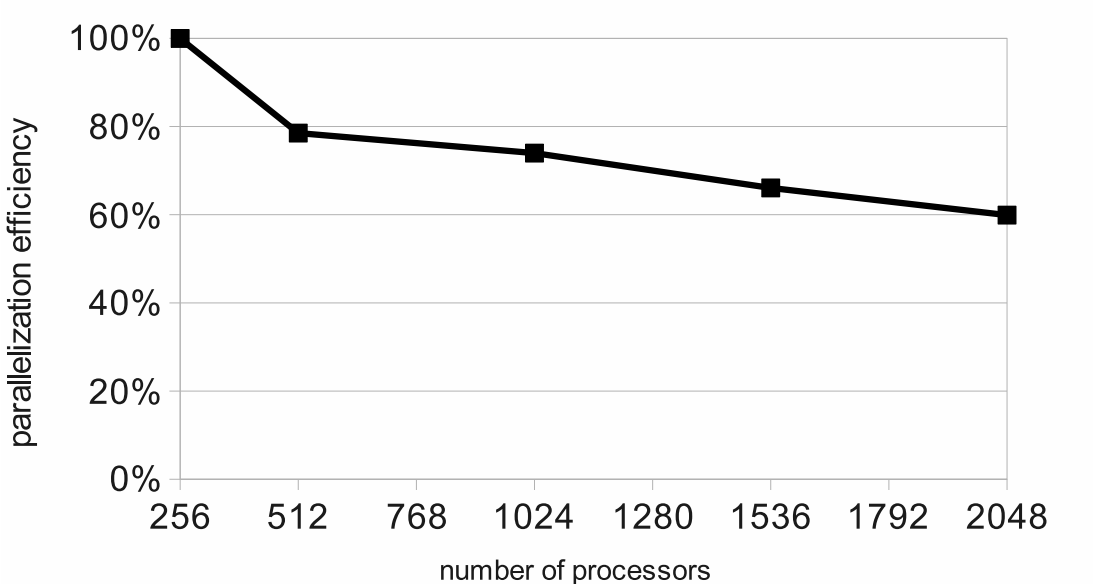


Рис. 2. Эффективность распараллеливания, вычисляемая по формуле 2. Расчеты проведены на МВС-100К, МСЦ РАН.

Основная цель создания параллельной программы для моделирования аномальной теплопроводности в плазме — возможность счета на больших сетках и с большим количеством модельных частиц. Поэтому эффективность распараллеливания вычислялась следующим образом

$$k = \frac{T_2}{T_1} \times \frac{N_1}{N_2} \times \frac{S_2}{S_1} \times 100\% \qquad (2)$$

здесь T_1 - время счета с использованием N_1 процессоров, T_2 – время счета с использованием N_2 процессоров, S_1, S_2 – характерные размеры задач, в данном случае число узлов сетки по координате X. При этом размер задачи увеличивается пропорционально числу процессоров, т.е. нагрузка на отдельный процессор не возрастает. Цель такого определения эффективности – понять, насколько увеличивается время счета при увеличении числа процессоров и неизменной нагрузке на один процессор. В идеале время должно остаться тем же самым ($k = 100\%$ в идеале).

За основу для сравнения при этом берется модель, требующая большого времени вычислений (сетка $256 \times 128 \times 128$ узлов, 150 частиц в ячейке). В расчетах по определению эффективности увеличивается только размер сетки по X, все остальные параметры остаются неизменными. Вопрос о том, насколько быстрее можно посчитать на суперкомпьютере задачу, которую можно посчитать и на настольной машине, в данной работе не рассматривается: такие задачи не представляют интереса с физической точки зрения. Из этого рисунка можно сделать вывод, что созданная программа способна эффективно использовать более тысячи процессорных ядер. С другой стороны, падение эффективности довольно заметное, и очевидно, что используемая двумерная декомпозиция не сможет обеспечить эффективность для десятков тысяч и более ядер.

2.5. Аномальная теплопроводность в вычислительных экспериментах

Актуальность настоящей работы связана с тем, что в экспериментах на многопробочной магнитной ловушке ГОЛ-3 (ИЯФ СО РАН) наблюдается понижение электронной теплопроводности на 2–3 порядка по сравнению с классическим значением. Известно множество работ по моделированию теплопроводности в термоядерных установках. Вычисление температуры в этих работах производится в основном с помощью гидродинамических уравнений. Таким образом функция распределения электронов по энергиям предполагается максвелловской, что может не соответствовать действительности. Для вычисления неравновесных распределений применяют, в частности, бессеточные модификации метода частиц. Однако для метода частиц в ячейках не известны критерии, позволяющие понять, насколько правильным является полученное в расчетах распределение температуры, и таким образом, корректность моделирования теплопроводности также оказывается под вопросом.

В результате проведения вычислительных экспериментов выяснилось, что движение электронов пучка, первоначально равномерное и однонаправленное, после релаксации пучка становится вихревым. Это приводит к тому, что поток энергии электронов

$$q = \left| \int dxdy \frac{mv_e^2}{2} \vec{v_e} \right|$$

также приобретает сложную структуру с модуляциями по осям X и Y, и более того, возникают подобласти, где поток энергии близок к нулю (рис. 3). На рисунке 3 линия уровня с наименьшим значением (0.01) соответствует 1 % начальной величины потока энергии электронов. Из рисунка видно, что в значительной части расчетной области поток энергии меньше начального. Более подробный анализ показывает, что в отдельных частях области поток энергии электронов на несколько порядков меньше начального.

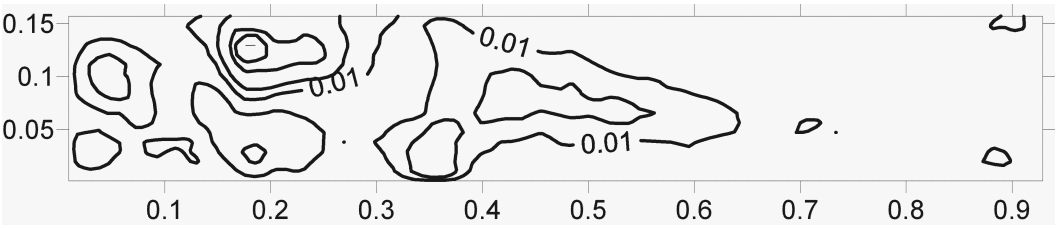


Рис. 3. Линии уровня потока энергии электронов в плоскости XY, $z = Z_M/2$, момент времени $t = 2.9$ (в единицах плазменного периода). Величина потока нормирована на его значение в момент времени $t = 0$. В расчете использована сетка $512 \times 64 \times 64$ узла, 50 модельных частиц каждого сорта (электроны пучка, электроны плазмы и ионы) в ячейке, 40 4-ядерных процессоров Xeon, кластер НКС-30Т, ИВМиМГ СО РАН, расчетная область поделена на 16 подобластей, как описано в разделе 2.3. Для расчета частиц каждой подобласти используется 10 ядер.

Это означает что удалось на качественном уровне воспроизвести эффект аномального понижения электронной теплопроводности. Однако для более точного моделирования и выяснения причин этого явления необходимо более высокое разрешения и увеличение количества модельных частиц.

3. Оценка потребностей в вычислительных ресурсах

В настоящее время проведены расчеты взаимодействия релятивистского элетронного пучка с плазмой, позволившие в квазиодномерном случае точно рассчитать инкремент двухпотоковой неустойчивости: получено $\gamma = 0.081$, точное значение $\gamma = 0.077$ [5]. Однако для

этого пришлось значительно увеличить число модельных частиц по сравнению с расчетом, представленным на рисунке 3, а именно до 1000 в одной ячейке. Число узлов сетки $120 \times 4 \times 4$, размер области 1.2 (в безразмерных единицах). При этом величина дебаевского радиуса 8.9×10^{-3} в тех же единицах. Таким образом длина области в дебаевских радиусах составляет 134.8.

Далее для оценки полного размера задачи могут быть высказаны следующие соображения

- Для адекватного моделирования плазменных неустойчивостей необходимо иметь не менее 16 узлов сетки на длине дебаевского радиуса.
- Число модельных частиц не может быть уменьшено, иначе полученные результаты будут носить лишь приближенный характер в известных физических случаях (упомянутая выше двухпотоковая неустойчивость), в новых же физических случаях вовсе ничего невозможно будет получить.
- Задача является существенно трехмерной. В силу того, что возбуждаемые пучком плазменные волны распространяются под углом к направлению движения пучка, задача не может быть сведена к двумерной или одномерной без потери существенной части физического смысла. Это означает, что число узлов сетки в направлениях, ортогональных направлению движения пучка (Y и Z), должно быть сравнимым с числом узлов по направлению X.

Таким образом, получаем следующую оценку размера сетки: $2156 \times 2156 \times 2156$ при 1000 модельных частиц каждого типа в ячейке. Это означает объем памяти 1.4 Петабайт и вычислительную нагрузку порядка 1.5 PetaFLOP (около 50 операций на каждую частицу) в течение одного временного шага.

Исходя из этого можно сказать, что для решения рассматриваемой задачи во всей полноте, а именно точного расчета спектра плазменной турбулентности, необходимо применение машин с производительностью более 1 PetaFLOPS, то есть перспективных экзафлопс–компьютеров. Может показаться, что применения современных суперЭВМ петафлопного класса будет достаточно, но во-первых, приведенные выше цифры оценочные, во-вторых, они показывают минимально необходимый объем вычислений (134 дебаевских радиуса, 16 ячеек на дебаевский радиус, 1000 частиц), в-третьих 1.5 PetaFLOP – это на один временной шаг, которых требуются десятки тысяч.

4. Перспективы достижения экзафлопс-производительности

4.1. Повышение масштабируемости: трехмерная декомпозиция

В настоящий момент соотношение времени счета и времени пересылок выглядит, как показано в таблице 1. Приведено две различных сетки: одна, на которой сейчас проводятся расчеты, и другая, намного меньше. Эта вторая сетка приведена для того, чтобы показать, что соотношение времени счета и времени пересылок остается примерно постоянным. Объем пересылаемых данных, связанных с частицами, составляет порядка 20 Кбайт для обоих сеток (пересылки выполняются между соседними подобластями), таким образом он также не растет всоедствие одинакового числа частиц в ячейке. Время пересылки измерено с помощью Intel Trace Analyzer&Collector на кластере НКС–30Т, ИВМиМГ СО РАН.

Выше уже говорилось, что используемая в настоящий момент одномерная декомпозиция области не может обеспечить достаточную масштабируемость для расчетов на десятках тысяч и более ядер. Невозможно это в силу того, что в такой декомпозиции минимальным фрагментом является слой сетки, и принадлежащие ему частицы. Это достаточно много (для приведенной выше оценки размера задачи, 2156×2156, 1000 частиц в ячейке – 660 Гб).

Таблица 1. Временные характеристики программы для моделирования взаимодействия электронного пучка с плазмой с одномерной декомпозицией расчетной области.

Размер сетки $N_X \times N_Y \times N_Z$	$512 \times 64 \times 64$	$16 \times 16 \times 16$
Число подобластей	16	4
Число процессов для расчета одной подобласти	10	1
Время счета, движения частиц, сек.	0.8	0.03
Время расчета электромагнитного поля, сек.	0.15	0.002
Время пересылки частиц Скорость пересылки, Мбайт/сек	2.8	302

Такой фрагмент очевидно не поместится в память одного процессорного элемента, а при обработке его группой процессоров будут сильно перегружены коммуникации. Поэтому, для того, чтобы иметь возможность делить расчетную область на подобласти сколь угодно малого размера, была реализована трехмерная декомпозиция. Отдельная подобласть также обрабатывается группой процессоров, так что топология имеет вид 4-мерной решетки с тремя периодическими и одним непериодическим измереним. Последнее (непериодическое) измерение соотвествует группе процессоров, назначенных для обработки подобласти. Эти процессоры выполняют между собой коллективные обмены, и только один из них поддерживает связь с другими подобластями.

Время работы программы с трехмерной декомпозицией было измерено на СКИФ-МГУ "Чебышев". При размере сетки в 200^3 узлов и увеличении числа подобластей по каждой координате от 2 до 8 соотношение времени решения уравнений электромагнитного поля и времени пересылок граничных значений находилось в интервале 7 - 10 (пересылка занимает от 10 % до 14% времени решения). В силу того, что время счета частиц значительно больше времени расчета поля, результаты измерений позволяют надеяться, что при трехмерной декомпозиции вычисления на больших сетках будут возможны и эффективны.

4.2. Сокращение времени счета: расчет на графических ускорителях

Расчет динамики частиц, выполненный на графической карте GeForce 9400, на 16 ядрах, сам по себе показал значительное повышение производительности по сравнению с расчетом на процессорах Xeon или Nehalem, таблица 2. Необходимо отметить, что время счета включает в себя также время обращения к памяти, и таким образом сильно различается на разных системах (аналогично, преимущество графических ускорителей достигается за счет быстрой локальной памяти).

Таблица 2. Сравнение времени счета движения частиц на универсальных процессорах и графическом ускорителе. Для процессора Xeon показано наименьшее время, полученное на кластере СКИФ-МГУ "Чебышев". Производительность оценивается исходя из того, что на одну частицы выполняется порядка 50 операций с плавающей точкой

Вычислитель	Время счета 1 млн. частиц, мс (на 1 поток)	Производительность, GygaFLOPS
GeForce9400	50	0.9
Tesla2050	2.3	22
Xeon	204.08	0.22
Nehalem	316.32	0.158

Таким образом, расчет с помощью одного ядра графической карты происходит на два порядка быстрее. Приведенные цифры относятся только к реализации метода частиц и не могут быть использованы для оценки и сравнения производительности указанных систем в общем случае.

Относительно перспектив реализации метода частиц на перспективных гибридных суперЭВМ необходимо сделать два замечания. Во-первых, для этого необходимо упорядочивание частиц по ячейкам, так чтобы один поток графической карты работал бы только с частицами внутри одной ячейки, или компактной группы ячеек. Во-вторых, все пересылки между подобластями должны выполняться только отдельно от вычислений: один, специально выделенный процессор взаимодействует с графическим ускорителем, другой одновременно с этим выполняет пересылки. Важно отметить, что вычислительные потоки CUDA ни в коем случае не выполняют пересылок (не возникает такой необходимости), и никак не взаимодействуют с потоками, связанными с другой подобластью.

Таким образом, если использовать графический ускоритель как замену группы процессоров (раздел 2.3), используемых для расчета движения модельных частиц в рамках одной подобласти, то можно во-первых, значительно ускорить расчет, во вторых, тем самым полностью исключаются коммуникации между потоками, связанными с одной подобластью (внутри одного графического ускорителя), что же касается пересылок между подобластями (связанными с разными графическими ускорителями), то они не могут быть более медленными, чем сейчас, когда данные пересылаются между группами процессоров, из-за меньшего количества физических связей в суперкомпьютере. Рассмотрим пересылку частиц между двумя соседними подобластями. Если сейчас с одной подобластью связано, например, 8 процессов (каждый занимает одно процессорное ядро), а пересылки частиц из одной подобласти в другую выполняются только между двумя выделенными процессами (условно «головными» процессами), то необходимо выстраивать маршрутизацию для сообщений в системе, физически состоящей из 16 ядер (или, с точки зрения MPI, из 16 потоков). В том случае, когда расчет выполняется на графическом ускорителе (в 8, 100, 200 потоков CUDA...), а пересылки будут выполнятся только между двумя MPI-потоками. Вероятно, такая пересылка займет не больше времени, чем в первом случае.

4.3. О перспективах экзафлопс–вычислений

Для рассмотрения того, как можно достичь экзафлопс–производительности в расчетах на основе мметода части–в–ячейках, возьмем за основу характеристики компьютера Tianhe-1A (№ 1 в списке Top-500 на октябрь 2010 года, [9]). Именно этот компьютер расматривается во-первых, из-за того, что, как показано выше, графические ускорители дают значительное преимущество в расчетах по методу частиц, во-вторых, в силу того, что в недавнее время компании NVIDIA и CRAY представили проекты экзафлопс-компьютеров именно на основе гибридной архитекруры, сходной с архитектурой Tianhe-1A (ссылки в докладе [8]).

Используя оценку производительности одного графического ускорителя Tesla на методе частиц–в–ячейках, полученную в предыдущем разделе (22 GygaFLOPS), оценку размера задачи (сетка $2156 \times 2156 \times 2156$ и 3×10^{13} частиц) и характеристики суперкомпьютера Tianhe-1A [9]: 7168 графических ускорителей Tesla и 14336 серверных процессоров Xeon, 32 Гб оперативной памяти в пересчете на один ускоритель, то производительность такого гибридного компьютера для метода частиц в ячейках будет 0.2 PetaFLOPS.

Если, как было описано выше,

1. **выделить для каждой подобласти один ускоритель Tesla** и один универсальный процессор

2. **считать, что необходимое количество частиц помещается в оперативную память узла** (в действительности этого нет, на каждый узел приходится 4.2 млрд. частиц или 201 Гб на узел, реально имеется 32 Гб на узел, и может быть размещено около 400 млн. частиц (эта цифра берется за основу для дальнейших рассуждений)

3. **предположить, что обмен данными между подобластями не превысит имеющийся сейчас**. При размещении на каждом узе 400 млн. частиц перелетают 5 % или 20 млн. частиц (1 Гб), что не превышает пропускную способность даже имеющихся сейчас сетей типа Infiniband.

то в таком случае масштабирование компьютера Tianhe-1A в 5 раз дало бы для метода частиц в ячейках производительность порядка 1 PetaFLOPS (эта же производительность могла бы быть достигнута при использовании порядка 700 тыс. 4-ядерных процессоров Xeon, в соотвествии с таблицей 2). Ясно, что в таком случае до производительности в 1 ExaFLOPS остается большое расстояние, тем не менее, с помощью использования графических ускорителей такая производительность достигается легче и быстрее, чем при использовании только лишь процессоров общего назначения.

Следует отметить, что безусловно, приведенные в этом разделе величины имеют характер грубой оценки. Тем не менее, можно считать, что они верны по порядку величины. Более того эти оценки основаны на свойствах метода частиц (локальность данных), а также на свойствах физической задачи (невозможность резких изменений плотности на масштабах 10-100 дебаевских радиусов), из чего следует невозможность пересылки большого количества частиц на одном шаге. Поэтому можно ожидать, что они останутся верными даже и при других характеристиках используемого оборудования.

5. Заключение

Приведены оценки необходимых вычислительных мощностей для решения задачи о взаимодействии электронного пучка с плазмой, а именно сетка $2156 \times 2156 \times 2156$ при 1000 модельных частиц каждого типа в ячейке. Исходя из свойств метода и особенностей физической задачи и предполагая возможность быстрой (по сравнению с временем счета сотен миллионов частиц на графическом ускорителе) пересылки модельных частиц между соседними подобластями, предлагается возможный путь достижения производительности

порядка 1 экзафлопс на суперкомпьютерах гибридной архитектуры в расчетах на основе метода частиц–в–ячейках.

Список литературы

1. ExaScale Software Study:Software Challenges in Extreme Scale Systems. DARPA report. September 14, 2009. http://www.er.doe.gov/ascr/Research/CS/DARPA exascale – software (2009).pdf

2. V.T.Astrelin,A.V.Burdakov, V.V.Postupaev. Generation of Ion–Acoustic Waves and Suppresion of Heat Transport during Plasma Heating by an Electron Beam. //Plasma Physics Reports. Vol. 24, No. 5, 1998, pp. 450-462.

3. В.А.Вшивков, К.В.Вшивков, Г.И.Дудникова. Алгоритмы решения задачи взаимодействия лазерного импульса с плазмой.//Вычислительные технологии, Том 6, № 2, 2001.

4. Н.Кролл, А,Трайвелпис. "Основы физики плазмы М: "Мир 1975.

5. К.В.Лотов, А.В.Терехов, И.В.Тимофеев. О насыщении двухпотоковой неустойчивости электронного пучка в плазме. //Физика плазмы, 2009, Том 35, № 6, стр. 567-574.

6. Ю.Н.Григорьев, В.А.Вшивков "Численные методы "частицы–в–ячейках". Новосибирск: "Наука 2000.

7. Ч. Бедсел, Б.Лэнгдон "Физика плазмы и математическое моделирование М:"Мир 1989.

8. А.С. Степаненко. Оценки ускорения вычислений гибридными системами. //Труды Пятой Международной Конференции «Параллельные вычисления и задачи управления». Россия, Москва. 26—28 октября 2010. Стр.29-38.

9. http://www.top500.org

Реализация проблемно-ориентированных вычислительных сервисов в среде MathCloud*

О.В. Сухорослов

Центр грид-технологий и распределенных вычислений ИСА РАН

В статье рассматриваются принципы реализации и основные компоненты сервис-ориентированной научной среды MathCloud. Целями данной среды являются предоставление унифицированного доступа к проблемно-ориентированным вычислительным сервисам и поддержка интеграции данных сервисов при решении прикладных задач. Во главу предлагаемого подхода к реализации среды MathCloud ставятся удобство разработки сервисов, простота доступа к сервисам пользователей и использование открытых технологий.

1. Введение

Современные грид-системы ориентированы на интеграцию высокопроизводительных вычислительных ресурсов для решения задач с предельно высокими требованиями к подобным ресурсам, а также задач, допускающих декомпозицию на множество небольших независимых подзадач. Представляется, что концепция грид-вычислений может быть использована для решения более широкого класса задач, отличительной особенностью которых является возможность их декомпозиции на относительно «крупные» типовые подзадачи. Данный класс фактически охватывает широкий спектр вычислительных задач математики, физики, химии, биологии и т.д. Для решения таких задач требуется набор сервисов решения типовых вычислительных математических задач, связанных друг с другом в соответствии со схемой решения исходной задачи. Появление возможности решения сложных задач путем композиции распределенных проблемно-ориентированных сервисов способно вывести распределенные вычислительные среды на качественно новый уровень.

Кроме того, все большее распространение получает модель научной кооперации и разделения труда, основанная на совместной работе территориально распределенных, часто - международных, коллективов исследователей. В подобных «распределенных» научных проектах остро стоит проблема совместного использования наработок сторон, вовлеченных в проект. Чаще всего речь идет о тех или иных вычислительных пакетах, приложениях и моделях, а также архивах и базах данных. Эта проблема также может быть решена путем преобразования данных наработок в удаленно доступные проблемно-ориентированные сервисы. Другими ситуациями, в которых может возникать потребность в создании подобных сервисов, являются публикация и обмен результатами научных исследований, внедрение и коммерциализация полученных результатов и создание образовательных ресурсов.

Предлагаемый подход, охватывающий фактически все этапы научных исследований, состоит в построении распределенных вычислительных сред нового поколения, предоставляющих доступ к проблемно-ориентированным сервисам и образующих универсальную инфраструктуру для научной кооперации. Данная инфраструктура базируется на сервис-ориентированном подходе: пользователи преобразуют свои приложения в удаленно доступные сервисы, которые могут быть обнаружены и использованы другими пользователями для решения интересующих их задач.

Актуальность данного направления исследований подтверждается сформулированной в 2005 году концепцией «сервис-ориентированной науки» (Service-Oriented Science) [1], автором которой выступил Иэн Фостер, один из основоположников грид-вычислений. В соответствии с данной концепцией, сервис-ориентированный подход позволяет организовать повсеместный доступ к разнородным научным ресурсам и автоматизировать процесс научных исследований,

* Работа выполнена при поддержке фонда РФФИ (грант № 10-07-00176-а) и Президиума РАН (программа П1).

тем самым, повышая производительность исследований и открывая новые возможности для науки в целом.

В настоящее время отсутствуют проработанные подходы к реализации сервис-ориентированных научных сред. Обусловлено это как относительной новизной данного направления, так и рядом технологических проблем, стоящих на пути реализации подобных сред, таких как:

- унификация и обеспечение интероперабельности сервисов на уровне протоколов, интерфейсов, форматов и семантики данных;
- организация безопасного доступа к сервисам, включая защиту передаваемых по сети данных, аутентификацию и авторизацию пользователей, учет использования ресурсов и т.д.;
- снижение технологического барьера для потенциальных разработчиков сервисов за счет упрощения процедур создания и размещения сервисов в сети;
- обеспечение масштабируемости сервисов, в том числе за счет динамического подключения внешних вычислительных ресурсов;
- снижение технологического барьера для потенциальных пользователей сервисов путем реализации проблемно-ориентированных интерфейсов («рабочих пространств»), скрывающих техническую сторону функционирования сервисов и сложность низлежащей вычислительной инфраструктуры;
- создание механизмов публикации, аннотации и оценки качества сервисов, позволяющих пользователю быстро найти интересующий его сервис;
- создание механизмов формирования сообществ пользователей (виртуальных организаций), образующих контексты для разделения сервисов.

Несмотря на имеющийся опыт решения подобных проблем при создании грид-систем, существующие технологии зачастую сложны в использовании и требуют своего пересмотра или поиска новых решений. Сервис-ориентированные научные среды, за счет расширения круга ресурсов и приложений, должны обеспечивать функционирование множества виртуальных сообществ среднего и малого размера. При этом требуется радикально упростить процедуры развертывания соответствующего ПО, формирования сообществ пользователей, разработки сервисов и их размещения в среде. Существующие средства разработки грид-сервисов, такие как Globus Toolkit, изначально ориентированы на разработчиков базовых сервисов грид, сложны в использовании и основаны на обладающих рядом недостатков спецификациях веб-сервисов.

В статье рассматривается распределенная среда MathCloud [2], реализующая концепцию сервис-ориентированных научных сред. Целями среды являются предоставление унифицированного доступа к проблемно-ориентированным вычислительным сервисам и поддержка интеграции данных сервисов при решении прикладных задач. Во главу предлагаемого подхода к реализации среды MathCloud ставятся удобство разработки сервисов, простота доступа к сервисам пользователей и использование открытых технологий.

2. Архитектура среды MathCloud

Архитектура среды MathCloud состоит из нескольких уровней, изображенных на **Рис. 1**.

2.1 Вычислительные ресурсы

Нижний уровень архитектуры MathCloud содержит вычислительные ресурсы, используемые для функционирования сервисов среды. Формально данный уровень не относится к самой среде MathCloud, а формируется из доступных разработчикам и пользователям ресурсов. Унифицированные механизмы интеграции и доступа к вычислительным ресурсам уже сформированы в рамках грид-инфраструктур. Однако стоит отметить, что данные инфраструктуры ориентированы на запуск ограниченных по времени вычислительных заданий и не поддерживают размещение постоянно функционирующих сервисов. В то же время, в рамках коммерческих систем, таких как Amazon EC2, активно развивается модель аренды виртуализованных аппаратных ресурсов (cloud computing), позволяющая размещать на этих ресурсах и сами сервисы.

В рамках среды MathCloud предполагается использование обоих видов вычислительных инфраструктур.

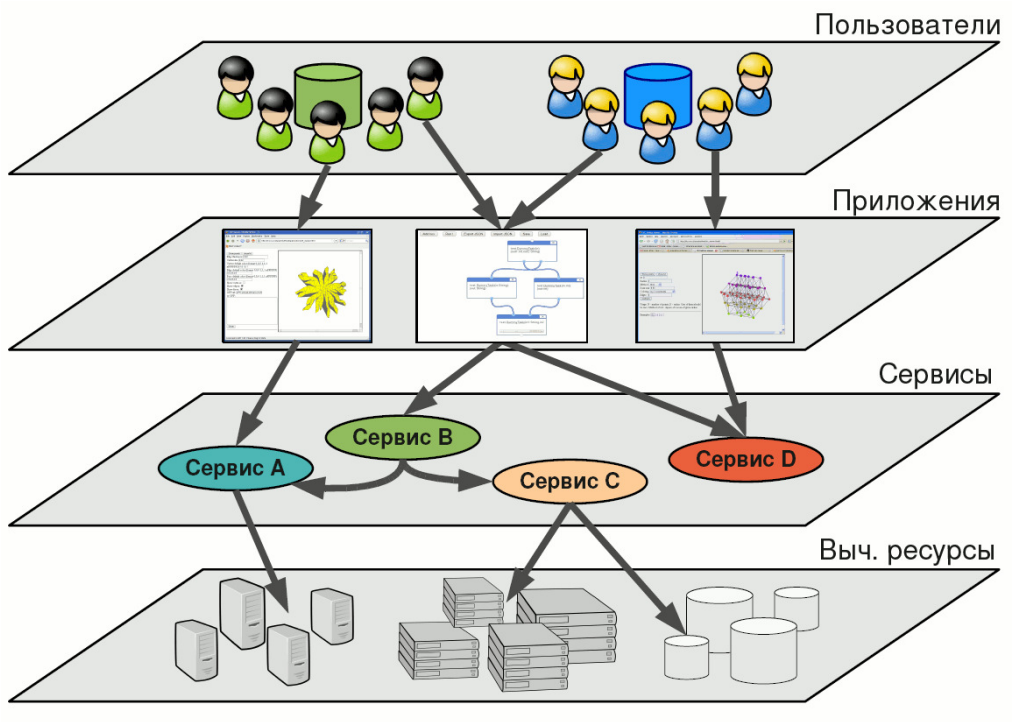


Рис. 1. Уровни архитектуры MathCloud.

Доступ к вычислительным ресурсам может осуществляться с помощью соответствующих интерфейсов менеджеров ресурсов и грид-инфраструктур. Однако, учитывая сложность освоения и использования промежуточного ПО грид-систем, в рамках MathCloud предполагается создание адаптеров, упрощающих взаимодействие сервисов с грид. Данные адаптеры позволят прозрачным для разработчика сервиса способом транслировать запросы к сервису в вычислительные задания, запускаемые в грид. Это также позволяет значительно упростить процесс преобразования в сервисы существующих грид-приложений.

2.2 Сервисы

На уровне сервисов реализуется удаленный программный доступ к некоторой востребованной пользователями среды функциональности. Проблемно-ориентированные сервисы, являющиеся главными компонентами среды, нацелены на решение определенного класса прикладных задач и служат основой для создания приложений. Помимо проблемно-ориентированных сервисов, в среде могут присутствовать системные сервисы, реализующие универсальную функциональность, такую как публикация и поиск сервисов. В дальнейшем под сервисами MathCloud будут подразумеваться проблемно-ориентированные сервисы. Уровень сервисов, как уже было отмечено, опирается на низлежащую вычислительную инфраструктуру, используемую для проведения сложных вычислений или хранения данных.

Остановимся подробнее на модели проблемно-ориентированного сервиса.

Сервис MathCloud представляет собой доступный по сети программный компонент, поддерживающий решение определенного класса задач с помощью соответствующих вычислительных алгоритмов. В соответствии с моделью клиент-сервер, сервис обслуживает приходящие к нему запросы клиентов на решение конкретных задач. Запрос клиента содержит параметризованное описание задачи, формулируемое в виде конечного набора входных параметров. После успешной обработки запроса сервис возвращает клиенту результат, оформленный в виде конечного набора выходных параметров.

Для взаимодействия с сервисом клиенту необходимо знать список входных и выходных параметров сервиса. Данная информация является частью описания сервиса, публикуемого в

общедоступном месте или предоставляемого сервисом по запросу клиента. Описание параметров сервиса определяет форматы сообщений (контракт), которым должны следовать клиент и сервис. Данное описание должно поддерживать машинную интерпретацию, например для валидации сообщений, генерации клиентского кода или реализации динамических вызовов.

Отметим, что описанная модель является довольно общей и может быть, вообще говоря, применена к большому классу сервисов, не обязательно предоставляющих доступ к вычислительным алгоритмам. В частности, это могут быть сервисы доступа к базам данных или сервисы, осуществляющие обработку и преобразование данных. В общем случае, речь может идти о произвольном программном компоненте, оформленном в виде сервиса. Тем не менее, в рамках среды MathCloud предполагается рассмотрение в первую очередь сервисов, нацеленных на решение научных и прикладных вычислительных задач. Рассмотрим характерные особенности вычислительных сервисов.

Во-первых, каждый запрос обрабатывается сервисом независимо от других запросов. Иными словами, результат запроса определяется исключительно значениями передаваемых клиентом входных параметров и не зависит от результатов других запросов. Фактически данная особенность играет роль ограничения, накладываемого на рассматриваемые сервисы. А именно, за рамками описываемой модели сервиса остаются клиент-серверные приложения, в которых требуется поддерживать состояние (сессию) между последовательными запросами клиента к серверу. Исключение из рассмотрения подобных случаев позволяет существенно упростить интерфейс и реализацию сервисов, а также, что самое главное, повысить масштабируемость и отказоустойчивость среды.

Вторая важная особенность вычислительных сервисов заключается в том, что обработка запроса клиента (решение задачи) может потребовать длительных вычислений, в том числе на многопроцессорных вычислительных комплексах или в грид. В этом случае сервис не может вернуть результат сразу же после получения запроса. Обработка подобных запросов должна вестись в асинхронном режиме, путем преобразования поступающих запросов в задания. В ответ на вызов клиенту передается идентификатор соответствующего задания, используя который клиент может опрашивать статус задания и получить результат. Клиент также может получать от сервиса уведомления об изменении статуса задания.

Кроме того, растет количество вычислительных приложений, принимающих на вход или генерирующих большие объемы данных. Для подобных сервисов необходимо организовать эффективную передачу параметров большого размера в виде файлов с использованием соответствующих механизмов передачи данных по сети. В этом случае запрос к сервису или возвращаемый сервисом результат могут содержать не сами значения параметров, а ссылки на соответствующие файлы.

Для упрощения повторного использования и композиции сервисов в рамках различных приложений требуется унификация механизма удаленного доступа к сервисам на уровне протоколов и форматов данных. Для этих целей предлагается использовать архитектурный стиль REST (Representational State Transfer), хорошо зарекомендовавший себя в рамках Web. Для среды MathCloud разработан и подробно описан унифицированный REST-интерфейс (см. раздел 3), который должны реализовывать сервисы среды.

Среда MathCloud является открытой распределенной системой, для участия в которой разработчику сервиса необходимо и достаточно реализовать унифицированный REST-интерфейс. При этом детали реализации сервиса остаются полностью на усмотрение его разработчика. Тем не менее, необходимо наличие готовых средств, упрощающих разработку сервисов, в первую очередь для разработчиков с минимальной квалификацией. Подобные средства особенно актуальны для быстрого преобразования в сервисы существующих приложений. Исходя из этих соображений, была реализована универсальная среда выполнения сервисов (см. раздел 4).

2.3 Приложения

На уровне приложений реализуется доступ пользователей к сервисам среды через проблемно-ориентированные интерфейсы. В качестве стандартной среды выполнения приложений предлагается использовать веб-браузер, что обладает рядом преимуществ: приложение сразу готово к работе без установки на компьютер пользователя, разработчик приложения сохраняет

полный контроль над ним, включая возможность обновлений и учета использования. Так же как в случае сервисов, архитектура не накладывает существенные ограничения на используемые для разработки приложения средства программирования.

Отметим, что реализация некоторого сервиса может быть уже снабжена интерфейсом для работы с сервисом пользователей. Однако в общем случае понятия сервиса и приложения следует различать. Если сервис предоставляет программный интерфейс для приложений, то приложение предоставляет интерфейс конечному пользователю. Таким образом, один и тот же сервис может использоваться в рамках нескольких приложений. И наоборот - в рамках одного приложения могут быть задействованы сразу несколько сервисов. Остановимся на последнем моменте подробнее.

Ключевой функцией предлагаемой сервис-ориентированной среды является поддержка объединения или композиции сервисов, позволяющая пользователям среды «собирать» из существующих сервисов новые приложения и, что особенно важно, новые сервисы, развивая, тем самым, среду. В общем случае, композиция сервисов может осуществляться с применением произвольных средств программирования, например скриптовых языков. Однако, также как и в случае с разработкой сервисов, в рамках MathCloud предусмотрены готовые средства, упрощающие композицию сервисов и доступные пользователям с минимальной квалификацией. Для поддержки интеграции сервисов среды при решении прикладных задач реализованы редактор композитных приложений и система управления сценариями на основе workflow-подхода (см. раздел 5).

2.4 Пользователи

На уровне пользователей сконцентрирован социальный аспект среды. Здесь решаются задачи, связанные с формированием и обеспечением функционирования сообществ пользователей. Предполагается, что данные сообщества, как правило, будут формироваться вокруг той или иной области исследований, аналогично виртуальным организациям в грид. Идеальной средой для формирования таких сообществ представляются Web-порталы с функциональностью социальных сетей, учитывающие научную специфику проекта. Будучи созданными, подобные сообщества могут образовывать контексты для разделения ресурсов и сервисов. Во-первых, в рамках сообщества могут коллективно вестись публикация, каталогизация и учет сервисов, относящихся к тематике данного сообщества. Во-вторых, аналогично грид, доступ к сервисам может регламентироваться на основе принадлежности пользователя к определенному сообществу. В-третьих, интерфейсы приложений могут быть интегрированы в портал сообщества, предоставляя удобный доступ к сервисам среды. Данный уровень архитектуры MathCloud является в данный момент наименее проработанным, поскольку текущие усилия сосредоточены на нижних, технологических уровнях среды.

3. Интерфейс сервиса MathCloud

Описанная архитектура носит общий характер и может быть реализована с помощью различных технологий построения распределенных систем. Выбор той или иной технологии зависит от ряда факторов, таких как распространенность, удобство разработки и открытость исходного кода. В рамках научной сервис-ориентированной среды, допускающей участие различных лиц и организаций, очень важно использование открытых стандартов и наличие нескольких независимых реализаций базовой технологии. Важно также, чтобы используемая технология как можно ближе соответствовала описанной выше модели проблемно-ориентированного вычислительного сервиса.

В настоящее время доминирующей технологией для построения сервис-ориентированных систем являются Web-сервисы на основе протокола SOAP и спецификаций WS-*, которые будем далее называть WS-сервисами. Спецификации WS-сервисов являются открытыми стандартами, изначально ориентированными на реализацию сервис-ориентированной архитектуры. Наконец, что немаловажно, данные технологии активно поддерживаются крупными компаниями-разработчиками, что привело к повсеместному их распространению и появлению множества реализаций, в том числе и с открытым кодом.

Распространенной критикой WS-сервисов является их чрезмерная сложность и некорректное использование протокола HTTP. При этом преимущества от использования WS-сервисов заметны только в определенном классе приложений, ориентированных на поддержку сложных бизнес-процессов, встречающихся в корпоративных и государственных системах и слабо распространенных в технических и научных проектах. Это привело к появлению альтернативных, более простых в использовании подходов к реализации Web-сервисов, основанных на прямом использовании протокола HTTP.

В 2000 году Рой Филдинг, один из главных разработчиков протокола HTTP и других спецификаций Web, опубликовал диссертацию [3] с описанием эталонной модели архитектуры Web. Данная модель или архитектурный стиль, получивший название Representational State Transfer (REST), содержит ряд ключевых принципов, определенных в виде накладываемых на архитектуру ограничений. Центральными понятиями REST являются понятия ресурса, идентификатора и представления ресурса. В рамках REST вводятся следующие ограничения на архитектуру системы: клиент-серверная архитектура, хранение состояния приложения на стороне клиента, кэширование ответов на запросы, унифицированный интерфейс доступа к ресурсам, многоуровневая архитектура. Данные ограничения позволяют обеспечить такие свойства архитектуры Web, как масштабируемость, расширяемость и открытость.

Благодаря унифицированному интерфейсу доступа к ресурсам, использованию открытых стандартов (HTTP, URI) и наличию множества проверенных временем реализаций (Web-серверы, библиотеки для работы с HTTP для всех современных языков программирования), REST обеспечивает максимальную свободу для независимой разработки Web-сервисов и соответствующих клиентских приложений. Немаловажным обстоятельством является простота разработки REST-сервисов в сравнении с другими рассмотренными технологиями. Все это позволяет максимально расширить как круг потенциальных разработчиков, так и пользователей сервисов, обеспечив при этом совместимость различных реализаций и широкое повторное использование сервисов. Это привело к широкому распространению Web-сервисов на основе стиля REST (т.н. RESTful Web-сервисов), особенно в рамках Web 2.0 приложений.

Исходя из вышеописанного, в качестве базовой платформы для организации удаленного доступа к сервисам MathCloud предлагается использовать протоколы и технологии Web, основанные на архитектурном стиле REST. Разработан и подробно описан REST-интерфейс сервиса MathCloud [4], использующий в качестве протокола доступа к сервисам протокол HTTP. Разработанный интерфейс учитывает характерные особенности вычислительных сервисов, такие как длительная обработка запросов и передача больших объемов данных в виде файлов. Также в соответствии с сервис-ориентированным подходом разработанный интерфейс поддерживает интроспекцию, т.е. получение информации о сервисе. Выбранные базовые технологии допускают независимые реализации описанного интерфейса на различных языках программирования. При этом детали реализации сервиса остаются полностью на усмотрение его разработчика.

В соответствии с принципами REST, интерфейс сервиса MathCloud образован совокупностью идентифицируемых при помощи URI ресурсов, поддерживающих стандартные методы протокола HTTP (Табл. 1).

Таблица 1. Ресурсы и методы интерфейса сервиса MathCloud

Ресурс	GET	POST	DELETE
Сервис SERVICE_URI	Получение описания сервиса	Запрос к сервису	—
Задание JOB_URI	Получение статуса и результатов задания	—	Отмена задания, удаление результатов задания
Файл результата задания FILE_URI	Загрузка файла	—	—
Сервер SERVER_URI	Получение списка сервисов, размещенных на сервере	—	—

Ресурс-сервис поддерживает два метода. Метод GET возвращает клиенту представление данного ресурса, содержащее описание сервиса. Метод POST служит для отправки сервису нового запроса. В теле запроса клиент передает набор значений входных параметров задачи. В ответ сервис создает новый ресурс-задание, являющийся подчиненным по отношению к ресурсу-сервису, и возвращает идентификатор ресурса-задания и его текущее представление клиенту.

Ресурс-задание поддерживает методы GET и DELETE. Метод GET возвращает представление данного ресурса, содержащее информацию о текущем статусе задания. Данная информация должна обязательно включать состояние задания, которое может принимать следующие значения:

- WAITING - выполнение задания еще не началось;
- RUNNING - задание выполняется;
- DONE - задание выполнено успешно;
- FAILED - выполнение задания завершилось ошибкой.

Если выполнение задания завершено успешно (состояние DONE), то в представление ресурса-задания также включается результат задания в виде набора значений выходных параметров. Часть значений выходных параметров может содержать идентификаторы ресурсов-файлов.

В случае если в процессе выполнения задания стали известны значения части выходных параметров, данные значения могут включаться в представление ресурса-задания еще до окончания выполнения задания.

Метод DELETE ресурса-задания позволяет клиенту отменить выполнение запроса или, если выполнение уже завершено, удалить результаты задания. После вызова DELETE данный ресурс-задание, а также подчиненные ему ресурсы-файлы, перестают существовать.

Ресурс-файл представляет собой часть результата задания и поддерживает метод GET для получения содержимого файла клиентом.

Дополнительный ресурс-сервер предназначен для случаев, когда в рамках одного HTTP-сервера размещено несколько алгоритмических сервисов. В этих случаях может быть полезным получение списка сервисов, размещенных на сервере. Для этих целей зарезервирован метод GET. Ресурс-сервер в данном случае является родительским по отношению к ресурсам-сервисам.

Описанный интерфейс поддерживает обработку запросов как в синхронном, так и асинхронном режиме. Действительно, если результат запроса может быть сразу возвращен клиенту, то он передается внутри возвращаемого в ответ клиенту представления ресурса-задания с указанием состояния DONE. Если же для обработки запроса требуется время, то это указывается внутри возвращаемого клиенту представления ресурса-задания с помощью указания соответствующего состояния задания (WAITING или RUNNING). В этом случае клиент использует переданный URI ресурса-задания для дальнейшего опроса состояния задания и получения результата.

4. Среда выполнения сервисов

Наличие готового и простого в использования инструмента для создания сервисов очень важно с точки зрения расширения круга потенциальных разработчиков сервисов. Для упрощения процесса разработки сервисов MathCloud была реализован контейнер сервисов Everest. Данный контейнер представляет собой универсальную среду выполнения сервисов, реализующую описанный выше интерфейс сервиса.

Универсальность контейнера заключается в том, что он позволяет легко, во многих случаях без написания программного кода, преобразовать в сервис MathCloud широкий спектр существующих приложений. Кроме того, реализованный в контейнере механизм адаптеров позволяет подключать произвольные реализации обработчиков запросов. В частности, путем преобразования запроса к сервису в вычислительное задание, на уровне адаптера может быть реализован доступ к вычислительной инфраструктуре.

Everest реализован на базе библиотеки Jersey, являющейся в свою очередь открытой эталонной реализацией спецификации JAX-RS (Java API for RESTful Web Services). На **Рис. 2** изо-

бражена архитектура Everest. Взаимодействие с клиентами осуществляется с помощью встроенного в контейнер Web-сервера Jetty. Входящие HTTP-запросы передаются библиотеке Jersey и, затем, реализации Everest. Сопряжение между Jersey и Everest осуществляется с помощью четырех Java-классов (ServerResource, ServiceResource, JobResource и FileResource), которые являются реализациями соответствующих ресурсов из REST-интерфейса сервиса MathCloud.

Контейнер осуществляет обработку запросов клиентов в соответствии с конфигурационной информацией. Компонент ServiceManager хранит список сервисов, размещенных на сервере, и их конфигурацию. Данная информация считывается при запуске сервера из конфигурационных файлов. Конфигурация каждого сервиса состоит из двух частей:

- внешнее описание сервиса, передаваемое клиентам (текстовая аннотация сервиса, описания входных и выходных параметров);
- внутренняя конфигурация сервиса, содержащая информацию о реализации сервиса (адаптер и его конфигурационные параметры).

Компонент JobManager осуществляет управление обработкой запросов к сервисам. Запросы преобразуются в задания и размещаются в очереди, откуда затем извлекаются конфигурируемым пулом потоков-обработчиков. При выполнении задания поток-обработчик вызывает тот или иной адаптер, в соответствии с внутренней конфигурацией сервиса.

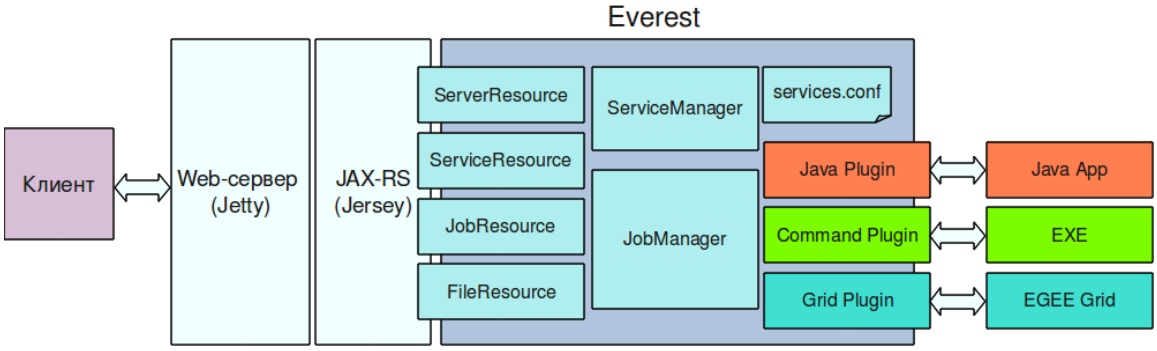


Рис. 2. Архитектура Everest.

Компоненты, реализующие обработку запросов к сервисам (заданий), оформляются в виде подключаемых адаптеров. Каждый адаптер реализует стандартный программный интерфейс, с помощью которого контейнер передает адаптеру параметры запроса, контролирует состояние выполнения задания и получает результаты. Реализация адаптера, как правило, преобразует полученный запрос в вызов внешнего приложения. После успешного завершения приложения адаптер сохраняет полученные результаты и изменяет состояние задания на DONE.

В настоящее время реализовано три универсальных адаптера, поддерживающих интеграцию следующих типов приложений:

- приложение с интерфейсом командной строки (адаптер Command);
- приложение на языке Java (адаптер Java);
- грид-задание, запускаемое в инфраструктуре EGEE (адаптер Grid).

Адаптер Command осуществляет преобразование запроса к сервису в запуск команды в отдельном процессе операционной системы. Во внутренней конфигурации сервиса необходимо указать запускаемую команду в специальном формате, содержащем информацию о том, каким образом параметры сервиса отображаются в аргументы команды и внешние файлы.

Адаптер Java осуществляет вызов заданного Java-класса в рамках текущей виртуальной машины, передавая в вызове параметры запроса к сервису. Используемый Java-класс должен реализовывать стандартный программный интерфейс. Во внутренней конфигурации сервиса требуется указать имя соответствующего Java-класса.

Адаптер Grid осуществляет преобразование запроса к сервису в запуск вычислительного задания в грид-инфраструктуре EGEE, основанной на ПО gLite. Данный адаптер может использоваться как для преобразования в сервис существующего грид-приложения, так и для переноса существующей реализации сервиса в грид, например, из соображений производительности и масштабируемости. Во внутренней конфигурации сервиса требуется указать виртуальную ор-

ганизацию грид, путь к файлу с описанием грид-задания в формате JD, а также информацию об отображении параметров сервиса в аргументы и файлы грид-задания.

Отметим, что адаптеры Command и Grid позволяют преобразовывать существующие приложения в сервисы без разработки дополнительных программных компонентов. Данная возможность позволяет быстро преобразовать в сервисы широкий спектр существующих приложений, не обладая при этом навыками программирования.

Каждый размещенный в контейнере сервис становится доступным через стандартный интерфейс сервиса MathCloud. Кроме того, Everest поддерживает дополнительную HTML-версию данного интерфейса, позволяющую пользователям непосредственно обращаться к сервисам через веб-браузер.

5. Композиция сервисов и система управления сценариями

Для поддержки интеграции сервисов среды MathCloud при решении прикладных задач реализована система управления сценариями на основе workflow-подхода [5]. Данная система реализует описание, хранение, выполнение и публикацию сценариев совместного использования сервисов среды. Сценарии описываются в виде ориентированных ациклических графов при помощи визуального редактора. Описанный сценарий может быть затем преобразован в новый сервис среды и запущен на выполнение путем вызова данного сервиса. Предлагаемый подход позволяет скрыть от пользователя детали реализации вызовов сервисов и передачи данных между ними, оставив только необходимость правильного соединения сервисов друг с другом. Таким образом, многие задачи, решение которых сводится к компоновке типовых сервисов, становятся доступными для пользователей, не обладающих навыками распределенного программирования. Графическое представление сценария позволяет сделать наглядными связи между сервисами. Кроме того, такое представление позволяет быстро вносить изменения в уже работающие сценарии - такие, как добавление новых блоков сценария или замена отдельных блоков другими, получая на выходе новые сервисы с новыми параметрами.

Разработанная система имеет распределенную клиент-серверную архитектуру. Клиентская часть системы включает редактор сценариев, а серверная - сервис управления сценариями и среду выполнения сценариев.

5.1 Редактор сценариев

Редактор сценариев реализует графический интерфейс, предназначенный для взаимодействия с системой пользователей. Редактор является клиентом сервиса управления сценариями и может функционировать на любой машине в сети. Основными функциями редактора являются просмотр и редактирование сценариев. Созданный с помощью редактора сценарий может быть сохранен и запущен на выполнение на серверной стороне. При этом в редакторе отображается состояние выполнения сценария.

Редактор сценариев реализован в виде веб-приложения на языке JavaScript. Благодаря этому редактор может использоваться без предварительной установки на любом компьютере, где установлен современный веб-браузер. Редактор создан с широким использованием технологий AJAX. В его основе лежат такие JavaScript-библиотеки, как WireIt, YUI и InputEx. Это позволило сделать интерфейс редактора легковесным, интуитивным и удобным для пользователя.

На **Рис. 3** изображен общий вид редактора сценариев. Правая часть редактора содержит список доступных сервисов и прочих элементарных блоков, из которых пользователь может компоновать сценарий. Верхняя часть представляет собой главное меню, предоставляющее доступ к основным операциям со сценариями - открытие, сохранение, запуск и т.д. В центральной части редактора размещается графическое представление сценария.

Сценарий представлен в виде ориентированного ациклического графа, вершинами которого являются элементарные блоки сценария, а ребрами - соединения, задающие потоки данных между блоками. Каждый блок имеет набор входов и выходов, отображаемых в виде круглых портов соответственно сверху и снизу блока. Блок реализует некоторую логику обработки поступающих на вход данных. Передача данных между блоками реализуется путем соединения

выхода одного блока с входом другого. С каждым выходом и выходом связан определенный тип данных. При соединении блоков проверяется совместимость типов входа и выхода.

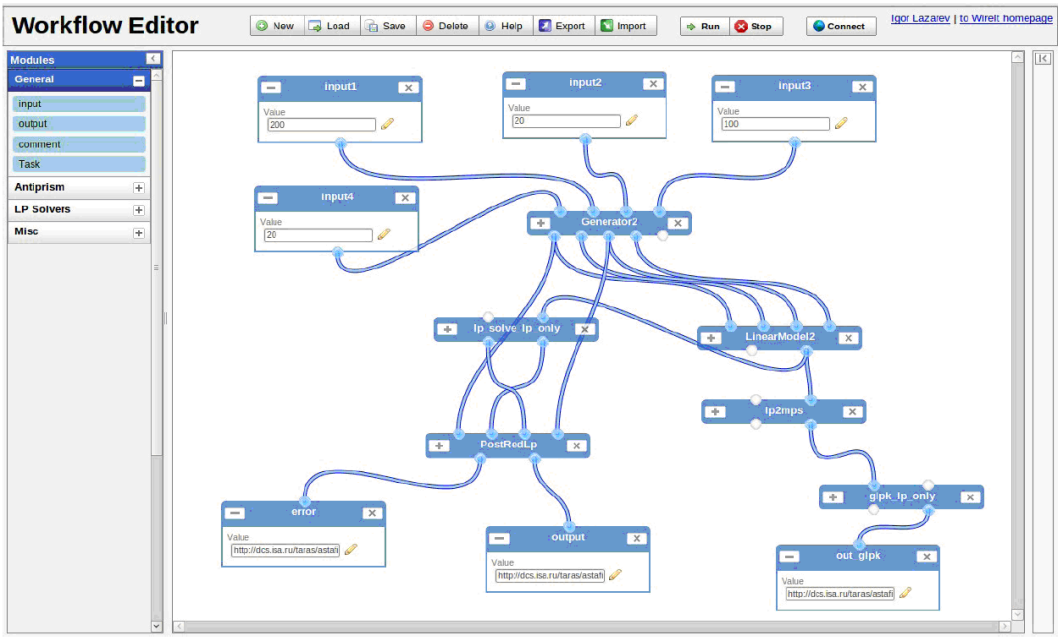


Рис. 3. Общий вид редактора сценариев.

В настоящее время реализованы следующие типы блоков сценария:

- Input - служит для задания значения входного параметра сценария;
- Output - служит для отображения значения выходного параметра сценария;
- Service - осуществляет вызов заданного сервиса MathCloud;
- JavaScript - осуществляет выполнение заданного кода на языка JavaScript;
- Python - осуществляет выполнение заданного кода на языке Python;
- Comment - служит для размещения комментариев к сценарию.

Добавление в сценарий сервиса MathCloud осуществляется путем создания нового блока Service и ввода пользователем URI требуемого сервиса. Редактор запрашивает описание данного сервиса, откуда извлекает информацию о количестве, типах и именах входных и выходных параметров сервиса. После чего динамически формируются соответствующие входы и выходы блока, а в заголовке блока отображается имя сервиса. В случае неудачи при соединении с сервисом блок окрашивается в красный цвет.

Для упрощения создания блоков Input и Output, редактор позволяет автоматически создавать данные блоки путем указания пользователем требуемого входа или выхода блока.

Блоки JavaScript и Python позволяют увеличить гибкость при описании сценария путем внедрения в него фрагментов кода (скриптов). Скрипт оформляется в виде функции, сигнатура которой используется для динамической генерации входов и заголовка блока. Выход блока соответствует возвращаемому значению функции. Выполнение скрипта осуществляется на серверной стороне средой выполнения сценариев.

Редактор поддерживает передачу и сохранение созданного сценария на стороне сервиса управления сценариями. Также с помощью редактора можно просматривать список сохраненных сценариев, открывать сохраненный сценарий, удалять сценарий, осуществлять импорт и экспорт описания сценария в формате JSON.

Важной функцией редактора является запуск сценария на выполнение. Для этого необходимо задать значения всех входных параметров сценария с помощью соответствующих блоков Input. После нажатия пользователем кнопки Run, редактор производит вызов сервиса-сценария с заданными входными параметрами. Далее редактор осуществляет периодический опрос состояния запущенного задания, содержащего информацию о состоянии отдельных блоков сценария. Данная информация отображается пользователю путем окрашивания блоков сценария в цвет, соответствующий текущему состоянию. После успешного завершения сценария, значения

выходных параметров сценария отображаются в соответствующих блоках Output. Каждый запущенный сценарий имеет уникальный URI, используя который можно в любой момент открыть в редакторе выполняющийся сценарий. Данная возможность особенно важна в тех случаях, когда сценарий выполняется в течение длительного времени.

5.2 Сервис управления сценариями

Сервис управления сценариями реализует удаленный программный интерфейс для хранения сценариев, созданных с помощью редактора. В соответствии с сервис-ориентированным подходом, сервис управления сценариями также обеспечивает развертывание каждого сохраненного пользователем сценария в виде нового сервиса среды MathCloud.

Удаленный интерфейс сервиса управления сценариями реализован на основе протокола HTTP и архитектурного стиля REST, что позволяет удобным образом взаимодействовать с сервисом из редактора сценария. В соответствии с принципами REST интерфейс сервиса образован совокупностью идентифицируемых при помощи URI ресурсов, поддерживающих стандартные методы протокола HTTP (Табл. 2).

Таблица 2. Ресурсы и методы интерфейса сервиса управления сценариями.

Ресурс	GET	POST	PUT	DELETE
Сервис управления сценариями WFMS_SERVICE_URI	Получение списка хранимых сценариев	Создание нового сценария	—	—
Сценарий WF_URI	Получение описания сценария	—	Обновление сценария	Удаление сценария
Сервис сценария WF_SERVICE_URI	Получение описания сервиса	Запуск сценария	—	—
Прокси WFMS_PROXY_URI	Получение описания сервиса с указанным URI	—	—	—

Сервис сценария реализует унифицированный REST-интерфейс сервиса MathCloud (см. раздел 3). Таким образом, с точки зрения клиентов сервис сценария ничем не отличается от обычных сервисов и запуск сценария на выполнение осуществляется путем отправки запроса к его сервису. Это позволяет, например, легко использовать существующий сценарий в качестве одного из элементов при создании нового сценария.

5.3 Среда выполнения сценариев

Среда выполнения сценариев осуществляет обработку запросов к сервисам-сценариям. Каждый подобный запрос приводит к запуску нового экземпляра соответствующего сценария. Среде выполнения сценариев передается описание сценария и значения его входных параметров. Среда осуществляет интерпретацию описания сценария, производит указанные в сценарии действия (в том числе вызовы внешних сервисов), отслеживает состояние выполнения сценария и возвращает результаты его выполнения. Основой среды выполнения является разработанная библиотека WorkflowRuntime на языке Java, поддерживающая выполнение произвольных workflow-сценариев, заданных в виде ориентированных ациклических графов.

6. Заключение

В статье рассмотрены принципы реализации и основные компоненты сервис-ориентированной научной среды MathCloud. Архитектура среды основана на представлении сервиса в виде функции с заданным набором входных и выходных параметров и применении стиля REST для описания унифицированного интерфейса сервиса. Разработанный интерфейс сервиса MathCloud учитывает характерные особенности вычислительных сервисов, такие как

длительная обработка запросов и передача больших объемов данных в виде файлов. Для упрощения процесса разработки сервисов MathCloud реализован универсальный контейнер сервисов, позволяющий быстро, без написания дополнительного программного кода, преобразовать в сервисы широкий спектр существующих приложений. Реализованный в контейнере механизм адаптеров позволяет подключать произвольные реализации обработчиков запросов к сервису. В частности, путем трансляции запроса в вычислительное задание, на уровне адаптера может быть реализован доступ к различным видам вычислительных ресурсов. Данный механизм позволяет упростить процесс преобразования в сервисы существующих параллельных и грид-приложений, а также перенос реализации сервиса на новую вычислительную инфраструктуру. Для поддержки интеграции сервисов среды при решении прикладных задач реализованы графический редактор и система управления сценариями на основе workflow-подхода.

Одним из важных аспектов архитектуры среды MathCloud, оставшимся за рамками данной статьи, является вопрос обеспечения безопасности. Предполагается, что многие из сервисов MathCloud будут открыты для публичного доступа. Тем не менее, очевидно, что в ряде случаев будут необходимы ограничения круга пользователей сервиса, и что жизнеспособность предлагаемой среды будет зависеть от наличия полноценного механизма безопасности. Сюда входят аутентификация пользователей и сервисов, авторизация пользователей и контроль доступа к сервисам, защита передаваемых по сети данных и делегация прав доступа. В настоящее время ведется разработка механизма безопасности среды MathCloud, основанного на применении протокола HTTPS и цифровых сертификатов. Также предусмотрен упрощенный способ аутентификации пользователей при помощи публичных провайдеров учетных записей и стандарта OpenID.

Другим важным компонентом среды MathCloud, работа над которым еще не завершена, является каталог сервисов, поддерживающий публикацию, поиск и мониторинг сервисов среды.

Данные вопросы будут подробно рассмотрены в рамках дальнейших работ.

Литература

1. Foster I. Service-Oriented Science // Science. 2005. Vol. 308, No. 5723. P. 814-817.
2. Астафьев А.С., Афанасьев А.П., Лазарев И.В., Сухорослов О.В., Тарасов А.С. Научная сервис-ориентированная среда на основе технологий Web и распределенных вычислений. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). Москва: Изд-во МГУ, 2009. С. 463-467.
3. Fielding, R.T. Architectural styles and the design of network-based software architectures. PhD Dissertation. Dept. of Information and Computer Science, University of California, Irvine, 2000.
4. Сухорослов О.В. Унифицированный интерфейс доступа к алгоритмическим сервисам в Web. // Проблемы вычислений в распределенной среде / Под ред. С.В. Емельянова, А.П. Афанасьева. Труды ИСА РАН, Т. 46. Москва: КРАСАНД, 2009. С. 60-82.
5. Лазарев И.В., Сухорослов О.В. Реализация распределенных вычислительных сценариев в среде MathCloud. // Проблемы вычислений в распределенной среде / Под ред. С.В. Емельянова, А.П. Афанасьева. Труды ИСА РАН, Т. 46. Москва: КРАСАНД, 2009. С. 6-23.

Перенос излучения, радиационное поле Земли и космические проекты: информационно-математический аспект и супервычисления (история и перспективы) *

Т.А. Сушкевич

Учреждение Российской академии наук
Институт прикладной математики им. М.В. Келдыша РАН
E-mail tamaras@keldysh.ru

Основное внимание уделено проблемам информационно-математического обеспечения проблем космонавтики: исторический экскурс и современные перспективы. Речь идет о проектах навигации, ориентации, стабилизации космических кораблей, упреждения стартов ракет из космоса, космического землеобзора, Лунном проекте, пионерских космических исследованиях по дистанционному зондированию Земли, которые проводились советскими космонавтами на пилотируемых космических кораблях и долгосрочных орбитальных станциях. Отмечена важнейшая роль М.В. Келдыша, который фактически определил основные направления становления и развития космических исследований и дистанционного аэрокосмического зондирования Земли как планеты, называемых в настоящее время во всем мире кратко REMOTE SENSING.

1. Введение

Настоящая статья посвящается ***100-летнему юбилею Главного Теоретика космонавтики академика М.В. Келдыша*** (10.02.1911–24.06.1978) — идеолога и организатора космических исследований и ***50-летию запуска в Советском Союзе 12 апреля 1961 года ПЕРВОГО в истории земной цивилизации человечества "корабля-спутника"*** — так называл Главный Конструктор космонавтики академик С.П. Королев искусственный спутник Земли с космонавтом — и ***полета в космическое пространство ПЕРВОГО в мире космонавта*** — гражданина Советского Союза Юрия Алексеевича Гагарина (09.03.1934–27.03.1968) — ***ГЛАВНЫМ событиям 2011 года, объявленного международным годом космоса***.

В ***2007 году*** прогрессивное научное сообщество отметило ***ТРИ ЭПОХАЛЬНЫХ юбилея***, которые разделяют ровно 50 лет:

- ***150 лет со дня рождения Э.К.Циолковского*** (05.09.1857-19.09.1935);
- ***100 лет со дня рождения С.П.Королева*** (12.01.1907-14.01.1966);
- ***50 лет со дня запуска первого искусственного спутника Земли*** (04.10.1957).

Одно событие в полвека — закон природы? Но в 2007 году этот закон нарушен, поскольку ничего эпохального в 2007 году не состоялось. Следует отметить, что в отличие от работ по созданию первой атомной бомбы и одной из ракет, где имели место элементы информированности о работах в США и Германии, в разработках и создании межконтинентальной ракеты и осуществлении запусков искусственных спутников Земли (ИСЗ), "кораблей-спутников" с космонавтами на борту (ПКК) и орбитальных долгосрочных станций (ДОС) ***советские ученые были первыми в мировой истории науки и техники, а специалисты США, Франции и других стран следовали за достижениями Советского Союза (СССР)***. На прошедшем 1–5 октября 2007 года ***Международ-***

*Работа выполнена при поддержке РФФИ (проекты 09-01-00071, 11-01-00021).

ном космическом форуме "Космос: наука и проблемы XXI века", посвященном 50-летию запуска первого ИСЗ, администратор НАСА Майкл Гринфилд свою речь посвятил успехам советских ученых — пионерам в области космонавтики и отметил огромное влияние советской космонавтики на космические проекты в США. По мнению НАСА и Европейского космического Агентства: ***в первые 10–15 лет космической эры советская космонавтика занимала лидирующее положение и многие достижения в космосе являлись пионерскими***.

Далее речь пойдет о космических исследованиях и аэрокосмическом дистанционном зондировании и мониторинге Земли, в частности, связанных с обнаружением стратосферных аэрозольных слоев, возникающих в результате извержений вулканов, мощных пожаров и последствий военных операций (война во Вьетнаме), которые учитываются при расчетах радиационных членов в климатических моделях и оценках радиационного форсинга на климат. ***ВПЕРВЫЕ*** такую проблему автору пришлось решать при математическом моделировании ореола Земли, ***ВПЕРВЫЕ*** увиденного из космоса Ю.А. Гагариным (первые визуальные наблюдения) и ***ВПЕРВЫЕ*** сфотографированного (первые инструментальные наблюдения) с пилотируемых космических кораблей в июне 1963 года Валерием Федоровичем Быковским на ПКК "Восток-5" и Валентиной Васильевной Терешковой на ПКК "Восток-6": ***космонавты ВПЕРВЫЕ сфотографировали дневной и сумеречный горизонты Земли и провели ПЕРВЫЙ научный эксперимент по исследованию Земли из космоса при участии космонавтов***.

2. Атомный и космический проекты

Двадцатый век в истории земной цивилизации — это век научно-технической революции (НТР), связанной с ***тремя великими открытиями***:

- ***проникновение в тайны и овладение ядерной энергией***;
- ***покорение космического пространства и выход человека в космос***;
- ***изобретение электронно-вычислительных машин (ЭВМ) и создание информационных технологий***.

Два эпохальных научных проекта — атомный и космический — способствовали колоссальному развитию советской науки, которая могла конкурировать с мировой наукой XX века. ***Компьютер*** явился главным действующим лицом, ***основным двигателем НТР***: использование ядерной энергии, полет в космос, информационные технологии, естественно, были бы невозможны без ЭВМ. Впервые для реализации инженерно-конструкторских проектов потребовалось решение больших задач на ЭВМ и были ***заложены основы новой технологии, которую позже назвали "математическое моделирование" или "computer science"***.

Разработка информационно-математических аспектов этих проектов привела к расцвету ***кинетической теории переноса нейтронов, заряженных частиц, излучения разной природы в широком диапазоне спектра длин волн***. В Институте Келдыша в 1955 году был создан Отдел № 7 "Кинетические уравнения", который принимал активное участие в работах по обоим проектам. Этот отдел основал профессор Евграф Сергеевич Кузнецов (13.09.1901-17.02.1966), который уже в 1952 году заложил фундамент и ныне работающего "Математического отдела" Физико-энергетического института (ФЭИ, г. Обнинск). В ФЭИ под руководством Е.С. Кузнецова проводились расчеты для запуска первой в мире атомной станции в 1954 году. Е.С. Кузнецов сотрудничал с Игорем Васильевичем Курчатовым, заместителем которого по вычислительным работам являлся академик Сергей Львович Соболев (06.10.1908-03.01.1989).

После Е.С. Кузнецова с 1955 года математическим отделом ФЭИ руководил академик Гурий Иванович Марчук, который работал по проектам создания атомных реакторов спе-

циального назначения. В Арзамасе многие вычислительные задачи по атомному проекту были выполнены под руководством Василия Сергеевича Владимирова.

В 50-ые годы В.С. Владимиров, Е.С. Кузнецов, Г.И. Марчук являлись главными специалистами по теории переноса, ориентированной на атомный проект и атомную энергетику.

Но ещё с 1925 года Е.С. Кузнецов занимался теорией переноса солнечного и теплового излучения в атмосфере и море в связи с проблемами авиации, климата, прогноза погоды, метеорологии, урожайности и т.д. Не случайно ученики Е.С. Кузнецова и сотрудники отдела "Кинетические уравнения" Института Келдыша были привлечены и приняли активное участие в информационно-математическом обеспечении первых и последующих космических проектов. Не случайно в этих проектах участвовал Институт Келдыша, поскольку М.В. Келдыш являлся Главным Теоретиком по космонавтике, а его заместитель А.Н. Тихонов (30.10.1906-07.10.1993) уже имел большой опыт по проведению вычислительных экспериментов и решению больших задач в рамках атомного проекта: в 1948 году Андрей Николаевич при активном участии Александра Адреевича Самарского провел ***ПЕРВЫЙ в мировой науке вычислительный эксперимент*** для решения большой и сложной задачи, связанной с моделированием взрыва водородной бомбы по модели "слойка" А.Д. Сахарова, да ещё с распараллеливанием вычислений.

Настоящая статья ориентирована на приложения теории переноса излучения в космических проектах. Сложность становления космических исследований и реализации космических проектов была обусловлена тем, что приходилось иметь дело с "замкнутым кругом":
— чтобы запустить на космические орбиты аппараты и измерить характеристики радиационного поля Земли, нужны предварительные оценочные расчеты этих характеристик на основе моделей теории переноса излучения с учетом многократного рассеяния и поглощения солнечного излучения, а также собственного излучения атмосферы и поверхности Земли;
— чтобы смоделировать перенос излучения в системе "атмосфера – земная поверхность", нужны были данные о пространственных и спектральных распределениях оптико-геофизических параметров атмосферы, описывающих взаимодействие солнечного и собственного излучения с компонентами земной атмосферы и земной поверхностью.

Радиационное поле системы "атмосфера – земная поверхность (суша, океан)" — это ***неотъемлемая компонента*** биосферы, экосистемы и климата Земли, с одной стороны. С другой стороны, радиационные характеристики системы являются ***носителями информации*** о состоянии окружающей среды, атмосферы, облачности, океана, гидрометеоров и всевозможных выбросов с загрязняющими примесями (последствия техногенных аварий, военных действий, лесных и степных пожаров, извержений вулканов и т.д.), а также промышленной и транспортной инфраструктуры. ***Электромагнитное излучение***, регистрируемое разными средствами, является ***основным источником информации*** о строении и физических свойствах планетных атмосфер и поверхностей при дистанционном зондировании. Для пассивных систем наблюдений ***источниками излучения*** являются ***внешний солнечный поток*** коротковолнового диапазона спектра (ультрафиолетовый, видимый, ближний инфракрасный) и ***собственное излучение*** планеты длинноволнового диапазона спектра (инфракрасный, миллиметровый), когда применимо квазиоптическое приближение теории переноса излучения.

В настоящее время возрастает роль "космического землеведения" как той дисциплины, которая должна объединить усилия различных специалистов и позволить им всем вместе "заговорить" на общем языке спутниковых исследований. Важным становится рассмотрение информационно-математических основ "космического землеведения", объединяющего междисциплинарные исследования, связанные
— с оценкой информационного содержания данных дистанционных и контактных измерений,
— с разработкой методов анализа и интерпретации аэрокосмических изображений,

— с оценкой состояния и пониманием проблем предсказуемости глобальных и региональных изменений природных сред на базе временных рядов регулярных спутниковых наблюдений,
— с исследованиями по оптимизации и эффективности систем наблюдений в интересах различных областей приложений.

"Космическое землеведение" — это оперативная информация о стихийных бедствиях и экологических катастрофах антропогенно-техногенного и естественно-природного происхождения и космический мониторинг глобальных изменений окружающей среды, включая экологические катастрофы замедленного действия.

В интересах международной кооперации по аэрокосмическому глобальному мониторингу Земли и международного глобального проекта по изучению эволюции Земли, климата и опасных явлений (в конце 2004 года около 50 стран подписали международное Соглашение) требуется развитие и разработка нового математического обеспечения для решения прямых и обратных задач теории переноса излучения в природных средах, реализуемого на высокопроизводительных суперкомпьютерах. Ежегодно проводятся саммиты по проблемам климата и загрязнения окружающей среды с участием руководителей правительств почти 200-х стран!

3. А как это начиналось?

Кто заложил основы космических исследований, дистанционного зондирования и получил первые результаты? Полезно почитать книгу В.С. Губарева "Русский космос" [1]. В мае 1946 года советским руководством было принято Постановление о развитии ракетостроения в СССР. В истории отечественной ракетно-космической техники решающая роль принадлежит Сергею Павловичу Королеву и созданному под его началом в 1947 году Совету главных конструкторов, не имевшему прецедента в истории мировой науки. Королев был признанным вождем, руководителем и полководцем советской космонавтики. Однако, очень велика была роль Мстислава Всеволодовича Келдыша. ***М.В. Келдыш*** считался Главным Теоретиком космонавтики и действительно был ***организатором математической школы***, которая обеспечила решение многих практических задач ракетодинамики, небесной механики, баллистики и навигации космических полетов и заложила основы космического землеобзора и дистанционного зондирования Земли.

В феврале 1954 года в кабинете М.В. Келдыша (с 1981 года это Мемориальный музей-кабинет академика М.В. Келдыша в Главном корпусе ИПМ РАН) прошло ***ПЕРВОЕ совещание по искусственному спутнику Земли***. В 1954 году М.В. Келдышем, С.П. Королевым и М.К. Тихонравовым было представлено письмо в ЦК КПСС и Совет министров с предложением о создании и запуске искусственного спутника Земли. 12 февраля 1955 года вышло Постановление о строительстве космодрома "Байконур".

О космических исследованиях заговорили в 1955 году. ***М.В. Келдыш — идеолог и организатор космических исследований***. По указанию М.В. Келдыша летом 1955 года из Академии наук СССР разослали письма ученым разных специальностей с одним вопросом: ***"Как можно использовать космос?"*** Мнений и предложений было много и разных. Для убеждения руководителей СССР в необходимости освоения космического пространства и запусков космических спутников и кораблей М.В. Келдыш выделил две главные задачи: **разведка и наблюдения Земли**, вокруг которых сформировались многие научно-исследовательские проекты. ***В ноябре 1955 года*** из АН СССР в ЦК КПСС и Совет Министров было направлено письмо с ***Программой космических исследований***. Актуально и в XXI веке.

Между СССР и США БЫЛ ДОСТИГНУТ ПАРИТЕТ по межконтинентальным баллистическим ракетам и остро стояла проблема разработки и создания ПРО (противоракетной обороны). М.В. Келдыш предложил концепцию УПРЕЖДЕНИЯ СТАРТОВ РАКЕТ из КОСМОСА. Этот фантастический проект до сих пор актуален и является мощным факто-

ром сдерживания военных конфликтов.

В январе 1956 года была организована Специальная комиссия по объекту "Д" . 30 января 1956 года М.В. Келдыш был назначен председателем Специальной комиссии Академии наук по ИСЗ; С.П. Королев и М.К. Тихонравов — его заместители. Первый спутник имел шифр "ПС-1" — "простой спутник первый" . Тогда уже мечтали о полете в космос человека и спутник с космонавтом С.П. Королев называл "корабль–спутник" , отражая мечту человечества о космических путешествиях.

17 сентября 1957 года в Колонном зале Дома союзов состоялось торжественное собрание, посвященное 100-летию со дня рождения К.Э. Циолковского — основоположника теории реактивного движения, мечтавшего о межпланетных полетах и завоевании космического пространства. С докладом о практическом значении научных и технических предложений К.Э. Циолковского для развития ракетной техники и запуска искусственных спутников Земли выступил член-корреспондент АН СССР С.П. Королев.

Это были годы жесткого противостояния СССР–США и конкуренции в ракетостроении. В докладе С.П. Королева прозвучало о произведенном успешном испытании сверхдальней межконтинентальной баллистической ракеты: "В наши дни сбываются замечательные предсказания Циолковского. Советские ученые работают над многими новыми проблемами ракетной техники, например, над проблемой посылки ракеты на Луну и облета Луны с возвращением ракеты (или некоторой части аппаратуры) на Землю, над проблемой полета человека на ракете, над глубоким исследованием космического пространства при помощи искусственных спутников. То, что было невозможно вчера, стало возможным сегодня."

18–19 сентября 1957 года в Московском Доме ученых состоялась Научно-техническая конференция отделения Технических и Физико-математических наук, посвященная развитию идей К.Э. Циолковского в области теории и практики реактивного движения и освоения космического пространства. В докладах обсуждались проблемы использования долговременных наблюдений на летающих лабораториях в виде искусственных спутников Земли. В докладе В.А. Егорова (ОПМ МИАН АН СССР) рассматривались некоторые задачи динамики полета к Луне, которые были реализованы в лунных проектах СССР. Через пару недель 04 октября 1957 года в СССР был запущен первый в истории человеческой цивилизации искусственный спутник Земли.

На встрече Нового 1961 года М.В. Келдыш произнес тост: "За космический год! И за полет человека!" ***12 апреля 1961 года советский космонавт Юрий Алексеевич Гагарин совершил первый полет человека в космос***. А уже 16 марта 1962 года запуск первого ИСЗ серии "Космос" положил начало осуществлению комплексной научной Программы оптических исследований и дистанционного зондирования околоземного космического пространства и Земли.

После первых запусков беспилотных космических кораблей состоялась ***встреча "Трёх К" (И.В. Курчатов, М.В. Келдыш, С.П. Королев)*** и далее атомный и космический проекты развивались параллельно, состоялась их "свадьба" , поскольку была поставлена грандиозная стратегическая задача "упреждения старта ракет из космоса" в рамках проекта создания "ракетно-ядерного щита" , который и в настоящее время является главным сдерживающим фактором глобальных войн.

Военные и гражданские, научно-технические и научно-исследовательские проекты выполнялись как взаимно дополняющие. Для принятия решений по программам космических полетов, обобщения информации и обмена данными космических наблюдений были созданы специальные Межведомственные Научно-Технические Советы с многочисленными секциями, в которых главную роль играли ученые и конструкторы из академических институтов и отраслевых организаций. После большого перерыва в октябре 2005 года вышла Федеральная программа по космосу, направленная на неотложные меры по реанимации космонавтики. В настоящее время ситуация меняется: подготовлена Федеральная программа по космосу на перспективу до 2020 и 2030 гг.

Отметим важные ключевые даты и должности М.В. Келдыша:
1953 г. – 1978 г. — директор ОПМ МИАН (1953-1966) и ИПМ АН СССР (1966-1978);
1953 г. – 1955 г. — Академик-секретарь Отделения физико-математических наук АН СССР;
1953 г. – 1960 г. — Член Президиума АН СССР;
30 января 1956 г. — назначен председателем Специальной комиссии АН СССР по ИСЗ;
1960 г. – 1961 г. — Вице-президент АН СССР;
1961 г. (19 мая) – 1975 г. (19 мая) — Президент Академии наук СССР.

28 января 1960 года Решением Правительства для координации работ был образован Межведомственный научно-технический совет по космическим исследованиям при Академии наук СССР и М.В. Келдыш назначен его председателем, С.П. Королев и М.К. Тихонравов — заместители председателя. Этот совет функционирует до сих пор. *Заслугой Мстислава Всеволодовича на этом посту было проведение сбалансированной программы исследований, обеспечившей органичное сочетание всех аспектов освоения космического пространства*. Подтверждением тому явились мировое признание успехов нашей страны, уважение и авторитет М.В. Келдыша, избранного в мае 1961 года Президентом Академии наук. Он руководил Академией 14 лет (с 19 мая 1961 г. по 19 мая 1975 г.).

С избранием М.В. Келдыша Президентом Академии наук СССР происходят существенные изменения как в работе самого Президиума, так и в общественном положении Академии наук в целом. Часто употреблявшееся тогда выражение ***"Академия стала штабом советской науки"*** все больше наполнялось реальным содержанием.

В титанической работе по решению атомной проблемы и ракетно-космических задач большой вклад Мстислава Всеволодовича состоял не только в руководстве научным коллективом, но и в личном участии как автора, *создателя новых вычислительных методов и алгоритмов*. Эти работы предопределили современное развитие в стране ***вычислительной математики*** и численных методов решения задач математической физики.

Для всей страны М.В. Келдыш был великим ее гражданином, выдающимся ученым и организатором науки, одним из "Трех К" — руководителей Программы "Ракетно-ядерный щит СССР", Главным Теоретиком космонавтики, Президентом Академии наук СССР, Кавалером трех золотых звезд "Герой Социалистического Труда" и многих других наград и званий (см. "Страницы памяти" www.keldysh.ru).

В ОПМ МИАН был организован Баллистический центр, который, начиная с запуска первого искусственного спутника Земли, успешно решает проблемы баллистико-навигационного обеспечения полетов пилотируемых кораблей, долговременных орбитальных станций "Салют", "Мир", МКС, многоразовой космической системы "Энергия-Буран", автоматических аппаратов научного назначения "Луна", "Венера", "Марс", "Фобос" и др., участвует в разработке и реализации международных космических проектов. В 1967 году за большие заслуги перед отечественной наукой и государством Институт был награжден орденом Ленина, в 1978 году Институту присвоено имя М.В. Келдыша.

В 1965 году был организован Институт космических исследований АН СССР. Организатором и первым директором ИКИ (1965–1973) по рекомендации Президента АН СССР М.В. Келдыша являлся академик Георгий Иванович Петров (1912–1987). В 1966 году, после кончины С.П. Королева, появилось открытое название Института Келдыша: "Отделение прикладной математики Математического института имени В.А. Стеклова АН СССР", основанное в 1953 году, приняло статус "Института прикладной математики АН СССР".

В 1961 году М.В. Келдыш стал Президентом АН СССР и сохранил свой пост Председателя Совета по космосу АН СССР. Далее по традиции Президенты АН СССР являлись Председателями Совета по космосу АН СССР. Как известно, последним Президентом АН СССР и последним Председателем Совета по космосу АН СССР являлся Гурий Иванович Марчук. Ни один космический проект не принимался даже к проектированию без экспертного заключения Совета по космосу АН СССР. В 1980 году Председателем Государственного Комитета по науке и техники СССР (ГКНТ) был назначен Гурий Иванович Марчук,

по инициативе которого ГКНТ стал играть существенную роль в подготовке и реализации космических проектов. Под руководством Гурия Ивановича были организованы Государственные научно-технические программы по развитию мирного космоса и использованию космических технологий для решения фундаментальных и народно-хозяйственных задач. В 1981 году академик Г.И.Марчук получил первую "Золотую медаль имени М.В.Келдыша" Академии наук СССР за выдающиеся результаты в области прикладной математики и механике.

4. Космические проекты и математическое моделирование

В течение тысячелетий человечество изучает звезды и планеты солнечной системы путем визуальных, а позднее фотографических и фотоэлектрических наблюдений. Только планета Земля до конца 50-х годов оставалась недоступной. Лишь по отраженному свету от поверхности Луны (пепельный свет) представлялось возможным оценить интегральное излучение Земли. Широкие возможности исследований радиационных характеристик нашей планеты появились в результате создания и развития ракетной и космической техники. Опыт осуществления в СССР космической программы подтвердил реальность тех перспектив, которые связаны с использованием ПКК, ДОС, автоматических межпланетных станций (АМС), космических аппаратов (КА), искусственных спутников Земли (ИСЗ) для исследования природной среды и природных ресурсов Земли из космоса. 20 ноября 1998 года состоялся запуск первого модуля "Заря" (Россия) первой Международной космической станции (МКС) — космической лаборатории настоящего и будущего.

Важной составной частью первых научных космических программ являлись оптические исследования [2–6]:
— визуальные наблюдения, фотометрические и спектральные исследования сумеречной и дневной атмосферы с целью изучения вертикальных профилей оптически активных компонентов (аэрозоль, озон, газовые примеси);
— исследования спектров отражения различных типов природных образований на поверхности Земли и оценка влияния атмосферы на спектральные яркости и контрасты природных объектов при наблюдениях (съемке) из космоса.

Анализ космических спектров природных образований (спектральных яркостей, коэффициентов спектральных яркостей, спектральных контрастов) показал принципиальную возможность решения ряда фундаментальных и практических задач "космического землеведения".

В хронологии пионерских работ советских ученых по дистанционному зондированию атмосферы и земной поверхности Земли особое место занимают достижения советской пилотируемой космонавтики, связанные с огромной ролью ПКК и ДОС с экипажами космонавтов, которые проводили пионерские уникальные космические эксперименты в контролируемых условиях. США предпочтение отдавали искусственным спутникам Земли, работающим в автоматическом режиме.

Полет Ю.А. Гагарина 12 апреля 1961 г. на ПКК "Восток", который совершил один виток за 108 мин. вокруг Земли. ***Это был ПЕРВЫЙ ВЗГЛЯД человека из космоса на Землю, т.е. ПЕРВЫЕ визуальные наблюдения поверхности и ореола Земли, когда ВПЕРВЫЕ человек увидел "КОСМИЧЕСКУЮ ЗАРЮ"***.

Полеты Г.С. Титова на ПКК "Восток-2" (август 1961), А.Г. Николаева на ПКК "Восток-3" и П.Р. Поповича на ПКК "Восток-4" (август 1962) расширили представления о возможностях визуальных наблюдений. Г.С. Титов 6 августа 1961 г. в начале второго витка ПКК "Восток-2" ***ВПЕРВЫЕ в мире провел киносъемку Земли из космоса***.

В.Ф. Быковский на ПКК "Восток-5" и В.В. Терешкова на ПКК "Восток-6" (июнь 1963) ***впервые сфотографировали дневной и сумеречный горизонты Земли — провели первый научный эксперимент из космоса***. Было положено начало инструментальным

исследованиям оптически активных компонентов атмосферы с ПКК. Теоретическое обоснование этих экспериментов провел Г.В. Розенберг. Математическое моделирование обеспечила Т.А. Сушкевич [4].

С ПКК "Союз-5" (Б.В. Волынов, Е.В. Хрунов, январь 1969) под руководством К.Я. Кондратьева начались спектрографические эксперименты. Были получены первые в мире спектры излучения атмосферы и поверхности Земли в видимой области спектра. Фотографирование и спектрографирование космической зари позволило одновременно получать дополняющие друг друга сведения о пространственной и спектральной структуре излучения и атмосферы Земли, в частности, об аэрозольных и озоновых слоях.

Под руководством К.Я. Кондратьева с ПКК "Союз-7" (В.Н. Волков, В.В. Горбатко, октябрь 1969) впервые осуществлен совмещенный эксперимент по фотографированию отдельных участков территории СССР с самолетов и из космоса в интересах изучения влияния передаточной функции атмосферы на результаты оптических наблюдений из космоса, а на ПКК "Союз-9" (А.Г. Николаев, В.И. Севастьянов, июнь 1970) в интересах метеорологического прогнозирования. Фотографирование геолого-географических объектов совмещалось с аэросъемками. С ПКК "Союз-12" (В.Г. Лазарев, О.Г. Макаров, сентябрь 1973) параллельно со спектрографированием земной поверхности проведена первая спектрозональная съемка отдельных участков Земли. Многозональное фотографирование и спектрометрирование атмосферы и поверхности Земли выполнено с ПКК "Союз-13" (П.И. Климук, В.В. Лебедев, декабрь 1973).

А.В. Филипченко и Н.Н. Рукавишников с ПКК "Союз-16" (декабрь 1974) впервые провели фотографирование земной поверхности и атмосферы в поляризованном свете на трассе протяженностью около 30 тыс. км. По программе "Союз-Аполлон" с ПКК "Союз-19" (июль 1975) оптические исследования проводились А.А. Леоновым и В.Н. Кубасовым. Эксперимент по наблюдениям последствий газовых и аэрозольных выбросов из вулкана в стратосферу подготовили Г.В. Розенберг и А.Б. Сандомирский, в обработке космических данных принял участие Ю.Д. Матешвили, а моделирование обеспечила Т.А. Сушкевич.

Отработка научно-технических методов и средств изучения из космоса поверхности Земли и ее геолого-географических характеристик проходила с ПКК "Союз-22" (В.Ф. Быковский, В.В. Аксенов, сентябрь 1976). Совмещение с самолетными съемками способствовало осуществлению первого полномасштабного эксперимента. Исследования акватории морей, океанов и поверхности суши, проведенные В.В. Коваленком и В.В. Рюминым с ПКК "Союз-25" (октябрь 1977), завершили научную космическую программу с ПКК.

После запуска в апреле 1971 г. ПЕРВОЙ ДОС "Салют" расширилась программа визуально-инструментальных оптических наблюдений Земли. 24 апреля 1971 года произошла ПЕРВАЯ стыковка ПКК "Союз-10" (В.А. Шаталов, А.С. Елисеев, Н.Н. Рукавишников) с ДОС "Салют". Начиная с ДОС "Салют-3" (июнь 1974) и на всех последующих ДОС "Салют-4" (декабрь 1974), "Салют-5" (июнь 1976), "Салют-6" (сентябрь 1977), "Салют-7" (апрель 1982), "Мир" (1986) выполнялась программа "космического землеобзора". В июле 1985 года прошел первый крупномасштабный комплексный международный эксперимент "Курск-85", когда наблюдения проводились одновременно с ДОС "Салют-7", ИСЗ, самолетов-лабораторий, вертолетов, наземных пунктов.

5. О супервычислениях и параллельных алгоритмах

К середине 70-ых годов благодаря работам советских и американских ученых фактически уже были заложены методические основы современных космических технологий дистанционного зондирования, которые в настоящее время являются массовыми и в них принимают участие ученые и специалисты из более 40 стран. Существенное отличие современных технологий от предыдущих касается, преимущественно, технологий приема, обработки и представления космических данных, т.е. лежит в области информационных тех-

нологий. ***Космические исследования — это такая область фундаментальных и прикладных работ, которая с первых шагов своего становления не могла развиваться без использования ЭВМ***.

Освоение космического пространства послужило значительным фактором совершенствования ЭВМ и формирования ***новых научных направлений, связанных с математическим моделированием радиационного поля Земли, теорией переноса изображения, теорией видения, теорией обработки и распознавания образов и т.д. Информационно-математическое обеспечение — обязательная составная часть любого космического проекта***. Вести "космические наблюдения" над чужой территорией запрещено международным правом, а потому наблюдения проводятся по наклонным и касательным направлениям. Для решения задач "ракетно-ядерного щита" и "космического землеобзора", а также "Лунной" программы с возвращением ракеты с Луны на Землю по её яркостному изображению и многих других приложений представляют интерес многомерные сферические и плоские модели радиационного поля [2].

Речь идет о кинетическом подходе к моделированию переноса электромагнитного излучения в природных и искусственных средах на основе общих краевых задачах для кинетического уравнения, которое является линеаризованным приближением уравнения Больцмана с бинарными столкновениями. Естественно, используется "дуализм", когда излучение можно рассматривать и как "электромагнитные волны" и как "частицы" (фотоны). До сих пор это одни из самых сложных задач теории переноса излучения, требующие огромных ресурсов и высокой производительности ЭВМ. В последнее десятилетие такие задачи решаются на суперкомпьютерах с распараллеливанием вычислений.

Для космических проектов и космических наблюдений с первых шагов освоения космического пространства необходимо было разрабатывать методологию решения двух основных классов многомерных задач теории переноса излучения [2]:
— прежде всего для сферической оболочки (сферическая Земля с атмосферой),
— а позже для 3D плоского слоя (атмосфера над земной поверхностью),
с двумя типами источников:
— внешним параллельным потоком солнечного (коротковолнового) излучения,
— собственным (длинноволновым, инфракрасным) излучением.

В Институте Келдыша Т.А. Сушкевич начинала на ЭВМ "Стрела" (1961 г.). После участия в сдаче ЭВМ "Весна" в августе 1964 года (параллельно осваивали "Восток", "М-20", "БЭСМ-4" и "АЛГОЛ") участвовала среди первых в освоении многих новых ЭВМ разных поколений и архитектур. Сферические многомерные модели переноса излучения, несмотря на их сложность и громоздкость численной реализации на первых поколениях ЭВМ (М-20, БЭСМ-4, БЭСМ-6, АС-6), в 60-е – 80-е годы имели исключительную актуальность в связи с проектированием и созданием ракетно-космических систем, освоением ближнего и дальнего космического пространства, организацией и проведением космических исследований и наблюдений. ЗАДАЧИ для сферической Земли на БЭСМ-6 считались по 300 часов (использовалась 14 МЛ и 6 МД, программа 25 тысяч перфокарт на Автокоде, 1965-1966 гг.) без потерь на внешние обмены, поскольку ВПЕРВЫЕ были реализованы возможности РАСПАРАЛЛЕЛИВАНИЯ расчета по ресурсам внутренней и внешней памяти: память была разбита на 4 "листа" и пока с одного "листа" памяти шла запись (чтение) на магнитную ленту, на другом "листе" накапливались результаты!

С 1989 года коллектив проводил испытания на всех поколениях и архитектурах суперкомпьютеров с распараллеливанием вычислений. Началось это под личным влиянием академика В.В. Воеводина и его первого доклада про параллельные алгоритмы в задачах алгебры на Конференции в МГУ в начале 80-ых годов. Это направление поддерживал академик Г.И. Марчук, будучи Председателем ГКНТ СССР, и по четвергам в "Зале коллегии" ГКНТ собирался Всесоюзный семинар, на котором обсуждались перспективные направления в развитии ЭВМ.

Созданный в Институте прикладной математики имени М.В. Келдыша АН СССР вычислительный аппарат использовался

- ***для фундаментально-поисковых научных исследований по разработке методов и средств космических наблюдений,***
- ***дистанционного зондирования из космоса,***
- ***ориентации, стабилизации и навигации КА, астронавигации ракет,***
- ***для интерпретации и анализа данных космических и комплексных экспериментов, проводимых на ПКК и ДОС, а также аэростатных, самолетных и наземных наблюдений.***

Под руководством Т.А. Сушкевич продолжены работы по информационно-математическому аспекту космических исследований и супервычислениям с параллельными алгоритмами. При дистанционном зондировании и мониторинге технических объектов и окружающей среды носителем информации об их состоянии и свойствах является электромагнитное излучение, регистрируемое различными средствами. Представляют актуальность перспективные гиперспектральные системы нанодиагностики природной и техногенной среды на основе данных аэрокосмического дистанционного зондирования атмосферы и поверхности (объектов техносферы на поверхности). Для решения таких проблем требуется информационно-математическое обеспечение, включающее прямые и обратные задачи теории переноса излучения, модели которых основаны на передаточном операторе [2].

Используются следующие приемы распараллеливания вычислений для задач с поляризацией:
(1) распределенные вычисления по физическим моделям:
— многоспектральные (по длине волны);
— по оптико-геофизической погоде (по коэффициентам общей краевой задачи);
— по источникам излучения;
(2) распределенные вычисления на основе методического распараллеливания:
— декомпозиции краевых задач:
— по моделям переноса излучения, т.е. по приближениям теории переноса излучения;
— по подобластям;
— по параметрам вектора параметров Стокса функций влияния;
— по параметрам вектора параметров Стокса пространственно-частотных характеристик;
— по компонентам матричных функционалов;
(3) алгоритмическое распараллеливание для многомерных моделей:
— однократное рассеяние по характеристикам;
— многократное рассеяние по интегралам столкновений;
— по квадрантам угловых разностных сеток;
— по подобластям с разными сеточно-характеристическими схемами.

Основные составные части математического обеспечения:
— банки данных по оптико-метеорологическим моделям атмосферы и земной поверхности;
— система автоматизированного расчета спектро-энергетических и других радиационных характеристик атмосферы и Земли в различных диапазонах спектра от УФ до ММВ;
— банки данных радиационных характеристик (функции влияния локальных возмущений параметров или источников в атмосфере, дымах, облаках, гидрометеорах, океане и на земной поверхности, пространственно-угловые и спектральные распределения яркости системы Земля-атмосфера, функции пропускания и сферическое альбедо атмосферы и т.д.);
– пакеты программ обработки, визуализации и диагностики результатов численного эксперимента и аэрокосмических данных.

С учетом источников и процессов трансформации излучения выделяются четыре основные физико-математические модели, отвечающие спектральным диапазонам:

— оптический диапазон (источник — Солнце, многократное рассеяние);
— ближний ИК-диапазон (источники — Солнце и собственное излучение, многократное рассеяние);
— ИК-диапазон (источник — собственное излучение, без многократного рассеяния, сложная структура спектров поглощения);
— ММВ диапазон (источник — собственное радиоизлучение, многократное рассеяние в гидрометеорах и облаках, сложные спектры поглощения).

Создаваемая система содержит три группы программных комплексов, соответствующих трем этапам решения задачи для создания оперативных баз данных (спектральных радиационных атласов):

Первая группа программ — формирование оптико-метеорологических моделей среды:
— программы работы с архивом и базами данных моделей атмосферы, облаков, дымов, земной поверхности, океана;
— банк спектров поглощения атмосферных газов;
— банк характеристик аэрозольного рассеяния и поглощения;
— формирование модели атмосферы;
— пакеты данных к программам расчета радиационных характеристик и т.д.

Вторая группа программ — численное решение уравнения переноса излучения быстрыми приближенными и репрезентативными высокоточными методами:
— для плоской и сферической геометрии,
— для системы свободная атмосфера-дымовая завеса,
— для системы атмосфера-океан,
— для системы атмосфера с многоярусными облаками,
— для функции влияния атмосферы, дымов, облачности, гидрометеоров, океана,
— для функции пропускания атмосферы, отягощенной многократным рассеянием, и т.д.

Третья группа программ — обработка и диагностика результатов расчетов:
— расчет функционалов;
— аналитическая аппроксимация и параметризация табличных функций;
— компьютерная графика и визуализация;
— решение обратных задач по восстановлению параметров среды и т.д.

Предложенная архитектура программного обеспечения с функциональным наполнением, ориентированным на решение задач мониторинга развития и оценки последствий воздействия техногенных аварий и природных катастроф, а также природно-ресурсных, экологических, геоэкоинформационных и т.п. задач, позволяет осуществлять модификацию и адаптацию вычислительно-информационной системы применительно к конкретным проблемам математического моделирования радиационных процессов в системе Земля-атмосфера или восстановления набора параметров зондируемой среды.

В настоящее время, в отличие от момента начала работ в 60-е годы, благодаря активному развитию теоретических и экспериментальных исследований по проблемам светорассеяния, а также систем космических наблюдений мы располагаем достаточно достоверными данными
— о тонкой структуре полос поглощения водяного пара и газовых компонент атмосферы и способах учета этих данных для математического моделирования радиационного переноса в поглощающей реальной атмосфере;
— о коэффициентах и индикатрисах рассеяния атмосферы с учетом аэрозольных примесей;
— об отражающих свойствах естественных поверхностей;
— о географических, сезонных, суточных распределениях, вариациях и статистических характеристиках влажности, давления, температуры, концентраций газовых и аэрозольных компонент и облачности, имеющих случайный характер и играющих основную роль в изменчивости радиационного поля Земли.

Каждая из этих моделей описывается совокупностью оптико-метеорологических (гео-

физических) характеристик атмосферы, облаков, подстилающей поверхности, которые являются входными физическими данными для уравнения переноса (через коэффициенты, граничные условия, источники).

6. Заключение

Всемирная система мониторинга и иерархия моделей — главные инструменты для прогнозирования изменений в природных процессах и разделения естественных и антропогенных воздействий.

Одно из важных направлений Федеральной программы по космосу связано с использованием космических данных для оптимального управления регионами с целью обеспечения их устойчивого развития.

В масштабах планеты стоит актуальная проблема создания международного глобального мониторинга Земли с целью исследования её эволюции и прогнозирования естественно-природных стихийных бедствий и антропогенно-техногенных катастрофических процессов.

Это грандиозные задачи, решение которых требует разработки нанотехнологий для космических проектов, в частности, гиперспектральной аппаратуры дистанционного зондирования и миниспутников и математического моделирования больших задач.

Литература

1. Губарев В.С. Русский космос (Сверхдержава. Русский прорыв). М.: АЛГОРИТМ, 2006. 464 с.

2. Сушкевич Т.А. Математические модели переноса излучения. М.: БИНОМ. Лаборатория знаний, 2005. 661 с.

3. Сушкевич Т.А. О пионерских работах по математическому моделированию радиационного поля Земли при освоении космоса / Методы и алгоритмы обработки спутниковых данных // Современные проблемы дистанционного зондирования Земли из космоса. Физические основы, методы и технологии мониторинга окружающей среды, потенциально опасных явлений и объектов. Институт космических исследований РАН. Сборник научных статей. Выпуск 5. Том 1. М.: ООО "Азбука-2000" , 2008. С. 165–180. ISSN 2070-7401.

4. Сушкевич Т.А. К истории первого научного эксперимента по дистанционному зондированию Земли на пилотируемом космическом корабле / Вопросы создания и использования приборов и систем для спутникового мониторинга состояния окружающей среды // Современные проблемы дистанционного зондирования Земли из космоса. Физические основы, методы и технологии мониторинга окружающей среды, потенциально опасных явлений и объектов. Институт космических исследований РАН. Сборник научных статей. Выпуск 5. Том 1. М.: ООО "Азбука-2000" , 2008. С. 315–322. ISSN 2070-7401.

5. Sushkevich T.A. Pioneering remote sensing in the USSR. 1. Radiation transfer in the optical wavelength region of the electromagnetic spectrum // International Journal of Remote Sensing, Vol. 29, N. 9. P. 2585–2597.

6. Sushkevich T.A. Pioneering Remote Sensing in the USSR. 2. Global spherical models of radiation transfer // International Journal of Remote Sensing, Vol. 29, N. 9. P. 2599–2613.

Использование усечённого варианта алгоритма SPIKE из библиотеки Intel Adaptive SPIKE-Based Solver для решения упругопластической задачи*

А.В. Толмачев, А.В. Коновалов, А.С. Партин

Институт Машиноведения УрО РАН

Исследованы возможности усечённого алгоритма SPIKE из библиотеки Intel Adaptive Spike-Based Solver для распараллеливания решения системы линейных уравнений внутри упругопластической задачи. Исследование проводилось на примере задачи сжатия цилиндра. Расчёты проводились на кластере um64 Института математики и механики УрО РАН. Решён ряд проблем, возникающих при использовании этой библиотеки из языка C++.

1. Постановка упругопластической задачи

Упругопластическая задача с большими пластическими деформациями физически и геометрически существенно нелинейная и требует большого количества времени для ее решения на персональном компьютере. На решение двумерной задачи затрачивается несколько часов, для трехмерной задачи это время увеличивается до нескольких суток. Существенно сократить время вычислений можно с помощью техники параллельных вычислений, в частности, решая данные задачи на кластерных системах.

Решение упругопластических задач методом конечных элементов осуществляется шагами по нагрузке, и на каждом таком шаге состоит из трех основных этапов [1]:

1. расчет локальных матриц жесткости для конечных элементов и формирование матрицы A (глобальной матрицы жесткости) и вектора F правой части системы линейных алгебраических уравнений (СЛАУ)

$$AX = F \qquad (1)$$

относительно искомого вектора X обобщённой скорости в узлах конечно-элементной сетки;

2. решение СЛАУ (1);
3. вычисление напряженно-деформированного состояния в конечных элементах в конце шага нагрузки.

Матрица A имеет ленточный вид. На каждом шаге нагрузки этап 1 выполняется один раз, а этапы 2 и 3 – от десяти до пятнадцати раз для удовлетворения итерационно с приемлемой точностью условию пластичности. При этом матрица жёсткости не меняется, а изменяется только правая часть системы уравнений.

Если этапы 1 и 3 легко распараллеливаются, то распараллеливание процесса решения СЛАУ является сложной задачей. Решение этой задачи итерационными методами рассмотрено в работе [2]. Исследование эффективности распараллеливания трёхдиагонального алгоритма LU-разложения из библиотеки ScaLAPACK [3] при решении получающейся СЛАУ приведено в работе [4].

Целью работы является исследование возможностей параллельного алгоритма SPIKE [5, 6] для решения СЛАУ в упругопластических задачах на кластерной системе.

Все численные эксперименты проводились на кластерной системе um64 Института математики и механики УрО РАН. Они представляли собой решение методом конечных элементов задачи сжатия цилиндра из упругопластического изотропного и изотропно-упрочняемого материала плоскими плитами, постановка которой приведена в работе [2].

*Работа выполнена в рамках программы Президиума РАН "Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация".

2. Описание алгоритма SPIKE

Использовали решатель систем линейных уравнений Intel Adaptive Spike-Based Solver (библиотеку программ), доступный по адресу http://software.intel.com/en-us/articles/intel-adaptive-spike-based-solver/. Лежащий в его основе алгоритм SPIKE для решения СЛАУ с ленточной матрицей состоит из трёх этапов: разбиения системы, разложения матрицы системы и решения системы с разложенной матрицы. На первом этапе систему уравнений разделяют на участки, удобные для дальнейшей работы, на втором этапе производится разложение матрицы системы на более удобные для решения СЛАУ матрицы, а на последнем этапе происходит непосредственно решение СЛАУ с использованием полученных матриц.

2.1. Разбиение системы

Рассмотрим СЛАУ вида (1) с ленточной матрицей A размерности $n \times n$ с узкой лентой и матрицей-столбцом F. Разобьём матрицы A и F на p участков горизонтальными линиями и распределим i-й участок разбиения на i-й процессор. Рис. 1 показывает распределение матриц A и F для $p = 4$. Выделим на i-м участке разбиения три блока: диагональный квадратный блок $A_i \left(i = 1,\ldots, p\right)$, который имеет размерность n_i; над-диагональный квадратный блок $B_i \left(i = 1,\ldots, p-1\right)$ и под-диагональный квадратный блок $C_i \left(i = 2,\ldots, p\right)$. Поскольку лента матрицы A узкая, то размерность блоков B_i и C_i, равная m, будет много меньше n_i.

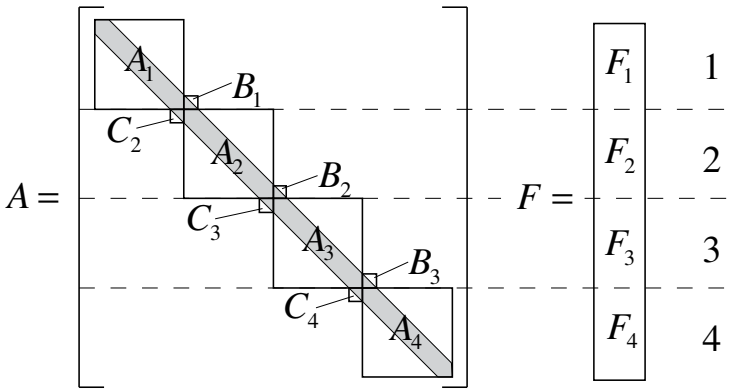


Рис. 1. Распределение матрицы системы A и правой части F на 4 процессора

2.2. Разложение матрицы системы

На данном этапе вычисляется разложение матрицы системы A в виде

$$A = DS \,. \qquad (2)$$

Здесь D – матрица, составленная только из диагональных блоков A_i,

$$D = diag\left(A_1,...,A_p\right),$$

а матрица S, показанная на рис. 2, вычисляется путём умножения i-го участка разбиения матрицы A на матрицу A_i^{-1} слева. Матрица S имеет на диагонали единичные матрицы I_{n_i} размерности n_i и матрицы $V_i \left(i = 1,\ldots, p-1\right)$ и $W_i \left(i = 2,\ldots, p\right)$ на месте внедиагональных блоков. Матрицы V_i и W_i называют шипами (spikes) из-за того, что они имеют размерность $n_i \times m$, то есть являются узкими и высокими.

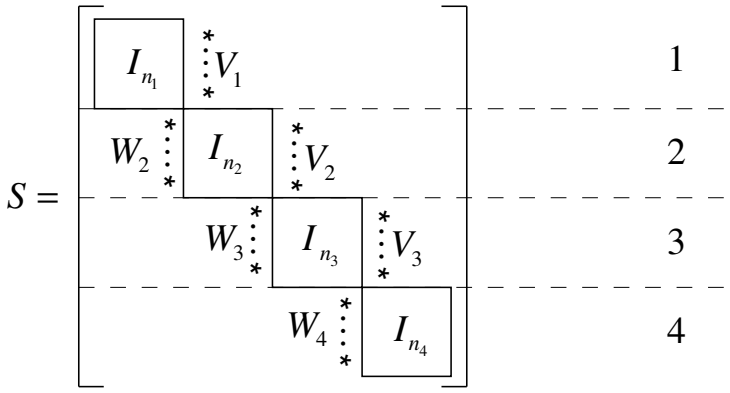


Рис. 2. Матрица S, распределённая на 4 процессора

Шипы V_i и W_i находятся из решения матричного уравнения

$$A_i \begin{bmatrix} V_i & W_i \end{bmatrix} = \begin{bmatrix} 0 & C_i \\ \vdots & 0 \\ 0 & \vdots \\ B_i & 0 \end{bmatrix}. \qquad (3)$$

2.3. Решение системы с разложенной матрицей

После разложения матрицы A в виде (2) решение системы (1) сводится к последовательному решению двух систем:

$$DG = F \qquad (4)$$

$$SX = G. \qquad (5)$$

Решение системы (4) может быть выполнено полностью параллельно, поскольку она состоит лишь из диагональных блоков.

Представим шипы V_i и W_i следующим образом:

$$V_i = \begin{bmatrix} V_i^t \\ V_i^r \\ V_i^b \end{bmatrix} \text{ и } W_i = \begin{bmatrix} W_i^t \\ W_i^r \\ W_i^b \end{bmatrix}$$

где V_i^t, V_i^r, V_i^b и W_i^t, W_i^r, W_i^b соответственно верхние m, средние $n_i - 2m$ и нижние m рядов шипов V_i и W_i. Аналогично локальные части матрицы неизвестных и матрицы правой части X_i и G_i разбиваются как

$$X_i = \begin{bmatrix} X_i^t \\ X_i^r \\ X_i^b \end{bmatrix} \text{ и } G_i = \begin{bmatrix} G_i^t \\ G_i^r \\ G_i^b \end{bmatrix}.$$

Из системы (5) формируется сокращённая система

$$\overline{S}\overline{X} = \overline{G}, \qquad (6)$$

состоящая из m рядов матричного уравнения (5), находящихся выше и ниже линий разделения матриц этой системы (это возможно т.к. $m << n$). Она имеет размерность $2m(p-1)$. Матрица сокращённой системы при распределении исходной системы уравнений на 4 процессора показана на рис. 3. В систему (6) входят только части матриц V_i, W_i, X_i и G_i с верхними индексами b и t. Блоки I_m являются единичными матрицами размерности m.

$$\hat{S} = \begin{array}{|c|c|c|c|c|c|}
\hline
I_m & V_1^b & & & & \\ \hline
W_2^t & I_m & & V_2^t & & \\ \hline
W_2^b & & I_m & V_2^b & & \\ \hline
 & & W_3^t & I_m & & V_3^t \\ \hline
 & & W_3^b & & I_m & V_3^b \\ \hline
 & & & & W_4^t & I_m \\ \hline
\end{array}$$

Рис. 3. Вид матрицы сокращённой системы для случая распределения матрицы A на 4 процессора

Матрица системы (6) является блочно-трёхдиагональной с $(p-1)$ диагональными блоками, i-й из которых имеет вид

$$\begin{bmatrix} I_m & V_i^b \\ W_{i+1}^t & I_m \end{bmatrix}.$$

Левый и правый i-й внедиагональные блоки соответственно выглядят как

$$\begin{bmatrix} W_i^b & 0 \\ 0 & 0 \end{bmatrix} \text{ и } \begin{bmatrix} 0 & 0 \\ 0 & V_{i+1}^t \end{bmatrix}.$$

Блок неизвестных и правая часть, соответствующие i-му диагональному блоку имеют вид

$$\begin{bmatrix} X_i^b \\ X_{i+1}^t \end{bmatrix} \text{ и } \begin{bmatrix} G_i^b \\ G_{i+1}^t \end{bmatrix}.$$

После нахождения решения $\hat{X}$ системы (6), общее решение системы (1) вычисляется по формулам

$$\begin{cases} X_1^r = G_1^r - V_1^r X_2^t \\ X_i^r = G_i^r - V_i^r X_{i+1}^t - W_i^r X_{i-1}^b, \; i = 2, \ldots, p-1. \\ X_p^r = G_p^r - W_p^r X_{p-1}^b \end{cases}$$

2.4. Усечённый вариант алгоритма SPIKE

В случае если матрица системы A имеет диагональное доминирование, то есть выполняется условие

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}|,$$

то шипы W_i и V_i практически полностью состоят из нулей [5]. Это позволяет при решении системы (6) отбросить её внедиагональные блоки. Полученная система называется усечённой. При этом достаточно вычислять только те части шипов, которые входят в диагональные блоки системы (6).

2.5. Параллельность решения

Этап разложения матрицы выполняется полностью параллельно, поскольку для вычислений достаточно данных, находящихся на локальном процессоре. При этом происходит вычисление LU-разложения [7] блоков A_i, после этого вычисляются шипы как решение матричного уравнения (3). Решение системы (4) так же выполняется полностью параллельно, так как матрица D состоит только из диагональных блоков.

Решение системы (6) в стандартном варианте алгоритма может быть выполнено различными способами, например итерационными методами, применением алгоритма SPIKE рекурсивно, и др. В случае если матрица A имеет диагональное доминирование, то вместо системы (6) решается её усечённый вариант без внедиагональных блоков. При этом в процессе решения требуется передать лишь блок W_{k+1}^t с $k+1$ на k-й процессор и получить обратно блок решений. Это увеличивает степень параллельности данного алгоритма. Изменённый таким образом вариант алгоритма называется усечённым алгоритмом SPIKE.

В нашем случае матрица системы не имеет диагонального доминирования, степень её диагонального доминирования δ, которая вычисляется как

$$\delta = \frac{|a_{ii}|}{\sum_{i \neq j} |a_{ij}|}$$

имеет порядок $0,4$. Поэтому сочли возможным использовать усечённый вариант алгоритма с контролем точности решения. Вычислительные эксперименты показали, что в нашем случае значение $\|F - AX\|$ имело порядок 10^{-12}. Здесь $\|\square\|$ обозначена ∞-норма, то есть максимальное по модулю значение компоненты вектора.

3. Особенности использования библиотеки из языка C++

Библиотека Intel Adaptive SPIKE Solver (версия от 23.02.2010 г.) рассчитана для использования из языков FORTRAN 90\95 и C, однако при использовании этой библиотеки из языка C++ возник ряд проблем.

3.1. Ошибки компоновки при использовании библиотеки Intel Adaptive Spike-Based Solver из языка C++

При использовании из языка C++ скомпилированных библиотек, написанных на языке C требуется, чтобы экспортируемые из данной библиотеки функции были помечены с использованием директивы `extern "C"`. Это приведёт к тому, что компилятор языка C++ будет игнорировать особенности разрешения имён, специфические для языка C++, такие как пространства имён, функции-члены, перегрузку функций и шаблоны.

В заголовочных файлах библиотеки Intel Adaptive Spike Solver данная директива отсутствует, поэтому при компоновке возникают ошибки невозможности разрешения имён. Это можно исправить либо добавлением указанных директив в заголовочные файлы, либо включать заголовочный файл, как показано на рис. 4.

```
extern "C" {
#include <spike.h>
}
```

Рис. 4. Способ включения заголовочных файлов библиотеки Intel Adaptive SPIKE-Based Solver из языка C++

3.2. Параллельное транспонирование ленточной матрицы

Библиотека Intel Adaptive SPIKE-Based Solver написана на языке FORTRAN, в котором исторически матрицы представляются в виде двумерного массива, причём используется нумерация элементов с ведущим столбцом. В языках C и C++, в свою очередь, обычно используется нумерация элементов матриц с ведущей строкой. При вызове функций библиотеки Intel Adaptive SPIKE-Based Solver в качестве данных требуется указывать описатели динамически выделяемых массивов языка FORTRAN. В поставке библиотеки присутствуют функции выделения, удаления и доступа к элементам таких массивов, однако это приводит к увеличению по-

требления памяти, поскольку сформированную матрицу жёсткости требуется переносить из массива, в который она формируется, в массив, понятный среде исполнения FORTRAN.

Для экономии памяти можно транспонировать полученную матрицу и заполнить структуры данных, описывающие эту матрицу для среды выполнения языка FORTRAN. При этом из матрицы, имеющей нумерацию с ведущими строками, получим матрицу, имеющую нумерацию с ведущими столбцами.

Операция параллельного транспонирования ленточной матрицы M размерности $n \times n$ с лентой полуширины β, хранимой в сжатой ленточной форме, происходит следующим образом. Допустим, что матрица M разбита на p частей M_i размерности $n_i \left(i = 1 \ldots n\right)$, $n_i > 2\beta$. Каждую матрицу M_i можно разбить на три подматрицы: $H_i \left(i = 1 \ldots n\right)$, которые можно транспонировать независимо друг от друга, и матрицы $J_i \left(i = 2 \ldots n\right)$ и $K_i \left(i = 1 \ldots n-1\right)$, данные в которых надо поменять местами для транспонирования матрицы M. Пример разбиения матрицы на три участка показан на рис. 5.

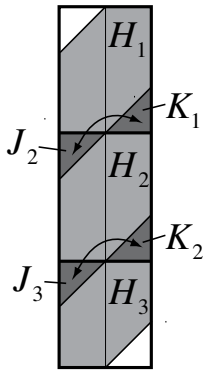


Рис. 5. Разбиение матрицы M для транспонирования при $p = 3$

Для улучшения параллельных свойств алгоритма требуется производить вычисления таким образом, чтобы иметь возможность совмещать вычисления и передачу данных. Таким образом, для i-го процессора параллельный распределённый алгоритм транспонирования ленточной матрицы будет выглядеть следующим образом:

1. Скопировать содержимое матриц J_i и K_i в буферы для передачи данных.
2. Запустить асинхронную передачу матрицы J_i на процессор с номером $i-1$, матрицы K_i - на процессор $i+1$; а также запустить асинхронный приём матриц K_{i-1} и J_{i+1}.
3. Транспонировать матрицу H_i.
4. Завершить асинхронную передачу данных или дождаться её завершения.
5. Скопировать полученное содержимое матрицы J_{i-1} на место матрицы K_i и содержимое матрицы K_{i+1} на место матрицы J_i.

4. Результаты вычислительных экспериментов

Решали задачу конечно-элементного моделирования сжатия упруго-пластического цилиндра при использовании регулярной конечно-элементной сетки с одинаковым количеством разбиений d по обеим осям.

Вычислительные эксперименты проводили на кластере um64 Института математики и механики УрО РАН. Кластер состоит из 32 двухпроцессорных двухядерных модулей на базе процессоров AMD Operton с тактовой частотой 2.6 ГГц. Для вычислительных экспериментов использовали версию библиотеки Intel Adaptive SPIKE-Based Solver от 23 февраля 2010 года, библиотеку ScaLAPACK, реализация которой входит в Intel MKL версии 10.0.010. Для межпроцессорной коммуникации использовалась библиотека OpenMPI версии 1.3.3 на сети InfiniBand.

Для тестирования были взяты сетки с параметрами, представленными в таблице, β - полуширина матрицы жёсткости.

Таблица. Параметры сеток, использовавшихся в вычислительных экспериментах

d	β	n	β/n
100	205	20402	0,0100
150	305	45602	0,0067
200	405	80802	0,0050
300	605	181202	0,0033

Представленные значения ускорений вычислялись по формуле $s = t_n / t_1$, где t_1 - время выполнения операции на одном процессоре, а t_n - время выполнения операции на n процессорах. Для сетки с $d = 300$ за t_1 приняли t_2, поскольку сформированная матрица жёсткости не помещалась в память одного процессора используемой кластерной системы.

4.1. Производительность усечённого алгоритма SPIKE при решении упругопластической задачи

На рис. 6, 7, 8 предоставлена зависимость значения ускорений s усечённого алгоритма SPIKE от количества процессоров p и количества разбиений сетки d соответственно для этапа разложения матрицы системы, этапа решения СЛАУ с уже разложенной матрицей и полного решения СЛАУ на шаге нагрузки упругопластической задачи. При полном решении СЛАУ происходит одно разложение и 15 решений с разложенной матрицей системы.

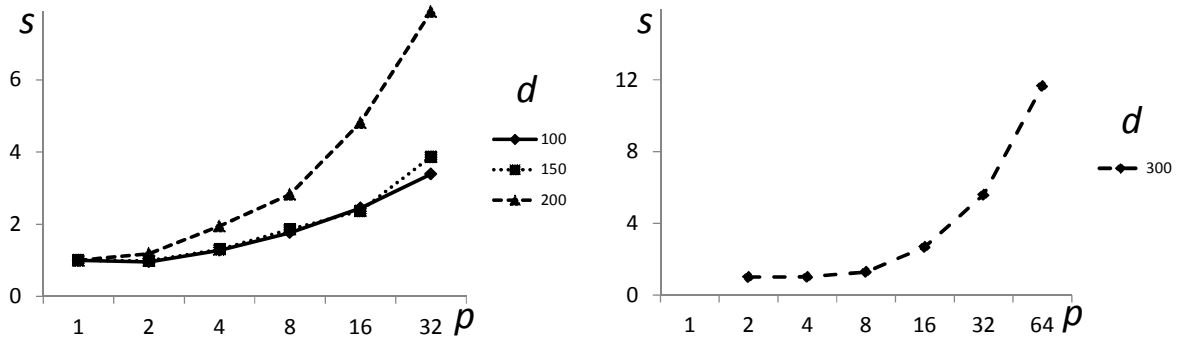


Рис 6. Зависимость ускорений на этапе разложения матрица системы алгоритма SPIKE от количества процессоров p и количества разбиений сетки d

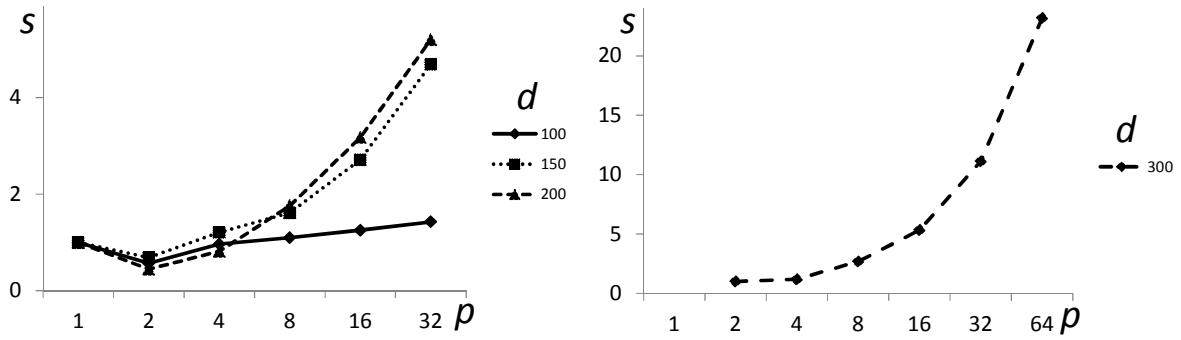


Рис. 7. Зависимость ускорений на этапе решения СЛАУ с разложенной матрицей алгоритма SPIKE от количества используемых процессоров p и количества разбиений d

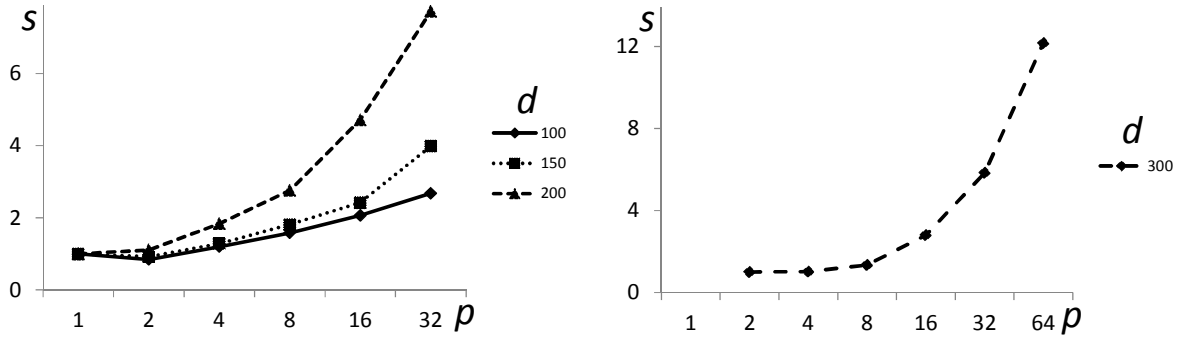


Рис. 8. Зависимость ускорений полного решения СЛАУ внутри шага нагрузки упругопластической задачи для случая одного разложения и 15 решений

Из рис. 6, 7, 8 видно, что рост производительности имеет место как при увеличении количества процессоров, так и при увеличении количества разбиений конечноэлементной сетки. Уменьшение производительности при использовании двух процессоров, наблюдающееся на рис. 7, а так же более пологий наклон графиков на этом рисунке по сравнению с графиками для этапа разложения матрицы для небольших сеток, вызван затратами времени на контроль точности решения.

4.2. Сравнение производительности усечённого алгоритма SPIKE и трёхдиагонального алгоритма LU-разложения из библиотеки ScaLAPACK

На рис. 9 предоставлено сравнение значений ускорений s решения СЛАУ с использованием трёхдиагонального параллельного алгоритма LU-разложения из библиотеки ScaLAPACK и усечённого алгоритма SPIKE, полученные при решении упругопластической задачи сжатия цилиндра с количеством разбиений $d = 200$. Исследование применимости трёхдиагонального алгоритма LU-разложения для решения упругопластической задачи рассмотрено в работе [4].

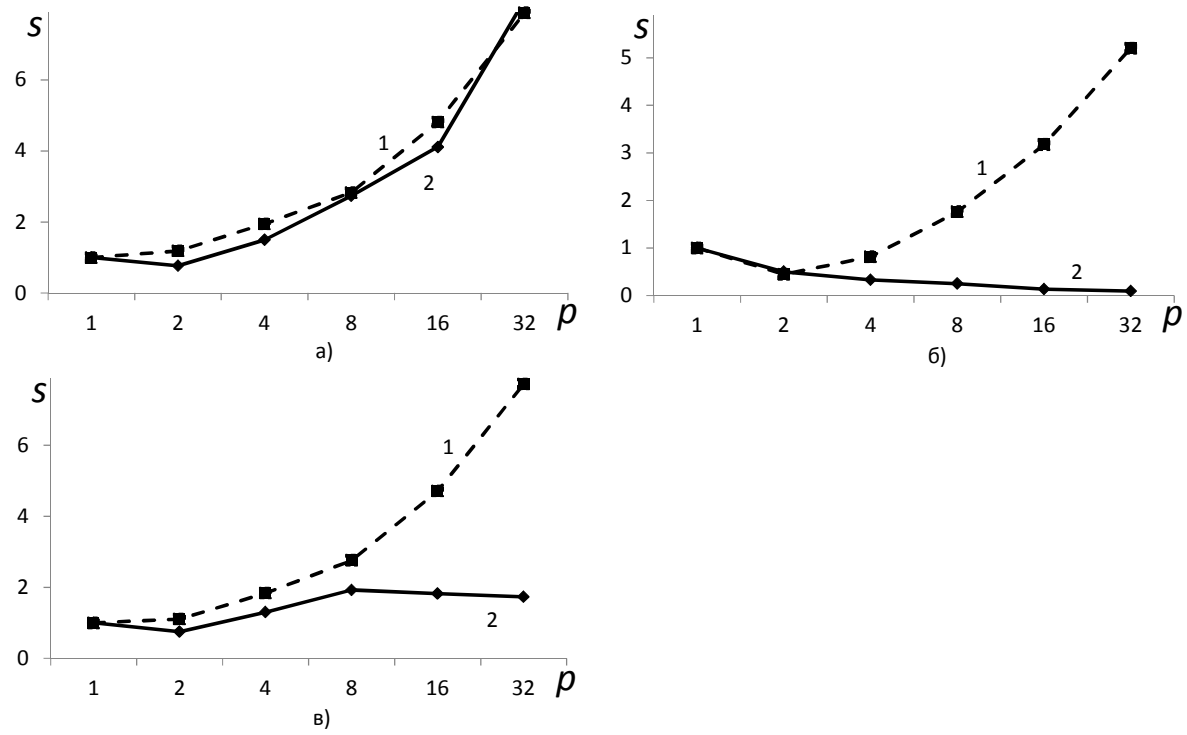


Рис. 9. Ускорение разложения (а); решения СЛАУ (б) и полного решения системы (в) для сетки с количеством разбиений $d = 200$: 1 – SPIKE; 2 – ScaLAPACK

Видно, что усечённый алгоритм SPIKE на сетке с количеством разбиений $d = 200$ имеет приблизительно одинаковую производительность вычисления разложения матрицы системы с трёхдиагональным алгоритмом LU-разложения из библиотеки ScaLAPACK. На этапе решения системы уравнений с использованием разложенной матрицы усечённый алгоритм SPIKE имеет значительно лучшую производительность, чем трёхдиагональный алгоритм. Трёхдиагональный алгоритм LU-разложения имеет коэффициент ускорения $s < 1$ при использовании более одного процессора, однако усечённый алгоритм SPIKE имеет коэффициент ускорения $s > 1$ при использовании более 4 процессоров. При дальнейшем увеличении количества процессоров коэффициент ускорения продолжает увеличиваться.

Литература

1. Поздеев А.А., Трусов П.В., Няшин Ю.И.. Большие упруго-пластические деформации. М: Наука, 1986 - 232с.
2. Демешко И.П., Акимова Е.Н., Коновалов А.В. Применение параллельных алгоритмов для решения системы линейных алгебраических уравнений с ленточной матрицей итерационными методами на кластерной системе // Труды международной конференции "Параллельные Вычислительные технологии (ПаВТ’2009)", Нижний Новгород, 30 марта – 3 апреля 2009. – С. 444 – 449. – Челябинск: Изд. ЮУрГУ, 2009.– 839 с. (электронное издание).
3. Blackford L. S., Choi J., Cleary A., D'Azeuedo E. at all. ScaLAPACK User's Guide. 1997: URL: http://www.netlib.org/scalapack/slug.
4. Коновалов А.В., Толмачев А.В., Партин А.С. Опыт применения параллельного алгоритма LU-разложения для решения линейных систем уравнений в упругопластических задачах // Труды международной конференции "Параллельные вычислительные технологии (ПаВТ’2010)", Уфа, 29 марта – 2 апреля 2010. – Челябинск: Изд. ЮУрГУ, 2010. – С. 498–506. (электронное издание).

5. Polizzi E, Sameh A. SPIKE: A parallel environment for solving banded linear systems. Computers & Fluids. 2007 Jan 0;36:113--120.

6. Polizzi E, Sameh AH. A parallel hybrid banded system solver: the SPIKE algorithm. Parallel Comput. 2006;32:177--194.

7. Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ. М., Вильямс, 2008. – 1296 с.

Воспроизведение атмосферной циркуляции на сезонных масштабах с помощью совместной модели атмосферы и океана

М.А.Толстых[1,2], Н.А. Дианский[1], А.В. Гусев[1], Д.Б. Киктев[2]

[1]Институт вычислительной математики РАН
[2]Гидрометцентр России

Впервые в России была реализована совместная модель атмосферы и океана, ориентированная на воспроизведение совместной циркуляции на временных масштабах до сезона. Для этого полулагранжева модель общей циркуляции атмосферы ПЛАВ (ПолуЛагранжев перенос Абсолютного Вихря), была соединена с σ-моделью общей циркуляции океана, получившей в международной практике название INMSOM (Institute of Numerical Mathematics Sigma Ocean Model). Пространственное разрешение ПЛАВ составляет 1.40625° по долготе, 1.125° по широте, 28 уровней по вертикали. Пространственное разрешение модели океана составляет 1° по долготе, 0,5° по широте и 40 неравномерно расположенных σ-уровней по глубине. Проведены расчеты модели общей циркуляции океана INMSOM за период с 1989 по 2010 гг. по реальному атмосферному воздействию из ERA-Interim и простой методики усвоения температуры поверхности океана для получения начальных данных по Мировому океану для каждого из 4-х сезонов за эти годы. Затем с помощью реализованной совместной модели были выполнены численные эксперименты по ансамблевому моделированию циркуляции атмосферы и океана на срок до 4 месяцев по реальным начальным данным для каждого из 4-х сезонов за 1989 - 2003 гг. для отработки методологии получения долгосрочного прогноза. Результаты показывают перспективность применения совместной модели для оперативного прогнозирования среднесезонных аномалий атмосферной циркуляции. С помощью совместной модели спрогнозировано возникновение и затухание явления Эль-Ниньо 1997-1998 гг.

Введение

Первые эксперименты по воспроизведению сезонных аномалий климата относятся еще к 70м годам прошлого века. Особенный интерес во всем мире эта тема вызвала в последние годы, что связано со значительным прогрессом в развитии моделей общей циркуляции атмосферы и океана в последние десятилетия, а также успехи в области усвоения данных дистанционного зондирования Земли. В первую очередь этот прогресс обусловлен бурным развитием вычислительной техники и технологий. Согласно определению Всемирной метеорологической организации (ВМО) [1], климатическую систему (КС) планеты Земля образуют следующие, взаимодействующие между собой, компоненты: (1) атмосфера - газовая оболочка Земли сложного состава (кислород, азот, углекислый газ, водяной пар, озон и т.д.), воздействующая на перенос к поверхности Земли солнечной радиации, поступающей на ее верхнюю границу, и являющаяся наиболее изменчивой составляющей рассматриваемой системы; (2) океан - главный водный резервуар в системе, состоящий из соленых вод Мирового океана и прилегающих к нему морей, поглощающий основную часть поступающей на его поверхность солнечной радиации и представляющий собой, благодаря высокой теплоемкости воды, мощный аккумулятор энергии; (3) суша - поверхность континентов с ее гидрологической системой (внутренние водоемы, болота и реки), почва (в том числе, с грунтовыми водами) и вечная мерзлота; (4) криосфера - континентальные и морские льды, горные ледники и снежный покров; (5) биота - растительность на суше и в океане, а также живые организмы в воздухе, море и на суше, включая человека. Сложность как самих компонентов КС, так и нелинейного взаимодействия между ними позволяет сделать вывод, что главным средством изучения КС является математическое (численное) моделирование с помощью глобальных климатических моделей. Основу современных моделей климатической системы Земли составляют глобальные модели общей циркуляции атмосферы и

океана. Поэтому совместную модель общей циркуляции атмосферы и океана (или для краткости просто совместную модель) мы будем отождествлять с моделью климатической системы Земли. При этом мы будем полагать, что модель атмосферы включает в себя описания основных процессов взаимодействия с деятельным слоем суши, криосферой и биотой.

Здесь следует отличать задачи прогноза долгопериодных изменений климата и долгосрочного прогноза погоды, одной из которых является тема настоящей работы - воспроизведение сезонных аномалий климата. Согласно оценкам межправительственной группы экспертов по изменению климата (МГЭИК) в последние десятилетия значительный вклад в климатические изменения климата вносит антропогенное воздействие [2]. К нему, прежде всего, относится сжигание ископаемого топлива, приводящее к изменению концентрации углекислого газа в атмосфере, изменение концентрации малых газовых примесей, контролирующих концентрацию озона в атмосфере, вырубка лесов, приводящая к изменению альбедо и процессу опустынивания, и многие другие воздействия. В формирование сезонных аномалий климата наибольший вклад вносит естественная изменчивость климата. К наиболее значимым проявлениям собственной изменчивости климатической системы Земли следует отнести такие явления, как Эль-Ниньо-Южное колебание (ЭНЮК), Северо-Атлантическое колебание, Арктическая осцилляция. Эти явления оказывают существенное влияние на текущее состояние атмосферы и океана и могут изменять свою интенсивность и повторяемость на фоне изменений климата.

С целью координации усилий по созданию систем долгосрочного прогноза погоды в рамках Всемирной программы по изучению климата (ВПИК) ВМО была создана Рабочая группа по прогнозам на масштабах от сезона до нескольких лет (WGSIP). Эта группа организовала международные проекты, посвященные сравнению моделей атмосферы по воспроизведению сезонных климатических аномалий SMIP и SMIP-2.

Во многих центрах атмосферных исследований и метеослужбах разработаны и реализованы системы сезонного прогноза на основе совместных моделей общей циркуляции атмосферы и океана (например, [3, 4]). Под прогнозом здесь, конечно, понимается не детерминистический прогноз состояния атмосферы, а прогноз среднесезонных аномалий атмосферной циркуляции по отношению к среднеклиматическим значениям для данного сезона. Типичное пространственное разрешение компонентов такой системы сезонного прогноза составляет 1,4-1,9 градуса по долготе и широте и 30-40 уровней по вертикали для модели атмосферы и 0,3-1 градуса по долготе и широте и около 30 уровней по вертикали для модели океана. Стандартным является применение ансамблевого метода с 20-60 участниками прогностической реализации.

Следует отметить, что эти совместные модели несколько отличаются от моделей для изучения долгопериодных климатических изменений, поскольку здесь важно конкретное воспроизведение состояния атмосферы и океана для заданного момента времени. Главное же для моделей климата является энергетически сбалансированное воспроизведение климатических характеристик атмосферы и океана [5].

Большой интерес к воспроизведению аномалий климата на масштабах от месяца до сезона проявили страны Юго-Восточной Азии в рамках проекта по разработке системы регионального сезонного прогноза на основе мультимодельных ансамблей. Этот интерес объясняется тем, что по оценкам потенциальной предсказуемости, полученных независимо разными исследователями, этот регион является одним из тех, где возможно получение практически полезных прогнозов.

Наличие оперативной прогностической технологии долгосрочного прогноза является в настоящее время одной из нормативных функций Мировых метеорологических центров Всемирной службы погоды во Всемирной метеорологической организации, один из которых находится в Москве. Именно в связи с этим в Гидрометцентре России была реализована совместная модель атмосферы и океанаориентированная на воспроизведение совместной циркуляции на временных масштабах до сезона. Для этого полулагранжевой модель общей циркуляции атмосферы ПЛАВ (ПолуЛагранжев перенос Абсолютного Вихря), разработанная в Институте вычислительной математики (ИВМ) РАН и Гидрометцентре России, была соединена с сигма-моделью общей циркуляции океана, разработанной в ИВМ РАН, получившей в международной практике название INMSOM (Institute of Numerical Mathematics Sigma Ocean Model). В нижеследующих разделах будет дано описание этой совместной модели и ее компонентов, а также будет уделено внимание вычислительным технологиям, используемым в моделях.

С помощью полулагранжевой модели атмосферы ПЛАВ, применяемой в данной статье, уже исследовалась потенциальная предсказуемость на сезонных временных масштабах [6, 7]. Для этого рассматривалось воспроизведение атмосферной циркуляции на сезонных временных масштабах с предписанной температурой поверхности океана (ТПО). Приводились оценки точности воспроизведения среднесезонной циркуляции на основе данных реанализа-2 NCEP/NCAR согласно протоколу международного эксперимента SMIP-2/HFP [8], в котором вместо предписанной (измеренной) ТПО использовалась простая модель эволюции начального состояния ТПО.

В настоящей статье исследуется воспроизведение среднесезонной циркуляции совместной моделью, которую будем называть, учитывая предназначение этой модели, совместной моделью долгосрочного прогноза (СМДП). Для этого с помощью реализованной СМДП были выполнены численные эксперименты по ансамблевому моделированию циркуляции атмосферы и океана на срок до 4 месяцев по реальным начальным данным для каждого из 4-х сезонов за 1989–2003 гг. для отработки методологии получения долгосрочного прогноза. Для сравнения такие же расчеты были проведены с моделью атмосферы ПЛАВ с простой эволюцией ТПО.

Следует отметить, что численные эксперименты с моделями общей циркуляции атмосферы и океана, а, следовательно, и с совместной моделью, требуют больших вычислительных ресурсов. Эта проблема усугубляется необходимостью применения ансамблевых расчетов.

1. Полулагранжева модель общей циркуляции атмосферы

В ИВМ РАН и Гидрометцентре России была создана вычислительно эффективная полулагранжева глобальная конечно-разностная модель общей циркуляции атмосферы (SL-AV или ПЛАВ). Полулагранжев метод представления адвекции позволяет использовать в модели шаг по времени в несколько раз больший, чем шаг, определяемый условием Куранта. Особенностями блока решения уравнений динамики атмосферы данной модели являются применение конечных разностей четвертого порядка на несмещенной сетке для аппроксимации неадвективных слагаемых уравнений и использование вертикальной компоненты абсолютного вихря и дивергенции в качестве прогностических переменных. Блок решения уравнений динамики атмосферы представлен в работе [9], а численные методы горизонтальной дискретизации модели более подробно описаны в [10].

Модель включает в себя набор параметризаций процессов подсеточного масштаба (коротко- и длинноволновая радиация, глубокая и мелкая конвекция, планетарный пограничный слой, торможение гравитационных волн, параметризация тепло- и влагообмена с подстилающей поверхностью), разработанный в Метео-Франс и метеослужбах консорциума RC-LACE (Limited Area modeling for Central Europe) (http://www.rclace.eu) для французской глобальной оперативной модели ARPEGE и региональной модели международного консорциума ALADIN [11].

Параметризация глубокой конвекции основана на подходе «потока массы» [12], но включает многочисленные усовершенствования [13], в том числе в схеме учитываются нисходящие потоки по краям облака [14], а также перераспределение импульса вследствие конвекции [15].

В отличие от оригинальной параметризации, в модели ПЛАВ в схеме глубокой конвекции применяется замыкание типа Куо в случаях, когда температура на нижнем модельном уровне ниже определенного порогового значения, иначе применяется замыкание на основе конвективной доступной потенциальной энергии (CAPE) [16].

Параметризация мелкой конвекции реализована как расширение вертикальной диффузии и представлена в [17] со следующей модификацией: мелкая конвекция может быть активизирована, только если существует влажная потенциальная неустойчивость. Диагностический расчет влагосодержания крупномасштабных и мелко-конвективных облаков основан на вычислении превышения концентрации водяного пара над значением насыщения.

Метод вычисления радиационных потоков основан на двухпотоковом приближении [18].

Поверхностные альбедо и излучательная способность Земли состоят из фиксированного фонового значения, зависящего только от сезона, и переменной составляющей, зависящей от глубины снежного покрова.

Вертикальный турбулентный перенос импульса, тепла и влаги в приземном слое описан с использованием теории Монина-Обухова для различных типов стратификации. Выше, в плане-

тарном пограничном слое, применяется K-теория с модифицированным числом Ричардсона. Схема турбулентного обмена на поверхности и в свободной атмосфере основана на работах [19] и включает в себя многочисленные усовершенствования.

Модель также включает параметризацию гравитационно-волнового сопротивления орографического происхождения (см. [20] и ссылки в этой работе) и схему параметризации процессов на поверхности суши ISBA.

Версия полулагранжевой модели атмосферы с разрешением 0,9 градуса по долготе, 0,72 градуса по широте успешно прошла оперативные испытания и внедрена в Гидрометцентре России в качестве основного численного метода среднесрочного прогноза.

Для целей настоящей работы используется версия ПЛАВ с пространственным разрешением 1,40625 и 1,125 градусов по долготе и по широте с 28 уровнями по высоте.

2. Сигма модель общей циркуляции океана

В качестве океанического блока СМДП используется разработанная в ИВМ РАН сигма-модель общей циркуляции океана INMSOM. Расчетный комплекс на основе этой модели может применяться для расчетных океанических областей сложных конфигураций, используя различные криволинейные ортогональные системы координат и конечноразностные аппроксимации на неравномерных сетках. Вместе с моделью общей циркуляции океана комплекс включает систему задания атмосферного воздействия на основе предписанных метеоданных, которыми могут служить и результаты расчета модели атмосферы. INMSOM можно применять в качестве океанического блока модели климата Земли, а также для решения научных и практических задач, связанных с расчетом циркуляции как всего Мирового океана, так и его отдельных акваторий. Разработанный программный комплекс может эффективно использоваться как на параллельных вычислительных системах с распределенной и общей памятью с использованием технологий MPI и OpenMP, так и на современных персональных компьютерах.

В основе INMSOM лежит полная система нелинейных уравнений гидродинамики океана в сферических координатах в приближениях гидростатики и Буссинеска. В качестве вертикальной координаты используется безразмерная так называемая сигма-координата, задаваемая как $\sigma=(z-\zeta)/(H-\zeta)$, где z – обычная вертикальная координата; H – полная глубина океана как функция долготы и широты, ζ – отклонение уровня океана от невозмущенной поверхности. Прогностическими переменными модели служат горизонтальные компоненты вектора скорости, потенциальная температура, соленость и отклонение уровня океана от невозмущенной поверхности. Использовано уравнение состояния из [21], учитывающее сжимаемость воды за счет давления столба воды, специально предназначенное для моделей циркуляции океана. В модель инкорпорирована модель динамики и термодинамики морского льда [22].

Особенностью этой модели, отличающей ее от ряда других моделей океана, таких как MOM (Modular Ocean Model) в z-системе координат или POM (Prinston Ocean Model) в сигма-системе координат, является использование в её численной реализации метода расщепления по физическим процессам и пространственным координатам [23–25].

Уравнения гидродинамики океана для компонентов горизонтальной скорости записываются в специальной, симметризованной форме [23, 26]. Она позволяет представить оператор дифференциальной задачи в виде суммы более простых операторов, каждый из которых является неотрицательным в норме, определяемой законом сохранения полной энергии. Это дает возможность расщепить оператор полной задачи на ряд более простых операторов и построить центрально–разностные пространственные аппроксимации так, чтобы закону сохранения энергии, выполняющемуся для исходной дифференциальной задачи, удовлетворяли бы и все расщепленные дискретные задачи.

Данный прием во многом устраняет сложность аппроксимации слагаемых, содержащих градиенты давления, плотности и рельефа дна в уравнениях движения, записанных в сигма-системе координат. Метод расщепления позволяет эффективно реализовывать неявные схемы интегрирования по времени для уравнений переноса-диффузии субстанций.

С целью более адекватного описания процессов динамики океана оператор боковой диффузии второго порядка для тепла и соли представлен в форме, в точности эквивалентной горизонтальной диффузии в обычной z-системе координат. Дополнительно, может использоваться опе-

ратор изопикнической диффузии, действующий вдоль поверхностей равной плотности (потенциальной). Для этого используется процедура диффузии, выписанная не методом конечных объемов, а диффузия, выписанная по принципу, схожему с полулагранжевым подходом, когда в процессе диффузионного перемешивания могут использоваться не только соседние узловые точки сетки, но и более удаленные по толще океана, согласно расположению изопикнических поверхностей.

Разностные аппроксимации по пространству строятся на смещенной сетке типа "C". Использование сетки "C" позволяет более адекватно аппроксимировать расчетную область в узких проливах, использовать условие скольжения на боковых границах, уменьшить коэффициент горизонтальной диффузии.

Океанический блок для СМДП практически тот же, что и для модели климатической системы ИВМ РАН [5]. Пространственное разрешение составляет 1° по долготе, 0.5° по широте и 40 неравномерно расположенных сигма-уровней по глубине. Чтобы устранить проблему Северного географического полюса, связанную со схождением меридианов в географической системе координат, модель глобального океана была реализована в криволинейной ортогональной системе координат. Последняя получена с помощью конформного преобразования исходной сферической системы координат с сохранением положения географического экватора. При этом полюсные точки новой системы координат расположены на материках за пределами расчетной области, так что один полюс располагается на Таймыре, а второй – в Антарктиде симметрично первому относительно экватора, таким образом, чтобы экватор в модельной системе координат совпадал с географическим. Топография дна, используемая в модели, была получена из данных ETOPO2, которые представляют собой топографию всей поверхности Земли с 2-х минутным разрешением.

Коэффициенты вертикальной вязкости и диффузии выбирались согласно параметризации Пакановского и Филандера как функция числа Ричардсона.

3. Объединение моделей общей циркуляции атмосферы ПЛАВ и океана INMSOM в совместную модель

Обмен информацией между моделями атмосферы и океана происходит каждый шаг по времени, определяемый атмосферной моделью, равный 36 мин. Этот промежуток времени служит и шагом по времени модели океана, хотя сама по себе модель океана может считаться и с большим шагом по времени. В процессе обмена в модель океана, рассчитываемые моделью атмосферы с шагом 36 мин на поверхности океана потоки явного и скрытого тепла, импульса, суммарные (приходящие и уходящие) потоки длинноволновой и коротковолновой радиации.

Пересчет полей с атмосферной на океаническую пространственную сетку осуществляется с помощью билинейной интерполяции с учетом сферичности. Для пересчета ТПО, сгенерированной моделью океана, на атмосферную сетку используется процедура пространственного весового осреднения. Она заключается в том, что ТПО во всех узлах океанической сетки, попадающих в ячейку атмосферной сетки, бралась с весом, пропорциональным площади пересечения соответствующей океанической ячейки с ячейкой атмосферной сетки. Предполагается, что при стыковке моделей атмосферы и океана не используется коррекция потоков тепла и импульса. При расчете потоков на поверхности океана в модели атмосферы температурой поверхности океана считается температура самого верхнего расчетного слоя океанической модели.

4. Программная реализация технологии расчетов по совместной модели

Прогноз среднесезонных аномалий температуры и осадков в силу неустойчивости атмосферы является, по сути, попыткой выделить слабый сигнал на уровне сильного шума. Поэтому стандартным является расчет ансамбля сезонных прогнозов с возмущенных начальных данных. Характерное количество участников ансамбля составляет 10-40. Хотя по отдельности программные комплексы моделей атмосферы и океана реализованы как с применением MPI, так и OpenMP [27], для расчета сезонных прогнозов по совместной модели применяется лишь

OpenMP. Таким образом, в зависимости от используемой вычислительной системы, один участник ансамбля может эффективно использовать 8-16 процессорных ядер. Общее требуемое число процессорных ядер для расчета одного ансамблевого прогноза на один сезон в зависимости от количества участников ансамбля составляет 80-640.

По сравнению с представленной в [27] параллельной реализацией полулагранжевой модели атмосферы, примененная в данных экспериментах версия модели имела ряд усовершенствований программного комплекса, направленных на повышение параллельного ускорения при использовании OpenMP. В частности, была выполнена локализация обращений к памяти путем введения дополнительных локальных рабочих массивов (ранее для этих целей в различных частях программного комплекса использовалась одна и та же область памяти).

Практические расчеты сезонных прогнозов выполнялись на вычислительной системе SGI Altix 4700, установленной в ГВЦ Росгидромета. Для запуска ансамбля прогнозов используется скрипт, который запускает расчет всех участников ансамбля через систему очередей PBSPro. Расчет сезонного прогноза по каждому участнику ансамбля занимал 8 часов на 12 ядрах (10 часов на 8 ядрах). Таким образом, расчет одного ансамблевого сезонного прогноза с помощью совместной модели требует минимум 800 процессоро-часов. Как описывается далее, практическим сезонным прогнозам с помощью совместной модели должен предшествовать расчет модельного климата (за 15-30 лет) для каждого из сезонов. Эти расчеты необходимо повторять при каждом изменении в совместной модели.

5. Численные эксперименты с совместной моделью

Предварительные расчеты сезонных прогнозов с помощью совместной модели показали, что ошибки среднесезонных аномалий метеорологических величин больше, чем в модели атмосферы с простой эволюцией температуры поверхности океана. Большая ошибка вызвана тем, что совместная модель для каждого из сезонов стартует не с согласованных между океаническим и атмосферным блоками состояний, а с начальных данных, полученных для каждого блока по отдельности. Это приводит к тому, что к сезонному ходу добавляется модельный тренд, вызываемый процессами подстройки друг к другу атмосферного и океанического блоков. Следует отметить, что этот тренд присущ всякой современной совместной модели климата: при длительном расчете климатические модели выходят на стационарное состояние, которое, однако, довольно заметно отличается от реального. В нашем случае, для того чтобы его убрать, необходимо выводить на реальные начальные условия непосредственно всю совместную модель, что, однако, невозможно при существующем уровне совместных моделей, в том числе и зарубежных. Поэтому была предложена методика исключения этого тренда для расчета сезонных аномалий.

Суть этой методики заключается в следующем. Необходимо провести ансамблевые расчеты СМДП для каждого из сезонов за несколько десятков лет. Затем, путем осреднения за все годы, вычислить модельный климатический квази-сезонный ход для каждого из сезонов для всех прогнозируемых параметров СМДП. Квази-сезонный ход будет содержать в себе как сам сезонный ход, так и модельный тренд. Затем, вычитая этот климатический квази-сезонный ход из прогнозируемых величин, можно получить прогностические аномалии нужных параметров, которые и будут служить прогнозом отклонений уже от климатического сезонного хода.

Таким образом, главным фактором предложенной методики служит ансамблевость и массовость расчетов. Для их проведения необходимо:

1. Подготовить реальный атмосферный форсинг для модели океана за несколько десятков лет.
2. С использованием этого форсинга и простой методики усвоения наблюдаемой ТПО рассчитать начальные состояния (НС) модели океана для начала каждого из сезонов.
3. Рассчитать ансамблевые состояния на эти же сроки модели атмосферы.
4. Просчитать СМДП для каждого из сезонов в выбранном интервале времени.
5. Путем осреднения за все годы, вычислить модельный климатический квази-сезонный ход для каждого из сезонов для всех прогнозируемых параметров СМДП, вычитая который из прогнозируемых величин получить прогностические аномалии нужных параметров, которые и будут служить прогнозом отклонений уже от климатического сезонного хода.

6. Обработать полученные прогностические сезонные аномалии.

Ниже приводятся главные этапы применения такого подхода.

5.1 Расчет модели океана для получения начальных данных для каждого из сезонов 1989-2010 гг.

Для подготовки метеоданных над океаном для расчета атмосферного воздействия в автономной модели океана использовался массив данных ERA-Interim, ECMWF доступный с сайта http://data-portal.ecmwf.int/data/d/interim_daily/.

Данные, находящиеся в этой базе имеют пространственное разрешение 1.5°x1.5° по широте и долготе. По времени же разрешение различное. Наиболее высокое разрешение в 6 часов из нужных данных для расчета атмосферного воздействия доступно только для температуры поверхности океана, температуры точки росы, давлению на уровне моря, скорости ветра на 10 м. Поэтому для данных по приходящим коротковолновой и длинноволновой радиации использовались данные с 12-часовым разрешением. Эти данные в определении ERA-Interim носят названия Step 0 (6 ч разрешение) и Step 12 (12 ч разрешение). Данные копировались для каждого года с 1989 по 2010гг. отдельно.

Расчеты начальных состояний Мирового океана проводились по методике, отработанной в ходе проведения предварительных экспериментов. Сначала INMSOM была «разогнана» на 60 лет с климатологии Левитуса с климатическим атмосферным форсингом из массива данных CORE для так называемого нормализованного годового хода. Эта база данных метеоинформации из проекта CORE (Forcing for Common Ocean-ice Reference Experiments) (http://data1.gfdl.noaa.gov/nomads/forms/mom4/CORE.html) предназначена именно для экспериментов с моделями океана, включающими в себя модуль параметризации морского льда. Пространственное разрешение данных составляет 1.875° по долготе и неравномерное по широте: от 1,9047354° на экваторе до 1,88878° вблизи полюсов. Данные собраны за период 1958-2006 гг. и включают в себя следующие величины: температура воздуха на высоте 10 м, влажность воздуха на высоте 10 м; скорость ветра на высоте 10 м и давление на уровне моря - временное разрешение этих параметров составляет 6 часов; падающая радиация (длинно- и коротковолновая) - временное разрешение 1 сутки; осадки - временное разрешение 1 месяц. Данные CORE включают в себя также сток рек, но в связи с тем, что он среднегодовой, в модели используются данные по стоку рек с разрешением 1 месяц из проекта OMIP (Ocean Model Intercomparison Project) (www.mpimet.mpg.de/Depts/Klima/natcli/omip.html). При этом данные CORE включают т.н. нормализованный год, который удобен для «разгона» модели до квазиклиматического состояния.

После расчета квазистационарного состояния циркуляции Мирового океана проводился сквозной счет модели океана с 1989 по 2010 гг. с сохранением необходимых для старта модели океана данных для каждого сезона для всех лет интегрирования модели. При этом использовался атмосферный форсинг, рассчитываемый по вышеописанным данным из ERA-Interim, ECMWF, с осуществлением жесткой привязки с параметром релаксации 1/(6 час) к наблюдаемой ТПО, имеющейся с шагом в 6 часов. Таким образом, осуществляется простое усвоение наблюдаемой ТПО в модели океана. Расчеты модели общей циркуляции океана INMSOM за период с 1989 по 2010 гг. были проведены для получения начальных данных по Мировому океану для каждого из 4-х сезонов за эти годы. Для каждого года из интервала 1989 по 2010 были получены начальные данные для модели океана на сроки 26 января (для весеннего сезона), 26 апреля (для летнего сезона), 26 июля (для осеннего сезона) и 26 октября (для зимнего сезона).

Эти начальные данные для модели океана использовались в следующих экспериментах с СМДП.

5.2 Расчеты сезонных прогнозов с помощью совместной модели и модели ПЛАВ с простой моделью эволюции ТПО для интервала 1989-2003гг.

Полученные начальные данные для модели океана использовались в следующих экспериментах с совместной моделью. В этих расчетах с помощью реализованной СМДП были выпол-

нены численные эксперименты по ансамблевому моделированию циркуляции атмосферы и океана на срок в 4 месяца по реальным начальным данным для 26-30 июля (прогноз на осень с заблаговременностью в месяц), 26-30 октября, 26-30 января и 26-30 апреля за интервал 1989-2003 гг.

С совместной моделью атмосферы и океана были выполнены расчеты сезонных прогнозов для всех четырех сезонов за период 1989-2003 гг. Все эксперименты выполнялись по ансамблевой технологии, в каждом ансамбле из 10 участников возмущались лишь начальные данные для модели атмосферы. Аналогичные эксперименты были выполнены с моделью ПЛАВ, в которой ТПО задавалась как сумма климатического сезонного хода ТПО и ее аномалии на день начала прогноза. Такой прогноз поведения ТПО зачастую называют также инерционным прогнозом.

Как было сказано выше, для анализа качества прогноза применялась следующая методика. Сначала путем осреднения за все годы вычислялся модельный климатический квази-сезонный ход для каждого из сезонов для всех прогнозируемых параметров СМДП. Затем эти климатические эволюции сезонных изменений вычитались из прогнозируемых величин для получения прогностических аномалий нужных параметров. Очевидно, что с помощью этой методологии устраняется не только модельный тренд, но и сам сезонный ход. Тем не менее, получаемые аномалии вполне могут служить прогнозом отклонений от климатического сезонного хода. Таким образом, для получения конкретного прогнозируемого с помощью СМДП сезонного хода какого либо параметра, нужно добавить наблюдаемые средне климатические сезонные изменения этого параметра к вычисленным его аномалиям. Были выполнены расчеты средней и среднеквадратической ошибок полей высоты поверхности 500 гПа (H500), давления на уровне моря, температуры на поверхности 850 гПа (T850), осредненных за период со второго по четвертый месяц прогноза. Оценивались как полные среднесезонные поля, так и поля, полученные прибавлением модельной аномалии к среднесезонным фактическим полям, осредненным за период 1989-2003 гг.

Результаты, осредненные за три сезона, приведены в таблице 1. По техническим причинам, оценки ошибок полей для зимнего сезона пока не выполнены. Мы видим, что ошибки «полных» полей в совместной модели несколько больше, чем для модели атмосферы с простой эволюцией ТПО, в основном за счет роста средних ошибок. Если эти ошибки убрать по описанной выше методике (столбцы с заголовком «аном»), то оказывается, что совместная модель имеет меньшие ошибки. Особенно заметно это уменьшение ошибок в тропиках. Это подтверждает тезис о том, что прогнозируемость в тропиках значительно выше, чем в средних широтах.

Таблица 1. Среднеквадратические (RMSE) и средние ошибки полей высоты поверхности 500 гПа (H500) [м], давления на уровне моря (MSLP) [мбар], температуры на поверхности 850 гПа (T850)[˚C], осредненные по всем численным экспериментам за три сезона (весна, лето, осень) и за 1989-2003 год для модели атмосферы ПЛАВ с простой эволюцией ТПО и СМДП, для прогностических полей и для полей модельных аномалий, добавленных к климату.

	ПЛАВ RMSE	СМДП RMSE	ПЛАВ средняя	СМДП Средняя	ПЛАВ RMSE аном	СМДП RMSE аном
H500						
20°-90° с.ш.	49.4	48.2	-20.8	-18.5	10.61	9.29
Тропики	10.8	14.1	-4.4	5.83	5.35	3.69
90°-20° ю.ш.	41.5	45.2	5.0	20.3	9.71	7.34
MSLP						
20°-90° с.ш.	3.12	3.25	-0.8	-1.51	0.908	0.756
Тропики	1.72	1.84	0.55	0.91	0.575	0.294
90°-20° ю.ш.	6.5	6.59	0.5	0.07	0.819	0.633
T850						
20°-90° с.ш.	2.43	2.63	-0.92	-0.6	0.534	0.479
Тропики	1.75	1.76	-0.3	-0.1	0.326	0.21
90°-20° ю.ш.	1.84	1.84	-0.22	0.7	0.365	0.255

5.3 Сравнительный анализ долгосрочного прогноза приповерхностной температуры в период Эль-Ниньо 1997-1998 гг.

Многими исследованиями подтверждается, что прогнозируемость в тропиках значительно выше, чем в средних широтах. Около трети населения мира проживает в странах тропического пояса. Многие из этих стран являются развивающимися, экономика которых во многом зависит от сельского хозяйства и рыболовства. Так, большой интерес к воспроизведению аномалий климата на масштабах от месяца до сезона проявили страны Юго-Восточной Азии в рамках проекта по разработке системы регионального сезонного прогноза на основе мультимодельных ансамблей. Этот интерес объясняется тем, что по оценкам потенциальной предсказуемости, полученных независимо разными исследователями, этот регион является одним из тех, где возможно получение практически полезных прогнозов.

Межгодовая изменчивости ТПО в приэкваториальном Тихом океане и связанные с ней явления Эль-Ниньо и Ла-Ниньа является одним из самых сильных сигналов естественной климатической изменчивости. Поэтому выяснению механизмов этой изменчивости и ее численному моделированию посвящено множество работ.

Современные климатические модели ОЦАиО способны воспроизводить многие черты наблюдаемой межгодовой изменчивости в тропиках Тихого океана [28].

В связи с важностью явления ЭНЮК, имеющего глобальный отклик практически во всех параметрах атмосферной циркуляции, интересно посмотреть качество долгосрочного прогноза этого явления, получаемого в наших экспериментах.

Эль-Ниньо 1997-1998 гг. было настолько сильным, что привлекло внимание мировой общественности и прессы. Тогда же распространились теории о связи Южной осцилляции с глобальными изменениями климата. Поскольку Эль-Ниньо 1997-1998 гг. в ходит в расчетный период настоящего исследования, были построены карты аномалий приповерхностной температуры (ПТ) - она же температура на уровне 2м, которая над океанами совпадает с ТПО. Эти аномалии были осреднены по сезонам, как того требует практика сезонных прогнозов. Это было сделано для сезонных прогнозов, выполненных с помощью СМДП, а также модели атмосферы ПЛАВ с простой схемой эволюции ТПО. Построенные карты ПТ сравнивались с реальными данными наблюдений, доступными с сервера NASA (http://data.giss.nasa.gov/gistemp/).

На рисунке 1 показана средняя за лето 1997 г. наблюденная аномалия ПТ, в отклонениях от среднего за 1989-2003 гг. для этих же сезонов согласно данным NASA, а так же показаны аномалия ПТ за этот же период полученная по результатам прогностического расчета СМДП и модели ПЛАВ с предписанной ТПО. Именно в этот сезон началось интенсивное Эль-Ниньо 1997-1998 гг. Эти аномалии вычислялись согласно вышеописанной методики. Следует отметить, что в модели ПЛАВ ПТ менялась только над сушей, а над океаном она задавалась как сумма климатического сезонного хода ТПО и ее аномалии ТПО на день начала прогноза. Далее в течение прогноза используется временная эволюция климата с постепенно ослабляющейся изначальной аномалий ТПО по экспоненциальному закону.

Сравнивая карты рис. 1, легко заметить, что аномалия ПТ в совместной модели более соответствует реальной, чем в модели ПЛАВ с простой эволюцией ТПО. Особенно это заметно в области тропиков Тихого океана. Следует обратить внимание, что аномалия ПТ в СМДП имеет большее сходство с наблюдениями не только над океаном, но и над сушей. Более того, главные ошибки прогноза и в СМДП и в модели ПЛАВ с простой эволюцией ТПО наблюдаются над территорией России. По-видимому, это обусловлено недостатками в блоке почвы в модели ПЛАВ, которые не проявляются в краткосрочных прогнозах погоды, но заметны в более протяженных расчетах. Требует, однако, объяснение тот факт, что над территорией Северной Америки прогнозируемая аномалия ПТ получается с меньшей ошибкой, чем над территорией России.

6. Заключение

Мы реализовали впервые в России совместную модель атмосферы и океана, ориентированную на прогноз среднесезонных аномалий атмосферной циркуляции и разработали технологию ее применения на параллельных вычислительных системах. В совместной модели долгосрочного прогноза происходит некоторое уменьшение ошибок воспроизведения среднесезонных аномалий полей H500, давления на уровне моря и температуры на поверхности 850 гПа, и в некоторых частях также температуры на уровне 2м, по сравнению с моделью атмосферы ПЛАВ с простой моделью эволюции ТПО.

СМДП существенно лучше дает сезонный прогноз аномалии ТПО, особенно заметный в периоды роста и падения интенсивности Эль-Ниньо. По-видимому, это связано с тем, что совместная модель воспроизводит не только положение траектории в фазовом пространстве своих характеристик, но и ее тенденцию траектории. Таким образом, при определенной доработке совместная модель может служить как инструмент прогноза крупномасштабных аномалий ТПО, особенно в тропиках.

Главная ошибка в прогнозе сезонных аномалий метеорологических полей наблюдается над сушей Евразии. По-видимому, в модели ПЛАВ необходимо улучшать параметризацию льда в почве, особенно в зонах вечной мерзлоты. Тем не менее, даже это улучшение не будет гарантировать достаточной оправдываемости долгосрочного прогноза аномалий в силу характера изменчивости атмосферы [290] в средних широтах, поскольку атмосферный отклик на аномалию ТПО не полностью определяет формирование изменчивости атмосферной циркуляции в средних широтах.

Таким образом, для уменьшения ошибок долгосрочного прогноза аномалий необходимо усовершенствовать схему параметризации процессов в почве в модели ПЛАВ. Для этого необходимо провести серию экспериментов с ПЛАВ с простой моделью эволюции ТПО для всех сезонов. Затем будут повторены ансамблевые расчеты с СМДП в период 1989-2010гг. Для улучшения воспроизведения поля ТПО моделью океана необходимо повышение ее пространственного разрешения. Кроме того, 10 участников ансамбля, использованных в настоящей работе, по современным меркам недостаточно. Новая версия технологии расчетов по СМДП потребует как минимум несколько тысяч процессорных ядер вместо нынешних 400 ядер.

Литература

1. Физические основы теории климата и его моделирования / Пер. с англ. под ред. А.С. Монина. Л.: Гидрометеоиздат, 1977.
2. IPCC Fourth Assessment Report // Intergovernmental Panel on Climate Change. Solomon S.D., Qin D., Manning M., Chen Z., Marquis M., Averyt K.B., Tignor M., Miller H.L. (eds.). Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA. 2007. 996 p.
3. Anderson D., Stockdale T., Balmaseda M., Ferranti L., Vitart F., Molteni F., Doblas-Reyes F., Mogenson K., Vidard A. Development of the ECMWF seasonal forecast System 3/ ECMWF Tech. Memo. 503. ECMWF, Reading, UK – 2007, 56p. (http://www.ecmwf.int/publications/library/ecpublications/_pdf/tm/501-600/tm503.pdf)
4. Шнееров Б.Е., Мелешко В.П., Соколов А.П. и др. Глобальная модель общей циркуляции атмосферы и верхнего слоя океана ГГО // Труды ГГО. 1997. вып. 544. С. 3-123.
5. Е.М. Володин, Н.А. Дианский, А.В. Гусев. Воспроизведение современного климата с помощью совместной модели общей циркуляции атмосферы и океана INMCM 4.0 // Известия РАН. Физика атмосферы и океана, 2010, Т. 46, № 4, С. 1–17
6. Тросников И.В, Казначеева В.Д., Киктев Д.Б., Толстых М.А.. Оценка потенциальной предсказуемости метеорологических величин при динамическом сезонном моделировании атмосферной циркуляции на основе полулагранжевой модели SL-AV // Метеорология и Гидрология. 2005. N12. С. 5-17.
7. Д.Б.Киктев, И.В.Тросников, М.А.Толстых, Р.Б.Зарипов. Оценки успешности прогнозов сезонных аномалий метеорологических полей для модели SL-AV в эксперименте SMIP-2// Метеорология и гидрология. 2006. N6. С. 16-26.

8. М.А.Толстых, Д.Б.Киктев, Р.Б.Зарипов, М.Ю.Зайченко, В.В.Шашкин. Воспроизведение сезонной атмосферной циркуляции модифицированной полулагранжевой модели атмосферы// Изв. РАН, сер. ФАиО. 2010. Т.46, N2. С. 149-160.
9. Толстых М.А. Полулагранжева модель атмосферы с высоким разрешением для численного прогноза погоды// Метеорология и гидрология. 2001. N4. С. 5-16.
10. Tolstykh M. Vorticity-divergence semi-Lagrangian shallow-water model on the sphere based on compact finite differences // J. Comput. Phys. 2002. V. 179. P. 180-200.
11. Geleyn J.-F., Bazile E., Bougeault P., Deque M., Ivanovici V., Joly A., Labbe L., Piedelievre J.-P., Piriou J.-M., Royer J.-F. Atmospheric parameterization schemes in Meteo-France's ARPEGE N.W.P. model // Procs. of ECMWF Seminar on Parameterization of subgrid-scale physical processes 5-9 September 1994. - Reading, UK: ECMWF. 1995. P. 385-402.
12. Bougeault P. A simple parameterization of the large-scale effects of cumulus convection // Mon. Weather Rev. 1985. V. 113. P. 2108-2121.
13. Gerard L., Geleyn J.-F. Evolution of a subgrid deep convection parametrization in a limited-area model with increasing resolution //Quart. J. Roy. Meteor. Soc. 2005. V. 131. P. 2293-2312.
14. Ducrocq V. and Bougeault P. Simulation of an observed squall line with a meso-beta-scale hydrostatic model // Mon. Weather Rev. 1995. V. 123. P. 380-399.
15. Gregory D., Kershaw R., and Inness P.M. Parameterization of momentum transport by convection - II: Tests in single-column and general circulation models // Quart. J. Roy. Meteor. Soc. 1997. V. 123. P. 1153-1183.
16. Tolstykh M. The use of combined closure in convection parameterization scheme /Research activities in atmospheric and oceanic modeling (Ed. J. Côté) WMO/TD 1161. 2003. Rep. 33. P.04-25 – 04-26.
17. Geleyn J.-F. Use of a modified Richardson number for parameterizing the effect of shallow convection // J. Met. Soc. Japan. 1987. Special 1986 NWP Symposium Issue. P. 141-149.
18. Ritter B. and Geleyn J.-F. A comprehensive radiation scheme of numerical weather prediction with potential application to climate simulations //Mon. Weather Rev. 1992. V. 120. P. 303-325.
19. Louis J.-F., Tiedtke M., and Geleyn J.-F. A short history of the operational PBL parameterization at ECMWF // Procs. of ECMWF Workshop on planetary boundary layer parameterization 25-27 November 1981. - Reading, UK: ECMWF. 1982. P. 59-80.
20. Catry, B., J.F. Geleyn, F. Bouyssel, J. Cedilnik, R. Brozkova, M. Derkova, R. Mladek : A new sub-grid scale lift formulation in a mountain drag parametarisation scheme Meteorologische Zeitschrift. 2008 **17**, Issue 2, 193-208.
21. Bryden D., San S., Bleck R. A new approximation of the equation of state for seawater, suitable for numerical ocean models// J. Geoph. Res. 1999. V. 104, No. C1. P. 1537–1540.
22. Яковлев Н.Г. Восстановление крупномасштабного состояния вод и морского льда Северного Ледовитого океана в 1948-2002 гг. Часть 1: Численная модель и среднее состояние // Известия РАН, ФАО, 2009. Т. 45, № 3. С.1-16.
23. Марчук Г.И. Методы расщепления. М., Наука, 1988. 264 с.
24. Залесный В.Б. Моделирование крупномасштабных движений в Мировом океане. М.: Отдел вычислит. мат. АН СССР, 1984. 158 с.
25. Дианский Н. А., Багно А. В., Залесный В. Б. Сигма-модель глобальной циркуляции океана и ее чувствительность к вариациям напряжения трения ветра // Изв. РАН. Физика атмосферы и океана. 2002. Т. 38. № 4. С. 537–556.
26. Дымников В.П. Вычислительные методы в геофизической гидродинамике. М.: Отдел. вычислит. мат. АН СССР, 1984. 148 с.
27. Е.М.Володин, М.А.Толстых. Параллельные вычисления в задачах моделирования климата и прогноза погоды// Вычислительная математика и программирование, 2007, Т.8, с. 123-129
28. Achuta Rao K., Sperber K.R. and the CMIP modeling groups. El-Nino Southern oscillation in coupled models. PCMDI report N61, 2000, p.1-50.
29. Kirtman B., Pirani A. WCRP Position Paper on Seasonal Prediction Report from the First WCRP Seasonal Prediction Workshop, 4-7 June 2007, Barcelona, Spain. WCRP Informal Report No. 3/2008 ICPO Publication No. 127

10.

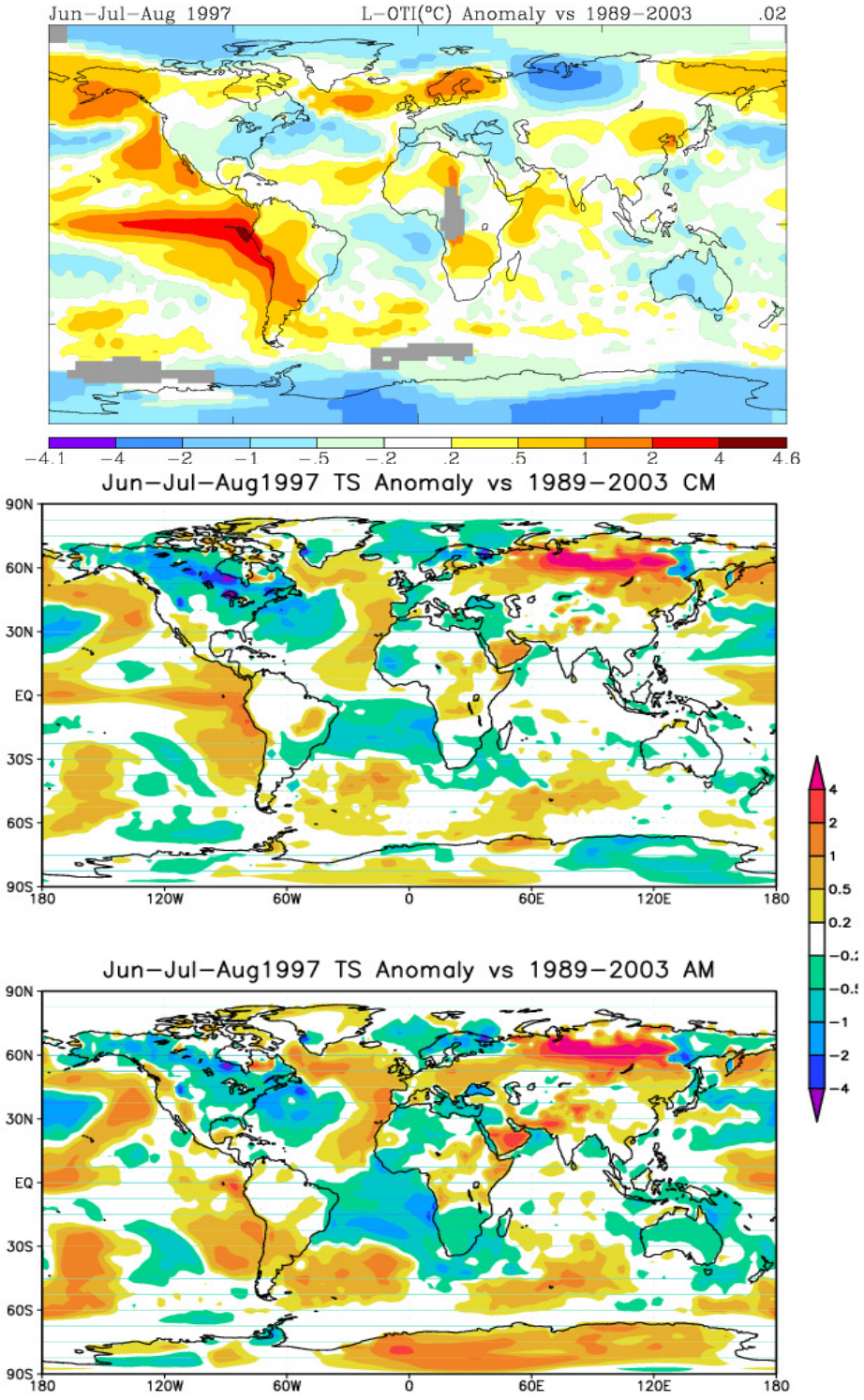


Рисунок 1. Вверху - средняя за лето 1997 г. наблюденная аномалия ПТ, в отклонениях от среднего за 1989-2003 гг. для этих же сезонов согласно данным NASA (http://data.giss.nasa.gov/gistemp/). Средняя за лето 1997 г. рассчитанная аномалия ПТ, в отклонениях от среднего за 1989-2003 гг. для этих же сезонов согласно СМДП (в середине) и модели ПЛАВ с простой эволюцией ТПО (внизу).

Применение высокопроизводительных вычислений для моделирования гидродинамических течений в сильно неоднородных гравитационных полях*

С.А. Хоперсков[1], А.В. Хоперсков[1], М.А. Еремин[1], А.В. Засов[2], Н.В. Тюрина[2]

Волгоградский государственный университет[1],
Государственный астрономический институт им. П.К.Штернберга, МГУ[2]

Была построена трехмерная численная схема TVD типа MUSCL для решения уравнений газовой динамики с учетом тепловых процессов и самогравитации в декартовой и цилиндрической координатных системах. Численный алгоритм был адаптирован для расчетов на компьютерах с массивно-параллельной архитектурой. Схема применялась для моделирования физических процессов, протекающих в галактических газовых дисках. Детально изучен механизм гофрировочной неустойчивости спиральной структуры в газовом диске, предложен новый, гидродинамический, механизм образования кольцеобразных галактических структур, проведено моделирование и сравнение полученных результатов с наблюдаемой облачной структурой Млечного Пути.

1. Введение

Проблема объяснения различных аспектов феномена природы спиральной структуры дисковых галактик сохраняет свою актуальность несмотря на 50-летнюю историю исследований. Для теоретического изучения широко применяются различные методики компьютерного моделирования. Однако, заметный прогресс был достигнут с привлечением высокопроизводительных суперкомпьютеров и использованием параллельных технологий.

Физические параметры, характеризующие свойства галактического газа, лежат в очень широких пределах, которые не имеют аналогов при решении типичных задач, возникающих в технике или «земной» физике. Межзвездная среда, являясь многофазной, имеет температуру от 3 до 10^5 K, огромные перепады плотности. Типичные значения скорости составляют 100–300 км/с, что дают числа Маха $\sim 10 \div 100$. Наблюдается иерархия пространственных и временных масштабов, которые существенно определяют свойства галактической системы. Например, газовый диск типичной галактики простирается до 20–40 кпк по радиусу. С другой стороны, мелкомасштабные структуры, формирующиеся в области спиральных галактических ударных волн составляют ~ 10 пк, и для их качественного разрешения необходимы масштабы до 1 пк. Характерные времена процессов, определяющих динамику газового диска в целом достигают 1 млрд лет, а необходимость одновременного моделирования быстропротекающих тепловых явлений, нелинейных этапов развития физических неустойчивостей (гравитационных и сдвиговых) требует рассмотрения динамики на временах до 10^4 лет.

Отличительной особенностью моделирования динамики межзвездной среды в пределах галактики является богатый спектр физических эффектов и процессов, существенно влияющих на динамику газа. Отметим целый ряд газодинамических и плазменных неустойчивостей различной природы: гравитационная, тепловая, конвективная, магнитная. Это вызывает необходимость дополнительного учета в уравнениях гидродинамики тепловых процессов, химических превращений, самогравитации, что делает задачу очень жесткой. Отмеченные выше временные характеристики задачи требуют большого числа временных шагов интегрирования $> 10^5$. Например, для самосогласованного описания динамики звездно-газового

*Работа выполнена при поддержке грантов РФФИ 10-07-97017-р_поволжье_а, гранта ВолГУ № 70-2011–а/ВолГУ, ФЦП Росбразование госконтракт П1248 (ФЦП НК)

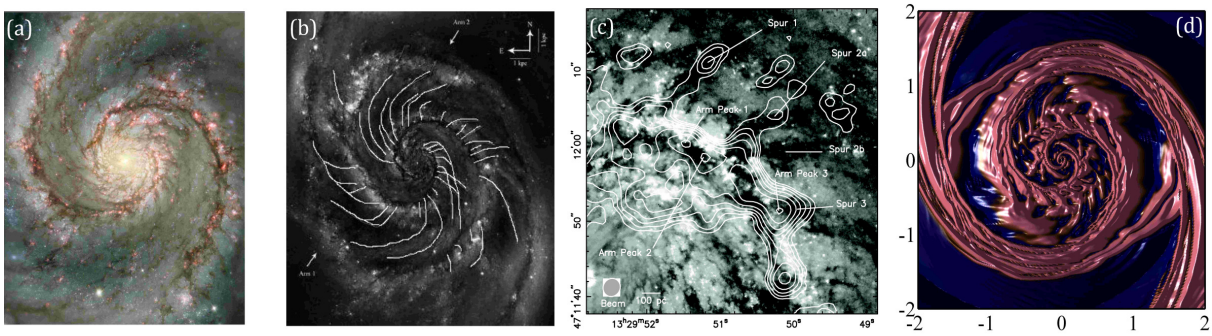


Рис. 1. a) Оптическое изображение галактики M51. b) Изображение M51 с выделением шпуров. c) Отдельные участки M51 со шпурами и оперением [2]. d) Пример результата компьютерного моделирования шпуров в галактических дисках

диска на уровне современной мировой науки необходимо для моделирования звездной подсистемы более 10^7 гравитационно взаимодействующих частиц, а число ячеек сетки для газовой компоненты должно превышать 10^8, что требует значительных ресурсов памяти и машинного времени. Вне использования параллельных технологий такого рода задачи не подлежат решению.

Выделим основные направления работы по исследованию динамики галактик с использованием параллельных технологий:

- Создание пакета программ для численного моделирования газовых течений с учетом тепловых процессов, внешних гравитационных сил и самогравитации.
- Адаптация численных алгоритмов для вычислений на компьютерах с массивно-параллельной архитектурой.
- Исследование развития гофрировочной неустойчивости ударных волн в моделях галактического газового диска с учетом спирального потенциала звезд. Моделирование кольцевых структур в дисковых галактиках.
- Изучение возможности формирования облачной структуры в модели газовой компоненты Млечного Пути. Сравнение результатов моделирования с наблюдениями.

Характерной мелкомасштабной особенностью большинства галактик с глобальным правильным спиральным узором являются шпуры (spurs), отходящие от спирального рукава почти перпендикулярно или пересекающие его. Типичная длина этих образований лежит в пределах $\sim 100 - 1000$ пк (рис. 1a,b,c). Такие структуры часто называют feathers в смысле «оперение», «выступ», «гребень» или spurs («шпора», «отросток»). Обычно под шпурами (spurs) понимают более мощные и развитые структуры, непосредственно примыкающие к основной части спирального рукава (положению ударной волны), под оперением (feathers) — более слабые и протяженные структуры, отчасти являющиеся продолжением шпуров. Наиболее важным наблюдаемым проявлением шпуров является повышенная плотность газа и интенсивные процессы звездообразования [1].

Большинство ближайших спиральных галактик демонстрирует наличие развитой системы шпуров, например, NGC 628, NGC 1232, NGC 3031, NGC 3184, NGC 4321, NGC 5194, NGC 5236, NGC 5457, NGC 4725, NGC 7424, IC0342. Изучение шпуров существенно осложняется сильной неоднородностью пыли в области спиральных рукавов, которая одновременно выступает в качестве одного из индикаторов самих шпур. Но в целом, нарушение гладкости спирального узора следует считать характерной особенностью спиральных галактик, хотя количественные различия существенны и трудно ожидать, что все они могут быть объяснены одним универсальным физическим механизмом.

2. Модель и численный метод

Динамика газового диска в общем случае описывается трехмерной системой уравнений гидродинамики:

$$\frac{\partial \rho}{\partial t} + \nabla(\rho \mathbf{u}) = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla(\rho \mathbf{u} \times \mathbf{u}) = -\nabla p - \rho \frac{\partial \Psi}{\partial r}, \tag{2}$$

$$\frac{\partial}{\partial t}\left(\rho\left[\frac{\mathbf{u}^2}{2} + \epsilon\right]\right) + \nabla\left(\rho\left[\frac{\mathbf{u}^2}{2} + \varepsilon\right]\right) = Q^+ - Q^-\,, \tag{3}$$

$\epsilon = \dfrac{p}{(\gamma-1)\rho}$ – удельная энергия газа, $\rho-$ объемная плотность газа, $p-$ давление, $\mathbf{u}-$ вектор скорости, Q^+ и Q^- - источниковые слагаемые, например, описывают процессы излучения в газе. $\triangle\Psi = 4\pi G\rho$ - уравнение Пуассона.

Для решения системы трехмерных уравнений газодинамики с учетом самогравитации, нагрева и охлаждения (1)-(3) была реализована явная трехмерная TVD (Total Variation Diminishing) схема типа MUSCL (Monotone Upstream-centered Schemes for Conservation Laws). Этот подход удовлетворяет условию невозрастания полной вариации, сохраняет монотонность численного решения для большинства задач, нет необходимости вводить искусственную вязкость, а также характеризуются отсутствием нефизичных осцилляций на разрывных решениях [3]. Численный метод является эйлеровым и построен для декартовой и цилиндрической систем координат. Использовался метод расщепления по пространственным переменным [8].

Для дискретизации был выбран не конечно-разностный подход, а наиболее эффективный интегро-интерполяционный (метод конечных объемов), в этом случае законы сохранения выполняются на сеточном уровне. При этом значения сеточных функций заменяются средними значениями по объему ячеек, а производные вычисляются по функциям на границах ячеек. Конечно-объемная аппроксимация физически величин также позволяет избегать особенностей при $r = 0$. Использовался третий порядок аппроксимации величин по пространству. Продвижение по времени целесообразно вести с помощью быстрого явного метода, который не понижал бы точность численной схемы. Наиболее удобен метод Рунге-Кутта второго порядка точности.

Для вычисления потоков физических величин через границы ячеек будем использовать решение задачи Римана, на котором основаны так называемые методы Годунова [4]. Использование точного решения задачи Римана является крайне ресурсоемким, поскольку требует в каждой ячейке расчетной сетки решения нелинейного алгебраического уравнения. Воспользуемся приближенным подходом, основанным на методе Хартена-Лакса-ван Лиира (HLLC), который позволяет одновременное учитывать наличие ударных волны, контактных и тангенциальных разрывов [5]. Данный метод был модифицирован для сквозного расчета границы газ-вакуум, что является важной особенностью для моделирования астрофизических систем, в которых могут возникать области крайне низкой концентрации вещества.

Получение реалистичных результатов численных расчетов требует высокого пространственного разрешения (рис. 2). Увеличение числа ячеек для газодинамических расчетов приводит с одной стороны к уменьшению шага интегрирования, с другой к росту количества операций для расчета большего числа узлов сетки. Для получения приемлемых результатов необходимо применение параллельных вычислений на суперкомпьютерах.

3. Распараллеливание вычислительного процесса

Распараллеливание численного кода выполнялось с помощью совместного использования стандартов MPI и OpenMP [7], [10]. Заданная расчетная область распределялась между

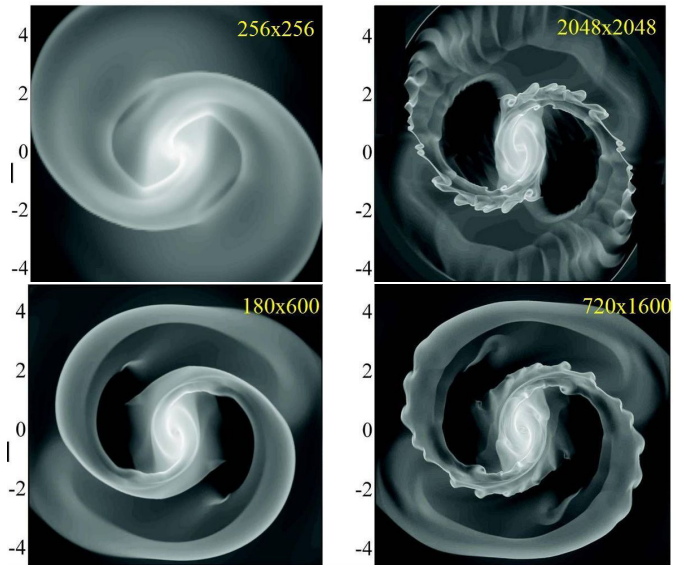


Рис. 2. Слева сравнение результатов расчетов на декартовой координатной сетке с различным пространственным разрешением. Справа тоже самое сравнение для цилиндрической системы координат. Очевидно, что результат расчетов чувствителен к параметрам расчетной сетки

процессорами [6]. Декомпозиция на подобласти проводилось вдоль всех трех координатных осей: вдоль радиуса, вдоль угла и вдоль вертикального направления (см. рис. 3).

Для получения решения во всей расчетной области необходимо обеспечивать непрерывную связь соседних процессов за счет обмена данными в фиктивных граничных ячейках. Поскольку разработанная численная TVD схема имеет третий порядок аппроксимации по пространству, то соседним процессам необходимо обмениваться всеми физическими величинами находящимися в двух приграничных слоях ячеек. В силу того, что данные граничных ячеек расположены в памяти непоследовательно, то вдоль радиуса можно пересылать границу целиком (плоскость), вдоль угла последовательность одномерных массивов и вдоль вертикальной координаты — поэлементно. Таким образом, время обмена границами на каждом гидродинамическом шаге вдоль координатных осей относится как 1:2:10. Для того, чтобы компенсировать потерю эффективности при пересылке граничных условий вдоль вертикальной координаты, по этому направлению декомпозиция должна быть наименее подробной. Дальнейшей модификацией обмена границами стало последовательное объединение данных, необходимых для отправки, и их дальнейшая пересылка. Для вычислительных кластеров, состоящих из многоядерных процессоров, эффективным оказалось использование стандарта OpenMP внутри каждого процесса. В этом случае происходит ускорение из-за отсутствия необходимости производить обмен фиктивными ячейками. Структура численного газодинамического кода позволяет распараллелить вычисления с помощью OpenMP в течение всего времени расчета на каждом процессе. В численном гидродинамическом алгоритме также проведена оптимизация распараллеливания за счет использования неблокирующей пересылки данных стандарта MPI [7].

Алгоритм расчета самогравитации является более ресурсоемким, нежели вычисление газодинамики на каждом шаге интегрирования (см рис. 4), поэтому важно адаптировать решение уравнения Пуассона под параллельные вычисления. Кратко опишем алгоритм нахождения гравитационного взаимодействия в газе. Совокупность узлов расчетной стеки могут быть организованы в иерархическую систему групп в форме древовидной структуры. Для сортировки частиц используется иерархия кубов, каждому узлу соответствует 8 потомков. В самом верхнем части дерева находятся все частицы. Деление кубов продолжа-

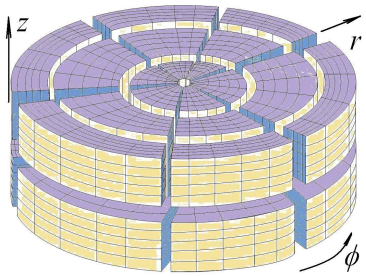


Рис. 3. Декомпозиция расчетной области вдоль трех направлений для цилиндрической системы координат

Рис. 4. Распределение времени расчета на одном газодинамическом шаге

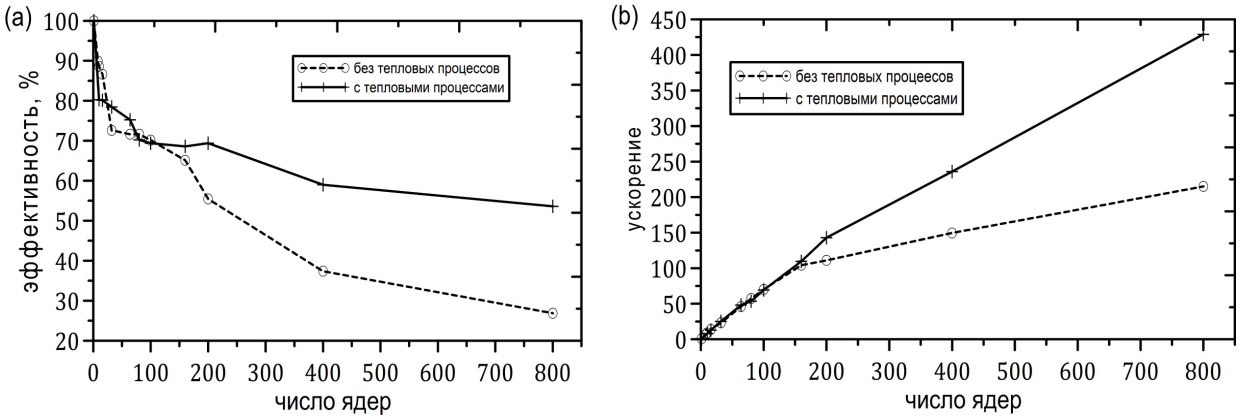


Рис. 5. (a) Графики эффективности параллелизма реализованного численного алгоритма для моделей с учетом термодинамических процессов и без них. (b) графики ускорения для тех же моделей

ется до тех пока в кубе не останется один узел, после чего для каждой ячейки или узла в дереве, необходимо найти общую массу и центр масс. Для более точного вычисления гравитации была реализована возможность учитывать квадрупольные слагаемые разложения гравитационного поля.

На рис. 4 показано, какую часть одного шага интегрирования занимают различные этапы вычислений. Эффективность распараллеливания численного алгоритма была оценена в серии экспериментов (число ядер $n_p = 8, 10, 16, 32, 64, 80, 100, 160, 200, 400, 800$) для двух моделей: с учетом термодинамических процессов и без них (см. рис 5a). На графике заметно, что наибольшая эффективность, около 70%, достигается при одновременном использовании в расчетах не более 200 ядер. Графики ускорения также построены для тех же наборов экспериментов (см. рис 5b). Ускорение значительно выше в моделях при учете тепловых процессов, чем для адиабатических моделей. Это связано с существенно большим объемом дополнительных вычислений, не влияющих на количество пересылаемых данных.

Вычисления проводились на суперкомпьютере СКИФ МГУ "ЧЕБЫШЕВ"(производитель Т-Платформы). Краткая характеристика кластера: пиковая производительность 60 TFlop/s, 1250 процессоров, 5000 ядер в системе, объём оперативной памяти 5.5 Тбайт, модель про-

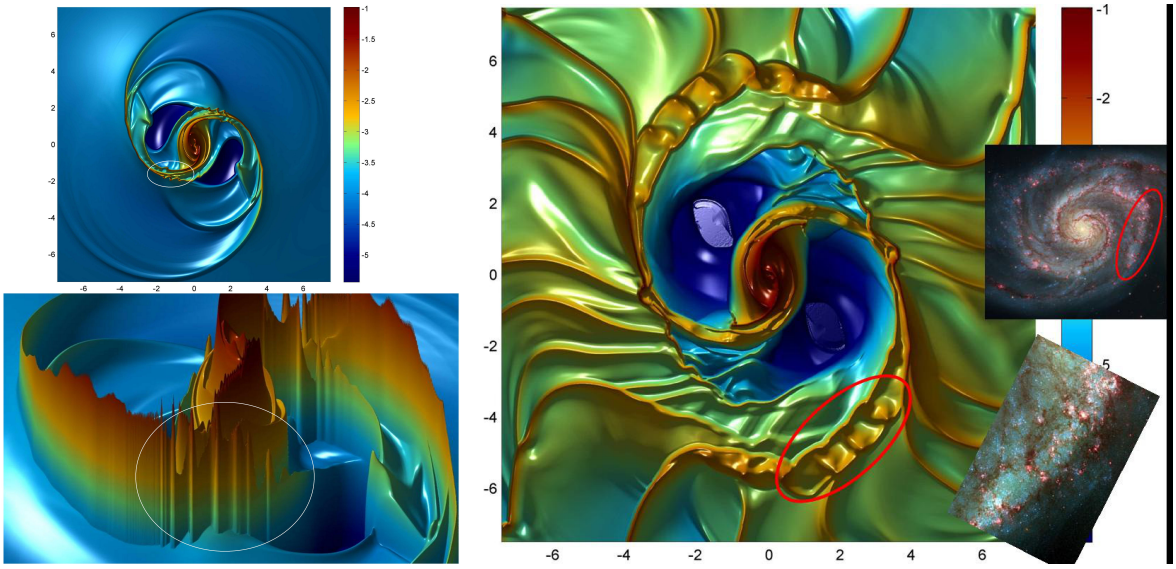


Рис. 6. Слева: начальный этап развития гофрировки фронта галактической спиральной ударной волны. Справа: нелинейный этап развития гофрировочной неустойчивости, отчетливо видны характерные протяженные шпуры (см. рис.1). На вставках изображения галактики M51.

цессора Intel Xeon E5472 3.0 ГГц.

Далее рассмотрим основные результаты численного моделирования, полученные с использованием описанного программного продукта.

4. Гофрировочная неустойчивость в газовом диске

Вначале рассмотрим возможность генерации шпуров за счет чисто гидродинамических эффектов, связанных с существенно нелинейной стадией развития гофрировочной (wiggle) неустойчивости фронта глобальной ударной волны в газовом диске при наличии спиральной волны плотности звездной компоненты [9].

В задаче имеется 8 свободных физических параметров модели, варьирование которых в широких пределах позволило рассмотреть формирование различных структур, связанных с развитием гофрировочной неустойчивости глобальной ударной волны. Значительное количество свободных параметров задачи приводит к необходимости проведения большого числа компьютерных экспериментов для исследования одной модели галактического газового диска. Так для одной модели проводилось до 10–15 расчетов с различными наборами параметров, что занимало в среднем 10000 процессор/час.

Возможность возникновения и скорость нарастания шпуров существенно зависят от их положения на спиральном рукаве. Как правило, очаг первоначального формирования нелинейной стадии гофрировки локализован на небольшом участке ударной волны (рис. 8a). По мере увеличения длины отростка газовой спирали он может достигать соседнего участка витка спирали, индуцируя возмущения на нем(рис. 8b). Такой механизм приводит к формированию наиболее интенсивных и протяженных шпуров с мощным оперением.

Рис. 7. Изображения галактик с кольцами (Hubble Space Telescope): слева Hoag's object , в центре NGC 1512, справа NGC 6782

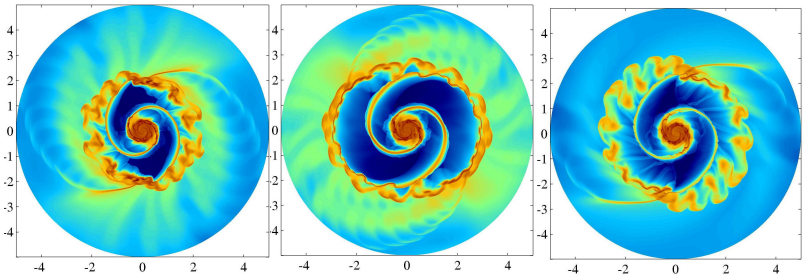


Рис. 8. Примеры колец в галактических газовых дисках в численных моделях с различными параметрами

5. Образование перьев в кольцеобразных галактических структурах

Известно значительное число галактик, у которых на некотором расстоянии от центра наблюдается кольцеобразная спиралевидная структура. Можно формально выделить три типа таких колец, не затрагивая различные механизмы их образования: ядерные кольца (NGC 1512, NGC 4314, NGC 6782), кольца на концах бара (NGC 1512, NGC 3081, NGC 4725, NGC 5905) и периферийные кольца в дисковой компоненте (M31, NGC 7742, AM 0644-741). В данной работе предложен гидродинамический механизм формирования кольцеобразных структур в галактических дисках.

На рис. 8 показаны распределения поверхностной плотности в моделях, в которых происходит образование кольцеобразных структур во внешней области диска, внутри которого имеется правильный двухрукавный узор со слабыми проявлениями нерегулярности. В то время как кольцо обнаруживает все признаки наличия шпуров и оперения различной степени интенсивности (рис. 8).

6. Моделирование облачной структуры Млечного Пути

Наблюдательные данные о газопылевых комплексах (ГПК) нашей Галактики показывают, что эти объекты концентрируются к спиральным рукавам Галактики, при этом свойства комплексов зависят от положения в Галактике [11] (рис. 9а).

Компьютерные модели, описывающие динамику газа в Галактике, должны адекватно описывать мелкомасштабную структуру и внутреннюю кинематику молекулярных облаков в галактическом диске. Это предъявляет особые требования к пространственному и временному разрешению, а также адекватный учет наиболее важных физических факторов.

Основываясь на результатах численных экспериментов проведенных на суперкомпьютере СКИФ МГУ "ЧЕБЫШЕВ"удалось продемонстрировать сценарии образования облачной структуры газа в окрестностях спиральных рукавов Галактики. Достигнутое разрешение

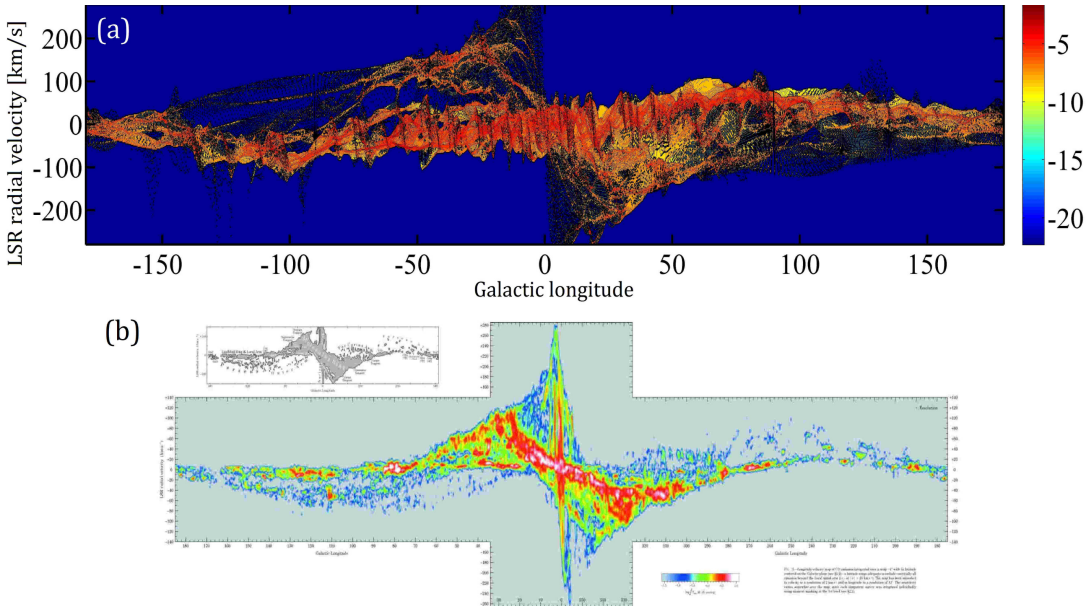


Рис. 9. Сравнение распределения молекулярного газа в численной модели с излучением газа Галактики [12]. Имеется качественное согласие результатов компьютерных симуляций и данных наблюдений. Отчетливо видны характерные морфологические особенности: молекулярное кольцо, внешний спиральный рукав и рукав Персея.

позволило изучать отдельные мелкомасштабные газовые комплексы. В моделях, учитывающих тепловые процессы (нагрев–охлаждение), хорошо выделяются газопылевые комплексы (рис. 10). В их плотных частях температура составляет $50-200$ К. При этом дисперсии турбулентных скоростей для таких образований порядка $3-5$ км/с. Параметры этих структур, полученные в численных экспериментах, соответствуют наблюдательным данным о гигантских молекулярных облаках и их окружении (рис. 9).

В рамках численных гидродинамических расчетов продемонстрирована возможность формирования ГПК, содержащих структуры типа гигантских молекулярных облаков. Структура и кинематика газа в построенных моделях качественно отражает наблюдаемые особенности распределения газопылевых комплексов.

7. Заключение

В рамках данного проекта было проведено около 200 численных экспериментов на вычислительном кластере СКИФ МГУ "ЧЕБЫШЕВ". На эти расчеты было затрачено порядка 200000 процессор/час. Благодаря использованию ресурсоемких вычислений на суперкомпьютере были промоделированы тонкие структуры в газовых дисках галактик. Результаты теоретических расчетов были сопоставлены со структурами наблюдаемыми в реальных спиральных галактиках. Сформулируем наиболее важные результаты:

1. Создание универсального программного продукта для моделирования газодинамических процессов различной геометрии с учетом самогравитации, тепловых процессов и внешних гравитационных полей. Данный пакет программ адаптирован для использования на ЭВМ с массивно-параллельной архитектурой.
2. Численные эксперименты позволили объяснить ряд наблюдаемых структур (шпуры, оперение) которые являются результатом развития гофрировочной неустойчивости ударных волн в газовом диске.
3. Предложен и численно-экспериментально изучен гидродинамический механизм образо-

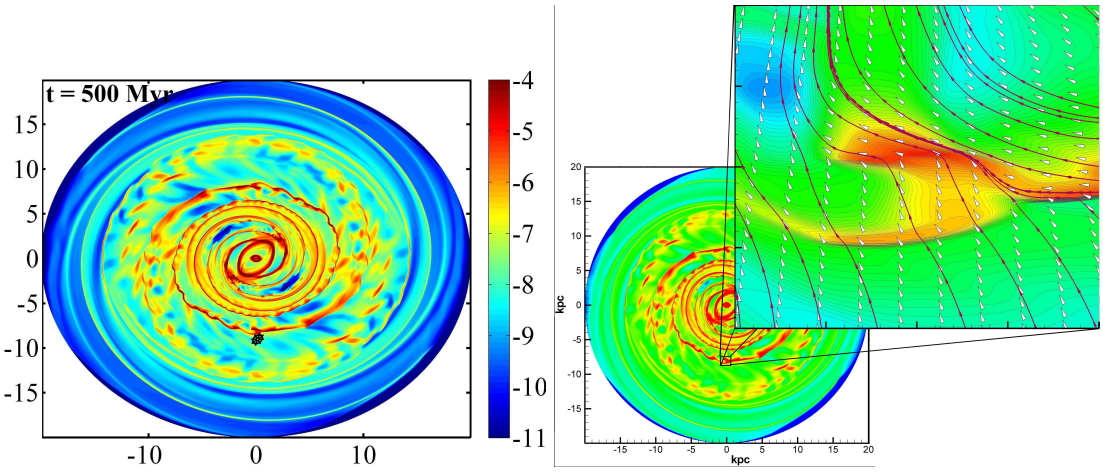


Рис. 10. Слева — пример развития облачной структуры в модели газового диске Млечного Пути, справа — поверхностная плотность и поле скоростей для отдельного облака на фоне всего галактического диска. Значком указано положения солнечной системы.

вания галактических колец в газовой компоненте галактик.
4. Продемонстрирован процесс формирования облаков в нашей Галактике под действием газодинамической и тепловой неустойчивостей.

Литература

1. Roberts W., Hausman M. 1984, Astrophysical Journal, 277, 744
2. Corder S., Sheth K., Scoville N.Z., Koda J., Vogel S.N., Ostriker E. 2008, Astrophysical Journal, 689, 148
3. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. - М.: Физмалит, 2001.
4. van Leer B. Journal of Computational Physics, 1979, V.32, P.101-136
5. Shengtai Li. Journal of computational physics, 2004, V.203, P.344-357
6. Кузнецов П.В., Кайгородов О.А. Адаптация схемы Ошера для параллельных вычислений, М.: 2002
7. Антонов А.С. Введение в параллельные вычисления, М.: 2002
8. Современные проблемы вычислительной математики и математического моделирования / Под ред. Бахвалова Н.С., Воеводина В.В. М.: Наука, 2005
9. Wada K. 2008, Astrophysical Journal, 675, 188
10. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. С.-Пб., 2002
11. Kalberla P.M.W., Kerp J. 2009, ARAA, 47, 27
12. Dame, T., Hartmann, D., Thaddeus, P. 2001, Astrophysical Journal, 547, 729

Метод анализа производительности распределенных приложений на основе эталонных моделей

А.С. Хританков

Московский физико-технический институт
Центр распределенных и Грид-технологий ИСА РАН

В работе рассматривается проблема анализа производительности распределенных приложений в системах метакомпьютинга и Грид. Предложена методика расчета характеристик производительности на основе сравнения с эталонными моделями, рассмотрены примеры применения для различных распределенных систем.

1. Введение

Грид для e-Science можно рассматривать, как вычислительную среду выполнения распределенных приложений для решения задач, требующих взаимодействия нескольких исследовательских коллективов. Исходная прикладная задача разбивается на проблемно-ориентированные части-подзадачи, которые решаются приложениями, предоставленными в виде сервисов для использования в Грид.

Для анализа производительности распределенных приложений в Грид необходимо рассматривать их как совокупность проблемно-ориентированных приложений и используемых ими вычислительных ресурсов в едином проблемно-ориентированном комплексе-сервисе, и исследовать производительность комплекса при решении конкретных типов задач или даже отдельных задач. Понятно, что вследствие характерного для распределенных вычислений отсутствия повторяемости условий вычислений, это реализуемо только на основе модельного описания процессов работы данных комплексов. Для широкого применения результатов анализа, необходимо обеспечить их объективность и воспроизводимость, а также, в какой-то мере, универсальность методов их получения. Поэтому для анализа необходима некая общая теория, положения и методы которой применимы для широкого спектра комплексов в составе Грид. Универсальность методов должна выражаться в возможности построения моделей различных существующих и будущих комплексов на основе одних и тех же положений и методов теории, а также в наличии единого механизма интерпретации получаемых результатов для всех моделей и анализируемых комплексов.

Другой особенностью Грид является вытекающая из целей его создания высокая специализация приложений, предоставляемых в качестве сервисов. Применение обычных методов "балансировки нагрузки" между приложениями, основанных на загруженности сервисов, в данном случае бессмысленно, так как часто идентичные альтернативные сервисы отсутствуют. Тем не менее, возможна оптимизация времени выполнения и используемых для решения задачи ресурсов путем выбора из набора сервисов, реализующих различные алгоритмы решения одного класса задач или подзадач, а потому обладающих различными характеристиками производительности.

В данной работе предлагается метод оценки производительности распределенных систем, основанный на сравнении модели процесса решения задачи распределенным приложением с различными эталонными моделями. Кратко рассмотрена суть метода, известные применения, сформулирована методика оценки производительности и проведена систематизация разработанных к настоящему времени моделей.

2. Метод эталонных систем

2.1 Общий подход к оценке производительности

При анализе производительности параллельных приложений на однородных вычислительных комплексах традиционно используются характеристики эффективности и параллельного ускорения [1]. Было предложено несколько способов переноса данных характеристик также и на неоднородные комплексы, сети рабочих станций [2-7]. Однако единой методики оценки производительности для всего спектра систем предложено не было. Наиболее распространенным методом оценки производительности вычислительных систем на данный момент является применение эталонных тестовых программ, то есть бенчмаркинг [25, 26], использование которого к распределенным приложениям затруднено.

В работах [9-17] был предложен и развит подход эталонных систем, в котором основополагающей идеей является сравнение процесса решения задачи системой с эталонной моделью данного процесса. Суть идеи состоит в следующем.

Модель вычислительной системы разделяется на модель системы $R(A) =< Y, X >$, которая включает измеряемые параметры Y комплекса и приложения при решении задачи $A \in \mathrm{A}$, и множество предположений X о вычислительном пространстве, структуре задачи и схеме управления процессом решения задачи; и эталонную модель $\overline{R}(\mathrm{A}) =< Y, \overline{X} >, X \subseteq \overline{X}$ процесса решения, которая использует измеряемые параметры Y и предположения $\overline{X}$ для расчета эталонного времени решения некоторого класса задач на этой системе.

Вследствие частичной доступности вычислительных ресурсов в распределенных комплексах традиционное понятие эффективности напрямую неприменимо. Тем не менее, его можно обобщить и адаптировать двумя способами.

Эффективность по времени определяется как отношение времени решения задачи эталонной моделью ко времени решения задачи в модели системы. Эффективность по стоимости определяется как отношение стоимости решения задачи в эталонной модели к стоимости решения задачи в модели системы.

Вследствие неодинаковой производительности элементов комплекса, вместо скалярного коэффициент параллельного ускорения следует использовать вектор ускорений по отношению к каждому элементу.

Вводимые таким образом характеристики выражают известные ранее понятия эффективности и ускорения для распределенных приложений и позволяют в дальнейшем адаптировать методы анализа производительности однородных параллельных комплексов к распределенным.

Следует отдельно рассмотреть удобный для распределенного решения класс декомпозируемых задач A_M, решение которых состоит в решении набора из M более простых однородных подзадач, зависимых друг от друга только по входным и выходным данным. К данному классу, например, относятся:

A) задачи поиска множества решений, удовлетворяющих заданному критерию, в том числе глобальной оптимизации;
B) задачи обработки информации, когда входные данные естественным образом представляются в виде набора несвязанных блоков, в том числе обработки потоков заявок.

Основным методом параллельного решения декомпозируемых задач является распределения подзадач между вычислителями комплекса и обеспечения доставки входных данных и результатов. В качестве примеров приложений, решающих декомпозируемые задачи, приведем системы добровольных вычислений (BOINC, SETI@Home, Folding@Home), проблемно-ориентированные системы (X-Com [24], BnB-Grid [8]).

2.2 Линейная эталонная модель

Под системой с расписанием $R(A)$ будем понимать модель распределенного приложения, которая включает совокупность N вычислителей совместно решающих задачу A из некоторо-

го класса A . Вычислители в процессе решения задачи доступны системе только часть времени согласно расписанию, задаваемому функцией $h_i(t):\Re^+ \to \{0,1\}$. При этом $h_i(t)=1$, если вычислитель доступен для системы, и $h_i(t)=0$ в противном случае. Введем характеристики *доступности* $\rho_i(t)$ и *времени доступности* $\tau_i(t)$ вычислителя i:

$$\rho_i(t)=\frac{1}{t}\int_0^t h_i(\tau)d\tau,\ \rho_i(t)\le 1,$$

$$\tau_i(t)=t\cdot\rho_i(t).$$

Временем решения задачи будем называть промежуток времени от начала процесса вычислений до прекращения использования всех вычислителей системы:

$$T(A)=\max_i\{t: h_i(t)>0\}.$$

Выберем некоторый эталонный алгоритм, с помощью которого принципиально возможно решить любую задачу $A\in \mathrm{A}$ на каждом вычислителе системы. Обозначим эталонное время решения задачи A вычислителем i при помощи эталонного алгоритма через $\overline{T_i}(A)>0$. Выберем некоторый способ оценки трудоемкости решения задачи $L(\cdot):\mathrm{A}\to\Re^+$. Например, *трудоемкость* $L(A)$ можно определить, как время решения задачи $A\in \mathrm{A}$ эталонным алгоритмом на одном из вычислителей системы. *Эталонной производительностью* $\pi_i(A)$ вычислителя i при решении задачи A будем называть величину $\pi_i(A)=L(A)/\overline{T_i}(A)$.

Рассмотрим простую модель учета затрат на вычисления, в которой задана *удельная стоимость* $c_i>0$ использования вычислителя i в единицу времени. Будем полагать, что стоимость использования $\Phi(t)$ системы R к моменту времени t складывается только из стоимостей использования вычислителей $\Phi_i(t)$:

$$\Phi_i(t)=c_i\cdot\tau_i(t),\ \Phi(t)=\sum_{i=1}^N\Phi_i(t).$$

Вычислительной *системой с расписанием* будем называть совокупность $R(A)=<\vec{h}(\cdot),\vec{\pi}(A),\vec{c}>$, где $Y=\{\vec{h}(\cdot),\vec{\pi}(A)\}$, $X=\{\vec{c}\}$. *Эталонной моделью* $\overline{R}_{\overline{X}}(\mathrm{A})$ системы с расписанием $R(A)$ будем называть совокупность $\overline{R}_{\overline{X}}(\mathrm{A})=<\vec{h}(\cdot),\vec{\pi}(\mathrm{A}),\{\vec{c}\}\cup\overline{X}>$, где $\vec{h}(\cdot)$ - функция расписания системы $R(A)$, $\vec{\pi}(\mathrm{A})$ - способ определения трудоемкости и эталонной производительности для задач $A\in \mathrm{A}$, $\overline{X}$ - множество допущений относительно вычислительного пространства, структуры задачи или системы управления вычислениями.

Линейной эталонной моделью $\overline{R}(\mathrm{A})$ будем называть эталонную модель, в которой $\overline{X}=\{\sum\overline{l_i}=L,\ \min T\}$. Обозначение «$\min T$» указывает цель моделирования – рассчитать минимальное время решения, которое и будет эталонным временем решения, «$\sum\overline{l_i}=L$» указывает, что решение задачи может быть произвольным образом разделено между вычислителями, а его трудоемкость не изменяется при переходе от последовательного к параллельному решению, а именно:

$$l_i(t)=\pi_i\cdot\tau_i(t),\ L(A)=\sum_{i=1}^n l_i(\overline{T}),$$

$$\overline{T}(A)=\arg\min_t\left\{t:\sum_{i=1}^N l_i(t)=L(A)\right\}.$$

2.3 Характеристики производительности

Введем характеристики эффективности систем с расписанием, используя эталонную модель в качестве образца таким образом, чтобы полученные характеристики являлись обобщением параллельной эффективности и ускорения, определяемых для однородных систем.

Коэффициентом эффективности по времени системы R будем называть отношение эталонного времени решения к измеренному:

$$E_t = \frac{\overline{T}}{T}.$$

Коэффициентом ресурсной эффективности системы R, или ресурсной эффективностью, будем называть отношение стоимости использования эталонной модели для решения задачи к реальной стоимости:

$$E_\phi = \frac{\Phi(\overline{T})}{\Phi(T)}.$$

Коэффициентом эталонного ускорения эталонной модели $\overline{R}$ по отношению к вычислителю i назовем отношение $\overline{S}_i = \overline{T}_i / \overline{T}$. *Коэффициентом ускорения* системы R будем называть величину $S_i = E_t \cdot \overline{S}_i$. Коэффициенты ускорения образуют *вектор ускорения по времени*:

$$\vec{S} = \left(\frac{\overline{T}_1}{T}, ..., \frac{\overline{T}_2}{T} \right).$$

Эффективность по времени и коэффициенты ускорения при «$\sum \overline{l}_i = L$» связаны соотношением [16]:

$$E_t = \left(\sum_{i=1}^{N} \frac{\rho_i(\overline{T})}{S_i} \right)^{-1}. \quad (1)$$

Покажем сводимость введенных характеристик производительности к известным понятиям эффективности и ускорения [16].

Рассмотрим неоднородную систему $R_{het} =< \vec{h} = \vec{1}, \vec{\pi}(A) >$. Так как вычислители доступны все время решения задачи, то $\phi_i(t) = t$, $\Phi(t) = t \cdot \sum_{i=1}^{N} c_i$, значит $E_t = E_\phi = E$.

Для параллельного комплекса с узлами одинаковой производительности воспользуемся моделью однородной системы $R_{par} =< \vec{h} = \vec{1}, \vec{\pi} = \vec{\pi}_0 >$. Поэтому $\vec{\pi} = \vec{\pi}_0$, и элементы вектора ускорения будут равны $\vec{S}_i = \overline{T}_i / T = L(A)/(\pi_0 T) = S_0$. Принимая во внимание (1), получим соответствие введенных характеристик принятым для однородных комплексов:

$$S = S_0, \; E = \frac{S}{N}.$$

2.3 Интервальная эталонная модель для декомпозируемых задач неизвестной трудоемкости

Задачу $A \in \mathrm{A}_M$ будем называть *декомпозируемой* на M подзадач $\{A_j\}_{j=1}^{M}$, если для решения задачи A необходимо решить все подзадачи. При этом суммарная трудоемкость решения подзадач равна трудоемкости решения задачи $L(A) = \sum_{j=1}^{M} L(A_j)$.

Рассмотрим систему $R_K(A_M) = R(A) \cup < \vec{n}, M, X_K >$, вычислительное пространство которой состоит из $K = |\vec{n}|$ кластеров C_k по n_k вычислителей в каждом, а решаемая задача A разбивается на M подзадач. Множество предположений X_K состоит из элементов:

- $\sum L_k = L$ - предполагаем, что задача является декомпозируемой;
- $\pi(A_k) = \pi$ - задача является *однородной*, то есть производительность всех вычислителей системы при решении подзадач равна производительности при решении всей задачи;
- $K : h_k, \pi_k, p_k$ - полагаем, что только p_k, $p_k \le n_k$, вычислителей кластера будут задействованы для решения подзадач. Вычислители кластера k имеют одинаковую эталонную производительность π_k для всех задач класса A_M и работают согласно общему для них расписанию $h_k(t)$, $k = 1..K$.

- B_{kj} - подзадачи A_i назначаются для решения на кластер k пакетами (bag) $B_{kj} \subseteq \{A_i\}_1^M$, $\sum_{k,j} B_{kj} = \{A_i\}_1^M$, по $b_{kj} = |B_{kj}|$ подзадач. Кластер приступает к решению подзадач из пакета $j+1$ только после окончания решения подзадач пакета j.

Модель системы $R_K(A_M)$, удовлетворяющую данным предположениям, будем называть *интервальной системой*.

Без ограничения общности можно полагать, что каждый кластер работает только один интервал $[t_k^0, t_k^1]$. Функция расписания будет иметь вид:

$$h_i(t) = \begin{cases} 1, t \in [t_k^0, t_k^1], \\ 0, t \notin [t_k^0, t_k^1], \end{cases} \quad i \in C_k .$$

В работе [13] предложена эталонная интервальная модель $\overline{R}_{K,m}(\mathrm{A}_{M,\mathrm{Pr}}) = R_K(A_M) \cup \overline{X}_{m,\mathrm{Pr}}$ и метод оценки производительности распределенных приложений при решении декомпозируемых задач класса $\mathrm{A}_{M,\mathrm{Pr}} \subseteq \mathrm{A}_M$ неизвестной заранее трудоемкости. Множество дополнительных предположений $\overline{X}_{m,\mathrm{Pr}}$ модели включает элементы:

- $p_k = n_k$ - все вычислители кластера участвуют в решении подзадач,
- $L_i \in \mathrm{Pr}$ - трудоемкости решения подзадач $\{L_i\}_1^M$ - независимые в совокупности случайные величины. Предполагается, что их распределение имеет положительный коэффициент асимметрии и конечный коэффициент эксцесса.
- $m_k \geq 1$ - на каждый вычислитель кластера k назначается $m_k \geq 1$ задач из пакета, $b_{kj} = m_k p_k$.

Вследствие того, что подзадачи имеют разную трудоемкость, вычислители, завершившие последовательный расчет m подзадач, будут простаивать, так как следующий пакет подзадач не может быть получен, пока не будет решена последняя подзадача из данного пакета.

Оценку влиянию данного эффекта на эффективность процесса решения задачи в распределенной системе дает следующее

Утверждение 1. Для ресурсной эффективности в среднем $< E_\phi >$ эталонной модели $\overline{R}_{K,m}(\mathrm{A}_{M,\mathrm{Pr}})$ по отношению к линейной модели $\overline{R}(\mathrm{A})$ справедлива оценка:

$$< E_\phi > \leq \sum_{k=1}^{K} \xi_k \varepsilon_\phi^k , \ \xi_k = \frac{c_k n_k \rho_k(T)}{\phi(T)} , \tag{2}$$

$$\varepsilon_\phi^k \approx \left(1 + \frac{n_k - 1}{\sqrt{n_k}} \ln n_k \frac{D_A}{\sqrt{m_k n_k}} \right)^{-1} , \tag{3}$$

где $\phi(T) = \Phi(T)/T$, n_k - количество вычислителей в кластере, m_k - количество подзадач назначаемых на один вычислитель, c_k - стоимость использования одного вычислителя кластера, D_A - константа, которая зависит от параметров распределения трудоемкости подзадач.

3. Методика оценки производительности

3.1 Структура моделей и применимость результатов

Построенные в рамках подхода эталонных систем модели и использованные при этом предположения представлены в таблице 1.

По определению, модель системы $R_1 =< Y_1, X_1 >$ *обобщает* модель системы $R_2 =< Y_2, X_2 >$, если $Y_1 \subseteq Y_2$ и $X_1 \subseteq X_2$, причем, по крайней мере, в одном из соотношений выполнено строгое включение. Будем говорить, что эталонная модель $\overline{R} =< \overline{Y}, \overline{X} >$ *моделирует* систему $R =< Y, X >$, если $Y \supseteq \overline{Y}$ и $X \subseteq \overline{X}$.

Рассмотрение структуры моделей и установление между ними отношений обобщения и моделирования позволяет указать, какие эталонные модели к каким системам применимы, и обеспечить возможность сравнения различных систем между собой. Справедливо [15] следующее

Утверждение 2. Эталонная модель $\overline{R}$, моделирующая систему R применима ко всем системам, обобщением которых является система R.

Таблица 1. Модели систем и эталонные модели

Модель	Входные параметры, Y	Предположения и ограничения, X
Системы с расписанием		
$R(A)$ [9]	$h(t)$, $\pi(A)$	c_i
$\overline{R}(\mathrm{A})$ [9]	$h(t)$, $\pi(A)$	c_i, $\sum \overline{l}_i = L$, $\min T$
$R_{het}(A)$ [16]	$h(t)$, $\pi(A)$	c_i, $h_i(t) = 1$
$R_{par}(A)$ [16]	$h(t)$, $\pi(A)$	c_i, $h_i(t) = 1$, $\pi_i = \pi_0$
$\overline{R}_{het}(\mathrm{A}_\beta)$ [16]	$h(t)$, $\pi(A)$	c_i, $h_i(t) = 1$, $\beta > 0$, $\sum \overline{l}_i = (1-\beta)L$, $\min T$
Интервальные системы		
$R_K(A_M)$ [10]	$h(t)$, $\pi(A)$, n_k, M	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$
$\overline{R}_{K,p}(\mathrm{A}_{M,a})$ [10]	$h(t)$, $\pi(A)$, n_k, M	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$, $L_k = a$, $m = 1$, $\min T$
$\overline{R}_{K,n}(\mathrm{A}_{M,a})$ [10]	$h(t)$, $\pi(A)$, n_k, M	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$, $p_k = n_k$, $L_k = a$, $m = 1$, $\min T$
$\overline{R}_{K,m}(\mathrm{A}_{M,\mathrm{Pr}})$ [13]	$h(t)$, $\pi(A)$, n_k, M	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$, $p_k = n_k$, $L_k \in \mathrm{Pr}$, $m \geq 1$, $\min T$
Кластерные системы		
$R_{K,T}(\mathrm{A}_M)$ [12]	$h(t)$, n_k, M, $\bar{t}(A_k)$	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$, $b_{kj} = 1$
$\overline{R}_{K,T}(\mathrm{A}_K)$ [12]	$h(t)$, n_k, M, $\bar{t}(A_k)$	c_i, $\sum L_k = L$, $\pi(A_k) = \pi$, $K : h_k, \pi_k, p_k$, $B_{kj}^{et} = B_{kj}^{sys}$, $\pi_k = L_k / (n_k \bar{t}_k)$, $\min T$

3.2 Методика оценки производительности

Метод оценки производительности [15] на основе эталонных моделей включает следующие шаги:

1) Определить цель исследования: сравнение производительности, выбор оптимальной схемы управления, оценка потерь и накладных расходов или анализ целесообразности перехода к распределенному решению задачи.

2) В зависимости от цели исследования, определить модель системы, которая описывает решение задачи комплексом, и выбрать эталонную модель. После этого применить модель системы к исследуемому комплексу, сопоставить элементы комплекса вычислителям модели.
3) Согласно выбранной системе следует определить эталонный алгоритм. Выбор зависит от доступных данных о процессе вычислений, возможности провести экспериментальную оценку производительности.
4) При необходимости, провести оценку производительности вычислителей.
5) Провести вычисления и собрать данные о процессе решения в соответствии с выбранной моделью системы.
6) Рассчитать характеристики производительности модели системы по отношению к эталонной модели.

Общая схема метода приведена на Рис. 1.

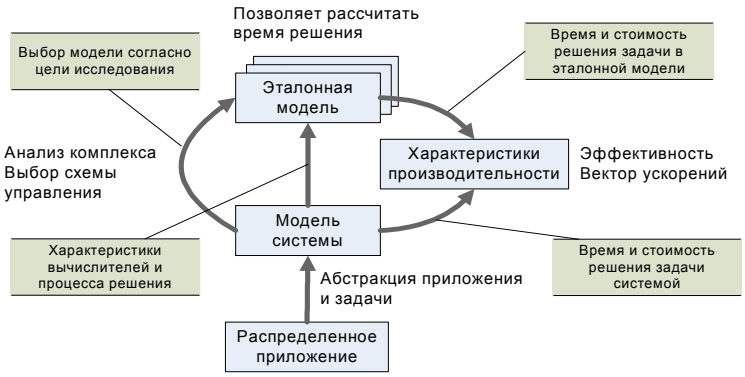


Рис. 1. Общая схема метода эталонных систем.

3.3 Программная реализация

Для применения метода эталонных систем к анализу производительности распределенных приложений был реализован комплекс программ [15], включающий модуль управления вычислениями, модуль сбора данных и модуль расчета характеристик производительности.

Комплекс программ реализован на Java в виде подключаемой к комплексу BnB-Грид программной библиотеки с возможностью его использования независимо для анализа производительности других распределенных комплексов.

4 Применение метода эталонных систем

С помощью метода эталонных систем была проведена оценка производительности и выработаны рекомендации по ее повышению следующих распределенных систем.

В работе [14] проведен анализ производительности системы BnB-Грид и продемонстрирована применимость линейных эталонных моделей систем с расписанием, а также обобщений характеристик эффективности и ускорения.

В работе [10] предложена модель интервальной системы для описания процесса решения задач в мультикластерных системах, разработан алгоритм управления процессом вычислений и получены оценки снижения эффективности вследствие ограничений на управление отдельными вычислителями.

В работе [11] были представлены результаты анализа производительности системы метакомпьютинга X-Com^2 [24] и представлены рекомендации по организации управления вычислительным пространством, реализация которых позволила бы повысить эффективность использования вычислительных ресурсов в полтора раза.

В квалификационной работе [15] приведены результаты оценки производительности вычислительного комплекса HPC-NASIS [23] и системы распределенных вычислений CAS Maxima DesktopGrid [22], для которой на основе разработанной ранее в работах [16, 17] модели

решения задач с последовательной частью было предсказано время решения задачи с точностью 1%.

В данной работе рассмотрим подробнее применение метода эталонных систем к анализу производительности распределенных приложений в средах на базе UNICORE и уточненную модель для вычислительной инфраструктуры BnB-Грид.

4.1 Кластерная модель ППО UNICORE

В работе [12] в рамках подхода эталонных систем построена модель кластерной системы $R_{K,T}(A_M)$. Предложенная модель использована в методе оценки производительности распределенных приложений, выполняемых в средах, основанных на промежуточном программном обеспечении UNICORE 6 [28].

Согласно информационной модели GLUE [27], приложения логически разделяются на активности (Activity). Выделение ресурсов для выполнения активностей происходит в рамках заданий (Shares), которые указывают количество выделенных логических процессоров и обязательства менеджера ресурсов по времени их предоставления.

Рассмотрим *кластерную систему* $R_{K,T}(A_M) = R_K(A_M) \cup < \bar{t}_k, b_{kj} = 1 >$, где $\bar{t}_k = \bar{t}(A_k)$ - измеряемое эталонное время решения задачи A_k кластером. *Эталонная кластерная модель* задается выражением $\overline{R}_{K,T}(A_K) = R_K(A_M)\big|_{M=K} \cup < \pi_k = L_k/(n_k \bar{t}_k), B_{kj}^{et} = B_{kj}^{sys} >$. При этом эталонная производительность вычислителей кластера k рассчитывается по формуле $\pi_i(A_k) = L(A_k)/(n_k \bar{t}_k)$, $i \in C_k$. Будем полагать, что назначение подзадач на кластеры в эталонной модели остается тем же, что и в модели системы, $B_{kj}^{et} = B_{kj}^{sys}$. Стоимость решения кластером k подзадачи A_k составляет $\Phi_k = (t_k^1 - t_k^0) c_k n_k$, эталонная стоимость решения подзадачи на данном кластере $\overline{\Phi}_k = \bar{t}_k c_k n_k$. Эффективность по стоимости:

$$E_\phi = \frac{\sum_1^M \overline{\Phi}_k}{\sum_1^M \Phi_k}.$$

Эффективность по стоимости в данном случае показывает, насколько накладные расходы на организацию вычислений влияют на затраты на решение задачи. Эффективность по времени по отношению к $\overline{R}_{K,T}(A_K)$ не является существенной характеристикой, так как будет близка к единице при больших k:

$$E_t = \frac{\max_k \{t_k^0 + \bar{t}_k\}}{\max_k \{t_k^1\}}.$$

Необходимые для расчета параметры n_k и K известны пользовательскому приложению, параметры $\vec{h}(\cdot)$, $\bar{t}_k$, c_k могут быть получены при помощи сервиса UNICORE CIS (Common Information Service) [28], реализующего информационную модель GLUE. Сопоставление параметров модели атрибутам в GLUE приведено в таблице 2.

Таблица 2. Сопоставление параметров кластерной модели атрибутам GLUE.

Параметр модели	Сопоставляемая / измеряемая величина
Задача A	Инициированная клиентом активность (Activity), возможно включающая под-активности, назначаемые на выполнение.
Кластер C_k	Активность (Activity) на выделение ресурсов.
Вычислитель в кластере	Слот, выделенный активности на ресурсе
Время начала интервала t_k^0	Атрибут время старта активности (StartTime)
Время завершения интервала t_k^1	Атрибут время завершения активности (EndTime)
Удельная стоимость вычислителя c_k	Индекс производительности (атрибут Value) ресурса согласно бенчмарку (Benchmark) для запрошенной активностью среды выполнения (ExecutionEnvironment)
Трудоемкость решения подзадачи $L(\cdot)$	Можно положить $L(\cdot) \equiv 1$

Эталонное время $\bar{t}(A_k)$ решения подзадачи A_k	Атрибут использованное процессорное время (UsedTotalCPUTime)
Время начала вычислений	Минимальное из времен принятия под-активностей к исполнению, атрибут SubmissionTime.
Время завершения вычислений	Максимальное из времен завершения выполнения под-активностей, атрибут EndTime.

4.2 Анализ производительности распределенного комплекса BnB-Грид

Программный комплекс BnB-Грид предназначен для распределенного решения декомпозируемых задач глобальной оптимизации. В работе [12] построена модель процесса решения задачи глобальной оптимизации в BnB-Грид, и в рамках подхода эталонных систем разработан метод оценки производительности на основе экспериментальных данных. Реализованный в комплексе алгоритм SMBH [8] основан на Basin-Hopping и состоит в следующем. Исходная задача A разбивается на M подзадач $\{A_i\}_1^M$. Подзадачей является нахождение локального минимума итеративным алгоритмом из начальной точки, выбираемой случайным образом в окрестности предполагаемого минимума. Трудоемкость решения подзадачи напрямую зависит от количества шагов локального алгоритма и сложности вычисления оптимизируемой функции.

Комплекс построен на основе трехуровневой архитектуры [8]. Управляющий узел верхнего уровня производит разбиение исходной задачи на подзадачи и направляет их узлам второго уровня блоками для решения. Получив блок подзадач, узлы второго уровня распределяют их для решения узлами третьего уровня. Узлы третьего уровня обычно представляют собой программы-решатели, расположенные на вычислительном кластере. Узлы второго уровня организуют решение блока подзадач в рамках одного кластера общего доступа, узел верхнего уровня управляет несколькими кластерами для решения одной задачи. В настоящей реализации BnB-Грид на втором и третьем уровнях используется параллельный комплекс решения задач оптимизации BnB-Solver, подключаемые через программу-посредника (proxy).

Для анализа производительности системы при решении задачи нахождения конфигураций атомных кластеров с минимальной потенциальной энергией была использована эталонная модель интервальной системы $\overline{R}_{K,m}(\mathrm{A}_{M,\mathrm{Pr}})$. Входные данные для модели были получены с помощью программной библиотеки управления процессом вычислений, реализованной для комплекса (см. раздел 3.3). Соответствие величин приведено в таблице 3.

Таблица 3. Сопоставление параметров интервальной модели измеряемым характеристикам.

Параметр модели	**Сопоставляемая / измеряемая величина**
Задача A	Совокупность $M \approx 500$ начальных точек алгоритма оптимизации
Кластер C_k	Узел второго уровня, экземпляр приложения BnB-Solver
Вычислитель в кластере	Процессорный элемент, доступный BnB-Solver
Время начала интервала t_k^0	Время получения сообщения о готовности BnB-Solver к решению подзадач.
Время завершения интервала t_k^1	Время завершения приложения BnB-Solver
Удельная стоимость вычислителя c_k	Принималась численно равной эталонной производительности
Эталонный алгоритм	Параллельный алгоритм, выполняемый в системе из одного кластера с одним вычислителем
Эталонное время решения задачи $\overline{T}_k(A)$	Оценка времени решения всей задачи на основе решения части подзадач $M_1 \approx \sqrt{M}$ или $M_2 \approx M$
Трудоемкость решения подзадачи $L(\cdot)$	Случайная величина с логнормальным распределением, параметры распределения определяются экспериментально при расчете эталонного времени решения
Время начала вычислений	Время старта решения задачи узлом верхнего уровня
Время завершения вычислений	Время освобождения всех выделенных ресурсов после решения всех подзадач

На Рис. 2 представлены значения ресурсной эффективности, полученные в серии экспериментов в системе BnB-Грид, и их сопоставление с теоретическим результатом (3). Сплошная линия соответствует верхним границам, получаемым при экспериментальной оценке трудоемкости задачи на $M_2 \approx M$ подзадачах, пунктиром указаны границы при оценке по $M_1 \approx \sqrt{M}$ подзадачам. Определение трудоемкости в последнем случае позволило бы обнаружить снижение эффективности при $p = 7..12$, что вполне достаточно для реальных задач, когда повторное решение не имеет смысла.

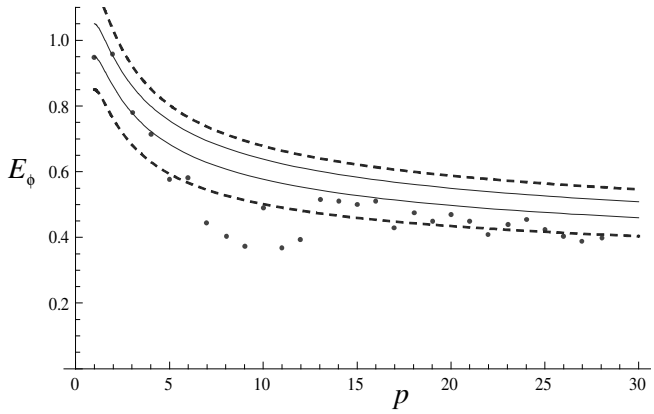


Рис. 2. Расчетная верхняя граница и экспериментальные значения ресурсной эффективности E_ϕ в зависимости от количества вычислителей в кластере p

5. Заключение

Для того, чтобы анализ эффективности распределенных приложений в Грид действительно проводился и применялись алгоритмы управления вычислениями, использующие результаты этого анализа, необходимо наличие следующих факторов:

a) разработаны и используются распределенные приложения, производительность которых недостаточна при решении поставленных прикладных задач, то есть имеется недостаток ресурсов или времени для решения прикладных задач. Например, ограниченные "вычислительные бюджеты", решение задач в реальном времени (*существует проблема недостаточной производительности*);
b) решение проблемы недостаточной производительности распределенного приложения актуально и необходимо, имеются "инфраструктурные ресурсы" для ее решения (*имеются ресурсы для совершенствования приложений*);
c) разработаны методы анализа производительности и оптимального управления, которые являются частью методологии разработки и построения распределенных приложений (*разработаны методы анализа и управления приложениями*);
d) применение методов оптимизации управления для повышения производительности оправдано, использование методов другого рода не позволяет эффективно решать прикладную задачу при сопоставимых затратах на применение (*проблему нельзя решить проще*).

В настоящий момент времени развитие концепции Грид переходит от теоретической в прикладную область, разработаны распределенные приложения [20, 21], объединяющие разнородные ресурсы, и специализированные научные сервисы [23] для решения прикладных задач. Вследствие расширения сферы применения технологий Грид следует ожидать актуализации первого фактора.

Введение образовательных программ в сфере суперкомпьютерных и Грид-технологий позволяет предполагать, что будет подготовлено достаточное количество специалистов, которые будут способны разрабатывать и совершенствовать распределенные приложения (например, [18. 19]).

Анализ существующих распределенных вычислительных комплексов и решаемых задач позволяет сделать вывод, что неэффективное управление распределенной средой может оказы-

вать значительное влияние на количество используемых для решения прикладных задач ресурсов и время вычислений [11, 13].

В настоящей работе предложен метод количественного анализа производительности различных классов распределенных комплексов на основе модельного описания процесса решения, что позволяет оценить степень достоверности результатов анализа, определить условия, при которых полученная оценка справедлива, и использовать результаты анализа при построении алгоритмов управления распределенными приложениями.

Литература

1. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. – 2nd ed. – USA: Addison Wesley, 2003.
2. Yan Y., Zhang X., Song Y. An Effective Performance Prediction Model for Parallel Computing on Non-dedicated Heterogeneous Networks of Workstations // J. of Parallel and Distributed Computing. – 1996. – vol. 38, no. 1. – p. 63-80.
3. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
4. Colombet L., Desbat L. Speedup and Efficiency of Large Size Applications on Heterogeneous Networks // In Proceedings of the Second international Euro-Par Conference on Parallel Processing, Volume II. Lecture Notes In Computer Science. – London: Springer-Verlag. – vol. 1124. – pp. 653-664.
5. Donaldson V., Berman F., Paturi R. Program speedup in a heterogeneous computing network // J. Parallel Distrib. Comput. – vol. 21, no. 3 (Jun. 1994). – pp. 316-322.
6. Robertazzi T.G. Networks and Grids: Technology and Theory. – New York: Springer, 2007.
7. Bharadwaj V., Ghose D., Robertazzi T. G. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems // Cluster Computing. – vol. 6, no. 1 (Jan. 2003). – pp. 7-17.
8. Посыпкин, М.А. Методы и распределенная программная инфраструктура для численного решения задачи поиска молекулярных кластеров с минимальной энергией // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). – Челябинск: Изд. ЮУрГУ, 2009. – 839 с., С. 269-281.
9. Афанасьев А.П., Посыпкин М.А., Хританков А.С. Аналитическая модель оценки производительности распределенных систем // Программные продукты и системы. – 2009. – № 4. – с. 60-64.
10. Хританков А.С. Об одном алгоритме балансировки вычислительной нагрузки в распределенных системах // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). – Челябинск: Изд. ЮУрГУ, 2009. – 839 с. – с. 778-783.
11. Хританков А.С. Анализ производительности распределенных вычислительных комплексов на примере системы X-Com // Труды Всероссийской суперкомпьютерной конференции «Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность» (г. Новороссийск, 21-26 сентября 2009 г.). – 2009. – с. 46-52.
12. Хританков А.С. Модель оценки эффективности распределенных приложений в среде «СКИФ-ГРИД». // Современные информационные технологии и ИТ-образование. Сборник докладов научно-практической конференции: учебно-методическое пособие. Под ред. проф. В. А. Сухомлина. – М.: ИНТУИТ.РУ, 2010. – 640 с. – с. 612-620.
13. Хританков А.С. Оценка эффективности распределенных систем при решении задач переменного размера. // Научно-технический вестник СПбГУ ИТМО. – 2010. – № 2(66). – с. 66-71.

14. Посыпкин М.А., Хританков А.С. О понятии ускорения и эффективности в распределенных системах // Труды Всероссийской научной конференции «Научный сервис в сети Интернет: решение больших задач». – 2008. – с. 149-155.
15. Хританков А.С. Оценка производительности распределенных вычислительных комплексов на основе модели эталонных систем. Диссертация на соискание ученой степени кандидата физико-математических наук. ИСА РАН. – 2010. – 148 с.
16. Хританков А.С. Математическая модель характеристик производительности распределённых вычислительных систем // Научный журнал «Труды Московского физико-технического института».– М., 2010. – Т. 2, № 1(5).– с. 110 – 115.
17. Хританков А.С. Закон Амдала для неоднородных вычислительных систем // Задачи системного анализа, управления и обработки информации : межвузовский сборник научных трудов. Вып. 3 / под общ. ред. Е. В. Никульчева. – М.: МГУП, 2010. – 198 с. – с. 188-197.
18. Суперкомпьютерный консорциум университетов России, http://hpc-russia.ru/.
19. Программа «СуперИнжиниринг» ЮУрГУ, http://supercomputer.susu.ru/SUSU_Intel/courses/superengineering/.
20. Среда распределенных вычислений MathCloud, http://www.mathcloud.org/.
21. Parallel Grid Runtime and Application Execution Environment (P-GRADE), http://www.p-grade.hu/.
22. Волошинов В.В. Символьное обращение плохо обусловленных матриц в Grid-среде сервисов доступа к системе компьютерной алгебры Maxima. // Тезисы докладов международной конференции «Математическое моделирование и вычислительная физика» (ММСР'2009) (Дубна, 7-11 июля 2009г.). – Дубна: ОИЯИ, 2009. – с. 175-176.
23. В.Н. Васильев [и др.]. Высокопроизводительный программный комплекс моделирования атомно-молекулярных наноразмерных систем // Научно-технический вестник СПбГУ ИТМО. Выпуск 54. Технологии высокопроизводительных вычислений и компьютерного моделирования. – СПб.: СПбГУ ИТМО, 2008. – с. 3-12.
24. Соболев С.И. Использование распределенных компьютерных ресурсов для решения вычислительно сложных задач // Системы управления и информационные технологии. – 2007. – №1.3 (27). – с. 391-395.
25. van der Wijngaart R. NAS Parallel Benchmarks Version 2.4: NAS Technical Report NAS-02-00 / NASA Ames Research Center. – Moffett Field, CA: (s.n), 2002.
26. Набор тестовый программ GridNPB 3, http://www.nas.nasa.gov/Resources/Software/npb.html
27. GLUE Specification v. 2.0, www.ogf.org/documents/GFD.147.pdf.
28. Romberg, M. The UNICORE Grid infrastructure. // Sci. Program., N. 10, V. 2 (Apr. 2002). – 2002. – pp. 149-157.

Автоматизация развертывания научного ПО в облачную инфраструктуру

М.Г. Щербаков[1,2], В.М. Соловьев[1], П.В. Ирматов[1]

Саратовский Государственный Университет[1], Grid Dynamics Consulting[2]

Проведено исследование возможности автоматизации развертывания научного программного обеспечения в облачной инфраструктуре. Показано, как упростить систему развертывания с использованием специализированного фреймворка Chef. Описан разработанный авторами программный комплекс, обеспечивающий развертывание высокопроизводительного вычислительного кластера в облачной инфраструктуре путем выполнения не более одной команды. В качестве примеров, разработана автоматизированная система развертывания фреймворка параллельного выполнения программ Hadoop и пакета конечно-элементного моделирования Elmer.

1. Введение

В современных условиях быстрого развития информационных технологий, в том числе программного обеспечения (ПО) для научных целей, необходимо обеспечение возможности апробирования ПО с минимизацией затрат времени и ресурсов на его развертывание. Поскольку среди вариантов, где можно развернуть ПО для оценки возможности его дальнейшего использования в рамках данных конкретных научных задач, зачастую имеется лишь единственный - персональный компьютер, то решение обычно затягивается. Особенно, если для этого необходима установка другой операционной системы (ОС). Кроме того, современные пакеты научного ПО могут требовать для работы кластеры высокопроизводительных вычислений. Апробирование нового научного ПО на специализированных вычислительных кластерах может представляться еще большими проблемами, как на бюрократическом уровне, связанными со всевозможными согласованиями, так и на техническом - проблемы совместимости совместимости с библиотеками и программами, уже работающими на кластере. Таким образом, возникает задача обеспечения возможности запуска ПО для целей апробирования и проведения научных расчетов на выделенных серверах под управлением совместимой с ПО операционной системой. По мнению авторов, решением поставленной задачи является использование облачных инфраструктур, то есть вычислительных ресурсов по заказу. На сегодняшний день существует уже достаточно большое количество общедоступных провайдеров облачных ресурсов, например: Amazon EC2, GoGrid, Rackspace. Облачные провайдеры предоставляют возможность загрузки виртуальных машин в их центрах данных и получения пользователями удаленного доступа с почасовой оплатой. Стоимость работы достаточна низкая, и на данный момент для не самых мощных компьютеров составляет порядка трех рублей в час. Пользователь, загружая необходимую для его экспериментов ОС, получает к ней удаленный доступ, например посредством протоколов RDP (Windows) или SSH (*nix). Далее пользователь работает с системой так, как будто бы это был его настольный компьютер. А именно, настраивает ОС и устанавливает необходимое ПО. Преимущество использования такого виртуального сервера в рамках задачи очевидно - устанавливается необходимая ОС (провайдеры предлагают на выбор большое число различных версий ОС, возможна загрузка собственного образа), система находится под полным контролем пользователя, не нужно никаких согласований и т.д. В том случае, если установка ПО вручную не является тривиальной задачей, тем более в случае распределенных приложений, и занимает значительное время, возникает вопрос автоматизации. Во-первых, это позволяет добиться безошибочного повторного развертывания ПО; во-вторых, позволяет сэкономить время и деньги. В случае несложных приложений, требующих за-

мены нескольких конфигурационных файлов в ОС, установки переменных окружения и установки приложения из пакетного репозитория, эта задача может быть достаточно легко решена путем написания bash скрипта в ОС *nix. Однако, когда список приложений для установки увеличивается, возникает связь между приложениями, тем более в случае установки распределенных приложений, быстро возрастает сложность кода скриптов. Отладка и поддержка таких скриптов отнимает значительное время, таким образом bash может быть использован в качестве инструмента настройки и установки приложений лишь для простых задач. В литературе [2] достаточно широко исследована работа по автоматизированному системному администрированию, включая анализ эквивалентности управления серверами и машины Тьюринга. На основе теоретических изысканий формируются фактические требования, предъявляемые к системе управления инфраструктурой серверов. В разное время были разработаны специализированные фреймворки, предназначенные именно для целей автоматизированного системного администрирования и установки приложений. Авторами рассмотрен достаточно новый фреймворк автоматизации развертывания приложений Chef Opscode [1] и показано, что фреймворк Chef удовлетворяет теоретическим требованиям, предъявляемым к инструменту автоматизированного управления серверами. Авторами также была разработана специальная утилита, которая совместно с Chef позволяет свести задачу развертывания распределенного приложения в облачной инфраструктуре к запуску одной команды с минимумом аргументов. Примеры и утилита находятся в свободном доступе [3], где рассмотрена автоматизация в *nix средах. Существует реализация Chef и для Windows, но она позволяет автоматизировать лишь ограниченное число задач.

2. Управление инфраструктурой серверов

Виртуальная машина в облачной инфраструктуре с точки зрения пользовательских программ и самого пользователя представляет собой ту же операционную систему, требующую настройки, администрирования, то есть управления, как и в случае физического сервера. Теоретические основы управления серверами базируются на практическом опыте [2]: *«Основываясь на проведенных нами исследованиях, не существует никакого инструмента написанного на каком-либо языке, способного предсказуемо администрировать инфраструктуру серверов без обеспечения детерминированного, повторяемого порядка изменений на каждом вычислительном узле. Среда времени выполнения для любого инструмента всегда работает в контексте целевой операционной системы; изменения могут повлиять на поведение самого инструмента, создавая круговые зависимости. Поведение таких изменений может быть сложно предсказуемо, следовательно, необходимо тестирование для проверки изменившихся вычислительных узлов. Как только изменения прошли тестирование, они должны быть реплицированы на рабочих вычислительных узлах в том же самом порядке, в котором они тестировались, из-за этих же круговых зависимостей.»*. Порядок изменений важен по следующим основным причинам:
– «круговая зависимость» — инструмент администрирования системы может выполнять код, который модифицирует сам этот инструмент и конечные пользователи могут не знать о таких зависимостях;
– непредсказуемость поведения в результате беспорядочных изменений на диске — сложнее проверить, чем предсказать поведение в результате детерминированного порядка.

2.1. Методы управления

Управление серверами (конфигурирование и администрирование) можно разделить на три категории: расходимость, конвергенция, и конгруэнция.
Расходимость — результат неверного управления. Расходимость характеризуется тем, насколько далеко отстоит существующая конфигурация сервера от требуемой или подразумеваемой. Симптомы расходимости включают в себя: непредсказуемое поведение сервера;

незапланированные нарушения работы; неожиданные проблемы с установкой пакетов и патчей; незакрытые проблемы безопасности; значительные временные затраты на устранение нарушений работы; высокая стоимость поиска источника нарушения работы и обслуживания. Причиной расходимости могут быть изменения на сервере «вручную» и другие беспорядочные изменения.

С **конвергенцией** большинство системных администраторов знакомятся, как только сталкиваются с расходимостью и начинают ручную синхронизацию критических файлов Затем они стараются автоматизировать этот процесс, котрорый и называется конвергенцией. В процессе конвергенции, вычислительный узел приближается к требуемому, предполагаемому поведению. Поскольку процессу конвергенции подлежит лишь некоторое подмножество файлов, а не система целиком, то все конвергентные инфраструктуры в некоторой степени содержат расходимость. Таким образом, основной проблемой администрирования инфраструктуры с изначальной расходимостью является отсутствие информации о том, когда сходимость, или конвергенция, будет завершена. В отсутствии такой информации, процесс никогда не будет завершен. Процедура развертывания программного обеспечения, уже пройденная успешно на одном из серверов, может провалиться на другом сервере из-за изначального расхождения в конфигурации. Установки ПО в этом случае превращается в итеративный процесс внесения изменений в инструмент конфигурирования и повторный запуск. Более того, необходимо убедиться в том, что внесенные изменения корректно работают на других серверах, где уже установлено это ПО. Не смотря на все вышеперечисленные недостатки, конвергенция обеспечивает большую надежность и предсказуемость, чем в инфраструктуре с расходимостью.

Конгруэнтность — управление серверами таким образом, чтобы их работа находилась в полном соответствии с предполагаемым, требуемым поведением. Конгруэнтность определяется состоянием битов на диске, а не поведением сервера, потому что состояние диска может быть однозначно определено, а поведение — нет [2]. По определению, расхождение от изначального состояния диска в конгруэнтной среде является симптомом сбоя кода, административных процедур или безопасности. В любом из этих трех случаев может не быть полной уверенности, какие области диска повреждены или изменены. В таком случае легче все три случая рассматривать как уязвимость в системе безопасности. Необходимо устранить причину сбоя, и переустановить вычислительный узел в конгруэнтном режиме. Этот режим позволяет восстановить сервер быстрее, чем каким-либо другим способом.

По мнению авторов, конгруэнтным инструментом также является инструмент, который хранит все инструкции по изменению системы, начиная (и включая) изначальную инсталляцию с образа. В случае выхода из строя вычислительного узла, все сделанные изменения на исходной системе должны быть повторены в таком же порядке на вновь созданной системе, используя те же инструкции. То есть, при соответствующем использовании инструментов конвергенции, а именно применении их на изначально идентичных системах, будет получена конгруэнтная система. Кроме того, все изменения в системе должны производиться с помощью инструмента и применяться одинаковым образом на вычислительных узлах, в противном случае в систему будет внесено расхождение с предполагаемым состоянием.

Для обеспечения конгруэнтного режима, инструмент управления сервером должен в том числе удовлетворять следующим требованиям.

1. Останавливать свою работу с отображением ошибки, в случае, если очередная инструкция не завершилась успехом. В качестве примера, покажем, что язык командной оболочки bash не обеспечивает такого поведения. Если инструкция

```
cd /storage/sources/
rm␣-fr
```

будет выполнена, например, с недостаточными привилегиями для смены директории, то мы получим следующий результат:

```
cd /storage/sources/
cd:␣permission␣denied:␣/storage/sources/
```

`rm -fr .`
Сообщение об ошибке может быть различным на разных ОС, однако результат одинаков: вместо удаления содержимого в `/storage/sources/`, будет удалено содержимое текущей директории.
2. Вносить изменения в одном и том же порядке.
3. Для большинства стандартных действий, таких как создание директории, пользователя, работа с сервисами и т.д., должны существовать специализированные ресурсы, облегчающие работу, и абстрагирующие от конкретной ОС.
4. Специализированные ресурсы должны быть идемпотентны [4].

С точки зрения программного кода, научное ПО ничем не отличается от какого-либо другого ПО, поэтому можно считать что автоматизированные системы установки и управления серверами одинаково применимы для любого вида программного обеспечения.

2.2. Установка и настройка приложений

Необходимые действия при установке приложения в ОС семейства *nix представляется более наглядным рассмотреть на конкретном примере, из которого выделить общие задачи. В качестве примера, рассмотрим Hadoop [5], фреймворк для запуска параллельных приложений с использованием технологии map-reduce. Кроме обеспечения параллельной работы приложений, Hadoop включает в себя распределенную файловую систему HDFS, высокопроизводительный координационный сервис для распределенных приложений Zookeeper, распределенную базу данных HBase и другое. Все перечисленные сервисы в общем случае могут устанавливаться на различные серверы, и некоторые из них имеют серверную и клиентские части. Сервисы не являются независимыми и взаимодействуют друг с другом, для чего должны быть созданы специальные конфигурационные файлы. В этих файлах указываются доменные имена узлов и порты, где запущены сервисы Hadoop, а также дополнительные настройки.

Hadoop написан на языке Java, и требует установленную Oracle JRE на сервере для своей работы. В простейшей конфигурации, в целях изучения основ Hadoop, на ОС не требуется никаких других пакетов, кроме JRE. Для использования Hadoop в режиме «production», система может потребовать специальных настроек (например, максимально допустимое число открытых соединений), настройка сетевого файрвола и т.д. В том случае, если требуется создание высокодоступного (Highly Available, HA) кластера Hadoop, необходима установка дополнительных пакетов в систему и их настройка. Если пакетов для данного дистрибутива нет, то потребуется сборка из исходного кода, что, в свою очередь, требует установленных пакетов компилятора и сборщика. Система установки должна работать без сбоев, то есть установка одного и того же приложения на один и тот же дистрибутив ОС должна заканчиваться успешно. В некоторых случаях необходимо обеспечить возможность работы приложения на другой ОС. В случае автоматизации, это означает обеспечение максимальной кроссплатформенности системы установки. При переходе на другую ОС, затраты на адаптацию системы установки должны быть минимальными, а код должен оставаться единым для установки на любую систему, следуя принципу DRY [6].

Обобщая, можно выделить следующие функциональные требования для инструмента управления серверами:
– повторяемая и предсказуемая работа на одной и той же платформе;
– работоспособность на всех необходимых платформах семейства *nix;
– установка пакетов в систему;
– внесение изменений в конфигурационные файлы системы;
– возможность отслеживания, на каких узлах установлен конкретный сервис.

С точки зрения программиста, для написания читаемого, кроссплатформенного кода необходимо обеспечение следующих требований:
– использование шаблонов для создания конфигурационных файлов;

– возможность доступа к системной информации (IP адреса, архитектура ОС и т.д.) независимым от ОС способом;
– возможность написания пользовательских функций;
– минимальное количество действий вручную для развертывания приложений на серверах;
– параллельная работа на различных серверах.

Для установки приложений в облачной инфраструктуре требуется их поддержка инструментом виртуализированных облачных сред. Кроме того, требование на минимальное количество действий вручную накладывает дополнительные функциональные ограничения на инструмент в плане работы с облачными инфраструктурами, а именно:
– автоматический запуск виртуальных машин облачного провайдера в соответствии с требованиями устанавливаемого ПО;
– ожидание готовности виртуальных машин;
– передача необходимых файлов и данных на виртуальные машины;
– активация установщика на виртуальных машинах (установка, конфигурация и запуск);
– параллельная работа с несколькими виртуальными машинами.

3. Фреймворк Chef

Выше были перечислены основные требования, предъявляемые к инструменту управления инфраструктурой серверов. Open Source продукт Opscode Chef удовлетворяет большинству из этих требований и является специализированным фреймворком по настройке и управлению серверами. Скрипты установки и настройки создаются на Ruby DSL и называются рецептами Chef. Абстрагируясь от ОС, в рецептах Chef предоставляются специальные **ресурсы**, позволяющие работать с системой. Например, ресурс `package` предназначен для установки пакетов в систему, `service` — для управления сервисами. Чтобы установить пакет «tar» в систему, в рецепте достаточно написать `package “tar"`; для запуска сервиса tomcat6:

```
service "tomcat6" do
  action :start
end
```

При выполнении кода запуска сервиса tomcat6, Chef сначала проверяет, не запущен ли уже сервис. Если сервис остановлен, Chef выполнит команду по его запуску, причем в зависимости от ОС команда может быть различной. Таким образом, фреймворк дает возможность программисту создавать скрипты автоматизированной установки, не привязываясь к конкретному дистрибутиву ОС. В некоторых случаях, однако, возникает необходимость выполнить различные действия в зависимости от ОС. Например, пакет Web сервера Apache в ОС RedHat называется httpd, а в Ubuntu - apache2. В этом случае можно использовать условное ветвление в зависимости от платформы:

```
package "apache2" do
  case node[:platform]
  when "centos","redhat","fedora","suse"
    package_name "httpd"
  when "debian","ubuntu"
    package_name "apache2"
  end
  action :install
end
```

Ресурсы в Chef обеспечивают идемпотентность. Однако, специальные проверки на полную идемпотентность отсутствуют. Например, следующий ресурс в Chef уже не будет идемпотентным:

```
bash "iptables␣startup" do
user "root"
  code <<-EOH
  echo ".␣/etc/iptables.sh" >> /etc/rc.d/rc.local
␣␣EOH
```

```
end
```
Таким образом, разработчик рецептов должен самостоятельно отслеживать и не допускать подобные ситуации.

На данный момент в Chef были доступны 24 ресурса, включая `package` и `service`, облегчающие конфигурирование системы и установку новых приложений. Так, Chef имеет ресурсы следующего предназначения:
– **HTTP Request**, отправка HTTP запросов;
– **Mount**, управление монтируемыми разделами;
– **Template**, создание файлов по шаблону.

Между ресурсами возможна передача сигналов. Это удобно, например, в следующем случае. Предположим, что в процессе выполнения рецепта изменяется конфигурационный файл Web-сервера. Для того, чтобы изменения вступили в силу, необходим перезапуск сервиса, однако нежелательно перезапускать сервис если конфигурационный файл не менялся. В Chef это можно реализовать следующим образом:

```
template "/etc/www/configures-apache.conf" do
  notifies :restart, "service[apache]"
end

service "apache"
```
Специальная инструкция `notifies` отправляет ресурсу `service "apache"` сигнал к действию, подлежащему выполнению.

Кроме рецептов, Chef оперирует также и другими сущностями. Все необходимые файлы для установки и конфигурации одного конкретного приложения, например Web сервера Apache, помещаются в единую структуру каталогов и такая структура называется Cookbook ("поваренная книга"). Типичное содержание каталога на примере Cookbook для установки Apache Tomcat имеет следующий вид:

```
.
|-- README.rdoc
|-- attributes
|   `-- default.rb
|-- definitions
|   `-- tomcat_app.rb
|-- files
|   |-- centos
|   |   `-- rightscale.repo
|   `-- default
|       |-- JVM-MANAGEMENT-MIB.mib
|       |-- dtomcat6
|       |-- logging.properties
|       `-- tomcat6
|-- libraries
|   |-- tomcat.rb
|   `-- tomcat_manager.rb
|-- metadata.json
|-- metadata.rb
|-- recipes
|   `-- default.rb
`-- templates
    `-- default
        |-- manager.xml.erb
        |-- tomcat-users.xml.erb
        `-- tomcat6.conf.erb
```
Имя корневого каталога, содержащей все эти файлы, и является именем "поваренной книги"для приложения. Рецепты хранятся в подкаталоге recipes. Кроме рецептов, Chef оперирует такими понятиями, как атрибуты (attribures/), определения (definitions/), файлы (files/), библиотеки (libraries/), шаблоны (templates/). **Атрибуты** в Chef можно рассматривать как некие глобальные переменные, причем во всей инфраструктуре, находящейся под управлением Chef сервера. Атрибуты содержат такие данные, как IP адреса, информацию об ОС, установленных пакетах и т.д. Во время запуска клиента Chef атрибуты формируют-

ся на клиенте и отправляются на сервер, где затем индексируются и становятся доступны для поиска. Это позволяет во время выполнения рецепта получить информацию о других серверах под управлением Chef, например, на каких еще виртуальных машинах выполняется или будет выполнен этот же рецепт. **Определение** в Chef позволяют создавать новый ресурс, содержащий в себе объединение нескольких других ресурсов. **Файлы** — файлы, которые требуется записать в систему без изменений. **Библиотеки** предоставляют возможность написания произвольного Ruby кода, расширяя возможности Chef. Например, это может быть библиотека для сбора данных с LDAP сервера для дальнейшего использования полученных данных в рецепте. **Шаблоны** — файлы со специальными переменными. Ресурс `template` передает в эти файлы значения подставляемых переменных, и записывает файлы в заданные места в ОС. Специальный файл `metadata.rb` содержит информацию о Cookbook в простом формате; `metadata.json` автоматически генерируется из `metadata.rb`, ручная модификация файла не требуется.

4. Утилита Clorun

Авторами была разработана утилита Clorun, основное предназначение которой состоит в уменьшении количества действий, необходимых для развертывания ПО, в том числе распределенного, в облачной инфраструктуре. На данный момент утилита работает только с облачным провайдером Amazon EC2, однако она может быть легко дополнена модулем сопряжения с другими облачными провайдерами, например Rackspace или GoGrid. Создание подобной утилиты показало принципиальную возможность сведения ручной работы к минимуму — программа автоматизирует все ручные действия, связанные с выбором и запуском виртуальных машин в облачной инфраструктуре, а также установку, настройку и запуск Chef клиента. Кроме работы в интеграции с Chef фреймворком, программа может быть легко адаптирована как для запуска произвольных bash скриптов, так и для работы с другими фреймворками. Требуемые изменения не должны затронуть больше, чем несколько строк конфигурации. Следовательно, утилиту можно считать универсальной для запуска виртуальных машин на Amazon EC2 и произвольного кода на этих машинах. Кроме запуска кластера виртуальных машин, утилита позволяет подобным же образом останавливать работу запущенных машин по окончании работ.

4.1. Принцип работы Clorun

Утилита представляет собой несколько файлов, написанных на Ruby, и конфигурационные файлы. Последние представляют из себя несколько файлов формата JSON (по количеству запускаемых машин), которые содержат информацию о запускаемых рецептах, а также некоторые дополнительные атрибуты, и также файл ec2.config, содержащий данные для запуска виртуальной машины на Amazon EC2 (тип машины, зона запуска, идентификационный номер образа). Принцип работы утилиты следующий:
– запрос виртуальных машин по количеству найденных JSON файлов в многопоточном режиме;
– запрос присвоенных IP адресов виртуальным машинам;
– создание JSON файлов конфигурации из уже существующих, включая в новые информацию о том, на каких IP адресах какие рецепты будут выполняться;
– ожидание перехода машин в статус "Running"и ответа по SSH протоколу;
– загрузка JSON файлов и заархивированных Chef рецептов по протоколу SFTP на каждую из машин в многопоточном режиме;
– выполнение на виртуальных машинах команды wget для загрузки рецептов, если они не были переданы по протоколу SFTP;
– распаковка рецептов на машинах;
– выполнение команды chef-solo для запуска Chef в режиме "только клиент". Пример кон-

фигурационного файла ec2.config имеет следующий вид:

```
--
:ami: ami-f95cba90
:instance_type: m1.small
:security_groups: mscherbakov
:keypair: mscherbakov
:key: /home/mike/keys/amazon/mscherbakov.pem
:availability_zone: us-east-1c
:EC2_URL:
:chef\textit{cooks}url: http://myserver/chef.tar.gz
```

Файл записан в формате YAML. Все параметры специфичны для облачного провайдера Amazon EC2, кроме `chef_cooks_url`. Этот параметр является необязательным. Если он присутствует, то должен содержать URL, по которому виртуальная машина сможет получить заархивированные рецепты Chef, выполнив команду wget.

Пример файла конфигурации машины формата JSON:

```
{
  "cloud": {
    "info": "Amazon EC2"
  },
  "recipes": [ "java6", "hadoop::namenode" ]
}
```

Атрибут `cloud` является необязательным, и лишь передает дополнительную информацию, к которой легко получить доступ во время выполнения рецепта. Атрибут `recipes` содержит массив рецептов для выполнения и является обязательным. Соблюдая формат JSON, возможно передать произвольные конфигурационные параметры, которые будут легко доступны из кода. Например, строка "Amazon EC2"может быть получена в коде рецепта следующим выражением:

```
node[:cloud][:info].
```

После получения информации об IP адресах всех запускаемых виртуальных машин, утилита Clorun перечитывает каждый из JSON файлов конфигурации (если их больше одного), дополняет их IP адресом, а также информацией о списке рецептов для запуска на каждой другой машине и ее IP адресом. Пример одного из полученных таким образом файлов имеет следующий вид:

```
{
  "cloud": {
    "info": "Amazon EC2",
    "name": "cassandra",
    "recipes": [ "cassandra": [ "204.236.196.147" ],
        "java6": [ "204.236.196.145", "204.236.196.147" ],
        "hadoop::namenode": [ "204.236.196.145" ] ],
    "roles": [],
    "ip": "204.236.196.145",
    "id": "i-c87175a0"
  },
  "recipes": [ "java6", "hadoop::namenode" ]
}
```

Модифицированные файлы сохраняются в директории `configs/<name_of_env>/`, где `<name_of_env>` — указанное параметром `-n` при запуске `clorun` имя кластера. Из полученных атрибутов достаточно легко получить IP адреса всех узлов, на которых будет выполнен конкретный рецепт. В коде рецепта это может выглядеть следующим образом:

```
cass_ips = node[:cloud][:recipes]["cassandra"] & \
    search(:node, "recipe:cassandra").map { |cfg| cfg["ipaddress"] }
```

В этом примере будет также произведена попытка поиска других узлов с рецептом `cassandra`, используя функцию search сервера Chef. Результатом будет объединение массивов IP адресов, полученных через переданные атрибуты утилитой Clorun, и найденными IP адресами на Chef сервере. В нашем случае Chef используется в режиме «соло», то есть без сервера, поэтому результат будет состоять только из IP адресов, переданных в виде атрибутов. Удобство именно такой записи в коде состоит в том, что позволяет без каких либо модификаций

работать рецепту в модели запуска Chef с сервером. Дополнительный атрибут «id», записанный в JSON файл, содержит идентификационный номер запущенной виртуальной машины, выданный облачным провайдером. Утилита использует этот id для остановки виртуальных машин.

После создания модифицированных конфигурационных файлов, утилита ожидает ответа серверов по SSH протоколу. Как только соединение установлено, в многопоточном режиме происходит передача файла конфигурации каждого из серверов из директории textttconfigs/<name_of_env>/, файла `solo.rb` и архива `chef.tar.gz` с рецептами Chef. Архив не передается, если `ec2.config` содержит атрибут `"chef_cooks_url"`. Файл `solo.rb` содержит конфигурационные параметры для Chef:

```
file\textit{cache}path "/root/chef"
cookbook_path "/root/chef/cookbooks"
role_path "/root/chef/roles"
```

После загрузки файлов, на сервере запускается Chef на выполнение рецептов. Утилита Clorun отображает на экране монитора стандартный вывод запуска Chef, и завершает свою работу только после окончания работы Chef на всех серверах.

Установка Hadoop, Elmer [8] или какого-либо другого научного программного обеспечения в разработанной системе развертывания сводится к написанию рецептов Chef и созданию конфигурационных файлов для работы утилиты Clorun. Разработанные авторами рецепты могут быть загружены с сайта [3], там же можно найти конфигурационные файлы утилиты Clorun для развертывания упомянутых приложений.

5. Заключение

В работе была представлена практическая реализация алгоритма автоматизированного процесса установки распределенного ПО в инфраструктуру облачного провайдера Amazon EC2, выполненная на базе Open Source фреймворка Chef и созданной авторами утилиты Clorun. Проведенные исследования показали возможность создания систем установки и управления инфраструктурой серверов, действующих без вмешательства человека в полностью автоматическом режиме, позволяя разворачивать несколько виртульных машин одновременно в облачной инфраструктуре. В случае сбоя по каким-либо причинам, не связанным с ошибкой в работе системы, самый простой способ разрешения — удалить виртуальный кластер, и попытаться создать его снова. Время создания кластера может варьироваться: около 5 минут для простой конфигурации серверов, около 10–15 минут для достаточно сложной, и дольше, например в случае пересылки большого объема данных для первоначального запуска. В работе использован Chef в качестве основного инструмента управления серверами, однако могут быть использованы и другие подобные фреймворки, удовлетворяющие теоретическим требованиям, например Puppet [7]. Для Puppet необходима дополнительная машина с установленным сервером Puppet, доступная для запускаемых клиентов. В дальнейшем планируется более детально исследовать соответствия теоретическим требованиям описанной системы установки на базе Chef, возможные сбои работы с облачной инфраструктурой и методы борьбы с ними. Кроме того, требуется дополнительное исследование на тему установки распределенных приложений со сложными связями, например, когда в процессе установки на одной из виртуальных машин потребуется, чтобы на другой уже был выполнен один из рецептов. Могут быть и более сложные ситуации, требующие распределенной блокировки.

Литература

1. Chef. Infrastructure Automation for the Masses: URL: http://www.opscode.com/chef (дата обращения: 14.12.2010).

2. Steve Traugott, Lance Brown. Why Order Matters: Turing Equivalence in Automated Systems Administration: URL: http://www.infrastructures.org/papers/turing/turing.html (дата обращения: 14.12.2010).

3. Scherbakov M. Public repositories of Mike Scherbakov: URL: http://github.com/mihgen (дата обращения: 14.12.2010).

4. Idempotence: URL: http://en.wikipedia.org/wiki/Idempotency (дата обращения: 14.12.2010).

5. Apache Hadoop: URL: http://wiki.apache.org/hadoop/ (дата обращения: 14.12.2010).

6. Эндю Хант, Дэвид Томас. Программист-прагматик. Москва: Лори, 2004. С. 22.

7. Docs: Introduction to Puppet: URL: http://docs.puppetlabs.com/guides/introduction.html (дата обращения: 14.12.2010).

8. Open Source Finite Element Software for Multiphysical Problems: URL: http://www.csc.fi/english/pages/elmer (дата обращения: 14.12.2010).

Virtual Dike and Flood Simulator: Parallel distributed computing for flood early warning systems*

N.B. Melnikova[1,2,3], G.S. Shirshov[1,3] ,V.V. Krzhizhanovskaya[1,2,3], N.N. Shabrov[1],

State Polytechnic University, St. Petersburg, Russia[1]
National Research University ITMO, St. Petersburg, Russia[2]
University of Amsterdam, The Netherlands[3]

The paper presents two simulation modules –Virtual Dike and Flood Simulator– developed for flood Early Warning Systems (EWS). The *UrbanFlood* EWS is a distributed system that (1) analyzes sensor data received in real-time from flood defenses (dikes, dams, etc.) and (2) simulates dike stability, breaching and flood propagation. Computational modules are invoked by workflow-based expert scenarios via the Common Information Space middleware. The Virtual Dike and Flood Simulator have been ported to the HPC Cloud system of SARA supercomputing centre in Amsterdam. Cloud system provides dynamic resource allocation and remote user control. Virtual Dike is a parallel FSI module for coupled simulation of porous flow and dike deformation, based on the finite element method. Efficiency of the distributed EWS and solver parallel performance are presented.

1. Introduction

Design of Early Warning Systems (EWS) for flood protection and disaster management poses a grand challenge to scientific and engineering communities. Development of an EWS includes [1]:

- Sensor equipment design, installation and technical maintenance in flood defense systems;
- Information and Communication Technologies in application to:
  - gathering, processing and visualizing sensor data;
  - developing Common Information Space (CIS) middleware for connecting sensor data, relevant documents, analysis tools, modeling software and advanced scientific visualization;
  - providing Internet-based interactive access to CIS for researchers;
- Development of computational models and simulation components for analysis of dike stability, evaluation of dike failure probability, prediction of flood dynamics and ways for evacuation;
- Development of a decision support system for public authorities and citizens that will help making informed decisions in case of emergency and in routine dike quality assessment, thus reducing flood risk and providing advanced tools for flood management.

Recent catastrophic floods around the world have spawn a large number of projects aimed at the development of stronger and "smarter" flood protection systems. Many projects, among which are FLOODsite, Flood Control 2015, International Levee Handbook [2], attempt to solve some of the EWS aspects listed above. The *UrbanFlood* EC FP7 project [3,4,5] is the first endeavor that unites the work on all the development aspects of a fully integrated EWS. One of the key challenges that we are concentrated on is the development of computational models. Here we describe two models: Flood Simulator and Virtual Dike.

Structural analysis of dike stability includes analyzing different possible failure mechanisms, such as macro-instability, surface erosion and piping. Macro-instability analysis requires modeling of flow through porous media and deformations in the dike -a highly nonlinear coupled fluid-structure interaction problem. Elastic and elasto-plastic models are commonly used to describe soil behavior [6]. In this work, elastic rheological model has been used for dike stability analysis.

In this paper, we briefly describe the *UrbanFlood* EWS workflow and implementation of the distributed system using HPC Cloud of SARA supercomputing centre. Further, we describe the

*The work is supported by [a] *UrbanFlood* EC FP7 project N 248767, theme ICT-2009.6.4 ICT for Environmental Services and Climate Change Adaptation www.urbanflood.eu;
[b] 'Leading Scientist Program' of the Government of the Russian Federation, under contract 11.G34.31.0019.

Virtual Dike and Flood Simulator computational models and parallel efficiency tests for the structural analysis simulations.

2. *UrbanFlood* early warning system computational workflow

Computational workflow of the EWS is organized as follows (see Figure 1): The 'Sensor Monitoring' component receives sensor data from the sensors installed in the dike. Raw sensor data is filtered by the 'Artificial Intelligence Anomaly Detector' that identifies abnormalities in dike behavior or sensor malfunctions. The 'Reliability Analysis' module calculates the probability of dike failure in case of abnormally high water levels or an upcoming storm and extreme rainfalls. If the failure probability is high then the 'Breach Simulator' predicts the dynamics of a possible dike failure, calculates the water discharge through the breach and estimates the total time of the flood. After that, the 'Flood simulator' models the inundation dynamics. Information from all the modules is visualized in the 'Decision Support System', which works as an intelligent interactive interface from the raw data to the users.

For the advanced research into dike stability and failure mechanisms, the 'Virtual Dike' component is available for experienced users. It provides more fundamental and accurate simulations, but requires substantial computing resources, ranging from a powerful computer cluster to supercomputers in case of 3D fully coupled fluid-structure interaction dynamics. The simulation modules and visualization components are integrated into Common Information Space. They are accessed from the interactive graphical environment of the multi-touch table or the web-based application.

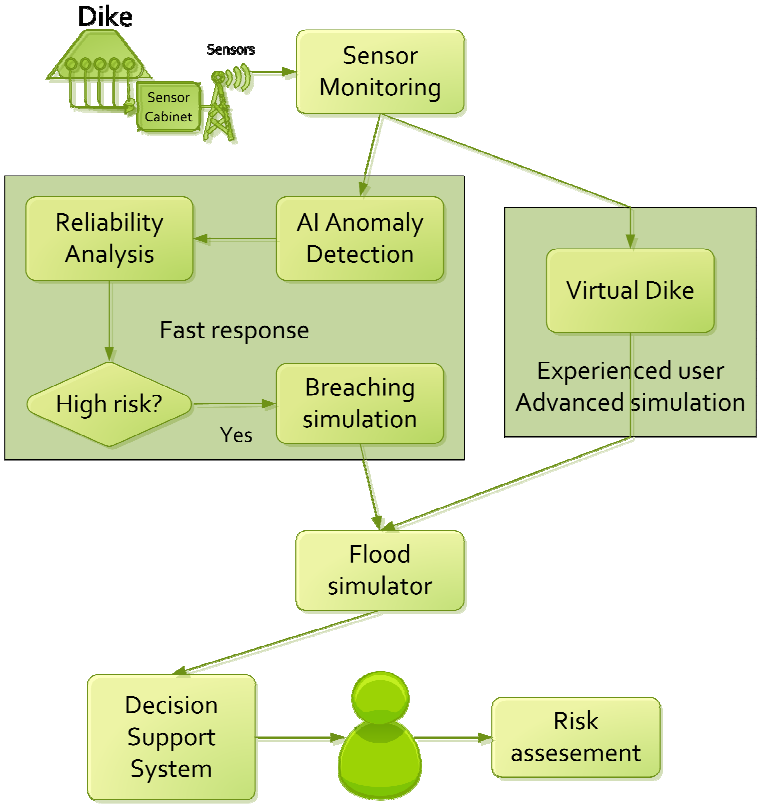


Figure 1. *UrbanFlood* early warning system workflow

3. Implementation of the distributed EWS and cloud computing

The EWS components have been wrapped as plug-ins and integrated in the *UrbanFlood* Common Information Space (CIS). CIS is a generic framework for creating and hosting EWS systems. CIS is used for:

- Monitoring: data is collected in real time from sensors and fed to an EWS.
- Analysis: one or more EWS-specific applications/workflows are invoked to perform analysis of sensor data streams, such as anomaly detection or simulation based on expert models.
- Value judgment: the outcome of this analysis is judged according to EWS-specific rules in order to estimate the risk level or determine the necessity of taking action.
- Advice/act: if mandated by the value judgment, further EWS-specific applications/workflows may be invoked in order to recommend actions for users interacting with a Decision Support System (DSS), or automatically take action to influence the system.

CIS runs on the integration platform PlatIn by *UrbanFlood*. PlatIn extends the engines and introduces additional capabilities which exploit the cloud environment managed by the Dynamic Resource Allocation (DyReAlla) module.

The EWS components Flood Simulator and Virtual Dike have been deployed on the clouds of the SARA BiG Grid High Performance Computing and e-Science Support Center [7]. The cloud is hosted on a 128-core cluster with the following characteristics: 16 compute nodes; CPU dual quad-core 2.2 GHz; 24 GB RAM per node; 500 GB local hard disk; 100 TB backup storage, network 1 Gb/s per node; 20 Gb/s aggregated connection from the cluster to storage. SARA uses OpenNebula open source cloud computing management toolkit. The Virtual machine software is KVM. Simulations are run under Ubuntu Linux.

4. Virtual Dike computational module

Virtual Dike is an advanced multiscale multi-model simulation lab for expert users and model developers. This virtual lab is used for validation of all the models involved in the modeling cascade, and serves as a research field for experiment planning and understanding the underlying physical processes influencing dike stability and failure. Comparison of simulation results with the experimental data allows determining the material properties and computational model parameters that best represent real-life dikes, with all their inhomogeneities and special features. In the first stage of the project, we have studied the structural stability of the LiveDike, a sea dike in Eemshaven. LiveDike is protecting a seaport in Groningen. This dike has been equipped with sensors, and data stream is available in real-time via the LiveDike Dashboard [8]. Pore pressure and dike inclination sensors are placed in four dike cross-sections. These cross-sections have been simulated in 2D models under tidal water loading. Simulation results have been compared with the pore pressure sensors data in order to calibrate soil properties, so that virtual and real sensors agree.

4.1 Modeling approach

The problem requires modeling of coupled fluid-structure interaction with non-linear dike material properties. The fluid part of the model describes the dynamics of flow through porous soil, with Richards equation for wetting and drying of the area above phreatic surface. Van Genuchten model [9] is used to describe the properties of unsaturated soil. As pores' volume expand with the expansion of the media, water content increases. This process is taken into consideration with additional source term in the right hand side of the Richard's equation:

$$(C + \theta_e S)\frac{\partial p}{\partial t} + \nabla \cdot [-\frac{K_S}{\mu} k_r \nabla (p + \rho g z)] = \frac{\partial \varepsilon}{\partial t}, \quad (1)$$

where p is pressure, [Pa]; $C = C(p)$ is specific moisture capacity [1/Pa], given by formula $C = \partial\theta/\partial p$; $\theta_e = \theta_e(p)$ is effective water content; S is a storage coefficient, [1/Pa]; t is time, [s]; μ is dynamic viscosity of water, [Pa·s]; K_S is saturated permeability, [m^2]; $k_r = k_r(p)$ is relative permeability, given

by Van Genuchten formulas; g is standard gravity, [m/s^2]; ρ is water density [kg/m3]; z stands for coordinate of vertical elevation, [m]; ε is volume expansion coefficient.

The structural part of the model describes deformation dynamics of the dike under tidal pressure load, gravity and volumetric pore pressure load obtained from flow simulation. Linear elastic constitutive relation is used for describing sand and clay properties:

$$\begin{cases} \nabla \cdot (\underline{\sigma} - p\underline{E}) + \rho_s \underline{g} = 0 \\ \underline{\sigma} = \lambda \varepsilon \underline{E} + 2\mu \underline{\varepsilon} \end{cases}, \quad (2)$$

Dike stability is evaluated by the Mohr-Coulomb failure criterion [10]. Harmonic approximation of water level sensor signal is applied as input parameter for the flow boundary conditions (to define pressure level at the sea side). At the land side, water stays at the constant average see level. The remaining boundaries are treated as impermeable. In the structural task, corresponding transient pressure is applied at the sea side and constant pressure below average sea level at the land side. The base of the dike is fixed. The remaining boundaries are free from structural loads.

4.2 Implementation and simulation results

Partial differential equations (1), (2) are solved by the finite element method using a commercial software package Comsol [12]. All the simulations are transient. Implicit time integration scheme is applied. Simulations are non-linear due to non-linear constitutive behavior of the soil and fluid-structure coupling phenomenon. Iterative Newton-Raphson method is used to linearize differential equations at each time step. Direct parallel UMFPACK and PARDISO solvers have been applied for solving system of linear algebraic equations within each iteration.

Simulated pore pressure field and structural displacements (at some chosen moment of time) are shown in Figure 2 and Figure 3, correspondingly. Pressure distribution is almost hydrostatic. Pressure changes with elevation mostly. The structural displacements are composed of: a) static soil settlement under gravity load (maximal at the top of the dike) and b) transient displacements resulting from tidal pressure at the seaside and volume pore pressure load. Total displacements are maximal at the top of the dike due to gravity settlement component.

Comparison of pore pressure dynamics for real and virtual sensors corresponding to section #1 of the dike is presented in Figure 4. Real sensor signals are shown with bold lines (left plot – sensor 1E4, right plot – sensor 1G2). “Virtual” sensor signals (obtained from simulation) are shown with thin lines. Comparison of the data for sensor 1G2 shows that the real sensor is placed in a less permeable zone than the zone of the virtual sensor is. In virtual model, permeability values will be corrected and G2 sensors will be surrounded with less permeable soil. Some local inhomogeneities of the soil must be taken into consideration in further simulations.

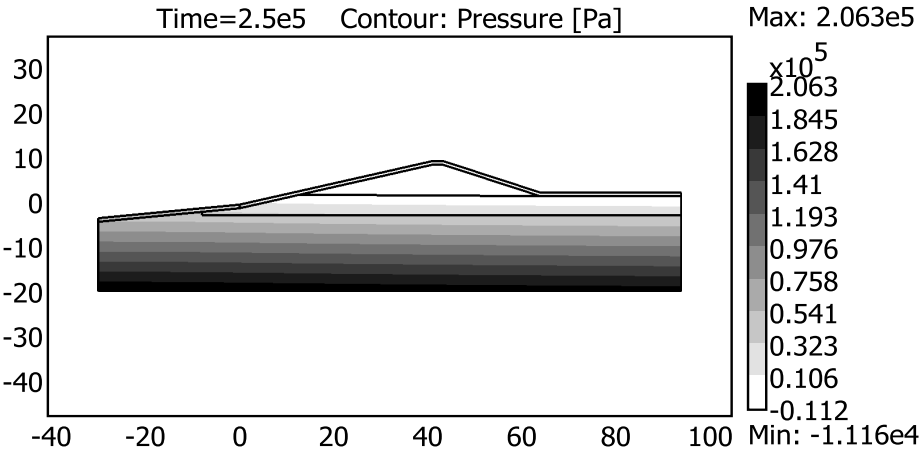


Figure 2. Pore pressure field in the dike

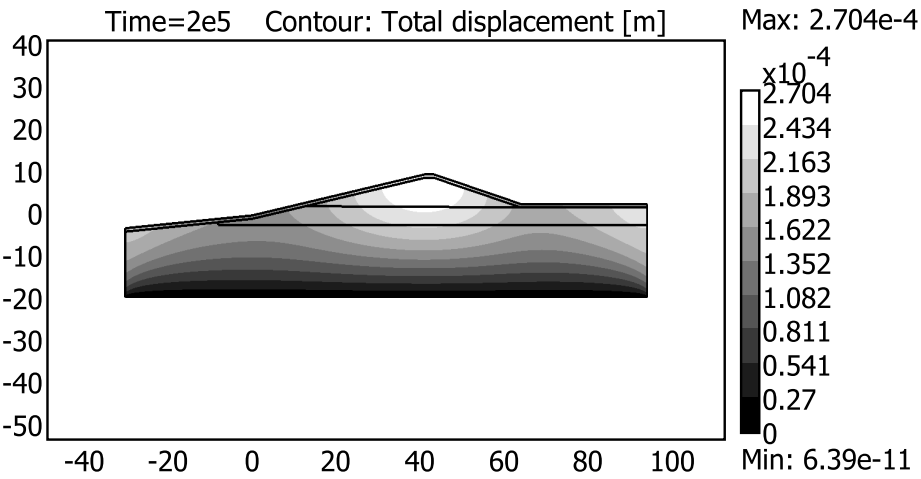


Figure 3. Structural displacement field in the dike

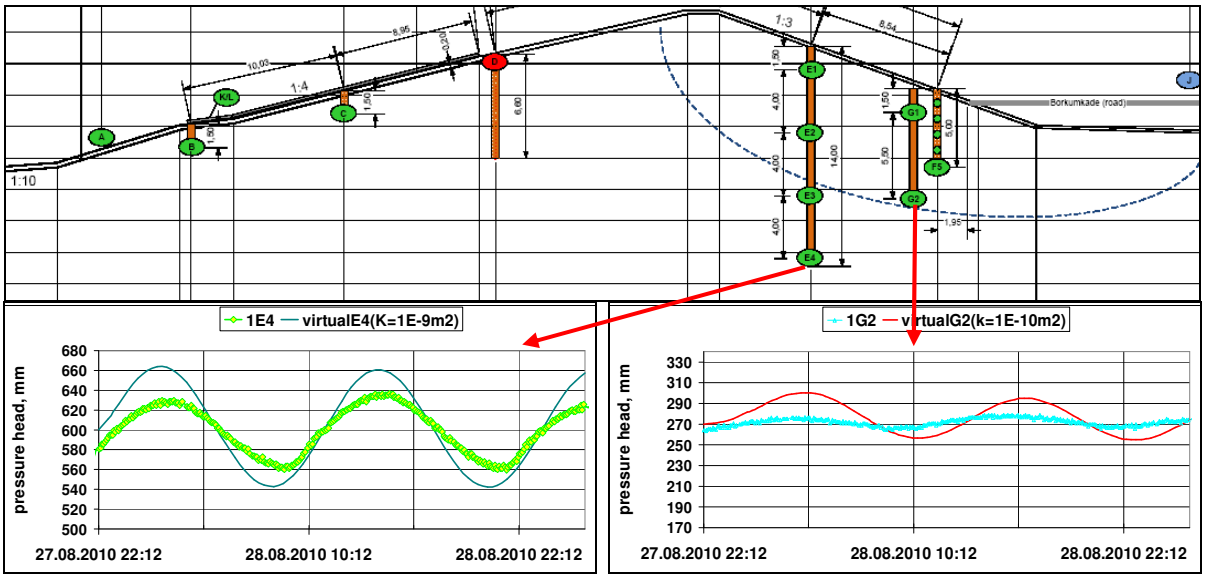


Figure 4. Comparison of pore pressure dynamics for real sensors and virtual sensors

5. Flood Simulator computational module

Flood Simulator module predicts flood dynamics. Given a set of points where a dike breach has occurred, and given the amount of water which will flow through each of these points, this model calculates the water heights of flooded areas over time. The model comprises two main components: a pre-process and a hydraulic simulation.

The objective of the pre-process is to generate the computational mesh which consists of a series of Impact Zones that are based on the floodplain topography and therefore typically irregular in shape. Inputs to this process are the floodplain topography in the form of a Digital Terrain Map (DTM). Creation of the calculation mesh speeds up the process of the simulation (regarding to the DTM data usage).This approach could be used in probabilistic flood risk analysis where multiple runs are required, or in real time situations (flood forecasting), where the model run time is critical.

A hydraulic simulation algorithm performs the following operations at each time step:

1. Boundary Conditions treatment
2. Sort Impact Zones on water level
3. Loop on every wet Impact Zone
   3.1. Calculate outlet discharge towards neighbours (using Manning or weir equations).
   3.2. Discharge limiters

3.3. Use inlet discharge from neighbours
3.4. Transfer volumes
3.5. Calculate velocity

4. Treatment of Impact Zones newly flooded
5. Update water level (volume/level curve) for all wet Impact Zones

Input data is specified in the form of a hydrograph representing a time dependence of water flow rate through the breach in the dike calculated by another simulation module (see Figure 5).

The output of the flood simulator is the table which contains the time series of depth and velocity in each Impact Zone (the saving time-step is defined by the user independently of the computational time-step). The output data is visualized. In the Figure 6 the flooding of the Science Park area of Amsterdam is shown.

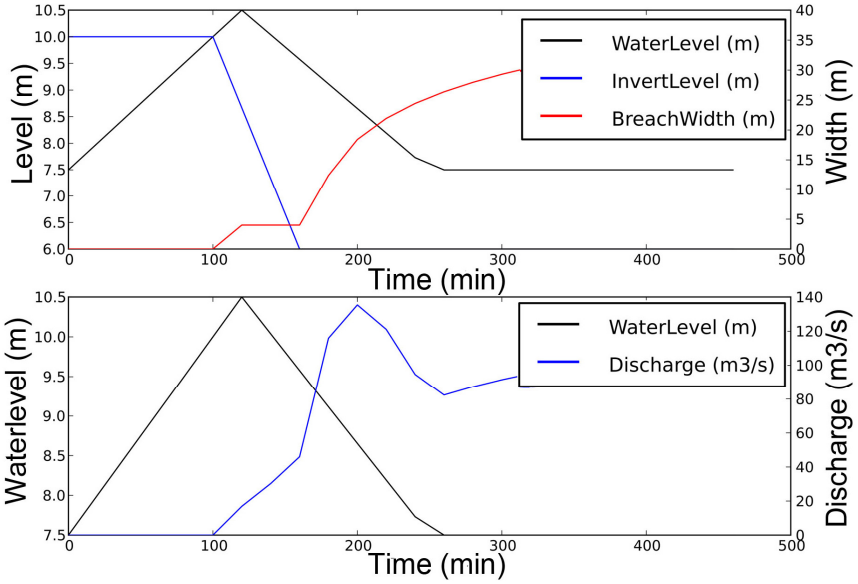


Figure 5. Time dependence of breach width and water amount coming through the breach

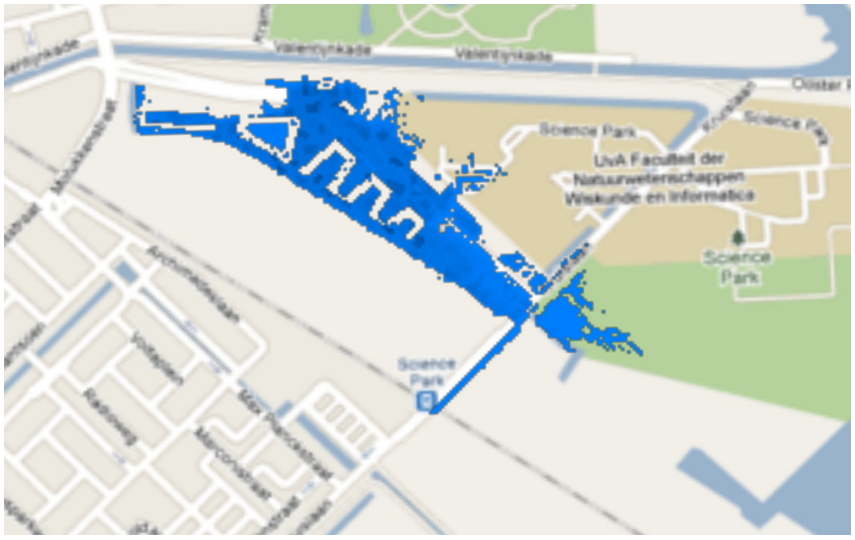


Figure 6. Inundation of the Science Park area - visualization

6. Performance tests of the distributed EWS system and parallel modules

The EWS prototype has been extensively tested and presented during the *UrbanFlood* Workshop [11]. Different EWS components are running on several servers located in 4 countries: Russia, the Netherlands, Poland and the United Kingdom. The modules communicate via the Common Information Space middleware. A user interface implemented on the Microsoft *Surface* multi-touch table provides access to all EWS components and allows interactive simulation steering. Tests of the EWS performance showed a real-time response of system components to user manipulations.

6.1 Parallel efficiency of the Virtual Dike module

For a relatively coarse mesh with 60 000 degrees of freedom (DOF) and maximum element size of 1 m, 2D porous flow simulations typically require up to 2 GB RAM in serial mode. Coupled fluid-structure problem requires up to 8 GB of memory in serial mode. The amount of occupied system memory increases when using parallel mode.

Comsol supports two modes of parallel operation: distributed mode and shared memory mode [12]. Distributed mode employs MPI library for process communications; it can be used on several nodes of a Linux or Windows cluster. Shared memory mode uses shared address space for inter-process communication. It can only be used on a single multi-core or multi-processor computational node. Presently one simulation uses only one computational node of SARA HPC Cloud, with dual quad-core CPU. Shared-memory parallelism has been tested for porous flow uncoupled modeling, employing UMFPACK and PARDISO solvers. Shared-memory parallelism has been used with the default processor usage mode. Besides the default mode, Comsol allows three specific modes to control processor usage: throughput mode (when several different processes are running actively at the same time), turnaround mode and owner mode (both are recommended for usage when no other processes than Comsol are active). We plan to study the efficiency of these modes in future work.

Parallel performance results are presented in Figure 7. PARDISO solver is recommended by Comsol as providing the best parallel efficiency in comparison with the other solvers. The results confirm this only for “small” problem size with number of DOF=60 000. For larger problem sizes UMFPACK scalability was better. At the same time, UMFPACK speed itself was significantly lower for large problem sizes (see Table 1). Hence, for large problems, it is preferable to use PARDISO as it is almost 50% faster.

Computational time varies from 30 min to 7 hours (Table 1), depending on the problem size, number of cores and chosen solver. The parallel speedup of a particular solver first improves as the number of DOF grows (as computational work portion grows higher relative to the portion of information exchange). After that, we get speedup saturation or even decrease for number of cores np equal to 8. The reason of low scalability and cases of decrease in speedup for np=8 may lie [a]: in high

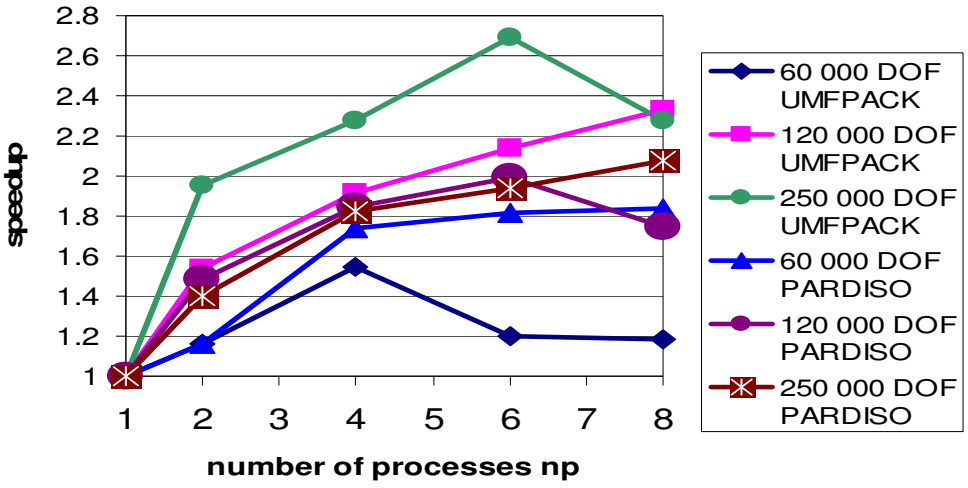


Figure 7. Virtual Dike module: Parallel performance test results

Table 1. Virtual Dike module: parallel simulation time, hours

number of cores	DOF 60000		DOF 120000		DOF 250000	
	UMFPACK	PARDISO	UMFPACK	PARDISO	UMFPACK	PARDISO
1	0.90	0.99	2.30	2.01	7.11	4.93
2	0.78	0.86	1.50	1.35	3.64	3.53
4	0.58	0.57	1.20	1.09	3.12	2.71
6	0.75	0.55	1.08	1.01	2.64	2.54
8	0.76	0.54	0.99	1.15	3.12	2.37

process synchronization costs and [b]: in a relatively high portion of sequential operations included in the algorithm. The solver writes down solution on the hard disk with specified periodicity. As the number of DOF grows, this sequential output work increases. Of course this I/O option could be withdrawn from the simulation tests but we were interested in a real speedup which would happen in working conditions. The output from the solver is going to get to CIS and other computational modules via the hard disk files.

In future research we plan to use a profiler to measure the cost of process synchronization, for different number of cores. A large portion of the computational engine in Comsol relies on BLAS (basic linear algebra subprograms) libraries which are included with the package. By default, Comsol employs MKL (Math Kernel Library) on Intel processors,. This default option can be overridden. An issue for future testing of parallel performance is to explore the efficiency of the alternative BLAS libraries.

7. Conclusions and future work

Design of the EWS for flood protection is in progress. System middleware and basic analysis tools of the EWS have been implemented. Overall efficiency of the distributed EWS has proved to be good. The system provides dynamic resource allocation and controls task scheduling.

We have built computational modules for modeling porous flow and deformations in the dikes; and simulation of inundation in case of dike failure. Parallel speedup of the structural Comsol solver was not high in our tests. The second problem of using Comsol is high memory requirements, which would demand several nodes of HPC cloud for performing distributed 3D simulations. As the primary aim of the analyses tools is to be able to perform simulation in a real-time, a possible solution is to develop a tailored finite-element code for modeling porous flow and structural displacements.

We plan to integrate the Virtual Dike in the Common Information Space to receive sensor input automatically and to produce real-time simulation results for displaying them graphically on a MultiTouch Table. For structural analysis of the dike, elasto-plastic rheological model will be implemented. We plan to perform 3D analysis. Flood condition is to be investigated, including porous flow modeling and dike macro-stability analysis. Parallel efficiency tests are to be continued in 3D case in “pure” conditions (without I/O operations), for different processor usage modes and different BLAS libraries.

Acknowledgements. This work is supported by the EC FP7 project *UrbanFlood*, grant N 248767, and a grant from the 'Leading Scientist Program' of the Government of the Russian Federation, under contract 11.G34.31.0019. The project is carried out in close collaboration with a number of organizations and individuals (listed in alphabetical order): AlertSolutions, particularly Erik Peters; BiG Grid, advanced ICT research infrastructure for e-Science; Deltares, particularly Andre Koelewijn; HRW Wallingford, particularly Ben Gouldby and Julien Lhomme; IJkDijk Association; Rijkswaterstaat, Ministerie van Verkeer en Waterstaat; SARA Computing and Networking Services, particularly Tom Visser and Floris Sluiter; TNO, particularly Jeroen Broekhuijsen and Erik Langius; UvA IBED-CGE, particularly Lourens Veen; UvA GIS , particularly Studio Guido van Reenen; WaterNet, particularly Rob van Putten; Waterschap Noorderzijlvest, particularly Christiaan Jacobs.

References

1. V.V. Krzhizhanovskaya, G.S. Shirshov, N.B. Melnikova, R.G. Belleman, F.I. Rusadi, B.J. Broekhuijsen, B.P. Gouldby, J. Lhomme, B. Balis, M. Bubak, A.L. Pyayt, I.I. Mokhov, A.V. Ozhigin, B. Lang, R.J. Meijer. *Flood early warning system: design, implementation and computational modules*. Proceedings of the International Conference on Computational Science, ICCS 2011. Procedia Computer Science 2011
2. Flood Control 2015 http://www.floodcontrol2015.com, FLOODsite http://www.floodsite.net, The International Levee Handbook project http://www.leveehandbook.net/about-project.asp
3. *UrbanFlood* EU FP7 project http://www.urbanflood.eu
4. V.V. Krzhizhanovskaya. A roadmap to multiscale modeling of flood defense systems: from sand grain to dike failure and inundation. Proceedings of ASME 2010 Computers and Information in Engineering Conference IDETC/CIE 2010, Montreal, Canada. Paper # DETC2010-28967
5. B. Gouldby, V.V. Krzhizhanovskaya, J. Simm, A.G. Hoekstra. Multiscale modelling in real-time flood forecasting systems: From sand grain to dike failure and inundation. Procedia Computer Science, V. 1, N 1, ICCS 2010, May 2010, p. 809
6. A.R. Koelewejn, M.A.Van. Monitoring of the test on the dike at Bergambacht: design and practice. Proceedings of XIII ECSMGE 2003, Prague, Czech Republic.
7. SARA Computing and Networking Services.
8. Livedike dashboard http://livedijk-www.ict.tno.nl/
9. M.T. van Genuchten. A closed form equation for predicting the hydraulic conductivity of unsaturated soils. Soil Science Society of America Journal 44: 892-898.
10. A. Verruijt. Soil Mechanics. Delft University of Technology, 2001. 315 pages.
11. Joint UrbanFlood & SSG4Env Workshop, Thursday 11 & Friday 12 November 2010 – Amsterdam, the Netherlands http://urbanflood.eu/urbanFloodWorkshop2010.aspx
12. Comsol Installation and Operations Guide http://math.nju.edu.cn/help/mathhpc/doc/comsol/install.pdf

Прямое численное моделирование турбулентной свободной конвекции, развивающейся во времени у нагретой вертикальной стенки*

А.Г. Абрамов[1], В.Д. Горячев[2], Е.М. Смирнов[1]

Санкт-Петербургский государственный политехнический университет[1],
Тверской государственный технический университет[2]

Представляются результаты прямого численного моделирования (DNS) турбулентной свободной конвекции воздуха, развивающейся во времени у нагретой вертикальной пластины, проведенного в условиях, отвечающих известным из литературы экспериментам. Использовалась вычислительно эффективная методика Time-developing DNS, основанная на подмене пространственного развития моделируемого течения развитием во времени. Получены обширные данные об эволюции полей скорости и температуры, пространственно-временных характеристиках возникающих в пограничном слое вихревых структур. Изучено влияние на условия ламинарно-турбулентного перехода в свободно-конвективном пограничном слое уровня начальных возмущений в среде. Выполнено сравнение осредненных и пульсационных составляющих рассчитанных полей с экспериментальными профилями для турбулентных режимов конвекции, показавшее хорошую степень согласованности результатов.

1. Введение

Появившийся в последние годы у научного сообщества широкий доступ к современным высокопроизводительным суперкомпьютерам и кластерам открывает новые качественные возможности по проведению высокоточных расчетов ресурсоемких задач из различных отраслей науки и техники, представляющих большой интерес, как с фундаментальных, так и с прикладных позиций. В области вычислительной гидродинамики и тепломассообмена одной из таких задач является моделирование переходного и турбулентного свободно-конвективного течения, развивающегося под действием сил плавучести у нагретой вертикально ориентированной пластины. Особый интерес представляет изучение "тонкой" структуры ламинарно-турбулентного перехода в свободно-конвективном пограничном слое, анализ влияния различных факторов на условия и общую картину переходного процесса.

До недавнего времени систематические исследования по проблеме проводились преимущественно на основе экспериментального подхода (см., например, [1, 2]), а численное моделирование выполнялось в двумерной постановке на основе осредненных по Рейнольдсу уравнений Навье-Стокса (Reynolds-Averaged Navier-Stokes, RANS) с привлечением различных моделей турбулентности [3]. Дальнейший прогресс в изучении течений данного класса, как и многих других, связан с применением вихреразрешающих подходов, требующих значительных вычислительных затрат, но при этом обладающих большей степенью информативности и потенциально большей точностью. Эти подходы, предполагающие проведение расчетов в трехмерной нестационарной постановке, известны как метод моделирования крупных вихрей (Large Eddy Simulation, LES) и прямое численное моделирование (Direct Numerical Simulation, DNS). Их применение в сочетании с современными методиками и программными средствами обработки результатов расчетов (визуализация трехмерных полей, статистический анализ и т.д.) позволяет существенно дополнить экспериментально полученную информацию новыми обширными данными и глубокими знаниями об особенностях переходного и турбулентного режимов конвекции, характеристиках эволюционирующих в процессе перехода в свободно-конвективном пограничном слое вихревых структур, которые трудно идентифицировать в эксперименте и невозможно описать в рамках вычислительных моделей более низкого уровня.

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (гранты №08-08-00977 и № 11-07-00135).

В настоящей работе представляются результаты прямого численного моделирования переходного и турбулентного режимов свободной конвекции воздуха, развивающейся во времени у нагретой вертикальной пластины. Соответствующие расчеты выполнены в условиях, отвечающих общепризнанным экспериментам [2].

2. Постановка и вычислительные аспекты задачи

Прямое численное моделирование турбулентной свободной конвекции несжимаемой ньютоновской среды с постоянными физическими свойствами проводилось на основе системы нестационарных трехмерных уравнений Навье-Стокса, дополненных уравнением баланса энергии [1]. Эффекты плавучести в поле силы тяжести учитывались в приближении Буссинеска.

Расчеты в размерной постановке выполнялись с использованием эффективной вычислительной методики, известной как Time-developing (Temporal) DNS (TDNS), которая основана на подмене пространственного развития моделируемого течения временным. Другими словами, время выступает здесь в роли координаты, в направлении которой происходит развитие основного течения. Метод TDNS позволяет, в сравнении с пространственным прямым численным моделированием (Spatial DNS), существенно экономить на размере расчетной области и, как следствие, на размерности расчетной сетки. Многообещающие результаты применения метода TDNS к расчету свободно-конвективного течения рассматриваемого типа представлены недавно в работе [4].

Весьма обстоятельное экспериментальное исследование развития свободно-конвективного пограничного слоя в воздушной среде у нагретой изотермической вертикальной пластины выполнено в работе T. Tsuji и Y. Nagano [2]. Медная пластина, генерирующая в эксперименте восходящее конвективное течение, имела высоту 4 м и ширину 1 м. Основной акцент в исследованиях был сделан на изучение режима развитой турбулентности, для которого производились измерения осредненных и пульсационных составляющих полей скорости и температуры по нормали к пластине (при нескольких значениях продольной координаты). Настоящие расчеты проводились в условиях экспериментов [2], при этом углубленно изучался переходный режим конвекции, которому было уделено меньшее внимание в экспериментах.

Расчетная область имела форму куба с длиной ребра 0.24 м (рис. 1) и была покрыта расчетной сеткой размерностью 3.2 млн. ячеек. Предварительный анализ экспериментальных данных показал, что выбранный по оси y размер расчетной области позволит смоделировать пространственное развитие пограничного слоя примерно до половины реальной высоты пластины, где уже развивалась турбулентная конвекция воздуха. Молекулярное число Прандтля полагалось равным Pr = 0.71.

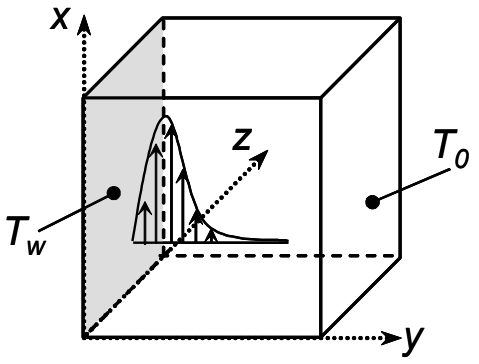


Рис. 1. Схема к постановке задачи.

Расчет начинался с состояния нулевых средних скоростей воздушной среды при одинаковой по всей расчетной области температуре T_0, равной 16°C. По результатам предварительно проведенного теоретического и численного анализа шаг по времени при выполнении основных расчетов был выбран равным 0.002 с. Температура нагретой вертикальной стенки, T_w, составляла 60°C. Параллельная стенке внешняя граница рассматривалась как выходная с заданным на ней постоянным давлением и температурой, T_0. В соответствии с методом TDNS, по однородным координатам (вертикальной, x, и трансверсальной, z) ставились условия периодичности. Следует заметить, что для рассматриваемой задачи применение DNS с моделированием пространственного развития конвекции потребовало бы существенного (более чем в 8 раз) увеличения размера расчетной области в продольном направлении (x), так что идентичная по параметрам расчетная сетка содержала бы более 25 млн. ячеек.

Изучалось влияние на процесс ламинарно-турбулентного перехода в свободно-конвективном пограничном слое контролируемого уровня начальных возмущений в среде. Это осуществлялось путем задания в расчетах определенной интенсивности и спектра вырождаю-

щейся изотропной турбулентности. Использовался массив данных по результатам описанного в [5] численного моделирования вырождения однородной изотропной турбулентности в кубической полости с периодическими границами. Выполнялась переинтерполяция предоставленных авторами [5] распределений пульсаций компонент скорости на топологически идентичную сетку, использованную в настоящих расчетах. При этом в разных рассчитанных вариантах для заданного спектра турбулентности варьировался уровень интенсивности пульсаций скорости, количественной характеристикой которых являлась величина среднеквадратичных пульсаций скорости $\overline{u_i'^2}$ (вычисленная путем осреднения по всей расчетной области). Начальное поле температуры всегда полагалось невозмущенным.

При сопоставлении расчетных и экспериментальных данных в качестве масштаба длины использовалась интегральная толщина развивающегося во времени скоростного пограничного слоя $\delta = \int_0^\infty u / U_m dy$, где U_m - максимальная скорость. Профиль вертикальной компоненты скорости, по которому вычислялось значение δ для каждого момента времени, находился путем осреднения рассчитанного поля скорости по однородным направлениям.

Дополнительная серия расчетов, нацеленная на исследование влияния вертикального размера области, была проведена при удвоенной ее высоте с соответствующим увеличением размерности расчетной сетки. Анализ результатов показал, что изменение размера области практически не повлияло на предсказываемые положения точки ламинарно-турбулентного перехода. Вместе с тем, в осредненных и пульсационных составляющих полей для турбулентного режима конвекции наблюдались некоторые различия. Это составляет один из предметов исследования в настоящее время.

3. Вычислительные средства: программный CFD-комплекс SINF и система визуализации HDVIS

Расчеты проводились с использованием развиваемого на кафедре гидроаэродинамики СПбГПУ программного комплекса (ПК) SINF (Supersonic to INcompressible Flows) [6], предназначенного для решения трехмерных уравнений Навье-Стокса. Данный CFD-код исследовательской направленности позволяет проводить расчеты стационарных и нестационарных, до- и сверхзвуковых течений жидкости или газа, развивающихся, в общем случае, в областях сложной геометрии. Пространственная дискретизация осуществляется по методу контрольного объема со вторым порядком точности. Для получения нестационарных решений применяется неявная схема второго порядка по физическому времени. На каждом временном шаге организуется итерационный процесс установления с введением искусственной сжимаемости. Параллельная версия ПК SINF базируется на применении декомпозиции расчетной области на блоки, концепции SPMD и коммуникационной библиотеке MPI. Параллелизованный код обладает высокими показателями эффективности параллелизации вычислений [7].

Расчеты выполнялись на вычислительном кластере кафедры гидроаэродинамики СПбГПУ, в настоящее время состоящем из 8 узлов, каждый из которых включает по два четырехъядерных процессора Intel Xeon Quad (модель - E5405, тактовая частота - 2.33 ГГц) и объединенных в общее вычислительное пространство сетью Gigabit Ethernet. В представляемых расчетах был задействован только один из узлов кластера, при этом типичная продолжительность одного варианта расчетов составляла около трех суток (для кубической области). Базовая расчетная сетка разделялась на 8 блоков одинаковой размерности, что обеспечивало сбалансированную загрузку вычислительных ядер узла.

Верификация корректности реализации метода TDNS в ПК SINF проводилась на задаче о развитии ламинарного конвективного пограничного слоя на нагретой вертикальной пластине. Профили скорости и температуры, построенные в автомодельных переменных, полностью согласуются с результатами работы [4] и очень близки к известному аналитическому решению.

Визуализация трехмерной структуры течения выполнена с помощью специализированной графической системы HDVIS (High Definition VISualization) [8]. В HDVIS обработка рассчитанных полей осуществляется на множестве данных, получаемых непосредственно с кластера.

На их основе строится и корректируется сцена визуализации: с выбором видов, сечений и выделенных областей анализа, с построением карт распределения векторных и скалярных полей, с генерацией изоповерхностей, отображением линий тока и т.д. Возможна обработка полученной первичной информации с выполнением встроенных в систему операций по расчету вторичных полей (производных скалярных и векторных полей, тензорных величин), с их связыванием и выделением особенностей течения, идентификацией вихревых структур, например, на основе вычисления Q- или λ_2-критерия и т.п. Автоматизация визуальной обработки позволяет монтировать представительный набор изображений, создавать законченные иллюстрации и анимационные ролики, комфортно вести сравнительный графический анализ результатов.

4. Результаты расчетов и обсуждение

4.1 Влияние начальных возмущений на ламинарно-турбулентный переход

Влияние уровня начальных возмущений в среде на ламинарно-турбулентный переход в свободно-конвективном пограничном слое изучалось при проведении вариантных расчетов для нескольких значений начальной интенсивности пульсаций поля скорости, в том числе и при отсутствии каких-либо задаваемых возмущений.

На рис. 2а показано изменение во времени интегральной толщины пограничных слоев для трех вариантов с разными значениями интенсивности начальных пульсаций (отмеченных цифрами на рис. 2в), на которых отчетливо виден момент резкого нарастания δ, соответствующий началу ламинарно-турбулентного перехода. На рис. 2б приведены соответствующие тем же вариантам кривые, отражающие зависимость от времени плотности теплового потока. На этих кривых также легко идентифицируется начало переходного процесса, а в турбулентном режиме конвекции наблюдаются низкочастотные колебания во времени теплового потока, которые обусловлены присутствием в течении крупномасштабных вихревых структур.

Для нахождения интересующего практику критического значения числа Грасгофа Gr_x выполнялся отдельный расчет пространственного развития двумерного стационарного ламинарного свободно-конвективного пограничного слоя. Это позволяет определить зависимость δ(x) и, по данным рис. 2а, найти критическое значение продольной координаты (отсчитываемое от нижнего края пластины). Вычисленные таким образом пространственные положения точек перехода для различных рассчитанных вариантов представлены на рис. 2в в форме зависимости от величины $\overline{u_i'^2}$ (экспериментально полученное значение $x_{cr} \approx 0.7$ м). Видно, что при отсутствии начальных физических возмущений в среде имеет место существенное "затягивание" перехода ($x_{cr} \approx 3.2$ м), а при наложении начальных возмущений увеличивающейся интенсивности пространственное положение точки перехода, после резкого смещения к нижнему краю пластины, стабилизируется, приближаясь к экспериментальному.

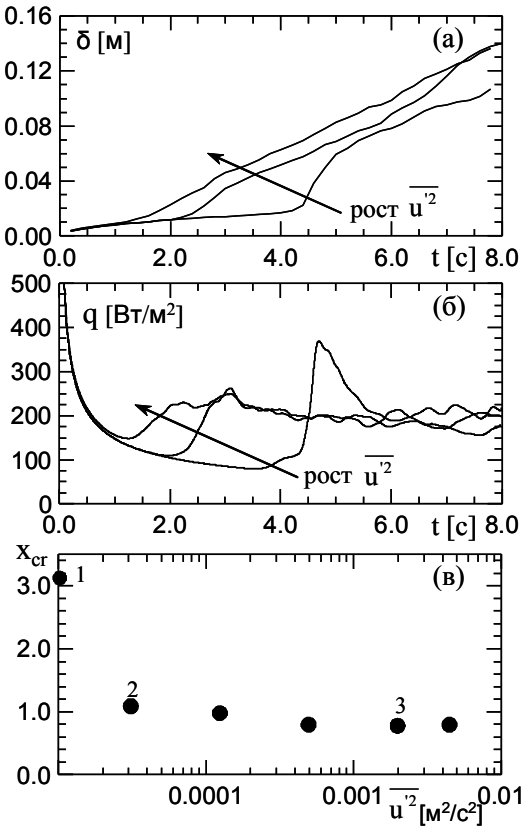


Рис. 2. Влияние начальных возмущений на ламинарно-турбулентный переход.

4.2 Анализ эволюционирующих полей течения

Развитие во времени моделируемого течения демонстрируется с использованием современных методик визуализации на рис. 3 (для области удвоенной высоты и $\overline{u_i'^2}$ =4.9·10^{-4} м^2/с^2). Основной акцент при этом сделан на максимально наглядном представлении процесса эволюции зарождающихся в свободно-конвективном пограничном слое вихревых структур: от момента начала ламинарно-турбулентного перехода до установления режима развитой турбулентности.

Численная шлирен-визуализация (рис. 3а) использована для показа полей градиента температуры в пограничном слое (в параллельном стенке сечении y ≈ 0.015 м, слева, и в нормальном к ней вертикальном сечении, справа). Иллюстрируется общая структура и интенсивность вихревых образований на разных этапах развития конвекции. Пространственное представление об эволюции во времени крупномасштабных вихрей дает рис. 3б, где изображены изоповерхности Q-критерия (Q = 200 1/с^2), широко используемого при визуализации турбулентных образований [9]. Там же показаны распределения локального теплового потока на пластине, заметная неоднородность которого связана с проникновением в пристенную область интенсивных вихревых структур. На рис. 3в иллюстрируются особенности течения в окрестности вихря, формирующегося на стадии переходного режима конвекции (показана изоповерхность Q = 300 1/с^2 и семейство линий тока на фоне распределения модуля завихренности).

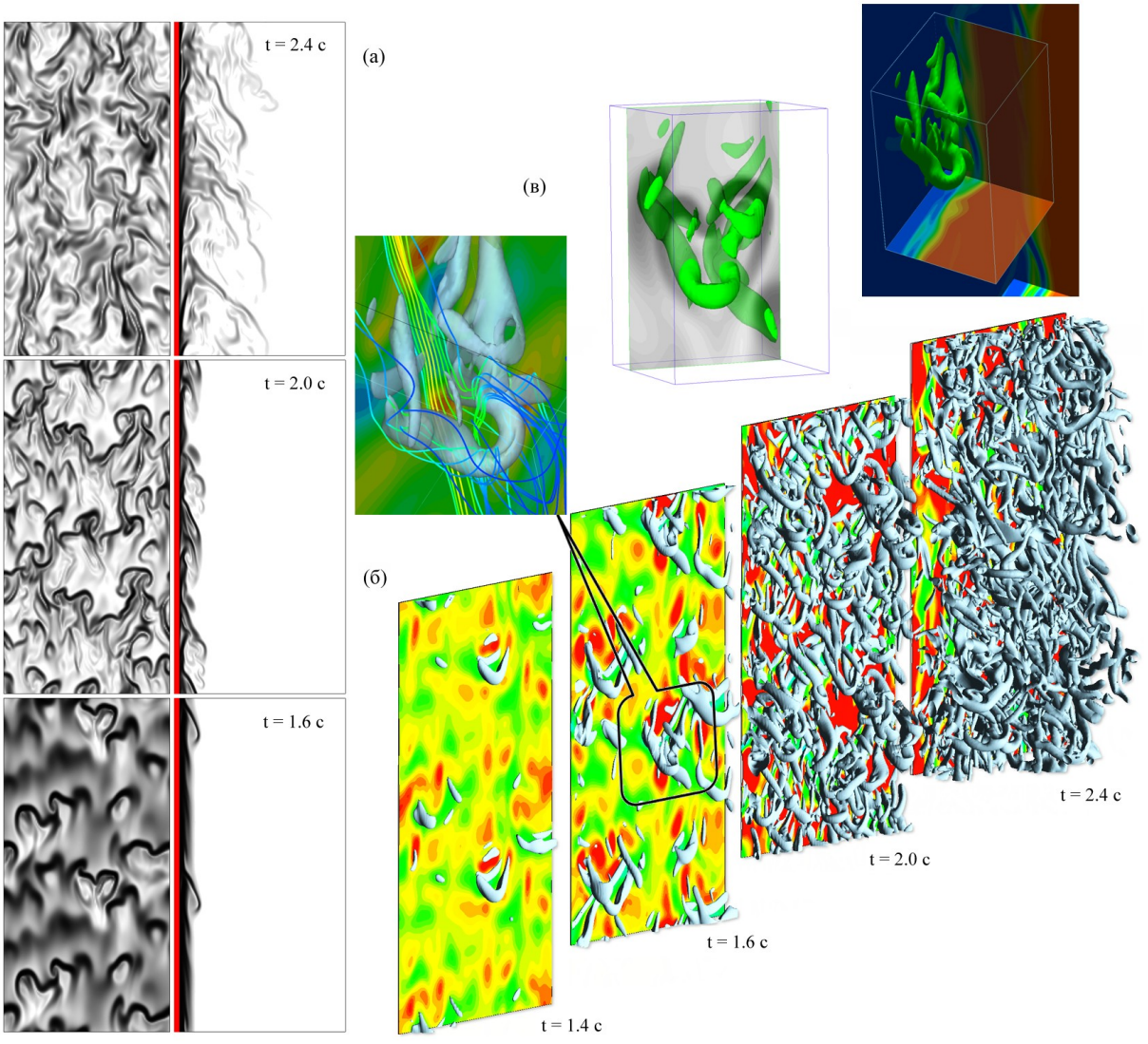


Рис. 3. Визуализация развивающегося во времени свободно-конвективного течения воздуха у нагретой вертикальной пластины.

4.3 Сопоставление с экспериментальными данными

График рассчитанной зависимости числа Нуссельта Nu_δ от числа Грасгофа Gr_δ, построенных по интегральной толщине скоростного пограничного слоя δ, показан на рис. 4. Для вариантов с заданием начальных возмущений среды наблюдается хорошее согласие с экспериментальными значениями и данными TDNS [4] во всех режимах конвекции.

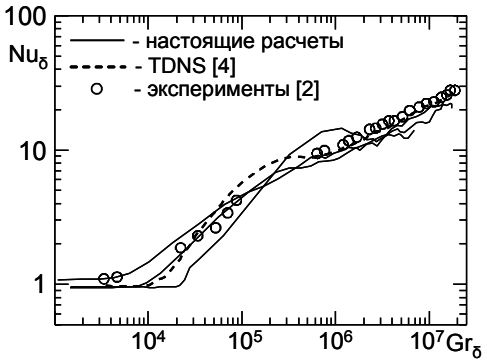


Рис. 4. Зависимость $Nu_\delta(Gr_\delta)$: сравнение с экспериментами [2] и расчетами [4].

Согласованное сопоставление результатов расчетов с данными экспериментов [2] для турбулентного режима конвекции также подразумевает переход от временного описания процесса к пространственному. Для этих целей, как показано ранее, в каждый момент времени вычислялась величина δ и сравнивалась со значениями, рассчитанными по экспериментальным профилям продольной скорости (измеренным для нескольких значений координаты x). Сравнение результатов производилось при совпадении расчетного и экспериментального значений δ.

На рис. 5 для $Gr_x = 1.55\cdot10^{10}$ (что соответствует $Gr_\delta = 1.45\cdot10^6$) производится сравнение нормированных экспериментальных [2] и расчетных профилей температуры, вертикальной скорости и пульсационных характеристик турбулентной конвекции (для варианта с $\overline{u_i'^2} = 4.9\cdot10^{-4}$ $м^2/с^2$). Соответствующие профили получены в расчетах путем пространственного осреднения полей (по двум однородным направлениям). Можно видеть хорошее согласие рассчитанных профилей основного течения и температуры, а также среднеквадратичных пульсаций температуры с данными экспериментов [2]. Следует отметить, однако что расчетное значение максимальной скорости U_m примерно на 40% больше экспериментального. Это, в частности, приводит к существенному занижению нормированных значений турбулентного потока тепла и касательного напряжения Рейнольдса. Примечательно, однако, что в размерном виде расчетные данные по турбулентным потокам близки к экспериментальным. Аналогичные выводы можно сделать и по результатам работы [4].

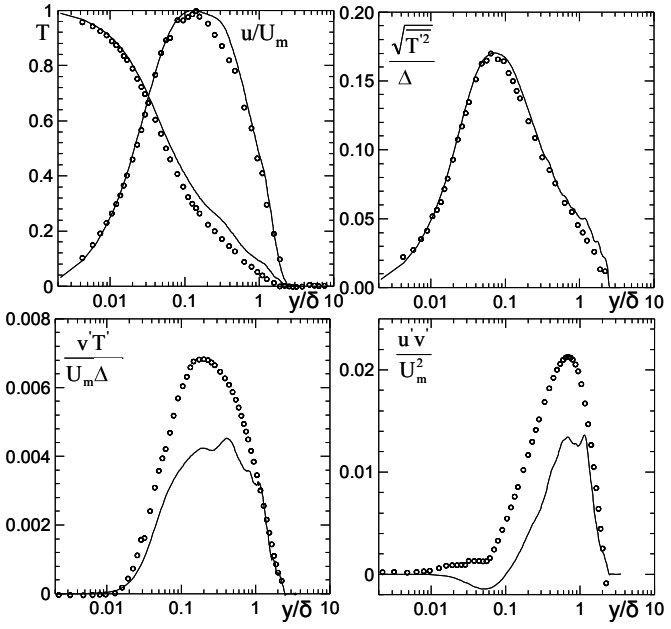


Рис. 5. Сравнение рассчитанных профилей осредненных и пульсационных величин с данными эксперимента [2].

5. Заключение

В условиях известных экспериментальных данных выполнено прямое численное моделирование турбулентной свободной конвекции воздуха, развивающейся у нагретой вертикальной пластины. При проведении расчетов последовательно апробирована и применена методика Time-developing DNS, которая показала свою высокую вычислительную эффективность. Получены новые данные, отражающие степень влияния уровня начальных возмущений на процесс ламинарно-турбулентного перехода в свободно-конвективном пограничном слое. Посредством

развитой визуализации течения проанализирован процесс эволюции зарождающихся в пограничном слое вихревых структур. Проведенное сравнение рассчитанных, осредненных по однородным направлениям полей скорости и температуры и пульсационных характеристик турбулентной конвекции с экспериментальными профилями, показало удовлетворительную степень согласованности результатов.

Литература

1. Гебхарт Б., Джалурия Й., Махаджан Р., Саммакия Б. Свободноконвективные течения, тепло- и массообмен. В 2-х книгах, кн. 2: Пер. с англ. - М.: Мир, 1991. - 528 с.
2. Tsuji T., Nagano Y. Characteristics of a turbulent natural convection boundary layer along a vertical flat plate // Int. J. Heat Mass Transfer. 1988. Vol. 31. No. 8. P. 1723-1734.
3. Henkes R.A.W.M. Natural-convection boundary layers. Ph.D. Thesis, Delft University of Technology, Delft, The Netherlands, 1990. 199 pp.
4. Abedin M.Z., Tsuji T., Hattori Y. Direct numerical simulation for a time-developing natural-convection boundary layer along a vertical plate // Int. J. Heat Mass Transfer. 2009. Vol. 52. No. 19-20. P. 4525-4534.
5. Shur M., Spalart P.R., Strelets M., Travin A. Detached-Eddy Simulation of an airfoil at high angle of attack // Engineering Turbulence Modelling and Experiments. 1999. Vol. 4. P. 669-678.
6. Смирнов Е.М., Зайцев Д.К. Метод конечных объемов в приложении к задачам гидрогазодинамики и теплообмена в областях сложной геометрии // Научно-технические ведомости СПбГПУ. 2004. №2(36). С. 70-81.
7. Smirnov E.M., Abramov A.G., Ivanov N.G., Smirnov P.E., Yakubov S.A. DNS and RANS/LES-computations of complex geometry flows using a parallel multiblock finite-volume code / In.: Parallel CFD: Advanced Numerical Methods Software and Applications, Elsevier, 2004. P. 219-226.
8. Горячев В.Д., Балашов М.Е., Смирнов Е.М. Визуализация нестационарных течений в вычислительной гидродинамике // Труды международной суперкомпьютерной конференции "Научный сервис в сети интернет: суперкомпьютерные центры и задачи" (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, 2010. С. 50-55.
9. Hunt J.C.R., Wray A.A., Moin P. Eddies, stream, and convergence zones in turbulent flows / In Proceeding of the Summer Program, Center for Turbulence Research, NASA Ames/Stanford Univ., 1988. P. 193-208.

Возможности оценки сложности параллельного программирования

В.Л. Авербух,[1] М.О. Бахтерев,[1] А.Ю. Казанцев,[1] В.В. Косенко[2]

ИММ УрО РАН[1], Уральский Государственный Университетет[2]

Работа предлагает постановку задачи оценки параллельного программирования. Рассматривается ряд подходов к оценке сложности параллельных программ, а также оценки необходимых трудозатрат программиста для создания надежного и эффективного параллельного кода.

1. Введение. Метрики программирования

Утверждение о том, что параллельное программирование сложно, стало общим местом в соответствующей специальной литературе еще с 80-ых годов XX века. Вместе с тем, необходимо разобраться, чем же оно сложно и как в этом плане соотносятся различные парадигмы параллельного программирования. Анализ сложности программирования полезен, прежде всего, при выборе инструментария для разработки новых программных комплексов. Одновременно интерес к такому анализу проявляют создатели новых средств параллельного программирования.

Известны основные аспекты языка программирования применимые к любому языку. Прежде всего, это уровень и охват (широта области применения) [1]. Уровень языка можно определить как универсальную меру количества деталей, которые программист должен указать для достижения желаемого результата. Язык является процедурным, если пользователю нужно последовательно, по шагам определять задание. Количество и размер требуемых шагов различается в разных процедурных языках. Для достижения одного и того же результата в высокопроцедурных (низкоуровневых) языках, например, ассемблерах требуется много малых детализированных шагов, а в низкопроцедурных языках (языках высокого уровня), требуется заметно меньше намного более крупных шагов с значительно меньшей детализацией.

Охват (широта области применения) языка подразделяет языки от языков общего или широкого назначения до узко специализированных языков.

Отметим, что ассемблер, в общем-то, может быть использован практически, в любых случаях. Таким образом, у него наиболее широкий охват, он имеет большую область применения чем, все остальные языки. Однако это не делает его наиболее удобным средством программирования.

В литературе хорошо представлены методы измерения качества и сложности программ.

Наиболее простыми являются количественные метрики программы, такие как среднее число строк для функций (классов, файлов) или среднее число строк, содержащих исходный код для функций (классов, файлов). Существуют подходы к оценке к количественной оценке сложности понимания программы, трудоемкость кодирования программы, уровню языка. Для анализа сложности программирования в рамках метрик Холстеда [2] используется понятие информационного содержания программы и оцениваются необходимые интеллектуальные усилия при ее разработке. Также существует метрика, характеризующая число требуемых элементарных решений при написании программы. Выведены формулы для данных метрик, в которые подставляются параметры уже готовых программ. Существуют также подходы к разработке метрик параллельных программ и метрик производительности параллельных программ. В последнем случае метрикой называется зависящая от времени функция, характеризующая некоторые аспекты производительности параллельной программы. Примеры простых метрик производительности - нагрузка центрального процессора, использование памяти, число операций с плавающей запятой. Надо учитывать также измеримые (объективные) показатели, например, скорость программирования, количество ошибок и время, необходимое для их обнаружения, время, необходимое для получения результата (работающей программы). Существуют и субъективные параметры типа “эстетического” удовлетворения процессом программирования, уровня утомление и пр. Среди возможных составляющих метрик сложности следует учитывать субъективные оценки того или иного языка. Можно отметить, что эстетическое удовлетворение от программирования получается, когда программа просто написана — код красиво смотрится, ничего лишнего (минимизированы абстракции) — и понятно на взгляд что происходит без

дополнительных комментариев. Эмоциональное удовлетворение возможно связано с простотой написания программы, когда минимизировано внимание к техническим сторонам программирования. Более объективным является учет таких параметров как длина программы, скорость ее написания, скорость отладки и пр.

В связи с нашими целями мы хотим получить не метрики качества или производительности уже готовых программ, а сравнение того насколько одна парадигма параллельного программирования сложнее или проще другой.

Для характеристики прагматических аспектов языков используются понятие адекватности. В случае средств параллельного программирования можно использовать понятие адекватности в описании параллелизма.

2. Явный и неявный параллелизм

Различается явный и неявный параллелизм. Неявный параллелизм программы учитывается распараллеливающими компиляторами. Распараллеливание компилятором представляется очень интересным. Мы пишем простой последовательный код, а компилятор за нас его преобразует в “правильный” параллельный. Однако на практике, чтобы компилятор мог что-либо распараллелить, нужно учитывать модель будущего параллелизма. Для получения эффективно работающей программы необходимо знание о методиках распараллеливания, которые использует данный компилятор. В тоже время больших результатов в плане эффективности может достичь специалист, не только владеющий навыками параллельного программирования, но и обладающий знаниями в данной предметной области, разбирающийся в сути использованных вычислительных методов и алгоритмов.

Существует возможность реализовать неявное распараллеливание за счет функциональных языков программирования. Распараллеливание основано на семантике понятия функции, не требующей вычисления ее аргументов в каком-то определенном порядке. Однако, отсутствие “привычного” императивного порядка исполнения порождает сложности в реализации ввода/вывода. В рамках функционального программирования нет адресов, указателей, перезаписываемых областей памяти, а только управление, данные и их метки.

Явный параллелизм задается создателем алгоритма и программистом. На всех этапах человек должен держать в голове ответ на вопрос “где и как это будет работать параллельно”. В рамках различных парадигм параллельного программирования существуют как весьма “актуальные” и широко известные и повсеместно используемые в настоящее время библиотеки - MPI и OpenMP, так и менее “актуальные” (по разным причинам), например, High Performance Fortran (HPF) (и близкие к нему среды программирования) или Cilk [3].

Можно заметить, что явный высокоуровневый параллелизм по данным (например, HPF) прост в использовании, но применим не для всех приложений. Более современные многонитевые среды (например, OpenMP) – несколько сложнее в использовании. Можно также указать на накладные расходы при обеспечении синхронизированного доступа к общей памяти и на недостаточную масштабируемость программ. При реализации Cilk’а была проведена большая работа по уменьшению накладных расходов. Авторы добивались удобства и простоты использования среды параллельного программирования. Вместе с тем, существуют проблемы переносимости на системы типа NUMA, связанные с моделью памяти, и на кластерные комплексы.

Поддерживающий парадигму обмена сообщениями MPI можно отнести к низкоуровневым средам (хотя и более высоким, чем прямое распараллеливание при помощи операторов fork/join).

Ниже мы проанализируем возможность оценки различных аспектов языков программирования для некоторых “параллельных” парадигм.

3. Анализ параллелизма

Каковы же возможные критерии для качества параллельного программирования? На что необходимо обращать внимание при проектировании средств параллельного программирования, и при их анализе? Сперва отметим, что не бывает операций без данных (или ещё точнее, без памяти, в которой эти данные записаны), а данные без операций существуют. Представляется, что это часть ментальной модели любого программиста. Если мы посмотрим на уровень интерфейса процессора, то увидим, что любая операция им совершаемая совершается на уровне работы с некоторыми областями хранения данных. Это могут быть регистры, биты состояния машины,

ячейки памяти. То есть, имеет место создание кода в том стиле, когда в исходном тексте указываются и операции, и указания на то, как получить данные для этих операций. Переменная не обязательно должна ассоциироваться с конкретной ячейкой памяти, она может быть плавающей, но имя переменной для компилятора всегда означает указание на то, как извлечь данные. Так обстоят дела в широко используемых языках программирования, будь то императивные языки или функциональные. Логические языки отрывают нас от простой ментальной конструкции (***у меня есть данные, я хочу их преобразовать в другие***), заменяя её более сложной конструкцией (***у меня есть логические высказывания, я хочу сделать из них всевозможные выводы и посмотреть, смогу ли я вывести из них некое другое высказывание***). Для некоторых задач такой уровень абстракций вполне адекватен, но многие алгоритмы не выражаются напрямую таким образом. (Или выражаются, хоть и понятно, но очень громоздко.)

Попытки отделить поток управления от данных предпринимались неоднократно. Такие языки действительно дают очень короткий код, действительно позволяют этот код повторно использовать. Однако так можно задавать лишь достаточно примитивные схемы композиции операций.

Распараллеливание подразумевает следующие действия: (1) выделение независимых участков кода; (2) описание (неким образом) процедуры получения обрабатываемых данных в этих участках кода; (3) независимое получение некоторых наборов данных, результатов параллельных шагов вычисления; (4) описание процедур сбора этих данных.

Когда мы пишем последовательные программы, мы не задумываемся о независимости и передачи данных, хотя в реальности и в этом случае происходит множество внутренних передач данных (современный процессор - это сложная сеть). Работу по организации передач, руководствуясь именами переменных, берёт на себя компилятор, а потом и процессор (руководствуясь кодом). Код явно не разбивается на независимые (напомним, что независимость и параллельность в контексте программирования - синонимы) части. Однако достаточно независимыми участками являются функции, на чём и построена основная концепция таких систем, как Cilk. Не особо заостряя внимание на этом, программист в традиционных языках весьма активно разбивает код на независимые участки, разбрасывает по ним данные, а потом собирает эти данные вместе.

Код не надо разбивать на независимые участки, потому что в программе определяются функции, то есть, в контексте последовательной программы, всё это делается неявно. Относительная свобода и легкость кодирования в этом случае связана с тем, в описание операций можно свободно вставлять ссылки на данные (имена переменных), которые, собственно, являются тоже кодами определённых операций для компилятора.

Отправная точка нашего анализа заключается в идее ***оценки уровня связи операторной части программы с данными***. Благодаря свободной манипуляции именами переменных у нас есть возможность достаточно легко производить декомпозицию и композицию привычных, последовательных программ. Именно она постоянно проводится при разбиении кода на строки, процедуры, функции, при перестановки всего этого местами, при распределении по модулям, и т.п. По нашему мнению такая свобода получается благодаря возможности свободно ссылаться на данные в выражениях. Для параллельного программирования ключевой момент заключен в композиции и декомпозиции кода.

Рассмотрим с позиций предложенного критерия известные среды MPI и OpenMP.

4. Анализ MPI

MPI своей системой демонов, библиотек, интерфейсов и пр. позволяет:

(1) абстрагировать программу от конкретной операционной системы и её особенностей и “тонких” мест, возникающих при работе со средой передачи данных (сетью). В рамках одной операционной системы абстрагироваться можно от конкретной сетевой среды.

(2) автоматически организовывать запуск процессов пользователя на машинах MPI-кластера, автоматически их нумеровать и объединять в MPI-сеть. Это, резко упрощает программирование по сравнению с низкоуровневым использованием tcp/ip-сокетов.

Важно также то, что пользователю позволяется писать всю программу в виде одного исполняемого файла, и запускать её на всей системе.

(3) MPI практически всё взаимодействие сводит к двум основным операциям обмена данными Send (*отправить*) и Recv (*получить*). Отправка и получение происходит на уровне сообщений.

Сообщения различаются системой по тройке (номер процесса-отправителя (rank), номер коммуникатора (comm), тэг (tag)). Чтобы получить сообщение, отправленного при помощи вызова Send(rank, comm, tag, ...), нужно использовать вызов Recv(rank, comm, tag,...) (*Запись вызовов схематична.*)

Благодаря MPI стало возможно сравнительно легко программировать для распределённых систем. Именно его использование открыло дорогу супервычислениям. Однако программирование на MPI низкоуровневое и требует существенных усилий для того, чтобы превращать ***данные*** для одних процессов в ***данные*** для других процессов. При этом, коды производящие это преобразование (управление памятью + сам MPI + схемы согласования сообщения) негибки. Перекраивать архитектуру вычисления очень сложно. Имеют место также “потери” в работе с данными по сравнению с традиционным последовательным программированием. С MPI мы теряем ссылки на данные. Более того, в какой-то мере мы теряем саму концепцию данных. Дело в том, что, сообщения не ведут себя как данные. Они ведут себя как нечто, что переписывается из адресного пространства одного процесса в адресное пространство другого. И после удачного переписывания исчезает. Сообщение - это не данные. Содержимое сообщений не может прямо использоваться в формулах, описывающих наше вычисление. Нельзя свободно модифицировать и составлять код, используя ссылки на эти данные в нужных местах программы, для того чтобы связывать процесс вычисления воедино. Сообщения существуют только в момент коммуникации, а превращать их в данные, которыми можно оперировать в выражениях - это уже задача программиста. Процедура этого превращения не такая простая: нужно оперировать памятью, оперировать MPI, и как-то устанавливать в двух процессорах соответствие между тройками (rank, comm, tag) и ссылками на доступные для обработки данные. В результате, при программировании на MPI мы очень сильно теряем в возможностях по композиции и декомпозиции программ. Теоретически, там полная свобода, но практически эта свобода даётся слишком большими усилиями, и поэтому, при практическом использовании MPI очень часто применяется ограниченно, с использованием только гораздо менее эффективных механизмов широковещательной рассылки данных типа Allgather. Анализируя популярность этого метода с позиций нашей основной идеи, можно сказать, что это самый простой способ превращать разбросанные по процессам блоки данных, в данные, которые достаточно свободно можно использовать в вычисляющих выражениях. (Просто в каждом процессе всё собирается в один буфер, свой для каждого процесса, но являющийся копией всех таких буферов в системе.)

Таким образом, достоинства MPI связаны с тем, что он скрывает сетевую сложность. Недостатки связаны с слишком большими трудозатратами, требующимися для того, чтобы превращать поток сообщений в данные.

5. Анализ OpenMP

Что даёт нам использование OpenMP? Прежде всего, (кажущуюся?) свободу от параллельного программирования. Как мы уже указывали, основная идея заключается в том, чтобы возложить работу по генерации кода для параллельных вычислений с общей памятью на компилятор, оставив программисту обязанность давать компилятору подсказки. Некоторые компиляторы могут включать режим трансляции с распараллеливанием даже без участия программиста, сами разбираясь в том, какие участки можно выполнять параллельно, какие нет, какие надо немного трансформировать, используя, например, политопные модели. Важным достоинством OpenMP является то, что мы можем взять уже существующую последовательную программу и попытаться проинструктировать компилятор о том, какие участки кода можно параллельно выполнять, а какие нет.

При этом модель выполнения у OpenMP достаточно проста. Есть основной поток исполнения, который, доходя до параллельного кода, запускает несколько нитей. Эти нити выполняют параллельный код. Основной поток исполнения тоже выполняет определённую часть вычислений. Потом все нити опять сходятся в одной точке, из которой продолжается выполнение последовательное. Имеет место использование простого принципа fork/join.

Укажем, однако, на ограничения данного подхода. OpenMP работает только с кодом. Он не занимается особым анализом данных. Нет возможности данные как-то перемешивать. Предполагается, что основная задача заключается в создании кода. А на данные особого внимания не обращается, тем более, они всё равно находятся в общей памяти и легко доступны.

OpenMP ограничивает контроль программиста и его свободу в манипулировании потоками данных и в композиции параллельных участков. Понижается понимание происходящего, потому что принципы распараллеливания достаточно сложны, а его детали скрыты. Особое внимание необходимо уделять управлению памятью. Сложно отлаживать ошибки, возникающие при директивах компилятору преобразовать некий принципиально “непараллельный” код в параллельный. (Хотя какие-то возможности предохраняющие от этого существуют.) Более того, в рамках OpenMP простые концепции (например, запустить процедуру умножения блоков матрицы на N процессов, с определенными блоками для процесса i) приходится выражать непрямым способом. Необходимо правильно расположить данные в массивах, дать правильные указания на распараллеливание цикла, а потом надеяться, что это даст эффект.

Кроме того, существуют известные архитектурные ограничения. Как известно, OpenMP не приспособлен для работы на кластере. Приходится его использовать вместе с MPI, что усложняет всё дело. MPI требует виртуозной работы с буферами в памяти, OpenMP скрывает детали кода, который с этими буферами работает, и надо как-то подгонять первое к неявным правилам работы второго.

6. Заключение

Как уже отмечалось, нас интересует возможность оценки сложности параллельного программирования. Именно программирования, а не самих готовых параллельных программ, несмотря на то, что и эти метрики также представляют значительный интерес. Наша работа представляет собой постановку задачи на исследование.

Кроме рассмотренных выше основных парадигм параллельного программирования существуют и другие подходы к распараллеливанию процессов.

Среди них следует обратить внимание на язык bluespec [4], в принципе созданный для проектирования электронной аппаратуры, но используемый и для параллельного программирования.

В ряде источников он характеризуется как функциональный язык, но это не совсем верно. Bluespec – пример языка асинхронного программирования (или иначе - программирования по событиям). В нем изначально заложена параллельная модель вычислений.

Программа на bluespec language содержит модули, имеющие интерфейсы. Модули содержат набор правил, с помощью которых описываются события, вызывающие реакцию программы. В языке поддерживается двухуровневая параллельность - на уровне модулей (код разных модулей может выполняться одновременно) на уровне операций внутри правил (если используем параллельную композицию 'a | a'). Модуль рассматривается как некий ресурс. То есть для некоторой задачи модуль X пишется так, чтобы на практике можно было использовать сразу несколько таких модулей одновременно. Разработчики языка bluespec, в частности, сообщают, что текст на этом языке на порядок короче и понятнее, чем на C++, хотя сама программа уступает коду на C++ по эффективности.

Еще одна парадигма параллельного программирования предлагается в рамках проекта RiDE [5]. Рассматривается методика и технические средства для программирования в параллельных распределённых средах. Цель разработки - упростить процесс создания параллельных программ, и сделать это не в ущерб эффективности исполнения вычислительных кодов. RiDE - это низкоуровневая система, обеспечивающая, несмотря на эту низкоуровневость, перенос языка композиции/декомпозиции с именованием данных и вычислений на уровень распределённой системы. Естественная декомпозиция операций, выраженная достаточно явным образом на языке RiDE, затем выполняется средой программирования по возможности параллельно. Сама декомпозиция может быть и не параллельной. Она может выполняться последовательно из-за специфичного потока данных в ней, но это не выходит за рамки RiDE. Параллельное же исполнение возникает само собой естественным образом, если это позволяют зависимости по данным. Конечно, параллелизм и здесь зависит от действий программиста. Программист может контролировать процесс исполнения, он знает архитектуру распараллеливания. Интерфейс к ней хоть и, не такой прямой, как в MPI, но всё же позволяет управление этим процессом. Прототип среды программирования показал работоспособность заложенных в ее основу идей. Интерес представляет большая компактность кода по сравнению с кодом, написанном на MPI.

В связи с анализом процесса распараллеливания возникает ряд соображений, связанных с визуальными языками параллельного программирования. Казалось бы, визуальное представление

параллельных программ жизненно необходимо из-за сложности самого параллельного программирования. Работы в данном направлении начались, практически, одновременно с разработкой систем визуального программирования [6] и достаточно активно продолжаются по сей день. Однако серьезных результатов не наблюдается. Очевидно, что попытки полностью перевести все операции параллельных языков в визуальную форму (как это было сделано в [7]) не дают желаемого эффекта. Возможно, что причины неудач связаны с тем, что визуализация должна быть адекватной задаче и описываемому процессу, а универсальный, чисто визуальный язык может породить излишнюю сложность и абстракцию в конкретной задаче. Не ясно как визуально отображать действия и операции. Возникают также проблемы с визуальным представлением данных, так необходимых для гибкой композиции вычислительных модулей.

Мы проанализировали возможности оценки распараллеливания с позиций оценки уровня связи операторной части программы с данными. Оценка сложности программирования в рамках той или иной парадигмы является важной, (хотя и вспомогательной) задачей, связанной с выбором инструментария программирования, с оценкой направлений развития области разработки сред параллельного программирования.

Кроме рассмотренных подходов к анализу параллелизма в рамках различных парадигм параллельного программирования, возможны и другие методики исследования, например, изучение ментальных моделей распараллеливания и методов их отображения в программу. Эти вопросы связаны с проблематикой психологии программирования и требуют дополнительного изучения.

Нами предполагается дальнейший анализ проблем параллелизма и поиск возможных способов оценки средств параллельного программирования.

Литература

1. Холстед М.Х. Начало науки о программах. М.: Финансы и Статистика, 1981.
2. Chang S.-K. Visual Languages: A Tutorial and Survey // Visualization in Programming. (Lecture Notes in Computer Science 282). Berlin. Springer-Verlag. 1987. Pp. 1-23.
3. Frigo M., Leiserson C. E., Randall K.H. The implementation of the Cilk-5 multithreaded language // PLDI '98: Proceedings of the ACM SIGPLAN 1998. 1998. New York, pp. 212-223.
4. Bluespec. Courses and Research Papers and Articles. http://www.bluespec.com/why-bluespec/technology-references.htm.
5. Бахтерев М.О., Васёв П.А. Методы распределённых вычислений на основе модели потока данных. Прототип системы // Тезисы XII Международного семинара «Супервычисления и математическое моделирование». Саров, РОСАТОМ, 2010, стр. 14-16.
6. Pong M.C. A Graphical Language for Concurrent Programming // Proceeding of the 1986 IEEE Workshop on Visual Languages (June 1986), pp. 26-33.
7. Al-Mulhem M., Ali Sh. Visual Occam: Syntax and semantics // Comput. Lang. 1997, V. 23, N 1, pp. 1-24.

Анализ эффективности масштабируемых подходов к решению задач с преобладанием ввода-вывода*

Д.Ю. Андреев[1,2], О.В. Джосан[1]

МГУ имени М.В.Ломоносова[1], ВЦ ДВО РАН имени А.А. Дородницина[2]

В данной работе описываются подходы к организации ввода-вывода в задачах с потоком входных - выходных данных и высокой загрузкой каналов, соединяющих вычислительные узлы и внешние системы хранения данных. Разработаны методы разделения функциональности вычислительных узлов в рамках модели иерархии коммуникаторов. Авторами предлагается библиотека, реализующая описанные методы. Исследование производительности разработанной библиотеки проводилось на системах BlueGene /P и «Ломоносов».

1. Введение

В настоящее время в мире решается множество задач по обработке данных, параллельные алгоритмы создаются и используются на различных архитектурах вычислительных систем. При реализации таких алгоритмов перед программистом встает вопрос об оптимальном способе загрузки данных из внешнего хранилища информации, распределении данных между узлами вычислительной системы.

Современные вычислительные системы содержат множество процессоров, которые могут одновременно обращаться за данными во внешнем хранилище. Увеличение числа вычислительных узлов приводит к усложнению архитектуры коммуникационной среды и систем хранения данных. При неэффективном использовании коммуникационных соединений или при неудачном сочетании обращений к жестким дискам во внешнем хранилище производительность загрузки данных может ухудшаться из-за ограниченности пропускной способности сети или скорости работы жестких дисков. Прикладному программисту сложно разрабатывать универсальные алгоритмы для всего разнообразия архитектур вычислительных систем.

2. Основные идеи и понятия

Не во всех задачах загрузка входных данных и выгрузка результатов занимает значительную часть времени работы. Проблема производительности операций ввода-вывода ярко выражена в задачах обработки потока данных большого размера, например при обработке последовательности изображений. В таких задачах часто можно встретить использование данных представимых в виде многомерных массивов. Существует не одно решение вопроса о формате хранения многомерных массивов, но наиболее популярными и признанными в научном мире являются библиотеки ориентированные на бинарный формат хранения netCDF[1] и HDF[2]. Многие современные архитектурные решения многопроцессорных вычислительных систем [8] предоставляют хранилище данных с возможностью параллельного доступа к файлам (с параллельными файловыми системами GPFS, PVFS, Lustre, и т. д.), существуют версии библиотек netCDF и HDF позволяющие использовать параллельный доступ к частям многомерных массивов. В концепции разрабатываемой библиотеке определено абстрактное понятие «источник данных». Указанные выше файлы на внешнем хранилище в форматах netCDF, HDF можно использовать в качестве источника данных.

В большинстве архитектур многопроцессорных вычислительных систем доступ к внешнему хранилищу неоднороден. Это обычно связанно с тем, что на несколько вычислительных узлов приходится один узел ввода-вывода. Например, в конфигурации системы BlueGene /P[3], установленной на факультете вычислительной математики и кибернетики МГУ имени М.В.

* Работа ведтся при поддержке Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" на 2009 — 2013гг.

Ломоносова, 128 вычислительных узлов используют один узел ввода-вывода. Неравномерное распределение между каналами ввода-вывода приводит к понижению производительности. В статье [4] демонстрируется, что в вычислительных системах с подобной организацией подсистемы ввода-вывода эффективным решением является выделение вычислительных узлов, через которые будут проходить все операции ввода-вывода. Естественным видится, что выделенные вычислительные узлы должны использовать максимальное доступное число каналов ввода-вывода. Вышеописанное приводит ко второму абстрактному понятию — «I/O вычислительные узлы».

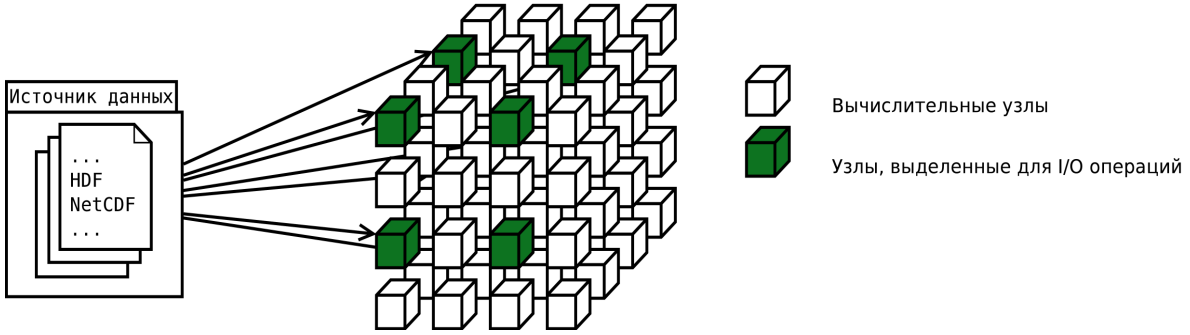


Рис. 1. Загрузка данных на выделенные узлы

Загрузив данные на выделенные вычислительные узлы, необходимо распределить их требуемым образом по вычислительным узлам обработки данных. В случае обработки потока данных, часто загрузка/выгрузка данных является более времязатратной операцией, чем обработка. Эффективным решением является продолжать загрузку/выгрузку данных одновременно с обработкой на вычислительных узлах, соответственно для обработки данных используется узлы не участвующие в операциях ввода-вывода. Назовём подмножество вычислительных узлов, обрабатывающих данные, областью обработки. Таким образом, в одной программе может быть несколько областей обработки, производящих разную работу над данными.

Схема работы программы в описанных терминах:

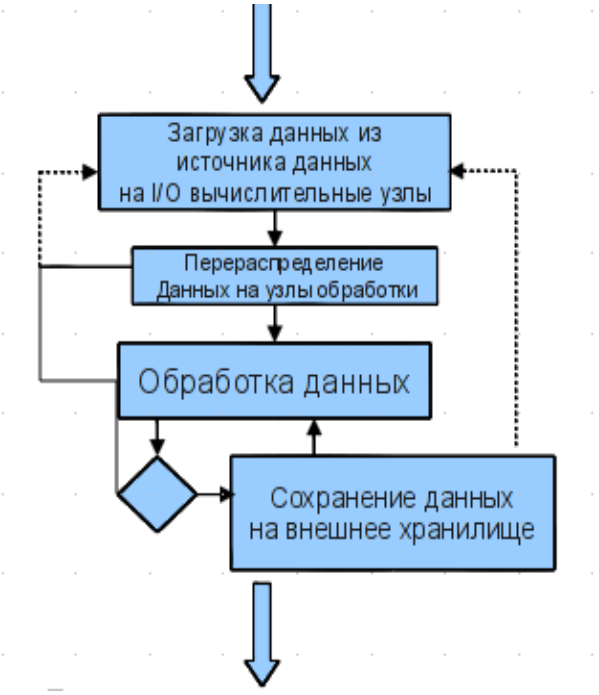


Рис. 2. Схема работы программы

При разработке библиотеки, реализующей описанные выше концепции, одной из целей было предусмотреть удобную возможность дальнейшей доработки. Части библиотеки, отвечающие разным этапам программы, связанны с друг другом только через интерфейс соответствующих классов или параметры функций. Внутренняя реализация библиотеки разрабатывается на языке C++ и имеет интерфейсы на языках C и C++.

В качестве библиотеки для организации многопроцессорного режима и обмена данными между вычислительными узлами используются самая популярная в научной среде библиотека Message Passing Interface. Message Passing Interface (MPI, интерфейс передачи сообщений) — программный интерфейс для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. В первую очередь MPI ориентирован на системы с распределенной памятью (большинство суперкомпьютеров являются системами с распределённой памятью).

3. Возможности разрабатываемой библиотеки работы с вводом-выводом

3.1 Схема применения библиотеки

Работа с библиотекой начинается с вызова функции инициализации библиотеки. Реализация функции инициализации может быть архитектурно зависима. Например, для системы BlueGene /P в момент инициализации библиотеки используются функции из библиотеки MPIX[5] для определения коммуникаторов, соответствующих множествам вычислительных узлов использующих один или наоборот разные узлы ввода-вывода. Исходя из полученных коммуникаторов, вычисляются возможные вычислительные узлы ввода-вывода. Функция инициализации должна вызываться на всех узлах, выделенных программе, т.е. в рамках коммуникатора MPI_COMM_WORLD.

Дальнейшую работу библиотеки пользователь может продолжить на всех процессорах, или указать число процессоров или коммуникатор, в рамках которого будет происходить работа библиотеки. Пользователю предоставляется возможность создать несколько независимых областей работы библиотеки в рамках подкоммуникаторов. В частности, библиотека может использоваться в системах визуализации данных [7].

В рамках каждого коммуникатора библиотеки возможно указывать источники данных, для этого предусмотрены функции вида «ParimprC_DataLoad_[Type]()», где в качестве типа источника указывается например netCDF или HDF; системный программист может добавлять источники данных добавляя функции подобного вида. Данные из источника загружаются на вычислительные узлы ввода-вывода, после чего доступны через поля структуры в списке всех источников данных.

С помощью функции «ParimprC_LayoutCreate()» может быть выделена область обработки. В зависимости от того, сколько MPI-процессов будет принадлежать области обработки, пользователь может создать множество независимых областей обработки. Системный программист может дорабатывать алгоритм данной функции, изменяя стратегию объединения MPI-нитей в области обработки. В зависимости от стратегии объединения, пользователь получит область обработки из соседних или удаленных вычислительных узлов. Указывая область обработки и типа распределения данных между узлами, пользователь с помощью функции «ParimprC_DataPrepare()» распределяет данные между вычислительными узлами области обработки.

В рамках области обработки могут быть вызваны функции-обработчики данных. Пользователь может создавать свои обработчики данных или использовать доступные в библиотеке. Обработчик данных работает в рамках MPI-коммуникатора, отвечающего области обработки.

3.2 Представление данных

В библиотеке определено абстрактное понятие «источник данных», для которого определен способ загрузки данных на узлы вычислительной машины. Данные могут иметь разный

формат и способ представления. В рамках данного этапа возможности библиотеки распространяются на работу с данными, представимыми в виде многомерных массивов.

Наиболее популярными в научной среде форматы хранения многомерных массивов во внешней памяти являются форматы NetCDF и HDF. Для этих форматов существует интерфейс взаимодействия на различных языках программирования, в частности C++.

Загрузка данных из файлов таких форматов происходит в три этапа:

- открытие файла
- определение имени и/или пути к данным внутри файла
- указание подмножества данных для загрузки

В результате для файла с массивом размерности N можно извлекать любые подмассивы размерности ≤ N. Требуется универсальное представление внутри оперативной памяти для массивов любой размерности.

Существует несколько решений для представления многомерного массива основанных на шаблонах, в параметрах шаблона указывается тип, размерность и максимальные значения индексов. Недостатком данного решения является то, что тип элементов массива определяется статически и невозможно изменить его в зависимости от того, какие данные были обнаружены в источнике данных в процессе выполнения программы.

Более универсальное решение — представление данных многомерного массива в виде проекции на одномерный массив типа char (или минимальной адресуемой единицы).

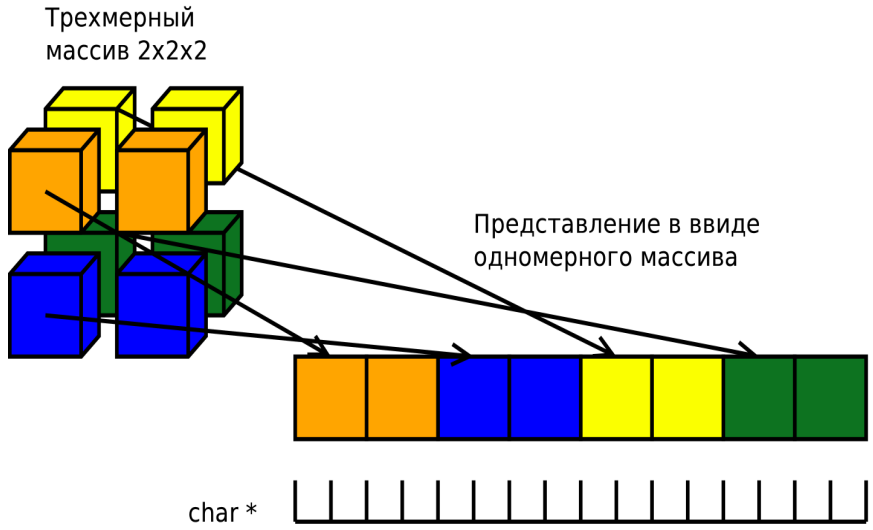


Рис. 3. Представление трехмерного массива

Разбиение данных на части по последнему измерению многомерного массива сводиться к разделению одномерного массива на последовательные части. Обработка произвольного подмассива произвольной размерности (меньшей изначальной) задача более сложная, требуются дополнительные операции, если функции обработки предполагают наличие последовательно расположенных элементов.

Для передачи данных средствами MPI определены стандартные типы данных, для передачи массива указывается тип данных и количество передаваемых элементов. Когда мы хотим передать произвольный подмассив многомерного массива, такой способ не подходит, так как данные не обязательно расположены последовательно.

Для подобного представления массива в MPI определены дополнительные типы. Для получения такого типа используется следующая MPI-функция: MPI_Type_create_subarray()

3.3 Выбор узлов ввода-вывода

Первая подзадача, которую требуется решить — выбор узлов ввода-вывода.

Каждый узел в вычислительной системе имеет ограниченное число оперативной памяти. Размер данных в источнике данных может многократно превосходить объем оперативной памяти отдельного вычислительного узла. Данные из источника распределяются по оперативной

памяти узлов ввода-вывода, от количества узлов ввода-вывода зависит возможный размер данных. Размер загружаемых данных определяется задачей и ее параметрами, поэтому заранее установить конкретное требуемое число узлов ввода-вывода на основе информации о вычислительной системе. Учитывая все вышесказанное, было решено оставить решение вопроса о количестве узлов ввода-вывода за пользователем. Во время инициализации библиотеки пользователь может указать, какое именно число процессоров он хочет задействовать в качестве узлов ввода-вывода.

После указания количества узлов ввода-вывода необходимо определить, какие именно вычислительные узлы будут отвечать за операции ввода-вывода. Подобный выбор может существенно повлиять на производительность дальнейшей загрузки и выгрузки данных.

Для системы BlueGene /P [3], установленной на факультете вычислительной математики и кибернетики МГУ имени М.В. Ломоносова, 128 вычислительных узлов используют один канал ввода-вывода. Если пользователю доступно для вычислений более 128 вычислительных узлов, то для выбора узлов ввода-вывода требуется определить группы процессоров, которые отвечают одному каналу ввода-вывода.

В MPI-реализации на BlueGene /P доступны три дополнительные функции, расширяющие функциональность механизма передачи сообщений с учетом особенностей аппаратуры системы. Они используются в рамках коммуникаций по сети трехмерного тора и упрощают отображение процессов на наборы процессоров.

Коллективная операция MPIX_Pset_same_comm_create (MPI_Comm *pset_comm) создает набор коммуникаторов, в каждый из которых входят все вычислительные узлы из данного набора процессоров. В результате обращения к коллективной функции MPIX_Pset_diff_comm_create(MPI_Comm *pset_comm) будут созданы коммуникаторы, объединяющие узлы, принадлежащие разным наборам процессоров, т.е. никакие два процесса каждого коммуникатора не будут иметь общего для них узла ввода-вывода.

Для выбора узлов ввода-вывода в разрабатываемой библиотеке определен алгоритм, который последовательно выбирает для узлов ввода-вывода процессы, для которых порядковый номер в коммуникаторах MPIX_Pset_same_comm_create() - минимален. При этом количества узлов ввода-вывода соответствующих одному каналу ввода-вывода различаются не более чем на единицу.

Выбранные узлы ввода-вывода объединяются в новый MPI-коммуникатор. Вызов функций загрузки-выгрузки данных может быть вызван на любом вычислительном узле, но на узлах не относящихся к узлам ввода-вывода никакой работы выполнено не будет. Такая модель защищает от ошибок вызова операций загрузки данных в рамках неправильного MPI-коммуникатора.

3.4 Выбор вычислительных узлов обработки данных

Пользователю библиотеки предлагается выбрать один из нескольких режимов работы:

1. Работа библиотеки на всех доступных задаче узлах вычислительной системы
2. Работа библиотеки в рамках отдельного коммуникатора

Первый и второй случай выглядят одинаково с точки зрения программы, поскольку запуск на всех доступных задачи узлах вычислительной системы равносилен запуску библиотеки в рамках глобального MPI-коммуникатора *MPI_COMM_WORLD*.

Другой принцип разделения режимов основан на выборе узлов обработки. Для запуска функций обработки пользователь должен определить коммуникатор, в рамках которого будет происходить распределение данных и непосредственно обработка распределенных данных. Процесс определения коммуникатора с вычислительными узлами обработки будем называть выделением области обработки. Существует два принципиально различных способа выбора областей обработки:

1. Области обработки выделяются с использованием вычислительных узлов, не задействованных в операциях ввода-вывода.
2. Области обработки выделяются, в том числе, с использованием вычислительных узлов отвечающих за ввод-вывод.

На данном этапе реализация библиотеки предполагает, что первый подход (использование узлов ввода-вывода для вычислений) используется только в том случае, когда обработка происходит на всех доступных задаче вычислительных узлах, иначе говоря, в рамках *MPI_COMM_WORLD*.

Пользователь может указать произвольный MPI-коммуникатор в качестве области обработки (MPI-коммуникатор не должен содержать узлов ввода-вывода).

3.4 Распределение данных на вычислительные узлы

После определения вычислительных узлы, на которых будет происходить обработка, необходимо расположить данные так, как предполагает их получать функция обработки данных. Возможны две стратегии решения указанной подзадачи:

1. Функции обработки разрабатываются так, что обладают встроенным параметром, в котором указано, какие данные должны находиться на каком вычислительном узле обработки.
2. Существует дополнительная функция, которая позволяет пользователю указывать желаемое распределение данных по узлам вычислительной системы.

Для данных в виде массива размерности N пользователь может указать список, в котором для каждого вычислительного узла будут указаны диапазоны координат или координаты отдельных элементов, все координаты указываются для N-мерного случая. Список должен иметь длину не больше, чем число MPI-нитей в области обработки, для которой происходит перераспределение данных. В соответствии с этим списком данные будут распределены по узлам вычислительной системы, предназначенным для обработки.

Поскольку, в общем случае, данные из источника распределены на узлах ввода-вывода независимо от последующего их использования в программе, процесс передачи определенного подмножества данных из источника на определенный узел обработки данных может потребовать передачи данных от нескольких узлов ввода-вывода.

Пример:

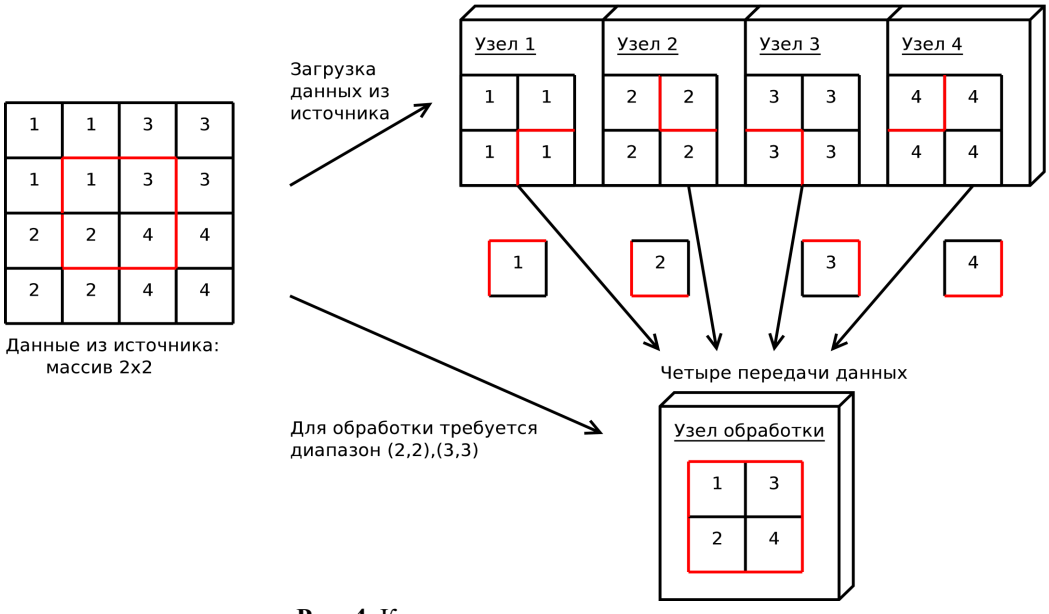


Рис. 4. Кусочное распределение данных

Данные из источника (двухмерный массив целых чисел) делится на равные порции и загружается на четыре узла ввода-вывода. Узлу обработки требуется подмассив из элементов (2,2), (2,3), (3,2), (3,3): потребуется четыре передачи данных. Стандартные MPI-функции передачи данных предполагают вызов функции приема данных для каждой порции. При этом требуется указывать буфер приема определенного размера, хотя порции данных от разных узлов ввода-вывода могут иметь разный размер.

Существует две возможности реализации подобного алгоритма:

1. Узлы обработки знают конкретное расположение данных на узлах ввода-вывода, тогда при узел обработки будет последовательно принимать данные от узлов ввода-вывода. Поскольку изначально предполагается, что узлы ввода-вывода могут работать независимо от узлов обработки, продолжая загрузку данных одновременно с обработкой, такой вариант потребует накладных расходов на извещение узлов обработки о конкретном расположении данных на узлах ввода-вывода.
2. Перед каждой передачей данных узел ввода-вывода извещает узел обработки о характеристиках последующей передачи (какие именно данных содержит узел ввода-вывода, размер данных). Такой способ создает дополнительную нагрузку на систему коммуникаций.

Для распределения данных на вычислительные узлы обработки используется функция односторонней передачи данных MPI_Put( ). Операция по сбору данных с узлов обработки на узлы ввода-вывода осуществляется с помощью MPI-функции MPI_Get().

После распределения на вычислительных узлах обработки доступны списки тех диапазонов данных, которые были указаны в функции распределения.

4. Анализ эффективности

Как описывалось выше, существует множество стратегий использования разрабатываемой библиотеки для обработки данных. Выше приведены аналитические оценки характеристик идеи выделения вычислительных узлов, отвечающих за все операции ввода-вывода. Необходимо исследовать практичность подхода на экспериментальных задачах.

Прежде всего необходимо определить характеристики по которым будет происходить сравнение. Конечным параметром сравнения разных подходов одинаковой обработки одни и тех же данных будет являться время выполнения.

В качестве функции обработки данных будет использована синтетическая тест-функция, которая принимает на вход несколько численных параметров: количество сложений, количество вычитаний, количество умножений и количество делений. Данная функция будет производить указанное число операций над каждым элементом многомерного массива, который был назначен на вычислительный узел обработки данных.

Поскольку требуется сравнить подход с выделением узлов ввода-вывода и обычный подход, когда все узлы могут использовать операции ввода-вывода, были разработаны дополнительные функции для формата HDF5, позволяющие вычислительному узлу получить произвольные данные из файла в указанном формате.

Экспериментальные данные создаются при помощи разработанного средства генерации многомерных массивов требуемого размера со случайными числовыми значениями его элементов. Данное средство генерации сохраняет выходные данные в виде одного файла в формате HDF5.

На основе разрабатываемой библиотеки, дополнительных функций и генератора экспериментальных данных было создано несколько режимов тестирования:

1. Полное использование библиотеки Parimpr с обработкой на всех узлах:
- Открытие файла, загрузка данных на узлы ввода-вывода, перераспределение данных на все вычислительные узлы, доступные тестовой задаче, происходит с использованием функций библиотеки.
- Запуск тестовой функции обработки данных происходит на всех вычислительных узлах, доступных тестовой задаче.
- После обработки результат собирается на узлах ввода-вывода и записываются в файл
2. Использование библиотеки Parimpr только для загрузки данных с обработкой на всех узлах:
    - Открытие файла, загрузка данных на узлы ввода-вывода, перераспределение данных на все вычислительные узлы, доступные тестовой задаче, происходит с использованием функций библиотеки.

- Запуск тестовой функции обработки данных происходит на всех вычислительных узлах, доступных тестовой задаче.
- Запись в файл происходит прямо с узлов обработки данных

3. Без использования библиотеки Parimpr:

- Файл открывается, данные загружаются непосредственно на узлах обработки данных, дополнительное распределение данных не требуется.
- Запуск тестовой функции обработки данных происходит на всех вычислительных узлах, доступных тестовой задаче.
- Запись в файл происходит прямо с узлов обработки данных

4. Полное использование библиотеки Parimpr без обработки данных на узлах ввода-вывода:

- Открытие файла, загрузка данных на узлы ввода-вывода, перераспределение данных на все вычислительные узлы не участвующие в операциях ввода-вывода, доступные тестовой задаче, происходит с использованием функций библиотеки.
- Обработка запускается только в выделенной области обработки
- После обработки результат собирается на узлах ввода-вывода и записывается в файл

5. Дополнение режимов 3 и 4: обработка потока данных. Операции, указанные в режимах 3 и 4, выполняются многократно над последовательностью файлов с данными, при этом для режима 4 предусмотрена загрузка данных из следующего файла одновременно с выполнением обработки текущих данных на остальных вычислительных узлах обработки данных.

Для каждого из режимов тестирования считается отдельное время работу загрузки данных, обработки и выгрузки.

Для каждого из режимов тестирования необходимо рассмотреть график зависимости общего времени работы тестовой задачи в зависимости от следующих параметров:

- размер исходных данных;
- количество операций в тестовой функции обработки данных.

Заключение

В данной работе рассмотрены подходы к организации ввода-вывода в задачах с потоком входных-выходных данных и высокой загрузкой каналов, соединяющих вычислительные узлы и внешние системы хранения данных. Разработаны методы разделения функциональности вычислительных узлов в рамках модели иерархии коммуникаторов. Авторами разработана библиотека, реализующая описанные методы. Исследование производительности разработанной библиотеки проводилось на системах Blue Gene/P и «Ломоносов».

Литература

1. NetCDF Tutorial
   URL: http://www.unidata.ucar.edu/software/netcdf/docs/ (дата обращения: 10.02.2011)
2. Сайт BlueGene /P, установленного в МГУ
   URL: http://hpc.cs.msu.su (дата обращения: 10.02.2011)
3. James Ahrens, Kristi Brislawn, Ken Martin, Berk Geveci, C. Charles Law, Michael Papka. Large-Scale Data Visualization Using Parallel Data Streaming. // IEEE Computer Graphics and Applications, July — Augus, 2001/ Vol. 21. No. 4. P. 34-41.
4. Alok Choudhary, Wei-keng Liao, Kui Gao, Arifa Nisar, Robert Ross, Rajeev Thakur and Robert Latham. Scalable I/O and analytics // Journal of Physics: Conference Series, 2009. Vol 180. No 1.

5. Hongfeng Yu, Kwan-Liu Ma, Joel Welling: I/O Strategies for Parallel Rendering of Large Time-Varying Volume Data. // 5th Eurographics/ACM SIGGRAPH Symposium on Parallel Graphics and Visualization (EGPGV 2004) ,June, 2004. P. 31-40, .
6. Библиотека mpix.
URL: http://www.mpix.com (дата обращения: 10.02.2011)
7. Джосан О.В., Мурынин А.Б., Попова Н.Н. Метод визуализации многомерных динамических данных на многопроцессорных комплексах // Вестник компьютерных и информационных технологий. 2009. № 8. сс. 8-12.
8. Корж А.А., Макагон Д.В., Оценка минимальных требований к аппаратуре и топологии при построении высокоскоростных коммуникационных сетей для суперкомпьютеров с общей памятью // Вычислительные методы и программирование: новые вычислительные технологии. 2008. Т. 9. № 2. сс. 26-31.

Исследование эффективности архитектуры CUDA для аппроксимации множества Парето с помощью метода роя частиц

А.Э. Антух, А.П. Карпенко, А.С. Семенихин

МГТУ им. Н.Э. Баумана

Во многих практически значимых случаях при решении задачи многокритериальной оптимизации предварительно целесообразно построить аппроксимацию множества Парето этой задачи. Рассматривается комбинация известного метода приближенного построения множества Парето «недоминируемая сортировка» и метода глобальной оптимизации роем частиц. Целью работы является исследование эффективности указанной комбинации методов при их реализации на графических процессорных устройствах с архитектурой CUDA.

1. Введение

Использование графических ускорителей (ГПУ) для научных расчетов началось относительно недавно [1]. Во многих прикладных областях до сих пор остается открытым вопрос об эффективности ГПУ по сравнению с классическими компьютерными системами. Целью данной работы является исследование эффективности ГПУ с архитектурой CUDA для задачи аппроксимации множества Парето с помощью метода роя частиц (PSO).

Основная масса публикаций, посвященных CUDA-вычислениям, ориентирована на задачи, обладающие явным параллелизмом по данным [2]. Такие задачи хорошо распараллеливаются и позволяют получать существенный прирост производительности вычислений по сравнению с последовательными вычислениями на хост-ЭВМ. В задачах многокритериальной оптимизации не всегда присутствует параллелизм по данным и эффективность решения таких задач на ГПУ с архитектурой CUDA мало освещена в литературе. Исследование эффективности CUDA-вычислений при решении задачи однокритериальной оптимизации выполнено, например, в работе [3].

В разделе 2 работы приведена математическая формулировка задачи многокритериальной оптимизации и дано определение множества Парето. Раздел 3 содержит краткое описание используемого алгоритма метода роя частиц. В разделе 4 приведено описание особенностей реализации алгоритма, а также дана постановка тестовой задачи многокритериальной оптимизации. Раздел 5 содержит результаты экспериментов и оценку эффективности алгоритма. В заключении подведены итоги работы и определены направления ее развития.

2. Постановка задачи

Задана совокупность частных критериев оптимальности $\phi_1(X),\phi_2(X),...,\phi_m(X)$, которые образуют векторный критерий $\Phi(X)$. Здесь X - n-мерный вектор варьируемых параметров [4]. Ставится задача минимизации каждого из указанных критериев в одной и той же области допустимых значений вектора варьируемых параметров $D_X \in \Pi \cap D$, где $\Pi = \left\{X \middle| x_i^- \le x_i \le x_i^+, i \in [1,n]\right\}$ - «технологический» параллелепипед, $D = \left\{X \middle| g_1(X) \ge 0, g_2(X) \ge 0,...\right\}$. Здесь $g_1(X), g_2(X),...$ - ограничивающие функции. Условно задача многокритериальной оптимизации записывается в виде

$$\min_{X \in D_X} \Phi(X) = \Phi(X^*). \quad (1)$$

Векторный критерий оптимальности $\Phi(X)$ выполняет отображение множества D_X в некоторое множество D_Φ пространства критериев, которое называется множеством достижимости

задачи (1). Введем на множестве D_Φ отношение предпочтения. Будем говорить, что вектор $\Phi^1 \in D_\Phi$ предпочтительнее вектора $\Phi^2 \in D_\Phi$ (или вектор Φ^1 доминирует вектор Φ^2), и писать $\Phi^1 \succ \Phi^2$, если среди равенств и неравенств $\phi_k(X^1) \geq \phi_k(X^2), k \in [1:m]$ имеется, хотя бы одно строгое неравенство. Выделим из множества D_Φ подмножество точек D_Φ^* (фронт Парето), для которых нет более предпочтительных точек. Множество $D_X^* \in D_X$, соответствующее множеству D_Φ^*, называется множеством Парето [4]. Таким образом, если $X \in D_X^*$, то $\Phi(X) \in D_\Phi^*$.

Если $\Phi(X_1) \succ \Phi(X_2)$, то будем говорить, что вектор X_1 предпочтительнее вектора X_2 (или вектор X_1 доминирует вектор X_2), и писать $X_1 \triangleright X_2$.

Ставится задача приближенного построения множества Парето D_X^* или, что то же самое, фронта Парето D_Φ^*, задачи многокритериальной оптимизации (1).

3. Методы роя частиц и недоминируемой сортировки

Множество частиц обозначим $\mathbf{P} = \{P_i, i \in [1:N]\}$, где N – число частиц в рое (размер популяции). В дискретный момент времени $t \in [0:T]$ координаты частицы P_i определяются вектором $X_{i,t} = (x_{i,t,1}, x_{i,t,2}, ..., x_{i,t,n})$, а ее скорость – вектором $V_{i,t} = (v_{i,t,1}, v_{i,t,2}, ..., v_{i,t,n})$; T - число итераций. Начальные координаты и скорости частицы P_i равны $X_{i,0} = X_i^0$, $V_{i,0} = V_i^0$ соответственно.

Итерации в каноническом методе PSO выполняются по схеме [5]

$$V_{i,t+1} = \alpha V_{i,t} + U_1[0,\beta] \otimes \left(X_{i,t}^b - X_{i,t}\right) + U_2[0,\gamma] \otimes \left(X_{g,t} - X_{i,t}\right), \quad (2)$$

$$X_{i,t+1} = X_{i,t} + V_{i,t+1}. \quad (3)$$

Здесь $U[a,b]$ представляет собой n-мерный вектор псевдослучайных чисел, равномерно распределенных в интервале $[a,b]$; $\otimes$ - символ покомпонентного умножения векторов; $X_{i,t}^b$ - вектор координат частицы P_i с наилучшим значением целевой функции $\Phi(X)$ за все время поиска $[0:t]$; $X_{g,t}$ - вектор координат соседней с данной частицы с наилучшим за время поиска $[0:t]$ значением целевой функции $\Phi(X)$; α, β, γ - свободные параметры алгоритма. Важнейшее в методе PSO понятие соседства частиц зависит от используемой топологии соседства и определено, например, в работе [5]. В процессе итераций вектор $X_{i,t}^b$ образует так называемый собственный путь (private guide) частицы P_i, а вектор $X_{g,t}$ - локальный путь (local guide) этой частицы.

Канонический метод роя частиц ориентирован на решение задач однокритериальной оптимизации. Для задачи многокритериальной оптимизации (1) используем модификацию этого метода, которая называется метод MOPSO (Multi Objective Particle Swarm Optimization) [6].

Важной частью метода MOPSO является определение глобально-лучшей (в смысле формулы (1)) частицы для каждого индивидуума в популяции. В силу специфики задачи многокритериальной оптимизации глобально-лучшая частица отыскивается на множестве Парето.

Псевдокод метода MOPSO представлен на рисунке 1. На каждой итерации в процессе выполнения шаге 3 метода происходит обновление архива частиц A_t. Функция *Update* выполняет сравнение частиц текущего поколения с недоминируемыми частицами из архива A_t и определяет те частицы, которые необходимо добавить в архив, а также те частицы, которые следует в архиве заменить. Функция реализует метод недоминируемой сортировки, предложенный в работе [7].

Выбор глобально лучшей частицы осуществляет функция *FindGlobalBest.* Существует несколько способов реализации этой функции. В данной работе используется метод «меняющихся» соседей» Хью и Эберхарта [6]. Рассмотрим суть этого метода на примере задачи двухкритериальной оптимизации. Поиск глобально лучшей частицы для каждой частицы популяции

осуществляется в пространстве критериев следующим образом. Сначала вычисляем расстояние от частицы P_i до других частиц, содержащихся в архиве A_t, используя значения первого («фиксированного») критерия оптимальности $\phi_1(X)$. Таким образом, для частицы P_i находим k ее ближайших локальных соседей. Затем, используя второй критерий $\phi_2(X)$, из числа указанных k соседей находим наилучшую частицу для частицы P_i, которая и полагается глобально лучшей частицей для частицы P_i.

Шаг 1: t=0
Шаг 2: Инициализация популяции $\mathbf{P}_t$ и архива частиц A_t
For i=1 **to** N
$X_{i,0} \coloneqq X_i^0$; $V_{i,0} \coloneqq V_i^0$; $X_{i,0}^b \coloneqq X_i^0$
End;
$A_t \coloneqq \{\}$;
Шаг 3: $A_{t+1} \coloneqq Update(\mathbf{P}_t, A_t)$;
Шаг 4:
For i=1 **to** N
$X_{g,t} \coloneqq FindGlobalBest(A_{t+1}, X_{i,t})$;
For j=1 **to** n
$v_{i,t+1,j} \coloneqq \alpha v_{i,t,j} + U_1[0,\beta] \otimes \left(x_{i,t,j}^b - x_{i,t,j}\right) + U_2[0,\gamma] \otimes \left(x_{g,t,j} - x_{i,t,j}\right)$;
$x_{i,t+1,j} \coloneqq x_{i,t,j} + v_{i,t+1,j}$
End;
If ($X_{i,t+1} \rhd X_{i,t}^b$) $X_{i,t+1}^b \coloneqq X_{i,t+1}$ **Else** $X_{i,t+1}^b \coloneqq X_{i,t}^b$
End;
Шаг 5: **If** (Критерий останова=false)
t:=t+1;
GOTO Шаг 3
End.

Рис. 1. Псевдокод метода MOPSO

Итерации продолжаются до тех пор, пока множество недоминируемых решений не перестанет меняться, либо до достижения заданного числа итераций.

4. Реализация алгоритма и тестовая задача

Была рассмотрена следующая двухмерная двухкритериальная тестовая задача, точные фронт и множество Парето для которой известны:

$$\begin{cases} \phi_1(X) = x_1^2 + x_2^2, \\ \phi_2(X) = (x_1 - 1)^2 + (x_2 - 1)^2; \end{cases} \tag{4}$$

$$D_X = \{X \mid -5 \le x_i \le 5, i = 1,2\}. \tag{5}$$

Исследование плотности покрытия множества Парето и фронта Парето задачи (4, 5) при использовании метода MOPSO рассмотрено в работе [8]. В данной работе исследуется эффективность указанного метода при его реализации на ГПУ.

Исследование выполнено с использованием в качестве хост-ЭВМ персональной ЭВМ, управляемой операционной системой Windows XP, в следующей конфигурации: процессор - Intel Pentium Dual E2160, 1,8 ГГц; оперативная память - 2 ГБ. При вычислениях только с использованием хост-процессора было использовано одно ядро. В качестве ГПУ использована плата NVidia GeForce 8500GT, содержащая 16 потоковых процессоров. Архитектура CUDA рассмотрена, например, в работе [9].

В практически значимых задачах многокритериальной оптимизации основной объем вычислений связан с вычислениями значений частных критериев оптимальности. Например, если

речь идет о задаче оптимизации сложной динамической системы, каждое такое вычисление требует численного интегрирования большой системы обыкновенных дифференциальных уравнений. Поэтому в работе принята следующая схема организации вычислений: метод MOPSO реализует хост-процессор системы, а вычисления значений частных критериев оптимальности – ГПУ.

Эффективность параллельных вычислений оценивается с помощью ускорения

$$S = \frac{T_{CPU}}{T_{GPU}}, \tag{6}$$

где $T_{ЦПУ}$ – время решения задачи с использованием только хост-процессора системы, T_{GPU} – аналогичное время при решении задачи с использованием ГПУ. Подчеркнем, что под ускорением S в данном случае понимается повышение производительности вычислений при использовании ГПУ относительно производительности вычислений, производимых только на хост-процессоре.

Важная составная часть исследования – исследование зависимости ускорения (6) от суммарной вычислительной сложности частных критериев оптимальности. Для моделирования разных вычислительных сложностей критериев (4) использовано их многократное вычисление. В качестве меры вычислительной сложности критериев использован коэффициент сложности C, равный числу их вычислений.

5. Результаты экспериментов

Исследование выполнено при варьировании следующих параметров:

- число итераций $T = 30;\ 50$;
- коэффициент сложности $C = 1;\ 10;\ 100;\ 1000$;
- число частиц в популяции $N = 32;\ 64;\ 128;\ 256$.

Поскольку эффективность метода существенно зависит от начальных параметров частиц X_i^0, V_i^0 результаты исследования усреднены по этим начальным параметрам; $i \in [1:N]$.

Некоторые результаты исследования представлены в таблице 1. Таблица показывает, что, как и следовало ожидать, при невысокой вычислительной сложности частных критериев оптимальности распараллеливание вычислений не эффективно, поскольку в этом случае велика доля коммуникационных расходов. При увеличении вычислительной сложности критериев, а также числа частиц в популяции, ускорение вычислений достигает величины 23.

Таблицу 1 иллюстрируют рисунки 2, 3.

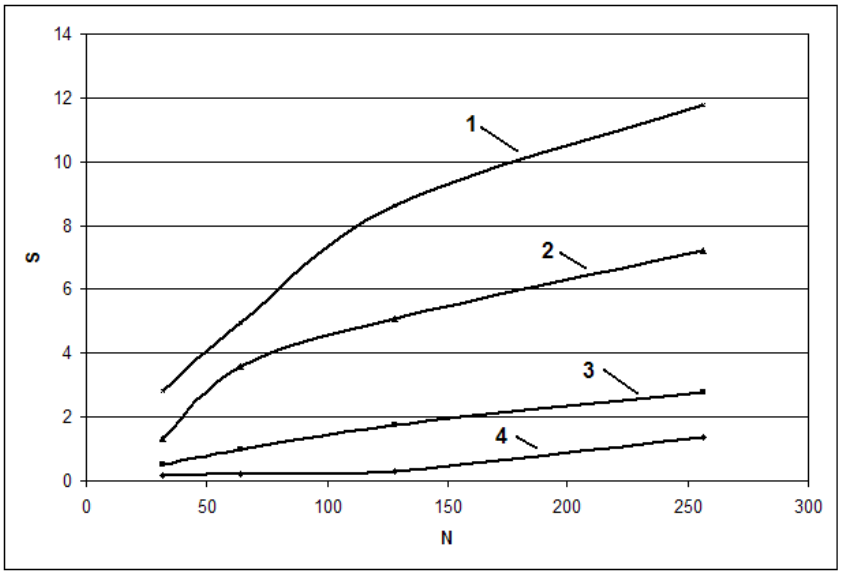


Рис. 2. Ускорение вычислений в функции числа частиц N при числе итераций $T = 30$:
1 - $C = 1000$; 2 - $C = 100$; 3 - $C = 10$; 4 - $C = 1$

Таблица 1. Некоторые результаты исследования

№	N	T	C	$T_{ЦПУ}$, с	T_{GPU}, с	S
1	32	30	1	0,026	0,156	0,167
2	32	30	10	0,087	0,175	0,497
3	32	30	100	0,531	0,406	1,308
4	32	30	1000	6,922	2,468	2,805
5	32	50	1	0,062	0,21	0,295
6	32	50	10	0,297	0,265	1,121
7	32	50	100	2,39	0,656	3,643
8	32	50	1000	25,64	4,343	5,904
9	64	30	1	0,062	0,265	0,234
10	64	30	10	0,296	0,296	1,000
11	64	30	100	2,515	0,703	3,578
12	64	30	1000	23,328	4,703	4,960
13	64	50	1	0,147	0,359	0,409
14	64	50	10	0,643	0,421	1,527
15	64	50	100	6,556	1,14	5,751
16	64	50	1000	69,828	7,641	9,139
17	128	30	1	0,13	0,421	0,309
18	128	30	10	0,856	0,484	1,769
19	128	30	100	7,312	1,437	5,088
20	128	30	1000	79,641	9,25	8,610
21	128	50	1	0,54	0,656	0,823
22	128	50	10	2,498	0,796	3,138
23	128	50	100	28,504	2,609	10,925
24	128	50	1000	219,031	15,406	14,217
25	256	30	1	1	0,736	1,359
26	256	30	10	2,7	0,968	2,789
27	256	30	100	18,25	2,531	7,211
28	256	30	1000	268,36	22,812	11,764
29	256	50	1	3,36	1,234	2,723
30	256	50	10	12,44	1,5	8,293
31	256	50	100	81,437	4,45	18,300
32	256	50	1000	690,44	29,375	23,504

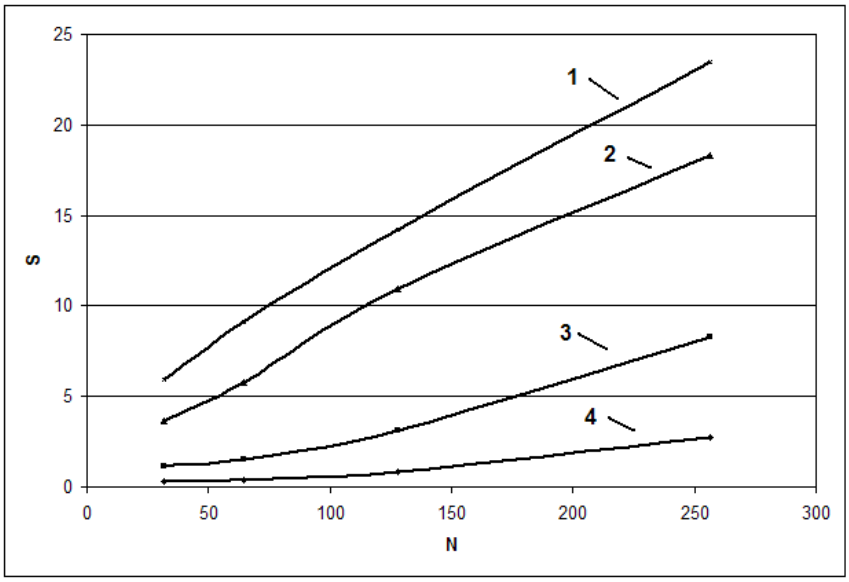


Рис. 3. Ускорение вычислений в функции числа частиц N при числе итераций $T = 50$:
1 - $C = 1000$; 2 - $C = 100$; 3 - $C = 10$; 4 - $C = 1$

Отметим, что рисунки 2, 3 показывают практически линейный рост ускорения с ростом числа частиц *N*. Это обстоятельство позволяет надеяться на получение ускорения, превышающего максимально достигнутое в рассматриваемых экспериментах (равное 23) при дальнейшем увеличении числа частиц в популяции.

6. Заключение

Выполнено исследование эффективности комбинации метода MOPSO и метода недоминируемой сортировки при приближенном построении множества Парето в задаче многокритериальной оптимизации с помощью ГПУ с архитектурой CUDA. Исследование показало высокий потенциал использования графических ускорителей и библиотек для работы с ними (CUDA) для эффективного решения указанной задачи.

В развитие работы предполагается расширение класса тестовых задач многокритериальной оптимизации; определение для ГПУ различных архитектур оптимальных значений свободных параметров рассматриваемых методов, обеспечивающих максимальное ускорение вычислений; разработка методов теоретической оценки ускорения при заданных значениях свободных параметров методов и архитектуре ГПУ.

Литература

1. Вычисления на GPU. http://www.nvidia.ru/page/gpu_computing.html (дата обращения 13.12.2010).
2. NVIDIA CUDA C SDK Code. http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html (дата обращения 13.12.2010).
3. Карпенко А.П., Селиверстов Е.Ю. Глобальная безусловная оптимизации роем частиц на графических процессорах архитектуры CUDA // Наука и образование: электронное научно-техническое издание, 2010, 4. http://technomag.edu.ru/doc/142202.html (дата обращения 13.12.2010).
4. Штойер Р. Многокритериальная оптимизация. Теория, вычисления и приложения. М., Радио и связь, 1992. 504 с.
5. Карпенко А.П., Селиверстов Е.Ю. Глобальная оптимизация методом роя частиц. Обзор // Информационные технологии. 2010. № 2. С. 25-34.
6. Hu X., Eberhart R. Multiobjective optimization using dynamic neighborhood particle swarm optimization // World Congress on Computational Inelligence. Proceeding: 2002. P. 1677–1681.
7. Srinivas N., Deb K. Multiobjective optimization using nondominated sorting in genetic algorithms // Evolutionary Computation. 1994. vol. 2. P. 221–248.
8. Антух А.Э., Семенихин А.С., Хасанова Р.В. Построение множества Парето методом роя частиц на графических процессорах архитектуры CUDA // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи, Труды международной суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). М.: Изд-во МГУ, 2010. С.274-280.
9. Фролов В. Введение в технологию CUDA . http://cgm.computergraphics.ru/issues/issue16/cuda (дата обращения 13.12.2010).

Предпосылки к созданию однокристального многопроцессорного компьютера ПС-2000М производительностью 1-10 TFlops

С.Е. Артамонов[2], Ю.С. Затуливетер[1], Е.А. Фищенко[1]

Учреждение Российской академии наук Институт проблем управления им. В. А. Трапезникова РАН[1], Общество с ограниченной ответственностью «ИДМ»[2]

Показано, что отсутствие высокоэффективных многопроцессорных архитектур становится ключевой проблемой развития высокопроизводительной элементной базы. Рассматривается однокристальная масштабируемая архитектура, которая развивает отечественную линию многопроцессорных компьютеров ПС-2000, впервые в мире выпущенных большой серией и промышленно апробированных на широких классах задач с массовым параллелизмом. Предлагается концепция реализации однокристального многопроцессорного компьютера ПС-2000М с масштабируемой архитектурой производительностью 1-10Tflops. Приводится сравнение ее архитектуры с архитектурами GP GPU, таких как nVIDIA Fermi и AMD Radeon, а также с архитектурами гетерогенных мультипроцессоров Intel Larrabee и IST Cell.

1. Введение

Технологии отечественного производства компьютерной элементной базы отстают от мирового уровня на несколько поколений. Наличие своей конкурентоспособной элементной базы необходимо для возрождения отечественного компьютеростроения, без которого инновационное развитие и модернизация экономики страны невозможно. Снижение зависимости от зарубежной элементной базы – еще и первоочередное требование национальной безопасности.

Современная индустрия интегральных схем в нормах 65, 45, 32 нм с уменьшением размера транзисторов дает, как и ранее, квадратичный рост плотности их размещения на кристалле. Чипы с 1-2 млрд. транзисторов и более перестали быть уникальными и серийно выпускаются разными производителями. Вместе с тем, в сферах массового производства элементной базы впервые за весь микропроцессорный период возникли серьезные барьерные проблемы фундаментального характера. Без их решения прежние темпы экспоненциального роста производительности массовых компьютерных устройств становятся недостижимыми, а это для компьютерного рынка весьма болезненно, поскольку сложившиеся за десятилетия балансы производства-потребления могут быть разрушены.

Выделим два аспекта этих проблем – технологический и архитектурный.

Технологический аспект. В диапазоне 65-32 нм и менее при увеличении плотности размещения транзисторов технологии столкнулись с опережающим ростом тепловыделения, что существенно ограничило подъем рабочих частот.

Это "вдруг" возникшее ограничение на рост частот лишает производителей микропроцессоров главного и привычного способа повышения производительности в рамках сложившихся за 3 десятилетия шаблонов одноядерных микропроцессорных архитектур. В их основе, как известно, лежит классическая модель последовательных вычислений Дж. фон Неймана, которая долго служила логическим и системообразующим стандартом универсальных вычислений, связывающих воедино индустрию массового производства компьютеров и программ. Теперь прежний темп роста частот, который позволял автоматически, в рамках привычных архитектур, увеличивать производительность всех ранее созданных программ, уже стал невозможным.

Архитектурный аспект. Быстрый (по закону Мура) рост числа транзисторов на кристалле по-прежнему сохраняется. Поэтому, в условия теплового барьера, главным способом увеличения производительности становится наращивание параллелизма компьютерных архитектур. Но и тут компьютерная индустрия "неожиданно" сталкивается еще с одной барьерной проблемой. Это структурное насыщение микропроцессорных архитектур [1], суть которого – исчерпание внутренних резервов того изначально ограниченного параллелизма, который допускается в

рамках микропроцессорных воплощений модели Дж. фон Неймана. Эффект структурного насыщения обнаружил себя [1] еще в середине 90-х, но микропроцессорная индустрия его проигнорировала, т.к. имела чисто технологические возможности коммерчески очень привлекательного наращивания производительности экспоненциальными темпами за счет рабочих частот.

Но сейчас ситуация кардинально изменилась. В диапазоне 10-50 млн. транзисторов они исчерпали свои структурные резервы параллелизма и не могут задействовать миллиарды транзисторов глубоко нанометровых технологий для эффективного наращивания производительности. "Архитектурный голод" [2] делает новые миллиарды транзисторов "безработными", а перепроизводство чего-либо на рынке – далеко не лучший способ стимулирования производителей. Одновременное исчерпание структурных резервов базовой архитектуры и технологических возможностей наращивания рабочих частот создают предпосылки фундаментального кризиса компьютерной индустрии, ставшей эталоном эффективности использования научно- технического прогресса.

"Архитектурный голод" обезоруживает передовые СБИС-технологии, которые, развиваясь по закону Мура и с опорой на микропроцессорные архитектуры, длительное время были локомотивом массовой компьютеризации. Сейчас, когда фон-неймановская монополия в массовом компьютеростроении себя исчерпала, требуются однокристальные архитектуры с массовым параллелизмом, которые позволят в широких классах применений в полной мере раскрыть вычислительный потенциал миллиардов пока еще "лишних" транзисторов. Также потребуются и качественно новые модели индустриального программирования, способные на массовом потребительском рынке обеспечивать новые компьютеры программами в экономически оправданных количествах.

2. Структурное насыщение микропроцессорных архитектур [1]

Микропроцессоры, реализуя последовательную модель вычислений, имеют ограниченные структурные резервы параллелизма. В линейке Intel эти резервы стали активно использоваться, начиная с i286. Прежде всего, в дело пошел внутренний параллелизм арифметических операций – увеличение разрядности слов и шин (i286 и далее) и схемная реализация "тяжелых" операций (i287, i387, i486), затем (i586 и далее) – их многофункциональность (разные параллельные блоки АЛУ, вплоть до MMX). Далее, начиная с i586, в ход пошли суперскалярные резервы параллелизма управляющих действий (конвейеризация и динамическое распараллеливание потока команд в ограниченном окне, предсказание условий). В поколениях после i586 удельная производительность (на транзистор), падала (рис.1).

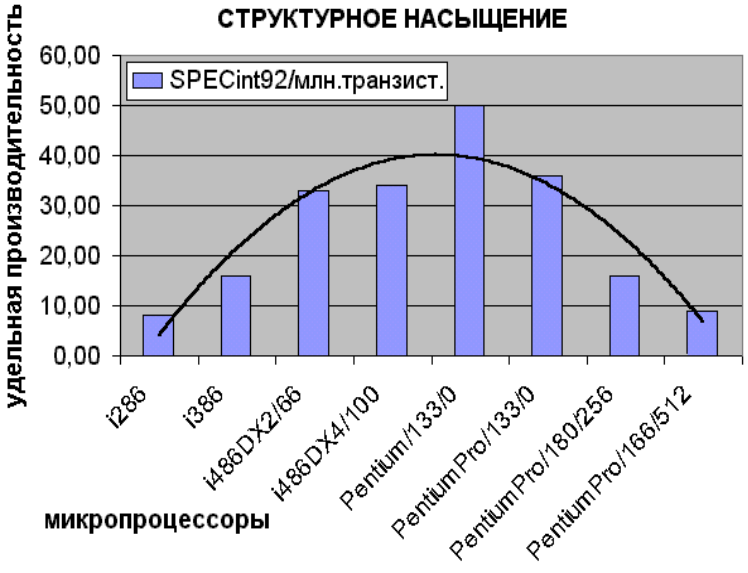


Рис.1 Удельная производительность микропроцессоров в расчете на транзистор

На последних стадиях исчерпание резервов завершилось обильным ростом кэш-памяти, которая "поглощала" все "излишки" транзисторов (в диапазоне 30-100млн.), сводя на нет их удельную производительность.

В конце 90-х годов со стороны уже было ясно видно, что структурные резервы параллелизма микропроцессорных архитектур почти полностью задействованы, но компьютерная индустрия в технологиях с нормами 130 и 90нм (200-700 млн. транзисторов) сконцентрировалась на резервах роста рабочих частот в диапазоне 0.7-3 Ггц. Лишние, с чисто вычислительной точки зрения, транзисторы до середины 2000-х легко укрывались в кэш памяти всех уровней.

На переходе от 90 нм к 65 нм и менее (от 0.7 к 1 млрд. и более транзисторов) тепловой барьер остановил рост частот. Массовые тиражи мобильных устройств вывели экономию энергопотребления на высшие приоритеты. В диапазоне 65 – 32 нм и менее увеличение производительности массовых компьютерных устройств за счет роста частоты при опережающем росте энергопотребления стало для новых поколений элементной базы неприемлемым.

Микропроцессорная индустрия с задержкой на десятилетие ответила на эффект структурного насыщения массовым выпуском двуядерных микропроцессорных кристаллов лишь под давлением двух суперпроблем – теплового барьера и "архитектурного голода".

Однако кристаллы со многими ядрами в виде универсальных микропроцессоров не дают роста производительности пропорционального числу транзисторов. На кристалле с миллиардом транзисторов, можно было бы размещать сотни микропроцессоров Pentium (3.1 млн. транзисторов). Но, как показала практика, число универсальных ядер на кристалле в большинстве случаев ограничено двумя (реже – 3 и 4), поскольку с увеличением числа ядер, изначально ориентированных на модель последовательных вычислений, производительность деградирует.

Кристаллы с несколькими универсальными ядрами не дали решения проблемы структурного насыщения.

Главным способом повышения производительности однокристальных вычислительных устройств, отвечающих требованиям эффективного использования вычислительного потенциала транзисторов, становятся масштабируемые многопроцессорные архитектуры, способные на широких классах задач с массовым параллелизмом наращивать эффективную производительность пропорционально количеству транзисторов на кристалле.

Такие архитектуры выходят за рамки системообразующих полномочий модели фон Неймана и наработанных способов индустриального программирования, что ставит компьютерную индустрию в сложное положение. Проигнорировав ранние признаки структурного насыщения, она поставила себя перед стрессовой необходимостью "на ходу" искать решение сложнейших задач не только поиска высокоэффективных многопроцессорных структур массовых применений, пригодных для СБИС-погружения, но и решения фундаментальных проблем разработки методов и средств их индустриального программирования, обеспечивающих свойство переносимости и кардинальное снижение трудоемкости.

3. СБИС-погружение многопроцессорных архитектур

Первые шаги в этом направлении были предприняты в рамках проекта развития отечественной архитектурной линии высокопроизводительных компьютеров ПС-2000 [2,3] еще в конце 80-х - начале 90-х годов [4,5]. Цель проекта – разработка на базе масштабируемой архитектуры ПС-2000 элементной базы в виде многопроцессорных СБИС. В основе проекта лежал уникальный опыт разработки и научного обоснования оригинальной архитектуры ПС-2000, инженерно-конструкторского проектирования ВК ПС-2000, организации их промышленного выпуска большой серией, а также программирования и широкого применения в разнообразных сферах. Логика обобщения этого опыта с учетом прогресса СБИС-технологий приводила к необходимости разработки новых поколений вычислительной элементной базы с масштабируемой многопроцессорной архитектурой широкого профиля, предназначенной для СБИС-погружения.

Оригинальная многопроцессорная архитектура ПС-2000 изначально разрабатывалась под широкие классы задач с массовым параллелизмом и балансировалась по экономическому критерию максимальной производительности в расчете на единицу стоимости. Во многом благодаря удачной, опередившей свое время архитектуре, компьютеры ПС-2000 стали одной из пер-

вых в мире многопроцессорных систем с массовым параллелизмом, которые выпускалась большой промышленной серией (более 240 комплексов) и стали доступными для широкого промышленного применения в разнообразных сферах народного хозяйства [2,3]. Особое сочетание достоинств SIMD- и VLIW-архитектур позволило ВК на базе ПС-2000 на многих задачах промышленной обработки данных и в научно-технических расчетах показывать производительность близкую к пиковой (80% и выше).

Прекращение финансирования в период "реформ" 90-х не позволило довести опережающие мировой уровень архитектурные решения в части высокопроизводительной элементной базы до аппаратного воплощения в виде масштабируемых многопроцессорных СБИС.

4. Однокристальные многопроцессорные ускорители

За рубежом тема СБИС-погружения многопроцессорных вычислительных устройств на индустриальном уровне актуализировалась лишь с середины 2000-х, т.е. спустя 15 лет после упомянутого выше проекта развития линии ПС-2000. И только тогда на рынке в больших тиражах появляются самостоятельные многопроцессорные кристаллы.

Наиболее известные решения представлены следующими однокристальными многопроцессорными архитектурами:

- графические процессорные устройства общего назначения (GP GPU):
  - nVIDIA Fermi (технология 40 нм, пиковая производительность 1Тфлопс);
  - AMD (ATI) Radeon (технология 40 нм, пиковая производительность 2,7 Тфлопс)
- гетерогенные мультипроцессоры:
  - Cell (IBM, Sony, Toshiba) (технология 90 нм, пиковая производительность 0.256 Тфлопс);
  - Intel Larrabee (технология 45нм, пиковая производительность 2 Тфлопс, на рынок не вышла).

Данные архитектурные решения дают высокую степень загрузки процессорных элементов (ПЭ) в основном для узких классов задач. В таблице 1 приведены примеры задач, решаемых на GP GPU nVIDIA. Представлено ускорение (g) относительно времени решения задачи на универсальном многоядерном CPU. Соотношение пиковых производительностей GP GPU и CPU приблизительно 200 раз. Отсюда КПД (процент от пиковой производительности): КПД~(g*100%)/200.

Таблица 1. Эффективность решения различных классов задач на GP GPU nVIDIA

Примеры задач, решаемых на GPGPU nVIDIA	**Ускорение, g**	**~КПД, %**
Гидрогазодинамика	10	5
Обработка изображений, кодирование видео, компьютерное зрение	3 -**160**	1.5-**80**
Сейсмическое моделирование (поиск нефти/газа)	30	15
Квантовая химия, моделирование белка	20-80	10-40
Моделирование взаимодействия объектов на молекулярном уровне	17	8.5
Базы данных, поиск, сортировка	2-6	1-3
Предсказание погоды, моделирование климата	20-40	10-20
Анализ и распознавание объектов, слежение за объектами	7-12	~5
Криптография и криптоанализ	1.7-20	1-10

Из таблицы 1 видно, что максимальное КПД=80% достигается на задачах обработки изображений, для которых GPU и разрабатывался. Изначально архитектура GPU ориентирована на обработку графики, что ограничивает возможности её высокоэффективного (по степени загруженности процессорных элементов) применения в других сферах. Такая узкопрофильность свидетельствует о наличии большого резерва в совершенствовании архитектурных решений в области высокопроизводительных вычислений общего назначения. Кроме того, она существенным образом ограничивает сферы потребления и не позволяет в полной мере раскрывать потенциальные возможности новейших СБИС-технологий.

Разработка компьютеров с гибкими, эффективными на широких классах задач высокопараллельными архитектурами, ориентированных на СБИС-погружение – одна из наиболее актуальных из нерешенных задач современного компьютеростроения.

Для дальнейшего продвижения требуются новые подходы, по крайней мере, в трех направ-

лениях:

- разработка и однокристальное воплощение многопроцессорных архитектур с массовым параллелизмом, обеспечивающих на широких классах задач производительность близкую к пиковой (свыше 80%);
- разработка новых подходов к программированию параллельных вычислений, отвечающих требованиям индустриального программирования (снижение трудоемкости; повышение эффективности машинных кодов, обеспечение переносимости программ);
- существенное снижение энергопотребления в расчете на единицу реальной производительности.

В этих направлениях мы обладаем конкурентоспособным научно-техническим заделом, представленным в масштабируемой архитектуре однокристального многопроцессорного компьютера ПС-2000М.

5. Об архитектуре однокристального компьютера ПС-2000М

В основу архитектуры ПС-2000М положен успешный опыт крупносерийного индустриального производства отечественного компьютера с высокопараллельной архитектурой ПС-2000 [2, 3]. Более 240 ВК на базе ПС-2000 использовались в нашей стране во многих сферах промышленной обработки данных с рекордно высокими показателями производительности в расчете на стоимость и энергопотребление. Его структура представлена на рис.2.

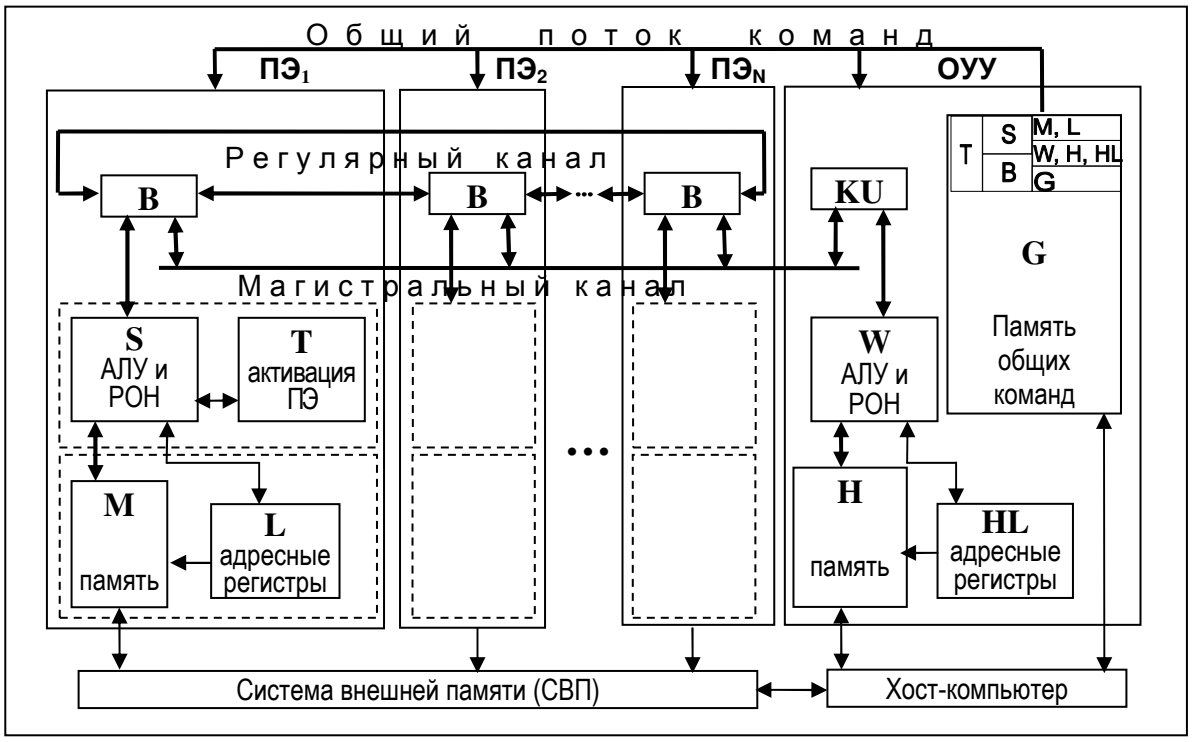


Рис.2. Структура многопроцессорного компьютера ПС-2000

Высокопараллельная многопроцессорная архитектура ПС-2000, в отличие от известных современных решений, изначально разрабатывалась и балансировалась под широкие классы задач с массовым параллелизмом. Её отличительные особенности, апробированные большой практикой:

- высокая гибкость и компактность управления (при минимальных аппаратных затратах обеспечивает эффективное сочетание разных видов машинного параллелизма);
- масштабируемость архитектуры и машинных программ;
- близкая к пиковой производительность на широких классах задач;
- высокая производительность в расчете на единицу стоимости, объема и энергопотребления.

Эти архитектурные качества, получившие дальнейшее развитие в ПС-2000М, актуальны и сейчас, поскольку до сих пор не имеют аналогов и, потому, способны составить конкуренцию известным реализациям архитектур nVIDIA, AMD, Cell.

Предварительный анализ показывает возможность реализации на технологии 65 нм однокристального компьютера ПС-2000М (512 ПЭ) с производительностью до 1 Тфлопс. По мере доступности более глубоких нанометровых технологий (32-12 нм) благодаря свойству масштабируемости архитектуры возможно построение программно-совместимого семейства однокристальных компьютеров ПС-2000М с достижением на кристалле производительности 10 Тфлопс.

ПС-2000М [6,7] предназначается для создания широкой номенклатуры высокопроизводительных вычислительных систем объединяющих стандартные микропроцессоры с однокристальной многопроцессорной системой, а именно, для создания:

- рабочих станций промышленной обработки данных (сейсморазведка, изображения, томография, криптография);
- мобильных и персональных теракомпьютеров с гибридной архитектурой (10-1000 Тфлопс);
- расширителей кластерных суперкомпьютеров (стационарных петакомпьютеров 1-1000 Пфлопс);
- встраиваемых систем реального времени: мультимедийных, телеметрических, гидроакустических, управляющих;
- систем ассоциативной обработки данных для систем управления большими базами данных и построения долговременных хранилищ данных.

В рамках проекта ПС-2000М с учетом обширного опыта промышленного программирования ПС-2000 развиваются и новые подходы к индустриальному программированию архитектур с массовым параллелизмом, позволяющие не только автоматизировать разработку ПО, но и выполнять комплексную оптимизацию по производительности и потреблению энергии.

На рис.3 изображен базовый вычислительный модуль (БВМ) [6,7], в основу которого положена SIMD-архитектура ПС-2000, см. рис.2.

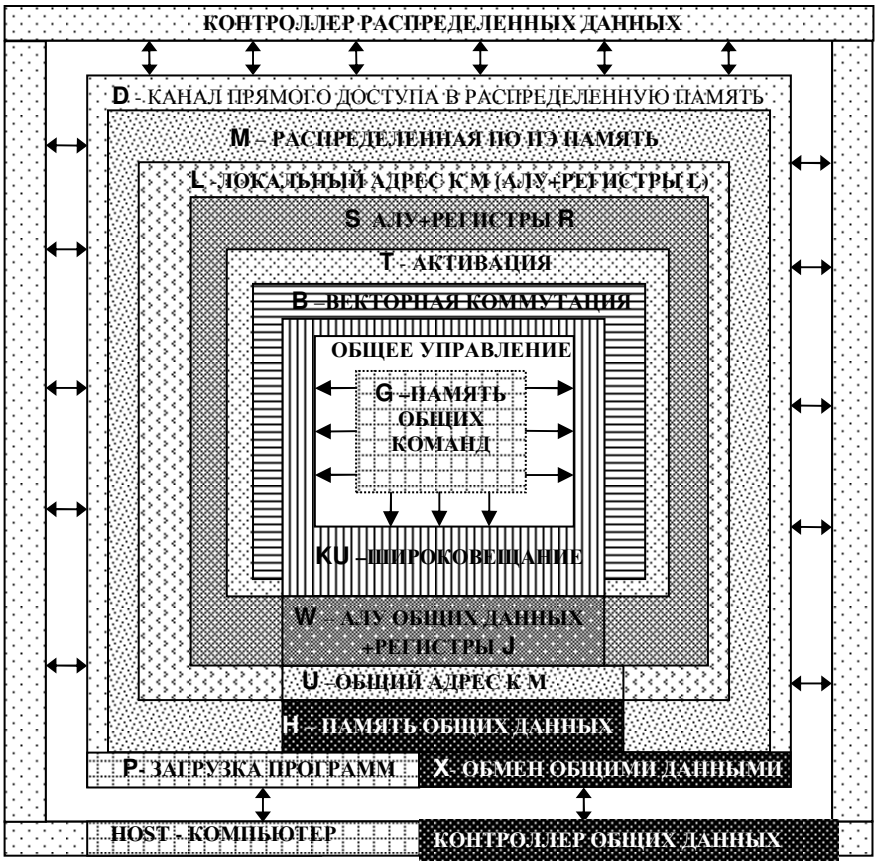


Рис.2. Структура БВМ

На рис.4 приведена Multi-SIMD – структура, которая позволяет программной реконфигурацией комплексировать БВМ внутри кристалла ПС-2000М.

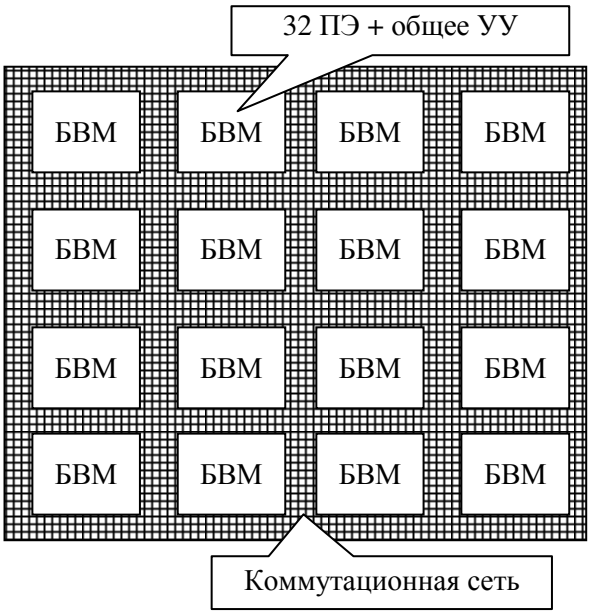


Рис.4. Multi-SIMD комплексирование БВМ внутри кристалла

В статье [8] об инновациях в архитектуре nVIDIA Fermi обсуждаются также ее недостатки и важнейшие проблемы построения сбалансированных компьютерных архитектур.

К недостаткам архитектуры nVIDIA Fermi отнесены:

- относительно небольшой объем памяти GPU;
- невозможность прямого ввода/вывода в память GPU;
- отсутствие возможности «бесшовного» комплексирования нескольких GPU.

ПС-2000М эти недостатки не присущи. Преимущества архитектуры состоят в следующем:

- широкие классы эффективно решаемых задач (с производительностью более 80% пиковой);
- структурная масштабируемость (по производительности, объемам памяти, пропускной способности памяти и канала ввода/вывода, энергопотреблению);
- больший объем и пропускная способность внутренней памяти;
- возможность комплексирования на внутри- и межкристальном уровнях;
- наличие канала прямого доступа в распределенную память в ориентации на использования масштабируемых высокоскоростных интерфейсов – QuickPath, PCI-Express, Hypertransport.

Эти качества дают основания полагать, что архитектура ПС-2000М обладает потенциалом конкурентоспособности в классе однокристальных компьютеров общего назначения (GP). В нем, как нам представляется, изначальная узкая специализация на графическую обработку (GPU) или игровые требования (Cell) является не только необязательной, но и излишней. Она заведомо "ущемляет" многие другие классы задач с массовым параллелизмом, для которых отсутствует необходимость в высококачественной онлайн визуализации, сопутствующей вычислениям (например, задачи ассоциативной обработки данных, распознавания, управления и др.).

Проблемы создания аппаратных средств эффективного решения различных классов задач (GP) и специальных ускорителей визуализации (GPU) можно (а, возможно, и нужно) рассматривать отдельно. Это позволит достигать предельных результатов в каждом из этих случаев, что в целом даст больший эффект, чем искусственное, не всегда оправданное, "натягивание" различных задач на вычислительные структуры, изначально предназначавшиеся под спецвычисления. Необходимость "приведения" задач к удобному для GPU виду существенно усложняет и без того непростые задачи параллельного программирования, ставит дополнительные препятствия на пути к получению высокоэффективных кодов программ машинного уровня.

6. Сравнение архитектур

В таблице 2 приведено сравнение архитектуры ПС-2000М с наиболее известными архитектурами, предназначенными для массового рынка высокопроизводительных вычислений:

- графических процессорных устройств GP GPU - nVIDIA Fermi и AMD (ATI) Radeon;
- гетерогенных мультипроцессоров - Intel Larrabee и IST Cell.

Из таблицы 2 видно:

- архитектура ПС-2000М превосходит другие архитектуры по объему внутренней памяти, которая имеет распределенную организацию, что значительно увеличивает ее пропускную способность (пропорционально количеству ПЭ) и позволяет достигать эффективную производительность на многих классах задачах близкую к пиковой;
- статическая организация вычислительного процесса дает предпосылки к его оптимизации на этапах компиляции, что существенно упрощает всю совокупность распределенных механизмов управления вычислениями, что позволяет кардинально снижать аппаратные и временны*е* затраты на управление, а вместе с ними и тепловыделение;
- наличие канала прямого доступа в распределенную память позволяет совмещать вычисления с вводом/выводом данных, что значительно повышает эффективную производительность на реальных задачах, а также обеспечивает комплексирование на межкристальном уровне;
- применение в распределенной памяти ПС-2000М кода Хемминга, повышает помехоустойчивость, живучесть и радиационную стойкость кристалла.

Таблица 2. Сравнение архитектур высокопроизводительных вычислителей для массового рынка

Характеристика	**NVIDIA** ***Fermi***	**AMD (ATI)** ***Radeon***	**Intel** ***Larrabee***	**IBM, Sony, Toshiba** ***Cell***	**ПС-2000М**
Технология, нм	40	40	45	90	65 – 12
Пиков. произв., Тфлопс	1	2,7	2	0,256	1 – 10
Число ПЭ	512∨256	320×5∨160×5	8∨16∨32∨48×(x86+БВМ)	CPU+8×БВМ×2∨4∨8∨16	512 (до 8192)
Вычисл. процесс	Динамика	Динамика	Динамика	Статика	Статика
Организация памяти	Общая	Общая	Распределенная	Распределенная	Распределенная
Объем памяти, МБ	1	0,6	8	2	16-256
КПДП	нет	нет	есть	есть	есть
Коррекц. ошибок (живучесть)	ECC	нет	нет	нет	код Хэмминга

Новизна архитектуры ПС-2000М по сравнению с известными на сегодня однокристальными решениями состоит в том, что она изначально сбалансирована под широкие классы задач с массовым параллелизмом. Параллелизм (и вместе с ним пропускная способность) арифметических ресурсов и ресурсов внутренней оперативной памяти масштабируется пропорционально росту количества ПЭ (в диапазоне 512-8192 ПЭ). Достижение этих качеств стало возможным благодаря опыту разработки, серийного производства, а также индустриального программирования и широкого использования в различных сферах народного хозяйства компьютера ПС-2000, оригинальную архитектуру которого по многим качествам можно отнести к классу широкопрофильных многопроцессорных компьютеров с массовым параллелизмом общего назначения.

Заключение

Высокопроизводительная элементная база на основе ПС-2000М может быть положена в основу единой, компьютерной платформы высокопроизводительных масштабируемых семейств вычислительных систем новых классов, оснащенных индустриальными технологиями

создания совместимого программного обеспечения. Такие вычислительные системы могут составлять основу различных мобильных устройств, систем управления и обработки данных реального времени, настольных суперкомпьютеров, игровых приставок новых поколений, рабочих и серверных станций, проблемно-ориентированных вычислительных систем, дата-центров, а также общедоступных суперкомпьютеров.

Своевременная реализация предложения по созданию конкурентоспособной отечественной элементной базы будет способствовать инновационному развитию и модернизации экономики, решению задач национальной безопасности, откроет новые пути развития экспорта российской высокотехнологичной продукции.

Литература

1. Затуливетер Ю.С. Компьютерные архитектуры: неожиданные повороты // HARD'n'SOFT. Компьютерный журнал для пользователей. 1996. №2. С. 89-94. URL: http://zvt.hotbox.ru/p2_z1.htm (дата обращения: 11.02.2011).
2. Затуливетер Ю.С., Фищенко Е.Ф. Компьютер ПС-2000: многопроцессорная архитектура, опередившая время // Пленарные и избранные доклады Четвертой международной конференции "Параллельные вычисления и задачи управления". М.: ИПУ РАН. 2008. С. 63 - 75.
3. Затуливетер Ю.С., Фищенко Е.А. Многопроцессорный компьютер ПС-2000 // Открытые системы. 2007. № 9. С.74-79. URL: http://www.osp.ru/os/2007/09/4570286/ (дата обращения: 11.02.2011).
4. Затуливетер Ю.С., Фищенко Е.А., Кротов В.А., Лементуев В.А. Параллельные высокопроизводительные ЭВМ на основе заказных СБИС с многопроцессорной архитектурой // Приборы и системы управления. 1996. №12. С.24 -26.
5. Артамонов С.Е., Затуливетер Ю.С., Козлов В.А., Лементуев В.А., Фищенко Е.А. Перспективные СБИС с многопроцессорной архитектурой // Тезисы докладов Второго межведомственного научно-практического семинара "Проблемы и технологии создания и использования космических систем и комплексов на базе малых КА и орбитальных станций". М.: ГКПНЦ им. Хруничева. 1998. С.131.
6. Затуливетер Ю.С, Фищенко Е.А. Многопроцессорная архитектура ПС-2000 на кристалле СБИС // Проблемы управления. 2007. № 4. С.30-35.
7. Артамонов С.Е., Затуливетер Ю.С., Фищенко Е.А. Предложение по созданию однокристального многопроцессорного компьютера ПС-2000М // Материалы докладов Международной научно-технической конференции «Суперкомпьютерные технологии: разработка, программирование, применение» Т.1.-Таганрог: Изд-во ТТИ ЮФУ. 2010. С. 22-26.
8. Patterson D. The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges, 30.09.2009 // URL: http://www.nvidia.com/content/PDF/fermi_white_papers/D.Patterson_Top10InnovationsInNVIDIAFermi.pdf (дата обращения: 11.02.2011).

Исследование и поиск наиболее эффективных подходов к параллельному моделированию плазмы методом частиц в ячейках на 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, предназначенного для моделирования плазмы методом частиц в ячейках (Particle-In-Cell) на современных кластерных системах. Исследование и поиск наиболее эффективных схем распараллеливания производится в рамках специфики приложений для решения наиболее актуальных задач современной физики сверхмощных лазеров. Приводятся результаты вычислительных экспериментов, показывающие корректность и характеризующие масштабируемость текущей реализации. Формулируются выводы и планы по дальнейшему развитию.

1. Введение

Теоретические исследования поведения вещества в экстремальных условиях, при которых происходит ионизация вещества, т.е. переход в состояние плазмы, приобретает все большее значение для решения многих прикладных и фундаментальных задач. Кроме традиционных задач вакуумной электроники, в последние годы активно развиваются исследования поведения плазмы, образующейся при ионизации вещества мишени, облучаемой сверхмощным лазерным импульсом. Этот процесс представляет большой интерес для целого ряда приложений, среди которых можно выделить лазерное ускорение заряженных частиц, концепцию быстрого поджига для управляемого термоядерного синтеза и генерацию излучения с уникальными характеристиками в труднодоступных частотных диапазонах. Новые результаты в этой области, в свою очередь, могут вывести на качественно новый уровень многие важные направления, среди которых создание компактных источников для адронной терапии при лечении онкологических заболеваний, создание фабрик короткоживущих изотопов для биоимиджинга, разработка приборов для исследования внутримолекулярных и внутриатомных процессов, а также поиск новых путей для фундаментальных исследований эффектов нелинейности вакуума.

Часто, в связи с высокой степенью нелинейности и геометрической сложностью задачи, исследование динамики плазменных структур основывается на моделировании плазмы методом частиц в ячейках [3]. Данный подход был развит и начал применяться уже более сорока лет назад, однако и сегодня продолжаются активные исследования в области разработки новых алгоритмов с целью повышения эффективности вычислительных схем, снижению негативного влияния численных эффектов и расширению границ применимости метода [1, 2]. Основная специфика метода частиц в ячейках заключается в одновременной обработке принципиально разнородных массивов данных, содержащих информацию о координатах и скоростях заряженных частиц плазмы, и об электромагнитном поле, заданном в узлах дискретной решетки, представляющей часть трехмерного пространства. В отличие от большинства задач численного моделирования в физике, где справедлив принцип локальности взаимодействия, отсутствие однозначных путей упорядочивания обращений к памяти обуславливает сложность эффективной программной реализации, как для классических многопроцессорных вычислительных систем, так и для гетерогенных систем с использованием графических процессоров.

*Работа выполнена в лаборатории «Информационные технологии» ВМК ННГУ при поддержке ФЦП «Научные и научно-педагогические кадры инновационной России», госконтракт № 02.740.11.0839.

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 кластерных системах.

2. Постановка задачи

Подход к моделированию плазмы методом частиц в ячейках основан 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 уравнений движения:

$$\begin{aligned} \frac{\partial \vec{p}}{\partial t} &= q\left[\vec{E} + \frac{1}{c}\left(\vec{v} \times \vec{B}\right)\right] \\ \frac{\partial \vec{r}}{\partial t} &= \vec{v} \\ \vec{v} &= \vec{p}\frac{1}{m}\left(1 + \left(\frac{\vec{p}}{mc}\right)^2\right)^{-\frac{1}{2}} \end{aligned}, \qquad (1)$$

где m, q, $\vec{r}$, $\vec{p}$ и $\vec{v}$ – масса, заряд, координата в трехмерном пространстве, импульс и скорость частицы соответственно; $\vec{E}$, $\vec{B}$ – векторы напряженности электрического и магнитного поля в точке нахождения частицы соответственно; $c \approx 2{,}998 \times 10^{10}\, см/с$ – скорость света. Эволюция шестикомпонентного векторного поля электромагнитных полей ($\vec{E}$, $\vec{B}$) описывается в рамках уравнений Максвелла, в которые входит векторное поле плотности тока $\vec{j}$, создаваемое частицами:

$$\begin{aligned} rot\vec{B} &= \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t} \\ rot\vec{E} &= -\frac{1}{c}\frac{\partial \vec{B}}{\partial t} \end{aligned}. \qquad (2)$$

Метод частиц в ячейках предполагает представление всего ансамбля частиц эквивалентным меньшим числом так называемых крупных частиц, имеющих такое же отношение заряда к массе, как у реальных частиц. Поскольку заряд и масса частицы входят в уравнения движения (1) только в виде отношения, такое представление обеспечивает полностью эквивалентную динамику плазмы в случае, если число крупных частиц достаточно велико. Обычно это позволяет обходиться моделированием динамики значительно меньшего числа частиц, чем реально присутствует в рассматриваемом объеме плазмы (известны задачи, в которых возникает необходимость моделирования динамики ~10^9 и более частиц в пространстве, представленном ~10^8 ячеек).

3. Метод решения

Расчетная область покрывается равномерной пространственной сеткой. Электрическое поле $\vec{E}$ и плотность тока $\vec{j}$ представляются значениями в узлах сетки, магнитное поле $\vec{B}$ – значениями в центрах ячеек сетки. Также хранятся данные о наборе частиц, каждая частица определенного типа характеризуется положением и скоростью. Масса и заряд определяются типом частицы.

Используется классическая схема метода частиц в ячейках [1], включающая следующие этапы: инициализация, вычислительный цикл (взвешивание токов, интегрирование уравнений поля, интерполяция полей, интегрирование уравнений движения частиц), завершение работы (Рис. 1). Каждая итерация цикла соответствует одному шагу по времени.

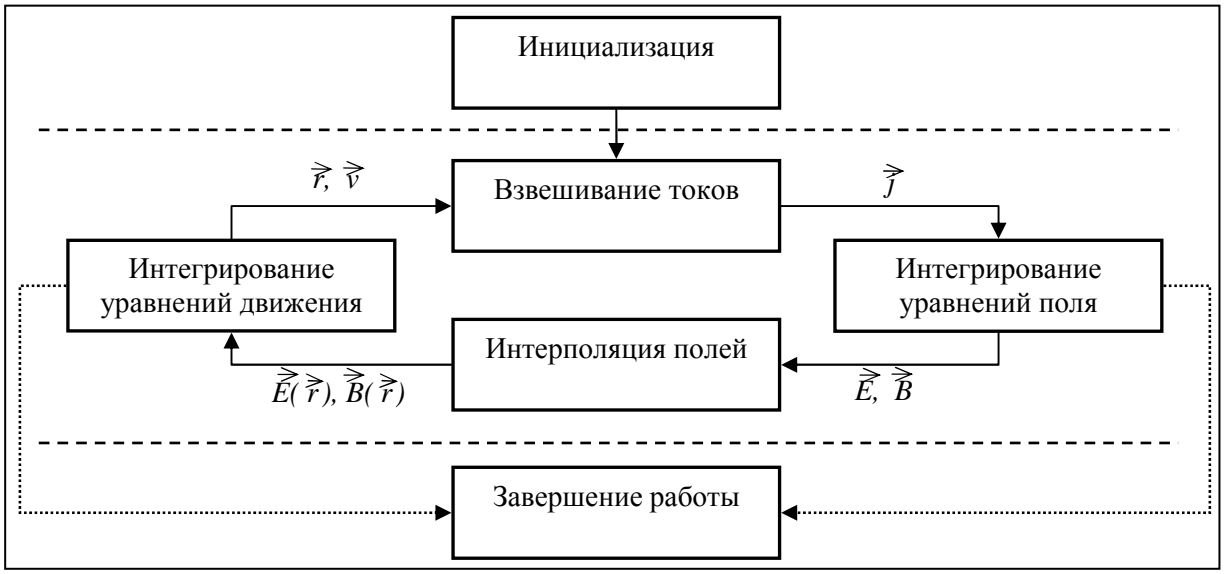


Рис. 1. Вычислительная схема метода

На этапе инициализации выполняются следующие подготовительные работы: чтение и интерпретация исходных данных, создание и инициализация необходимых структур данных. Взвешивание токов состоит в определении сеточных значений плотности тока $\vec{j}$ 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 [3]. Далее выполняется шаг численного интегрирования уравнений Максвелла (2) по времени, определяются новые сеточные значения электрического и магнитного поля; используется метод FDTD [4]. 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 (1); для интегрирования используется метод Boris по описанию в [1].

4. Параллельная реализация с использованием MPI

Параллельная реализация разрабатывается для использования на кластерных системах. Предлагается следующая схема работы:

1. Распределение задачи по узлам вычислительного кластера осуществляется при помощи интерфейса MPI.
2. На каждом узле кластера при помощи OpenMP выполняется распределение нагрузки между вычислительными ядрами. В настоящий момент реализован вариант, основанный на использовании MPI.

Декомпозиция задачи осуществляется по территориальному принципу: расчетная область разбивается на подобласти (домены) равного размера, операции над которыми выполняются параллельно. Вычислительный узел, 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 на всех вычислительных узлах, обрабатывающих такие домены. Кроме того, вычислительный узел хранит данные о магнитных полях в центрах ячеек, граничащих с обрабатываемым им доменом. На Рис. 2 приведен пример разбиения расчетной области на домены в двумерном случае: ячейки сетки изображены прямоугольниками, увеличенная толщина линий соответствует границам доменов, заштрихована подобласть, составляющая один из доменов, пунктиром выделена область хранимых значений электрического и магнитного поля.

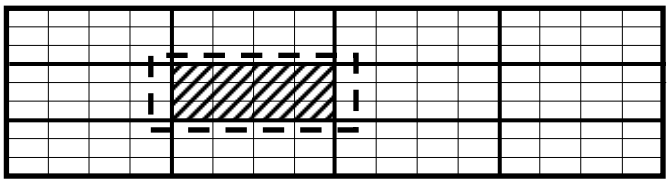


Рис. 2. Разбиение расчетной области на домены в двумерном случае

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. Для краткости далее будем называть это обменом токами, полями и частицами, соответственно. Вычислительная схема параллельной версии приведена на Рис. 3.

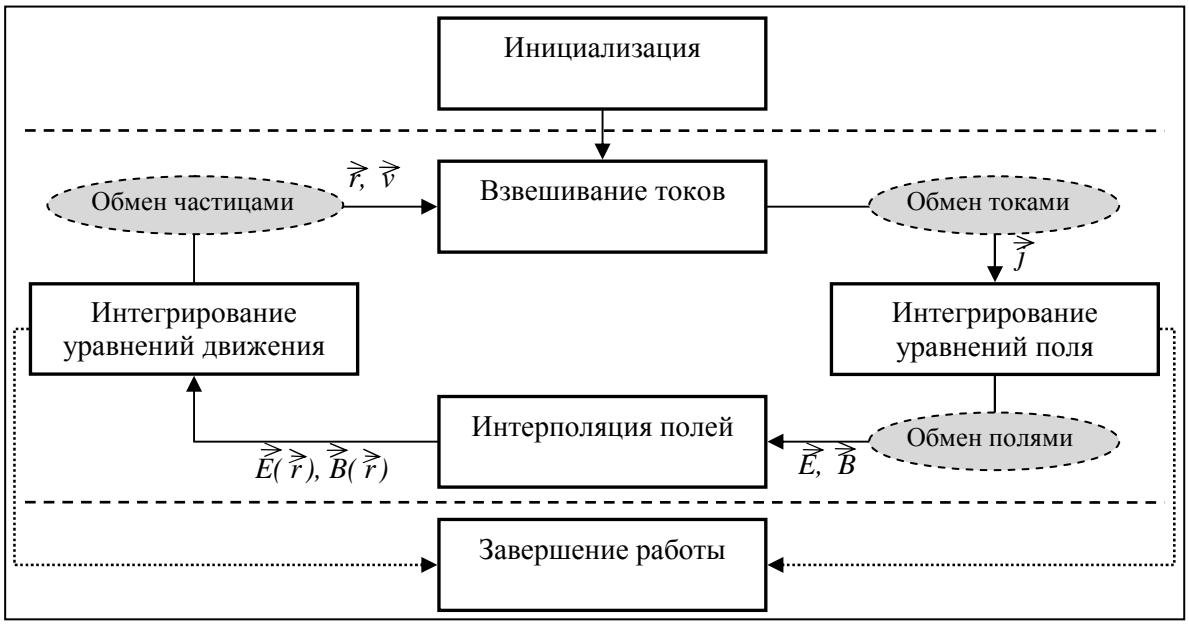


Рис. 3. Вычислительный цикл параллельной реализации метода

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 соседних доменов). Таким образом, все обмены данными осуществляются между вычислительными узлами, обрабатывающими соседние домены.

5. Тестовые задачи и результаты экспериментов

Для проверки правильности реализации метода частиц в ячейках традиционно используется ряд тестовых задач, для которых известно аналитическое решение:

1. Релятивистское движение заряженной частицы в постоянном электромагнитном поле (проверяется алгоритм численного решения уравнений движения (1)).
2. Распространение электромагнитной волны в вакууме (проверяется алгоритм численного интегрирования уравнений Максвелла (2)).
3. Ленгмюровские колебания плазмы (проверяются процедуры взвешивания токов, интерполяции полей и учет плотности тока в уравнениях Максвелла (2)).

Ввиду отсутствия коллективных эффектов и, следовательно, технической тривиальности проверки в настоящей работе мы не будем приводить результаты решения первой тестовой задачи, подтверждающие соответствие численных расчетов теоретическим предсказаниям.

В рамках второй тестовой задачи был смоделирован процесс распространения в вакууме гармонической электромагнитной волны с линейной поляризацией. Как известно, аналитическим решением уравнений (2) является распространение синусоидальной волны с сохранением пространственного профиля со скоростью света, что соответствует гармоническим осцилляци-

ям напряженности электрического поля в каждой точке пространства с частотой $\omega = kc$, где k – волновое число. На Рис. 4 приведено сравнение результирующей частоты осцилляций, полученной при моделировании (с использованием периодических граничных условий), с аналитическим результатом для различных значений волнового числа, минимальное значение соответствует 120 шагам пространственной сетки на длину волны, максимальное – 4 шагам на длину волны. Отличие от аналитического результата при больших значениях волнового числа является известным для метода FDTD следствием неточного описания при недостаточно малом по сравнению с характерными масштабами физического процесса пространственном шаге сетки.

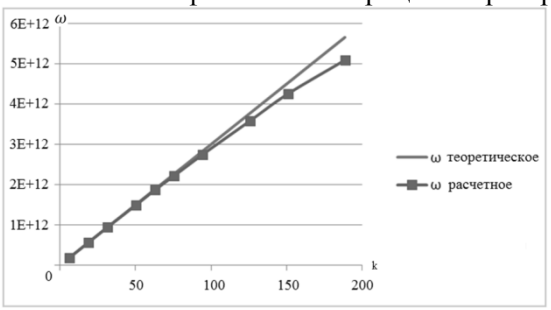


Рис. 4. Сравнение частоты осцилляций полученной при численном моделировании с аналитическим результатом для разных значений волнового числа

В рамках третьей тестовой задачи моделировались Ленгмюровские колебания двухкомпонентной плазмы, вызванные небольшим синусоидальным (в пространстве) отклонением концентрации электронов от равновесного состояния в начальный момент. Ионы при этом моделировались как неподвижные частицы. В соответствии с известным результатом физики плазмы, при малой амплитуде колебаний должны наблюдаться гармонические осцилляции концентрации электронов и напряженности электрического поля с плазменной частотой, определяемой выражением $\omega_{theoretical} = \sqrt{4\pi n_e e^2 / m_e}$, где e и m_e – заряд и масса электрона соответственно, n_e – равновесная концентрация электронов в плазме. Для сравнения результата моделирования с аналитическим решением тестовой задачи на Рис. 5. приведен спектр временной эволюции напряженности электрического поля, который имеет ярко выраженный пик, соответствующий Ленгмюровским колебаниям. Относительное отклонение частоты, соответствующей максимуму спектра, от теоретического значения составляет 2,8%, что с учетом параметров моделирования соответствует ожиданиям.

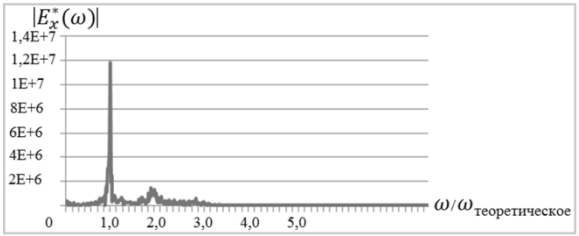


Рис. 5. Спектр временной эволюции напряженности электрического поля при моделировании Ленгмюровских колебаний холодной плазмы

Вычислительные эксперименты выполнены на кластере Нижегородского госуниверситета, включающего 64 двухпроцессорных двухъядерных сервера Intel Xeon 3.2 GHz, 4 GB RAM, сетевое оборудование на основе Gigabit Ethernet. Использовался Microsoft HPC Pack 2008 SDK, Microsoft Visual C++ 2008 SP1.

Для определений характеристик масштабируемости рассматривались две тестовые задачи (Ленгмюровские колебания плазмы) разной размерности. Параметры задач и результаты экспериментов приведены в таблицах 1 и 2. Анализ результатов позволяет сделать следующие выводы:

1. В первом тесте наблюдается существенное падение эффективности с ростом числа процессов по причине снижения вычислительной нагрузки в расчете на один процесс. К

достижению количества процессов значения 32, время, потраченное на вычисления, становится меньше времени, потраченного на передачу данных.

2. Во втором тесте наблюдается достаточно хорошая эффективность распараллеливания вплоть до использования 64-х процессов (16 двухпроцессорных двухъядерных узлов кластера).
3. Учитывая, что второй тест отличается от первого бóльшим размером сетки и количеством частиц, можно предполагать, что при решении практических задач с размерами сетки, значительно превышающими тестовые параметры, будет наблюдаться приемлемая эффективность на существенно большем числе вычислительных узлов.

Таблица 1

Тест 1: сетка 128x16x16, 983 039 частиц, 7168 итераций						
Количество процессов	**1**	**2**	**4**	**8**	**16**	**32**
Количество узлов	1	1	1	2	4	8
Вычисления, сек.	3466,70	1754,85	856,17	426,77	212,85	105,39
Передача, сек.	231,32	220,78	173,25	221,79	171,66	229,51
Итого, сек.	3698,01	1975,63	1029,42	648,56	384,51	334,90
Ускорение		1,87	3,59	5,70	9,62	11,04
Эффективность, %		**93,59**	**89,81**	**71,27**	**60,11**	**34,51**

Таблица 2

Тест 2: сетка 256x32x32, 7 864 319 частиц, 1024 итерации							
Количество процессов	**1**	**2**	**4**	**8**	**16**	**32**	**64**
Количество узлов	1	1	1	2	4	8	16
Вычисления, сек.	3902,38	1987,77	985,84	489,26	242,32	123,38	68,02
Передача, сек.	259,57	208,38	167,11	87,16	46,86	24,92	13,68
Итого, сек.	4161,94	2196,15	1152,95	576,42	289,18	148,30	81,70
Ускорение		1,90	3,61	7,22	14,39	28,06	50,94
Эффективность, %		**94,76**	**90,25**	**90,25**	**89,95**	**87,70**	**79,59**

6. Заключение

Разработана параллельная версия программного обеспечения для численного моделирования плазмы методом частиц в ячейках. Данная реализация на стандартных тестах показывает результаты, соответствующие теоретическим предсказаниям. Проведенные вычислительные эксперименты демонстрируют работоспособность параллельной версии, построенной на основе MPI и ориентированной на кластерные системы. В параллельной версии достигается эффективность ~80% при решении тестовых задач.

Целью дальнейших исследований является модификация параллельной версии с целью улучшения ее производительности и масштабируемости. Основное направление работ – создание OpenMP-реализации для распараллеливания решения задач на каждом узле кластера. Планируется переработка кода для гетерогенных систем с учетом возможностей современных графических процессоров.

Литература

1. Hockney R., Eastwood J. Computer Simulation Using Particles. – IOP, Bristol and New York, 1989.
2. Tskhakaya D. The Particle-in-Cell Method // Computational Many-Particle Physics. Lecture Notes in Physics, 2008, Volume 739/2008. – Berlin: Springer. – P. 161–189.
3. Бэдсел Ч., Ленгдон А. Физика плазмы и численное моделирование: Пер. с англ. – М.: Энергоатомиздат, 1989. – 452 с.

4. Taflove A. Computational Electrodynamics: The Finite-Difference Time-Domain Method. – London: Artech House, 1995. – 599 P.

Методика распределенных вычислений RiDE

М.О. Бахтерев, П.А. Васёв, А.Ю. Казанцев, И.А. Альбрехт

ИММ УрО РАН

RiDE - это методика для программирования в параллельных распределенных средах, основанная на модели потока данных (dataflow). RiDE превосходит подходы MPI и OpenMP по простоте использования. Также RiDE превосходит более узкие парадигмы, например Map/Reduce и Task Flow, позволяя решать более широкий класс задач. Методика обладает такими преимуществами, как автоматическая балансировка процесса вычисления, распространение счета при добавлении/удалении узлов "на лету", полная остановка счета и освобождение ресурсов с дальнейшим возобновлением, автоматическое создание контрольных точек, работа в гетерогенных и гибридных (CPU/GPU/CELL/ПЛИС/etc) средах.

1. Введение

Распространенные средства параллельного программирования, такие как MPI [1], OpenMP [2] и GLOBUS [3], требуют от программиста подробного описания большого количества сущностей. Необходимо заботиться о распределении вычислительных задач, синхронизации, обмене данными и так далее. В связи с этим, программирование с использованием этих средств является трудоемкой задачей, отвлекающей на себя существенную часть рабочего времени программиста-математика. Программы создаются долго, получаемые коды сложны и громоздки, их сопровождение и развитие оказывается затратным процессом.

Другой проблемой распространенных средств параллельного программирования является их ориентация на конкретные классы вычислительных систем: с общей памятью, кластерные системы либо распределенные системы. Программы, написанные с помощью таких средств, способны выполняться на подразумевавшемся типе параллельной системы, но не способны эффективно работать с системой другого типа. Кроме того, данные средства, как правило, не содержат встроенной поддержки ускорителей (GPU, ПЛИС и т.д.), а также концепций облачных вычислений и SaaS.

Существуют различные подходы к упрощению процесса программирования и исполнения параллельных вычислений. С одной стороны, создаются универсальные средства по автоматизации распараллеливания программ (как для исполнения в системах с общей памятью, так и в многомашинных конфигурациях, например DVM [4] и Т-Система[5]). С другой стороны, создаются среды для решения определенных классов задач (в основном это касается задач, для которых применим параллелизм «по данным», например Map/Reduce [6]). Также разрабатываются универсальные инструменты, пытающиеся упростить технические аспекты процесса программирования (например Intel TBB [7]).

Таким образом, поиск решений в сфере обозначенных проблем представляет большой интерес всего сообщества. Иногда при создании подобных решений используется модель потоков данных (Dataflow, [8]). В различных вариантах методики, основанные на моделях потоков данных, применяются для создания процессорных архитектур, суперкомпьютеров в целом, для программной организации вычислительных потоков в рамках одного процесса и взаимодействия процессов в распределенной вычислительной среде.

В настоящей работе представлена методика программирования в распределенных вычислительных средах, имеющая целью устранить показанные проблемы.

Методика основана на анализе различных, в том числе и авторских, моделей потока данных, и возникла в результате продолжительной теоретической работы над архитектурой операционной системы для распределенных вычислений [9].

2. Требования к методике распределенных вычислений

Основные критерии, на которые ориентировались авторы при разработке описываемой методики, следующие:

1) методика должна предоставить универсальный механизм параллельного программирования, более простой в применении, чем существующие универсальные средства;
2) вычислительные программы, реализованные с помощью разрабатываемой методики, должны выполняться не менее эффективно, чем при использовании других средств;
3) методика должна ориентироваться на создание и эффективное исполнение программ на всех типах вычислительных систем, в первую очередь в распределенных вычислительных средах;

Под универсальными средствами параллельного программирования понимаются средства, способные обеспечить создание разнообразных параллельных программ, не привязанных к какому-либо методу обеспечения параллелизма. Это важное и серьезное требование к разрабатываемым средствам программирования, с одной стороны позволяющее решать максимально широкий спектр задач, но затрудняющее разработку этих средств с другой стороны.

Требование более простого режима использования, чем существующие универсальные средства, не нуждается в отдельном комментарии. Аналогично не нуждается в пояснении и требование эквивалентной эффективности исполнения вычислительных программ.

Обратимся к последнему критерию. Как отмечено во введении, существующие универсальные средства параллельного программирования, как правило, ориентированы на какой-то один конкретный тип вычислительной среды. Например, OpenMP разработан для программирования систем с общей памятью, MPI – для кластерных сред, Globus Toolkit – для грид-систем, OpenCL [10] – для управления нестандартными устройствами, в первую очередь GPU. (Система MPICH-G2 [11] не вполне соответствует «кластерному» MPI, т.к. требует особого внимания программиста к нюансам используемой распределенной среды)

Вместе с тем особый интерес представляет создание такого универсального средства, которое бы обеспечивало возможность разработки и эффективного исполнения программ во всех типах вычислительных сред, а также в их смешанных конфигурациях.

Наиболее сложной задачей из встречающихся в разработке параллельных программ является программирование распределенных вычислительных сред. В работе [12] показаны отличительные особенности данного типа сред и трудности их преодоления:

1. *Масштабность* – суммарное число процессоров, объем памяти и т.д. может на порядки превосходить показатели кластерных систем.
2. *Распределенность* – расстояние между вычислительными узлами может исчисляться тысячами километров с соответствующей шириной канала и латентностью.
3. *Неоднородность* – узлы могут обладать существенно различными характеристиками.
4. *Динамичность* – вычислительные узлы могут включаться и отключаться в произвольные моменты времени, более того, можно рассматривать узлы как ненадежные.
5. *Различная административная принадлежность*.

Разрабатываемая методика параллельного программирования должна учитывать эти особенности. Кроме того, необходимо учесть и аспекты, связанные с другими типами вычислительных сред.

При работе в кластерных средах необходимо учитывать и эффективно использовать наличие быстрых связей между узлами и возможно – быстрых параллельных файловых хранилищ. В системах с общей памятью – необходимо задействовать возможность доступа к данным без их копирования, складирования или передачи по сети. Разумеется, необходимо обеспечить и возможность совместного, смешанного варианта использования различных типов вычислительных систем: например, распределенная среда может состоять из набора кластеров, каждый из которых, в свою очередь, объединяет набор многоядерных машин с общей памятью.

В качестве дополнительного аспекта необходимо учесть и наличие таких устройств, как GPGPU, ПЛИС, Cell. Желательно, чтобы в рамках методики можно было бы поддержать программирование и данных устройств с помощью удобных средств.

Итак, создаваемая методика должна учитывать всю обозначенную выше совокупность аспектов работы различных вычислительных сред.

3. Методика RiDE

Методика базируется на понятиях ***хранилища***, ***задач*** и ***правил***.

Хранилище содержит в себе именованные данные, по отношению к которым доступны три операции – создание (запись), чтение и удаление. Хранимые данные являются самодостаточными - это не очереди, но некие цельные единицы информации с уникальными именами. Допускаются операции частичного чтения данных.

Задачей называется программа, которая во время исполнения считывает данные с определенными именами из *хранилища* и в результате своего исполнения формирует новые данные, которые записываются в хранилище.

Правилом называется такая конструкция, которая определяет условия и параметры запуска *задач*. Правило содержит в себе:

1. Список имен данных, которые необходимы для выполнения задачи.
2. Список соответствия глобальных имен данных (находящихся в хранилище) локальным именам (с которыми и будет работать задача).
3. Список задач (программ), которые необходимо запустить.
4. Действия, совершаемые в случае успешного выполнения задач (3).

Правило считается готовым к исполнению, когда в хранилище присутствуют все данные с именами из списка (1). После успешного исполнения правило удаляется из списка выполняемых правил.

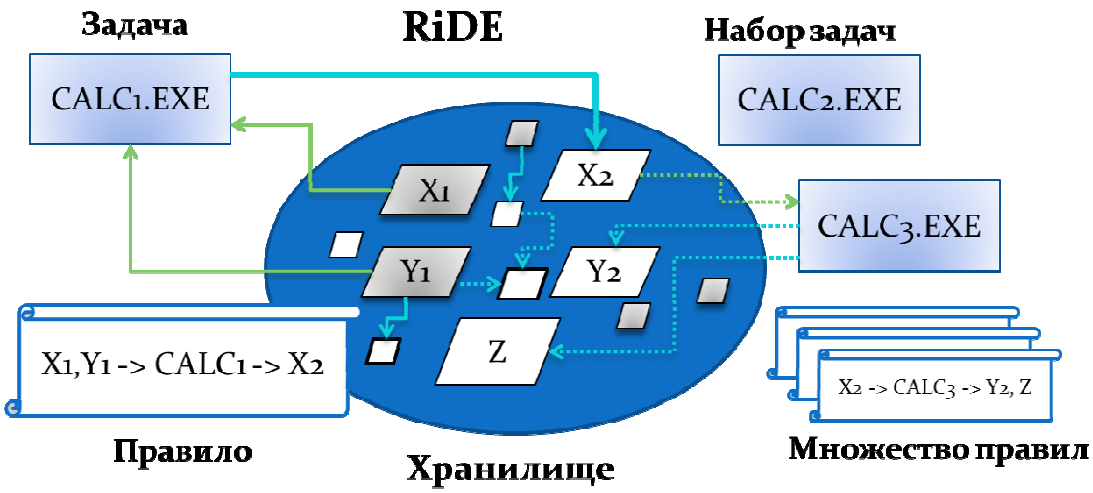


Рис. 1. Участники процесса вычислений в RiDE

На рис. 1 представлена общая схема участников описываемой методики. В центре находится хранилище. Белым выделены данные, которых в хранилище пока нет, серым - которые уже есть. Показан пример правила, которое гласит: при наличии данных X1 и Y1 необходимо запустить программу Calc1.exe, подать ей на вход эти данные, а результат работы записать в данные с именем X2. Показано и другое правило, которое требует выполнить другую программу при наличии элемента данных с именем X2. Очевидно, что это правило сработает только после того, как будет завершено первое правило. Это отмечено с помощью пунктира, который говорит, что обозначенный запуск и как результат порождение новых данных пока невозможно, но свершится в будущем. Вполне вероятно, что на рисунке есть и правила, которые могут выполняться независимо от представленных двух. Степень независимости и определяет меру параллелизма, с которой может быть произведено вычисление.

Процесс программирования и проведения вычислений в рамках RiDE происходит следующим образом. Прежде всего, разрабатываются программные коды задач, из которых состоит вычислительный эксперимент. Каждая такая задача на этапе инициализации должна считать данные из хранилища, а затем по ходу выполнения сформировать и записать новые данные в хранилище. Отметим, что в рамках одного вычисления могут использоваться любые комбина-

ции языков, а также целевых аппаратных сред для создания задач. Например, часть задач можно реализовать на графических ускорителях, а часть – на обычных процессорах.

После создания вычислительных программ (задач) программистом формируется файл инициализации, в котором описываются начальные правила системы. В дальнейшем эти правила могут дополняться – при выполнении задач или финализации правил. Кроме правил, в файле инициализации указываются начальные данные, которые помещаются в хранилище.

После подачи команды на запуск вычислительная среда ищет правила, готовые к исполнению, и запускает указанные в них задачи на подходящих свободных вычислительных ресурсах. В результате часть правил исполняется, формируя новые данные и освобождая ресурсы для других правил. Среда продолжает поиск и выполнение правил вплоть до исчерпания всех правил, приостановки работы с внешней стороны или выявления ошибки.

4. Преимущества, обеспечиваемые методикой RiDE

Программная реализация предлагаемой методики способна обеспечить следующие преимущества.

- Разделение уровней вычисления и взаимодействия, что обеспечивает более ясный процесс создания, отладки и сопровождения вычислительных программ. Вычислительные коды группируются в задачах; коммуникационные – в описаниях правил.
- Поддержка всех типов вычислительных сред – с общей памятью, кластерных и распределенных. Работа во всех средах может быть реализована эффективно, без необходимости переработки вычислительных программ, включая системы с общей памятью, где чтение и запись данных могут быть реализованы без накладных операций копирования.
- Возможность включения и отключения вычислительных ресурсов «на лету», что позволяет максимально эффективно задействовать вычислительные ресурсы и гибко планировать их распределение. Это свойство обеспечивается тем, что все обмены данными происходят через интерфейсы хранилища.
- Использование любых языков программирования для создания вычислительных программ. В программах должны присутствовать только два типа RiDE-функций – чтение и запись данных в хранилище.
- Отсутствие ограничений на внутреннюю сложность задач, вызываемых при срабатывании правил – задача, например, сама может быть параллельной MPI-программой или даже закрытой коммерческой программой, написанной вне рамок предлагаемой методики. В последнем случае взаимодействие программы с RiDE-окружением реализуется через файлы.
- Возможность участия в рамках одного вычисления программ различных платформ (различные ОС, языки программирования, процессоры и ускорители). Назначать правила на исполнение можно сообразно требованиям кодов задач к аппаратным характеристикам вычислительных узлов. Более того, в правиле можно указать различные версии вычислительных программ, написанные для разных целевых платформ; выбор конкретной версии может осуществляться исходя из имеющихся свободных вычислительных ресурсов.
- Возможность реализации поддержки программирования ускорителей. Программные средства уровня OpenCL [10] очень сложны в использовании; в рамках методики можно реализовать механизмы, упрощающие загрузку-выгрузку данных из вычислительных устройств;
- Встроенная поддержка контрольных точек. Весь обмен данными, и таким образом, текущее состояние счета, размещается в хранилище. Снимок состояния хранилища и формирует контрольную точку. При этом состоянием оперативной памяти считающихся в текущий момент задач можно пренебречь, осуществив при необходимости их пересчет.
- Возможность полностью остановить счет и в будущем продолжить его. Продолжение счета может быть осуществлено на других вычислительных ресурсах.
- Автоматическая оптимизация вычислений путем оптимального размещения правил на вычислительных узлах. Распределение правил по узлам может осуществляться на основе анализа а) статистики по предыдущим запускам, б) текущего размещения данных, в) зависимостей данных, описанных в правилах (заметим, что это описание – явное).

5. Вопросы реализации методики

Рассмотрим возможный вариант архитектуры системы, реализующей методику RiDE. Принципиально она состоит из двух уровней: среды выполнения и системы хранения. Среда выполнения отвечает за поиск готовых правил и их вычисление, система хранения отвечает за хранение всех данных, создаваемых и читаемых задачами. Также система хранения содержит и внутреннюю информацию, необходимую для работы системы в целом.

Среда выполнения состоит из следующих частей:

- ***Runner*** *(координатор)* – считывает файл исходных правил, выявляет готовые правила и размещает их для выполнения на доступных *Runsite*-узлах.
- ***Runsite*** *(область запуска)* – представитель вычислительного узла или группы узлов. Ведет учет доступных ресурсов (процессорных ядер, аппаратных ускорителей и т.д.), их типов и характеристик (архитектура, ОС, объем памяти и т.д.). Получает задания на запуск правил от модуля *Runner*.
- ***Runvisor*** *(трамплин)* – модуль, отвечающий за вычисление одного правила. *Runvisor* запускается модулем *Runsite* в результате размещения команд на выполнение правил.

Представленные участники системы и связи между ними изображены на рис. 2.

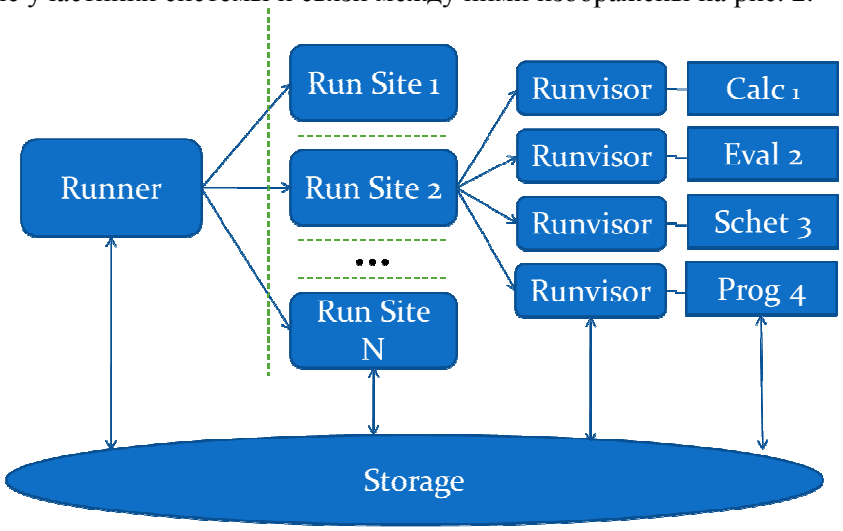


Рис. 2. Возможная архитектура системы

Модель исполнения в рамках данной архитектуры описывается следующим образом.

Модуль Runner считывает исходные правила. Также он считывает список адресов Runsite-узлов, доступных для использования. Runner определяет готовые правила, и производит поиск подходящего Runsite-узла среди множества доступных узлов. Искомый Runsite-узел должен обладать необходимым количеством свободных ресурсов и подходящими характеристиками (тип процессора, объем памяти и т.д.). Если подходящий свободный Runsite- узел обнаружен, то правило размещается на нем для исполнения. В случае, если на текущий момент таких узлов нет, то правило откладывается для последующих попыток исполнения.

При размещении правила на Runsite-узле этот узел а) производит синхронизацию вычислительных задач правила, получая при необходимости более новые версию (набор бинарных или текстовых файлов с вычислительными кодами) от модуля Runner и б) производит запуск процесса Runvisor и делегирует ему исполнение правила.

Runvisor производит процедуры подготовки запуска правила. Одним из важных этапов инициализации является настройка перенаправлений имен данных хранилища. Подразумевается, что в правилах указываются имена в глобальном пространстве имен. Однако вычислительные программы проще создавать, используя имена в локальном пространстве. Поэтому Runvisor должен позаботиться о том, чтобы передать запускаемым задачам информацию о соответствии глобальных и локальных имен.

Затем Runvisor производит запуск задач, указанных в правиле, и ожидает их завершения. В случае успешного завершения всех задач данного правила, Runvisor отмечает это правило как завершенное. При этом в ходе выполнения задач формируются новые данные, которые записываются в хранилище с применением глобальных имен. Также в ходе работы задач или на этапе финализации текущего правила могут формироваться новые правила. Таким образом, возможна генерация графа вычислений «на лету».

Вычисление считается законченным, когда Runner выявляет, что список правил, готовых к выполнению, исчерпался.

Предложенная архитектура среды выполнения опирается на распределенное хранилище данных. Рассмотрим вопросы создания такого хранилища.

6. Требования к распределенному хранилищу данных

Система хранения, необходимая для эффективной работы RiDE, должна удовлетворять определенным требованиям. Анализ существующих проектов хранилищ данных, включая различные распределенные базы данных, файловые системы, распределенные хэш-таблицы (DHT) показал, что в настоящий момент нет готовой системы, которая удовлетворяла бы всем выдвигаемым требованиям. В настоящее время авторский коллектив ведет проработку проекта распределенной системы хранения, удовлетворяющей следующим критериям:

1. Хранение как маленьких (от 1 байта), так и больших (несколько гигабайт) по объему элементов данных.
2. Хранение большого количества элементов данных (несколько миллиардов). Это связано с необходимостью поддержки вычислений, таких как сеточные расчеты; при этом необходимо хранить данные для нескольких последних итераций работы алгоритмов.
3. Работа в распределенном режиме, что включает поддержку неоднородности связей между узлами хранения.
4. Работа с неоднородными узлами и неодинаковой степени нагрузки. Большинство современных проектов распределенных хранилищ подразумевают равномерное распределение данных между узлами. Очевидно, что в случае RiDE такая политика является скорее вредной. Данные должны распределяться сообразно а) источникам этих данных, б) потребителям данных, в) возможностям машин, которые разнятся от узла к узлу.
5. Работа в многопользовательском режиме с различными пространствами имен. Требование связано с необходимостью поддержки одновременных расчетов, проводимых разными пользователями, и хранением результатов после проведения расчетов.
6. Возможность включения и исключения узлов хранения по ходу работы. С одной стороны, это требование касается условий ненадежности узлов. С другой стороны оно необходимо для реализации гибкого планирования и подключения узлов к вычислениям по ходу работы.
7. Использование отображения файлов в память для минимизации дисковых операций и разделяемой виртуальной памяти с механизмом копирования страниц при записи для дедубликации данных при одновременном доступе к одним и тем же данным из разных процессов.
8. Отслеживание появления новых данных с целью уведомления среды о готовности правил. В рамках одного вычисления в RiDE подразумевается наличие миллионов (возможно и миллиардов) экземпляров правил и соответственно, не меньшего числа данных. Соответственно для эффективного обнаружения готовых правил разумно использовать саму систему хранения, полностью или частично возложив на нее эту работу.
9. Эффективный поиск элементов данных по имени или части имени. Данное требование при реализации пункта (8) не является обязательным для RiDE, однако в общем случае такая функциональность была бы полезной.

7. Эксперимент по внедрению методики

Одним из хороших примеров применения методики RiDE является портирование с MPI rho-алгоритма Полларда для поиска дискретных алгоритмов. Основным преимуществом применения RiDE было сокращение исходного кода и упрощение логики параллельного выполнения.

Базовый алгоритм основан на поиске цикла (пересечения) в некоторой, специальным образом определенной, последовательности элементов, для чего используется алгоритм Флойда поиска циклов. Основным преимуществом алгоритма является минимальное требование к памяти - O(1), но он является очень сложным для параллельного вычисления и эффективность распараллеливания составляет sqrt(m), где m – количество вычислительных процессов.

Для оптимизации параллельной работы данного алгоритма была использована модификация rho-алгоритма Полларда со «специальными» точками, описанная в [14]. Данная модификация позволяет добиться практически линейного масштабирования, но требует большого объема памяти для эффективного выполнения (time-memory tradeoff).

Приведем краткое описание выполнения алгоритма для различных реализаций.

1. Выбрать начальные значения и некий критерий определения «специального» значения;
2. Вычислять значения последовательности пока не найдется значение, подпадающее под критерий «специального» значения;
3. Проверить, существует ли данное значение в хранилище;
4. Если значения нет в хранилище – записать его в хранилище и перейти к пункту 2;
5. Вычислить результат.

Рис. 3. Линейный алгоритм

1. Выбрать начальные значения (различные для каждого процесса) и общий критерий определения «специального» значения;
2. Каждый процесс:
   - 2.1. Ищет новое «специальное» значение, независимо от других процессов;
   - 2.2. Проверяет новые, входящие точки, от других процессов (каждые несколько секунд) и ищет входящие значения в локальном хранилище. Если значение найдено, то переходит к пункту 3;
   - 2.3. При нахождении нового «специального» значения процесс должен проверить, существует ли оно в локальном хранилище и перейти к пункту 3 если существует, либо отослать новое значение всем процессам в противном случае;
   - 2.4. Перейти к пункту 2.1;
3. Вычислить результат.

Рис. 4. MPI реализация алгоритма

1. Выбрать начальные значения (различные для каждой задачи), получить параметры алгоритма из хранилища и выбрать общий критерий определения «специального» значения;
2. Каждая задача:
   - 2.1. Находит новое специальное значение;
   - 2.2. Ищет его в распределенном хранилище данных и

```
        переходит к пункту 3 если такое значение суще-
        ствует;
    2.3. Порождает новое правило повторения итерации и
        заканчивает работу;
3. Вычислить результат.
```

Рис. 5. RiDE реализация

Приведенные выше алгоритмы показывают, что реализация с использованием методики RiDE позволила:

- Существенно облегчить логику каждой отдельной задачи;
- Исключить необходимость пересылки и проверки новых входящих значений между процессами;
- Сократить размер кода. На практике, более 60% кода было просто удалено и заменено несколькими клиентскими вызовами RiDE на чтение и запись данных;

Все изменения не повлияли на производительность исходной реализации алгоритма [14].

8. Заключение

В работе представлена методика параллельного программирования, которая может быть использована при программировании распределенных вычислительных сред, кластерных систем и систем с общей памятью.

Методика позволяет достаточно просто и эффективно реализовать проведение вычислительного эксперимента на гибридных архитектурах, позволяет осуществлять динамическое изменение количества вычислительных узлов по ходу вычислений, автоматизирует создание контрольных точек, приостановку и продолжение вычислений прозрачным для программиста образом, а также обеспечивает ряд других преимуществ.

Одной из целей, которые были поставлены при создании методики, являлось упрощение процесса создания суперкомпьютерных вычислительных программ, по отношению к средствам класса MPI и OpenMP. Первые эксперименты показывают, что методика достигла этой цели.

На основе предложенной методики авторами прорабатываются вопросы реализации среды параллельного программирования. Первые эксперименты показывают реализуемость предлагаемых идей и лаконичность программных конструкций для описания правил. Авторы выражают уверенность, что в результате развития этой среды удастся достичь главной цели – сделать процесс создания распределенных вычислительных программ более простым и эффективным.

Информация о ходе исследования размещается на сайте www.ridehq.net.

Литература

1. Dongarra J. J., Hempel R., Hey A. J. G., and Walker D. W. A proposal for a user-level, message passing interface in a distributed memory environment. Technical Report TM-12231 // Oak Ridge National Laboratory, February 1993.
2. OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface 1.0, 1997.
3. Foster I., Kesselman C. Globus: A Metacomputing Infrastructure Toolkit. // Intl Journal Supercomputer Applications, 1997, 11(2): P. 115-128.
4. Konovalov N.A., Krukov V.A., Mihailov S.N. and Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development // Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience, Institute for System Programming, RAS, Moscow, 1994, P. 124-133.

5. Абрамов С.М., Васенин В.А., Корнеев В.В., Московский А.А., Роганов В.А. Организация распределенной общей памяти в Т-системе с открытой архитектурой (рус.) // ИПС РАН, ЦНТК. — 2003. — С. 13.
6. Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters // OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
7. Michael Voss. Demystify Scalable Parallelism with Intel Threading Building Block's Generic Parallel Algorithms // DevX.com, Jupiter Media, October 2006.
8. Dennis J. Data Flow Supercomputers // Computer, 1980, Vol.13, No.11, P.48-56.
9. Бахтерев М.О. Описание параллельных вычислений при помощи замыканий // Тезисы 10-го Международного семинара "Супервычисления и Математическое моделирование", РФЯЦ-ВНИИЭФ, Саров, 2008, С. 31-32.
10. Jaejin Lee et al. An OpenCL framework for heterogeneous multicores with local memory // In PACT '10: Proceedings of the 19th international conference on Parallel architectures and compilation techniques, 2010, P. 193-204.
11. Karonis N., Toonen B., and Foster I. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface // Journal of Parallel and Distributed Computing, 2003, P. 1-22.
12. Воеводин Вл.В. Решение больших задач в распределенных вычислительных средах. //Автоматика и Телемеханика. 2007, N5, С. 32-45.
13. Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications // 1996 September 23, P. 1-30.
14. Albrekht I. Some methods for DLP solving, and time estimation // Information Security Conference, Yekaterinburg, 2005, P. 20-25.

Параллельная реализация комплекса программ для задачи определения параметров электрического диполя *

А.К. Богушов, А.В. Панюков

Южно-Уральский государственный университет

Рассматривается задача идентификации параметров произвольно ориентированного электрического диполя над плоскостью с бесконечной проводимостью по его электромагнитному полю, индуцируемому в точке наблюдения. Данная задача входит в комплекс математических моделей практически важной проблемы прогнозирования развития грозовых очагов. Для решения задачи, из-за ее плохой обусловленности, предлагается использовать параметризованное семейство алгоритмов, а окончательное решение принимать по результатам статистического анализа. В докладе представлена параллельная реализация комплекса программ для портативных компьютеров с использованием технологии OpenCL.

1. Введение

Электрический диполь над плоскостью с бесконечной проводимостью является математической моделью, адекватно представляющей внутриоблачные молниевые разряды. Поэтому задача идентификации местоположения электрического диполя по его электромагнитному полю индуцируемому в точке наблюдения над плоскостью с бесконечной проводимостью актуальна. Работы посвященные данной проблеме публикуются в ведущих международных журналах [1], [2], [3], [4], [5].

Рассматриваемая задача, как и большинство обратных задач математической физики, является плохо обусловленной. Следствием этого является высокая чувствительность алгоритма к погрешностям в исходных данных и погрешностям вычисления.

Во всех эмпирических измерениях, существует определенная вероятность погрешности между измеренным и истинным значением. На результаты измерения могут оказать влияние как случайные факторы, так и неточность моделирования. Одним из способов устранения подобных погрешностей является многократное измерение или расчеты с использованием множества алгоритмов. Затем полученные результаты можно представить в виде гистограммы, получив таким образом эмпирическое распределение вероятности для величин измеряемых параметров. Исходя из центральной предельной теоремы, разумным представляется принятие гипотезы о нормальном законе распределения. В этом случае по моде распределения можно отсеять заведомо неподходящие алгоритмы, а по оставшимся результатам проверить гипотезу нормальности, и получить оценки параметров и их доверительные интервалы.

При наблюдениях электромагнитного поля из одного пункта множественное измерения параметров источника излучения можно получить применением различных алгоритмов. Среди возможных методов определения параметров положения произвольно ориентированного электрического диполя, в работе будет рассмотрена параметризация прямого метода [1], [3], [7], как наиболее простого в реализации. Однако заметим, что для повышения статистической значимости оценки параметров следует использовать по возможности большее число алгоритмов.

*Работа выполнена при поддержке РФФИ, проект № 10-07-96003-р_урал_а

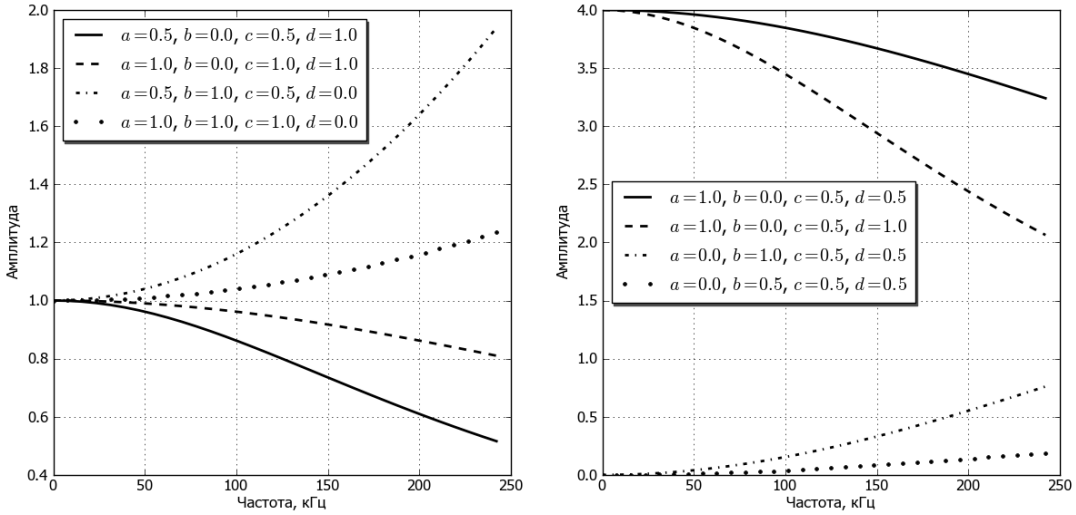


Рис. 1. Примеры АЧХ используемых передаточных функций

2. Прямой алгоритм и его параметризация

Известно [1], что задача идентификации параметров диполя сводится к задаче определения параметров u, v и α системы (1) по наблюдаемым сигналам $e(t)$ и $h(t)$.

$$e(t) = q(t)v + q'(t)\frac{v}{\alpha} + q''(t)\frac{u}{\alpha^2}, \qquad h(t) = q'(t)\frac{v}{\alpha} + q''(t)\frac{u}{\alpha^2}. \tag{1}$$

где $\alpha = c/r$, c – скорость света, r – расстояние от наблюдателя до диполя, $q(t)$ – неизвестный дипольный момент.

Прямой алгоритм, рассмотренный в работах [1], [3], имеет вид:

$$\alpha = \sqrt{\frac{h_1e_2 - h_2e_1}{h_2e_0 - h_1e_1}}; \quad u = \frac{e_0e_2 - e_1^2}{h_2e_0 - h_1e_1}; \quad v = \frac{3h_0u\alpha - e_0\alpha - g_0}{2h_0\alpha}, \tag{2}$$

где

$$e_k = \int_0^{+\infty} e^{(k)}(t)h^{(k)}(t)dt; \quad h_k = \int_0^{+\infty} (h^{(k)}(t))^2dt; \quad g_k = \int_0^{+\infty} e^{(k+1)}(t)h^{(k)}(t)dt; \quad k = 0, 1, 2.$$

С прямым алгоритмом было проведено большое количество численных экспериментов [6]. В качестве входных данных в этих экспериментах выступали сигналы полученные с помощью системы имитации сигналов $e(t)$, $h_x(t)$ и $h_y(t)$ [8].

Для повышения степени разнообразия сигналов от источника с заданными параметрами положения можно использовать предобработку измеренных сигналов $e(t)$, $h(t)$ линейными фильтрами. В работе для предобработки входных сигналов $e(t)$ и $h(t)$ использовано параметризованное семейство фильтров (см. рис. 1) с передаточными функциями

$$F(\omega) = \frac{a + j\omega b}{c + j\omega d}, \quad a, b, c, d \in [0;\ 1], a + b \neq 0,\ c + d \neq 0.$$

Данное семейство фильтров достаточно представительное: оно содержит фильтры в разной степени усиливающие и(или) ослабляющие как верхние, так и нижние частоты.

На модельных примерах было замечено, что идентификацию удаленных диполей лучше осуществлять на нижних частотах, а ближних диполей – на верхних частотах. Поскольку

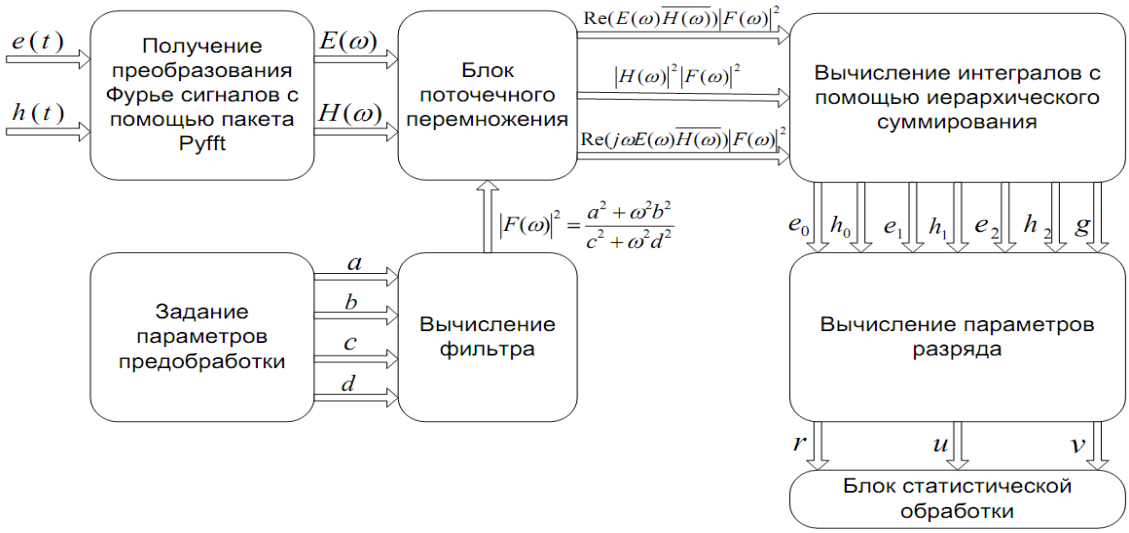


Рис. 2. Схема обработки

при обработке реальных сигналов не представляется возможным определить какой из фильтров необходимо использовать для получения наилучшей оценки, то оценка параметров u, v, α осуществляется по результатам статистической обработки всех результатов.

С целью повышения эффективности алгоритма, значения переменных e_k, h_k, g_k целесообразно вычислять в терминах преобразования Фурье. Из равенства Парсеваля имеем:

$$e_k = \int_{\omega_0}^{\omega_{N/2}} (\omega)^{2k} \mathrm{Re}\left(E(\omega)\overline{H(\omega)}\right)|F(\omega)|^2 d\omega, \quad h_k = \int_{\omega_0}^{\omega_{N/2}} (\omega)^{2k}|H(\omega)|^2|F(\omega)|^2 d\omega,$$

$$g_k = \int_{\omega_0}^{\omega_{N/2}} (\omega)^{2k} \mathrm{Im}\left(\omega E(\omega)\overline{H(\omega)}\right)|F(\omega)|^2 d\omega, \quad k = 0, 1, 2, \tag{3}$$

где $\omega \in [\omega_0; \omega_{N/2}]$, ω_0 – нижняя частота среза полосового фильтра, $\omega_{N/2}$ – верхняя частота среза полосового фильтра.

3. Реализация параметризованного семейства алгоритмов с использованием технологии OpenCL

Чаще всего алгоритмы определения местоположения грозового разряда используются в автономных грозопеленгаторах. Специфика подобных устройств накладывает существенные ограничения на их аппаратную часть. Данные устройства должны быть энергоэффективными, компактными и в тоже время обеспечивать обработку данных в реальном времени. В связи с этим использование суперкомпьютеров или мощных настольных вычислительных машин не представляется целесообразным. Следует отметить, что в последнее время большую популярность приобрели компактные и энергоэффективные портативные платформы, например Intel Atom. Кроме того, производители оснащают данные устройства довольно мощными блоками графических процессоров, которые поддерживают технологии массивно-параллельных вычислений, таких как Nvidia CUDA и OpenCL. Поэтому, даже если центральный процессор портативной платформы и не обладает большой производительностью, возможность его использования в связке с портативными блоками графических процессоров, поддерживающими технологии Nvidia CUDA и OpenCL, позволяет с успехом решать поставленные задачи.

Схема обработки сигнала и вычисления параметров положения диполя представлена на рис. 2. На вход системы подаются два сигнала $e(t)$, $h(t)$. Далее для этих сигналов вычисляются преобразования Фурье, по которым можно определить $\mathrm{Re}(E(\omega)\overline{H(\omega)})$, $|H(\omega)|^2$,

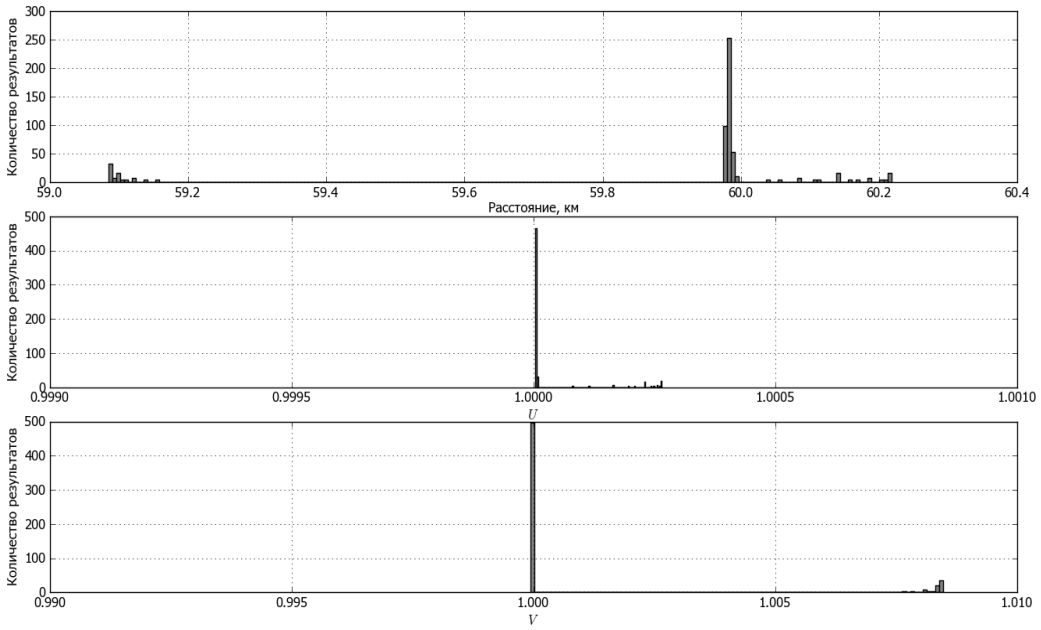


Рис. 3. Результаты статистической обработки

$\mathrm{Im}(E(\omega)\overline{H(\omega)})$. Затем для заданных параметров алгоритма a, b, c, d, в соответствии с (3), определяются значения параметров e_k, h_k, g_k, затем в соответствии с (2) находятся параметры r, u, v разряда, которые отправляются в блок статистической обработки. Блок статистической обработки по всей совокупности параметров a, b, c, d определяет оценки параметров r, u, v и их доверительные интервалы.

4. Вычислительный эксперимент

В качестве языка программирования используется Python. Pyopencl выступает в качестве API для проведения массивно-параллельных вычислений на графическом процессоре. Преобразование Фурье сигналов $e(t)$, $h(t)$ выполняется с помощью пакета Pyfft, который в свою очередь также использует пакет Pyopencl. Тестирование алгоритмов проводилось на портативном компьютере: центральный процессор Intel Core 2 Duo T5750 2ГГц, 2 Гб оперативной памяти, графический адаптер – Nvidia GeForce 9200M GS (содержит 8 вычислительных ядер).

Для иллюстрации работы алгоритма взяты сигналы от имитатора, входящего в библиотеку [8]. Была использована аппроксимация эквивалентного дипольного момента для атмосфериков нормального типа [9]

$$P(t) = P_0 \left[(a_1 t)^3 \exp^{-a_1 t} + a_2 (a_3 t)^3 \exp^{-a_3 t} \right], \tag{4}$$

со средними значениями параметров $a_1 = 0.023$ мкс$^{-1}$, $a_2 = 3$, $a_3 = 0.0035$ мкс$^{-1}$. Эквивалентный дипольный источник был размещен на расстоянии 60 км, $u = 1$ и $v = 1$. Результаты работы представлены на рис. 3.

Параметры a, b, c, d выбирались из множества $\{0.0, 0.25, 0.5, 0.75, 1.0\}$. В рассматриваемом случае высокая точность измерения: $\sigma_r = 0.3$ км, $\sigma_u = 7.7 \cdot 10^{-5}$, $\sigma_v = 0.003$, – объясняется идеальностью (отсутствием шумов) имитируемых функций $e(t)$, $h(t)$. Для реальных сигналов величина дисперсии оказывается выше.

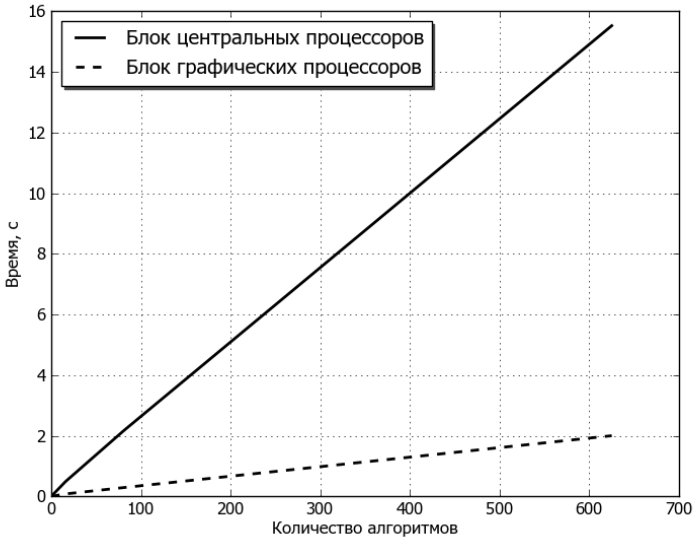


Рис. 4. Время обработки разряда

На рис. 4 представлены результаты измерения времени работы алгоритма реализованного на блоке центрального процессора и блоке графических процессоров. Как видно, реализация алгоритма с помощью технологии OpenCL позволяет уменьшить время обработки разряда в 7 раз (в нашем случае пропорционально числу процессоров в графическом блоке). Это в свою очередь позволяет увеличить количество разрядов обрабатываемых за единицу времени.

5. Заключение

Таким образом предложенный подход к решению проблемы определения параметров электрического диполя показал свою состоятельность. Использование в данной методике других базовых алгоритмов, например [2, 6], увеличит статистическую значимость результатов, получаемых таким образом.

Литература

1. Panyukov A.V. Estimination of the location of an arbitrary oriented dipole under single-point direction finding // Journal of geophysical researche. 1996, June 27. Vol. 101. № D10.

2. Panyukov A.V., Strauss V.A. A Method to determine parameters of a linear functional equation set and its application to location systems // Parameter identification and inverse problems in hydrology, geology and ecology. / J. Gottlieb and P. DuChateau (eds.). – Kluwer Academic Publishers, Printed in the Netherlands. –1996. – P. 199 – 209.

3. Panyukov A.V. Analysis of the error of a direct algorithm for determining the distance to an electric dipole. // Radio physics and Quantum Electronics. - Vol. 42. - No 3. - 1999. - P. 239 - 248.

4. M. Popov, S.He. Identification of a transient electric dipole over a conducting half space using a simulated annealing algorithm // Journal of Geophysical Research. 2000. Vol. 105. № D16. P. 20821–20831.

5. B.Z. Taibin. An approach to define parameters for localization of thunderstorm // IEEE Antennas and Propagation Magazine. 2006. Vol. 48. P.48–54.

6. Панюков А.В., Будуев Д.В. Алгоритм определения расстояния до местоположения молниевого разряда // Электричество 2001, апрель. Т. 4. С. 10–14.

7. Панюков А.В., Будуев Д.В., Малов Д.Н. Системы пассивного мониторинга грозовой деятельности // Вестник Южно-Уральского государственного университета. Серия: Математика, Физика, Химия. 2003. № – 8(24). С. 11–20.

8. Панюков А.В., Будуев Д.В. Библиотека методов определения местоположения дипольного источника излучения // Программа для ЭВМ, базы данных, топология интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам №2(1). – 2002. – С.149 – 150.

9. Кононов И.И., Петренко И.А., Снегуров В.С. Радиотехнические методы местоопределения грозовых очагов. – Л.:Гидрометеоиздат, 1986.

Математическое моделирование сложных строительных конструкций и сооружений с использованием суперкомпьютеров

Ю.Я. Болдырев, Д.В. Климшин, А.С. Шанина

ГОУ «Санкт-Петербургский Государственный Политехнический университет»

На данный момент развитие компьютерных технологий достигло такого уровня, что появилась возможность с достаточно большой степенью точности моделировать сложные реальные объекты, уменьшить объем натурных экспериментов или, вообще, обходиться без них. С помощью высокопроизводительных вычислений стало возможно моделировать распределение воздушных потоков в масштабе целого микрорайона, поведение грунтовых массивов с использованием моделей грунтов, учитывающих их нелинейные свойства (drucker-prager, mohr-coulomb, cam-clay)на различных этапах строительства в 3-мерной постановке. Безусловно, это все очень вычислительно ресурсоемкие процессы [1], но их использование насущное требование практики. Для стандартных строительных конструкций такие затраты, как правлило, неоправданны, но при проектировании уникальных объектов, высотных сооружений без таких технологий сегодня не возможно обойтись.

1. Введение

Существующие тенденции в строительстве (усложнение форм и нагрузок, индивидуальность, комбинированность конструкций и материалов), недостаточность нормативной базы, аварийность, принципиальная непредсказуемость разного рода воздействий и их сочетаний, уникальность каждого грунтового основания, геометрии самой конструкции, неопределенность и неполнота знаний о материалах и нагрузках вынуждают инженеров при проектировании проводить все более детальные исследования самих сооружений, их оснований и фундаментов. Причем большинство расчетов уже давно выходят за пределы установленных нормативных документов и используют сложные модели и современный математический аппарат. Разработка и совершенствование современных инженерных методов и математических технологий и их внедрение в сферу проектирования в строительной индустрии проводится с целью повышения надежности, безопасности и экономической эффективности зданий. На данный момент развитие компьютерных технологий достигло такого уровня, что появилась возможность с достаточно большой степенью точности моделировать сложные реальные объекты [2], уменьшить объем натурных экспериментов или, вообще, обходиться без них. С помощью высокопроизводительных вычислений стало возможно моделировать распределение воздушных потоков в масштабе целого микрорайона, поведение грунтовых массивов с использованием моделей грунтов, учитывающих их нелинейные свойства (drucker-prager, mohr-coulomb, cam-clay) [11] на различных этапах строительства в 3-мерной постановке. Безусловно, это все очень вычислительно ресурсоемкие процессы, но их использование насущное требование современной практики. Для стандартных строительных конструкций такие затраты неоправданны, но при проектировании уникальных объектов, высотных сооружений без таких технологий сегодня не возможно обойтись.

В данной статье приведен ряд задач строительной индустрии, для решения которых применены современные технологии математического моделирования в областях механики грунтов, вычислительной аэродинамики, строительной механики.

2.1 Расчет нелинейного поведения подкрановых колонн за пределом пластичности

Здесь нашей задачей является анализ устойчивости подкрановых колонн в нелинейной постановке с учетом упругопластической работы всех элементов [6]. Отметим, что такие колонны являются важными элементами конструкций в цехах промышленных предприятий. Данная за-

дача является особо актуальной при анализе возможности продления ресурса старых подкрановых колонн в промышленных цехах. При этом важным вопросом является возможность учета изменения толщины стенок профилей из-за ржавчины. Проводится сравнение результатов численного моделирования с аналитическими оценками (**1**) значения критической силы (**Рис. 1**):

$$P_{кр.аналит} = \frac{\pi^2 EJ}{l^2} \frac{1}{1+\frac{\pi^2 EJ}{l^2}\left(\frac{ab}{12EJ_b}+\frac{a^2}{24EJ_c}\right)}, \quad J = 2J_c + F_c\frac{b^2}{2}, \qquad (1)$$

где E - модуль упругости материала, Fc - площадь поперечного сечения вертикальной ветви, Jb - момент инерции поперечного сечения распорки, Jc- момент инерции поперечного сечения вертикальной ветви относительно центральной оси параллельной оси изгиба.

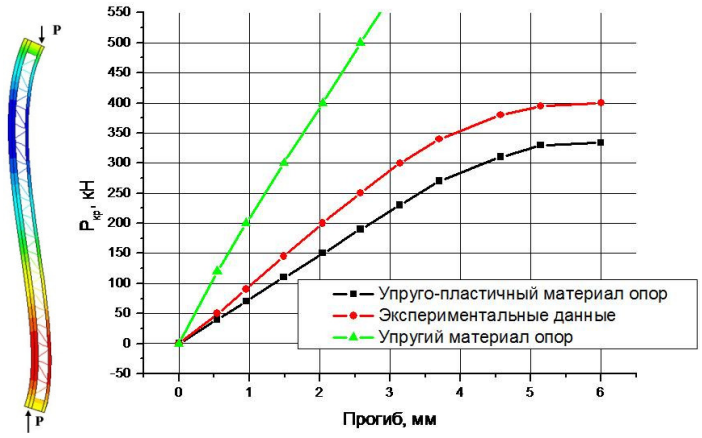


Рис. 1. Сравнение экспериментальных данных с результатами вычислений

2.2 Численное моделирование поведения уникальной металлической купольной конструкции диаметром 127 метров

Здесь представлены некоторые результаты расчета уникального металлического купола диаметром 127 метров (**Рис. 2**) [14], проведен выбор наилучшей конструктивной схемы [3], продемонстрирован расчет сложных узловых соединений в пространственной твердотельной постановке. Выявлено существование ряда ограничений и допущений, существенно влияющих на ход проектирования уникальных металлических конструкций. В частности, это касается методики учета жесткости узловых соединений конструкций [4], где необходимо проведение тщательного анализа того, что любой реальный узел металлической конструкции обладает некоей конечной жесткостью во всех направлениях, что не всегда учитывается в моделях современного проектирования. Представленные методики расчета позволяют учитывать реальную жесткость узловых соединений, анализировать поведение конструкции в нелинейной постановке.

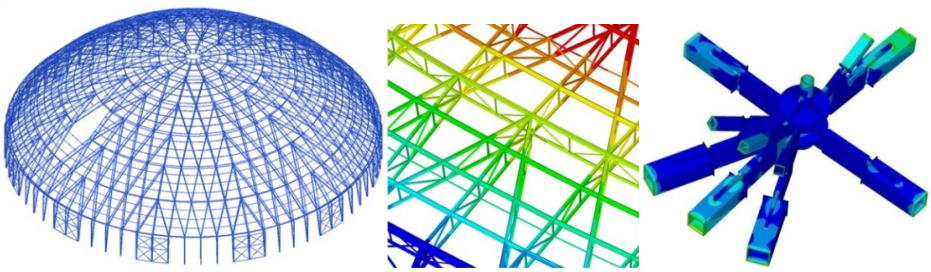


Рис. 2. Большепролетный металлический купол, результаты моделирования деформаций модели и поле напряжений отдельного узла в составе конструкции

2.3 Численное моделирование поведения бетонных конструкций со сложной геометрией с учетом напрягаемых элементов в теле бетона

В данном разделе представлена методика расчета бетонных конструкций со сложной геометрией с учетом напрягаемых элементов в теле бетона (**Рис. 3, 4**). Показано, что данная задача не может быть решена в большинстве программных комплексах из-за весьма существенных ограничений, как в инструментарии геометрического моделирования, так и в возможностях учета нелинейного взаимодействия бетона с напрягаемыми тросами. Отмечено, что применение напрягаемых тросов в конструкциях сооружений, является одной из наиболее сложных задач современного проектирования уникальных зданий, требующей дальнейших исследований [5]. В частности одной из проблем является методика учета скольжения троса в теле бетона, для чего в данной работе применяется специальное условие скользящего сопряжения (**2**):

$$tg\alpha = \frac{U_X^Б - U_X^T}{U_Y^Б - U_Y^T}; U_Z^Б = U_Z^T$$ (Б – бетон, Т - трос) и жесткого сопряжения (**2**)

$$U_X^Б = U_X^T; U_Y^Б = U_Y^T; U_Z^Б = U_Z^T$$

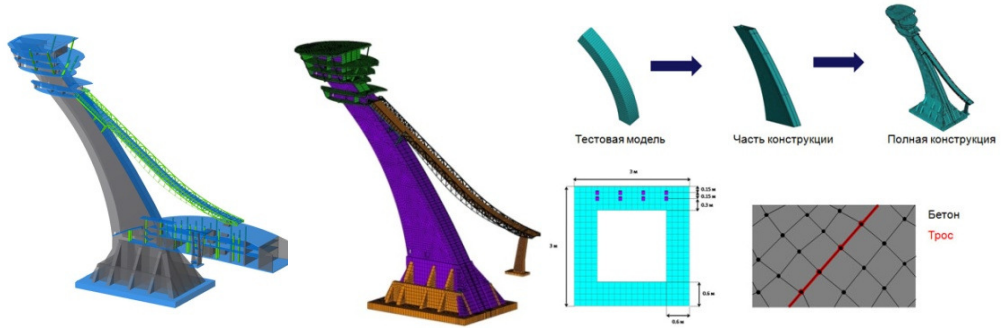


Рис. 3. Сооружение бетонного трамплина

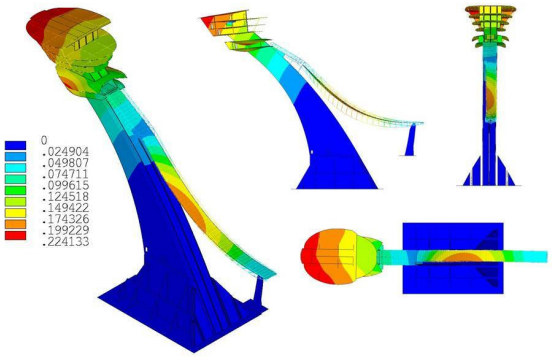


Рис. 4. Результаты применения разработанной методики – перемещения элементов сооружения (м)

С учетом возрастающей сложности возводимых сооружений, одним из наиболее перспективных направлений становится проектирование зданий с напрягаемыми элементами, которое представлено в работе. Основными преимуществами данного направления являются суммарная экономическая эффективность проведения монолитных работ и возведения сооружения в целом, сокращение расхода стали и бетона до 50%, а следовательно, и облегчение здания в 2-3 раза, уменьшение толщины перекрытия, значительное увеличение безопорных пролетов перекрытия [7], приобретение высокой несущей способности перекрытий всего за 3-4 дня, снижение суммарных затрат на строительство более чем в 2 раза.

2.4 Численное моделирование ветрового воздействия на комплекс трамплинов

Численное моделирование ветрового воздействия на здания и сооружения относится к классу задач прикладной аэродинамики. При проектировании высотных зданий важно полу-

чить детальную картину обтекания их ветром [7], при этом для объектов сложной формы, таких как рассматриваемый многофункциональный комплекс, такая задача, при решении ее в полном объеме (т.е. с учетом всех особенностей архитектурного решения фасада и ветрового профиля, - нарастания скорости ветра с высотой) достаточно трудоемка даже для современных программных комплексов и требует применения суперкомпьютерных ресурсов. Такой класс задач относится к области турбулентных течений за плохообтекаемыми телами, включая и отрывные течения.

Важным обстоятельством является то, что для данных объектов требуются испытания в аэродинамических трубах. Подобные эксперименты имеют ряд существенных недостатков таких как высокая стоимость их проведения, сложность изготовления моделей объектов, ограниченное количество обдуваемых направлений, и как следствие, качество оценки результатов.

Современные методы математического моделирования с использованием суперкомпьютеров позволяют:

- существенно сократить количество испытаний в аэродинамических трубах и в перспективе отказаться от них;
- уделить особое внимание тонким вопросам вибрационного и шумового воздействия ветрового потока в зонах углов здания и выступающих элементов, которые трудно реализуемы в физическом эксперименте;
- определять оптимальную по ряду критериев форму, размеры и ориентацию здания (например, исходя из минимизации отрицательного воздействия наружных климатических условий на энергетический баланс здания);
- разработать комплексную оценку влияния сооружений на окружающую застройку.

В данной работе представлено моделирование обтекания трамплинов спортивного комплекса. Используемый программный комплекс: ANSYS CFX (входит в состав вычислительной среды инженерного анализа ANSYS Workbench)

Проведен численный расчет турбулентного несжимаемого течения на основе решения системы уравнений Навье-Стокса (**3**), в состав которой входят уравнение неразрывности и уравнение переноса импульса:

$$\frac{\partial \rho}{\partial t}+\frac{\partial \rho u_j}{\partial x_j}=0 \quad , \qquad (\mathbf{3})$$

$$\frac{\partial \rho u_i}{\partial t}+\frac{\partial \rho u_j u_i}{\partial x_j}=-\frac{\partial p}{\partial x_i}+\frac{\partial \tau_{ij}}{\partial x_j} \quad .$$

где u_j - (j=1,2,3) компоненты вектора скорости, p и ρ - давление и плотность воздуха, соответственно, а τ_{ij} - тензор напряжений для Ньютоновской жидкости (**4**):

$$\tau_{ij}=2\mu S_{ij}-\frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij} \quad , \ (j,i=1,2,3),\ (k=1,2,3) \qquad (\mathbf{4})$$

где μ – динамическая вязкость воздуха, а S_{ji} – тензор скоростей деформаций (**5**):

$$S_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right) \qquad (\mathbf{5})$$

Приведены результаты моделирования движения воздушных потоков в масштабе всего комплекса с учетом спортивного стадиона с трибунами, 2 трамплинов и рельефа местности с существенными перепадами высот (**Рис. 5**)[12]. Расчеты проводились на основе численного решения трехмерных уравнений аэродинамики с учетом турбулентности внешнего ветрового потока на основе модели k-ε. Уравнения движения преобразуется к виду, в котором добавлено влияние флуктуации средней скорости (в виде турбулентной кинетической энергии) и процесса уменьшения этой флуктуации за счет вязкости (диссипации). В данной модели решается 2 дополнительных уравнения для переноса кинетической энергии турбулентности и диссипации турбулентности.

Приведены результаты решения задач, в рамках которых определены аэродинамические коэффициенты ветрового давления на здания с учетом взаимного влияния сооружения друг на друга.

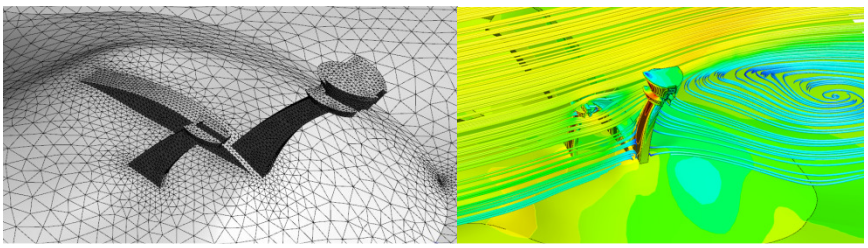


Рис. 5. Конечно - элементая сетка и характерное распределение воздушных потоков вокруг трамплина

2.5 Численное моделирование осадки комплекса трамплинов

Применение технологий математического моделирования [12] используется для расчета грунтовых массивов под сооружениями лыжных трамплинов с учетом взаимного влияния фундаментов [8,9]. Для этого разработана конечно-элементная модель грунтового массива, находящегося под трамплинами спортивного комплекса (**Рис. 6**). Проведены сравнительные расчеты в программных комплексах ANSYS/CIVILFEM и PLAXIS [13]. Также произведен линейный и нелинейный расчеты по деформациям массива грунта при совместном взаимодействии основания со всеми зданиями.

Осадки одного из трамплинов комплекса (Hs140) и примыкающего к нему зданию, полученные при учете нелинейных свойств, не удовлетворили нормативным требованиям [8,10].

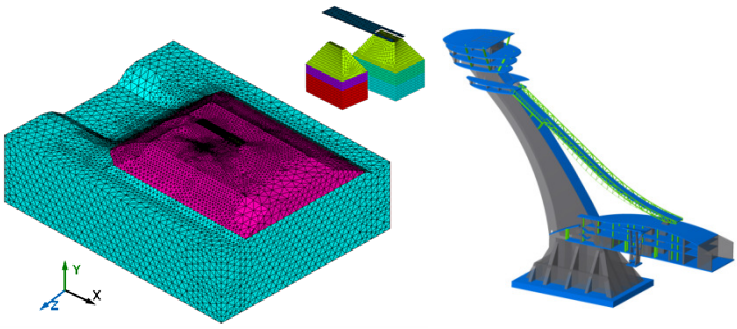


Рис. 6. Конструкция трамплина и насыпной грунтовый массив.

В результате, было предложено усиление насыпи и усиление подстилающего слоя, либо замена свайного фундамента на кессонный.

2.6 Численное моделирование поведения спуска в подземный пешеходный переход и влияние последствий строительства на пролегающий под ним канализационный коллектор

Был выполнен комплекс вычислительных исследований, направленный на определение условий, при которых возможно строительство спуска в подземный пешеходный переход, расположенного непосредственно над тоннельным канализационным коллектором, а также мероприятий по сохранности:

1) существующего тоннельного канализационного коллектора при строительстве и эксплуатации подземного пешеходного перехода;

2) подземного пешеходного перехода в случае аварии на тоннельном канализационном коллекторе.

Рассмотрены последовательно следующие этапы: до начала строительства ППП (подземного пешеходного перехода), разработка строительного котлована, нормальная эксплуатация ППП и ТКК (тоннельный канализационный коллектор) и аварийная ситуация при частичном обрушении коллектора. Предложено конструктивное решение защитного моста между ТКК и

ППП. Ниже показаны (**Рис.7**) показаны КЭ модель нормальной эксплуатации ППП, конструктивное решение подземного защитного моста.

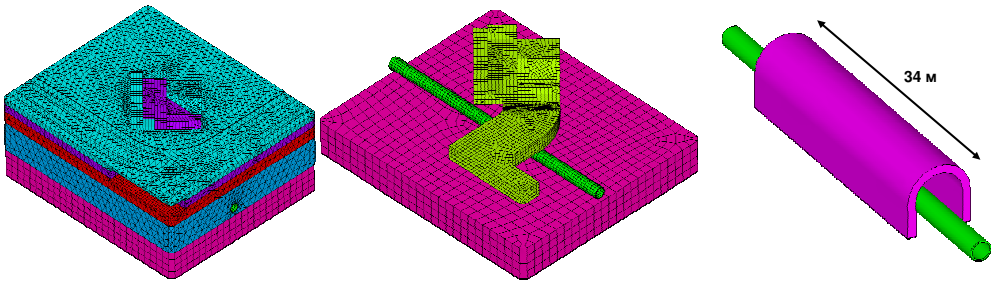


Рис. 7. КЭ модель нормальной эксплуатации ППП , конструктивное решение подземного защитного моста.

Результаты данного исследования использованы НИИ «Севзапинжтехнология» при расчетном обосновании проекта подземного пешеходного перехода у станции метро «Академическая» в Санкт - Петербурге и при разработке патента № 60166 на полезную модель “Экранирующая конструкция между инженерным сооружением и находящимся под ним трубопроводом“ (патентообладатель – Научно-исследовательский и проектно-изыскательский институт «Севзапинжтехнология»).

3. Выводы

Важнейшим выводом, который можно сделать по результатам данной работы является вывод о возможности и даже более о насущной необходимости применения современных технологий математического моделирования с использованием суперкомпьютеров для решения междисциплинарных задач инженерного анализа в сфере строительства. О насущной необходимости имеет смысл говорить по той простой причине, что «по большому счету» такому подходу нет альтернативы, поскольку большинство расчетов, как отмечалось, уже давно выходят за пределы установленных нормативных документов, используют сложные численные модели, современный математический аппарат и требуют мощных вычислительных ресурсов. Будучи уже весьма востребованными сегодня и, несомненно, все более востребованными в будущем, такие технологии требуют пристального внимания с многих точек зрения. Это и вопросы корректного применения программных комплексов и корректных же постановок задач в них, вопросы адекватности моделей и их адаптации к отечественной строительной сфере. Но только с их помощью возможно решать уникальные и не имеющие аналогов задачи проектирования зданий и сооружений и обеспечить конструкциям надлежащую надежность, безопасность и экономическую эффективность.

Литература

1. Болдырев Ю.Я. Суперкомпьютерные технологии - как современное воплощение междисциплинарного подхода в научно-образовательной деятельности// Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. № 4(103) 2010. СПб., Изд-во Политехнического ун-та. 2010. С. 99-105.
2. Болдырев Ю.Я, Лупуляк С.В, Петухов Е.П., Шиндер Ю.К. Моделирование на суперЭВМ обтекания затвора судопропускного сооружения С1 системы защиты Санкт - Петербурга от наводнений. Книга. Суперкомпьютерные технологии в науке и образовании (ред. В.А.Садовничий, Г.И.Савин, В.В.Воеводин). Изд-во МГУ, Москва, 2009. С. 89-95
3. Городецкий А.С., Евзеров И.Д. Компьютерные модели конструкций. – К., издательство «Факт», 2005. – 344 с.

4. Александров А.В. Лащеников Б.Я., Шапошников Н.Н; под редакцией Смирнова А.Ф. Строительная механика. Тонкостенные пространственные системы: учебник для вузов. М., Стройиздат, 1983-488с., ил.

5. Современное высотное строительство. Монография. М., ГУП "ИТЦ Москомархитектуры", 2007г. 400 с., ил.

6. Городецкий А.С. Вопросы расчета конструкций в упругопластической стадии с учетом применения ЭЦВМ.//ЭЦВМ в строительной механике. Труды первого всесоюзного совещания по применению ЭЦВМ в строительной механике (г. Ленинград, 1963г.). Л., М.., Издательство литературы по строительству, 1966.

7. Николаев С.В. Безопасность и надежность высотных зданий - это комплекс высокопрофессиональных решений//UST-Build - 2004. Инф. сб., 2004. С. 8-18.

8. СП 52-102-2003 «Проектирование и устройство свайных фундаментов»

9. Терцаги К., Пек. Р. Механика грунтов в инженерной практике. М., 1958.

10. Малышев М.В. Прочность грунтов и устойчивость оснований и сооружений. М.., Стройиздат, 1980

11. Schanz, T., Vermeer, P.A., Bonnier, P.G., (1999). The Hardering-Soil model: Formulation and verification. In: R.B.J. Brinkgreve, Deyond 2000 in Computational Geotechnics. Balkema, Rotterdam: 281-290

12. Болдырев Ю.Я., Климшин Д.В., Романов С.В., Шанина А.С.. Современные технологии математического моделирования для инженерного анализа и проектирования в строительстве. // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. № 4(103) 2010. СПб., Изд-во Политехнического ун-та. 2010. С.106 -111.

13. Белов В.В., Болдырев Ю.Я., Романов С.В., Шанина А.С.. Опыт применения математического моделирования грунтовых оснований зданий и сооружений // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. № 5(108) 2010. СПб., Изд-во Политехнического ун-та. 2010. С. 103 - 108.

14. Белов В.В., Болдырев Ю.Я., Климшин Д.В., Шанина А.С. Современные технологии математического моделирования при расчетах большепролетных металлических сооружений// Научно-технические ведомости. Информатика. Телекоммуникации. Управление. № 5(108) 2010. СПб., Изд-во Политехнического ун-та. 2010. С. 151 - 156.

Параллельные методы глобальной оптимизации в идентификации динамической балансовой нормативной модели экономики Нижегородской области*

В.П. Гергель[1], Н.Н. Оленев[2], В.В. Рябов[1], А.И. Фетинина[3]

Нижегородский государственный университет им Н.И. Лобачевского[1], Учреждение Российской академии наук Вычислительный центр им. А.А. Дородницына РАН[2], Вятский государственный университет[3]

Построенные в ВЦ РАН динамические модели экономики содержат большое число неизвестных параметров, идентифицировать которые можно с помощью верификации по историческим данным с нелинейным критериям близости расчётных и статистических данных с значительным числом локальных экстремумов. В работе идентифицируется математическая модель экономики Нижегородской области на основе параллельного индексного метода глобальной оптимизации, разработанного в ННГУ. Метод использует редукцию размерности на основе кривых Пеано и информационно-статистический подход, дополненный схемой построения множественных отображений с эффективным использованием сотен процессоров. Для ускорения поиска используются также разные модификации индексного метода.

1. Введение

В работе рассмотрена математическая модель региональной экономики, содержащая неизвестные параметры, которые можно найти косвенно, решая задачу минимизации отклонения между расчётными временными рядами макроэкономических показателей и соответствующими им статистическими историческими данными. Таким образом, возникает задача идентификации математической модели экономики, которая в общем случае является задачей глобальной оптимизации [1].

При построении и идентификации модели решаются задачи анализа и структуризации исходных данных в соответствии со структурой модели, так что по ходу построения модели решается задача выбора и системного анализа исходных статистических данных. Идентифицированную модель можно использовать для проверки тех или иных сценариев возможного развития экономики региона на основе численных экспериментов с моделью. Выводы и предсказания, полученные с помощью идентифицированной математической модели, в принципе могут быть экспериментально опровергнуты, а гипотезы, на которых основана модель, могут быть обоснованно оспорены. Это позволит в дальнейшем модифицировать модель и на постоянно совершенствующейся модели получать более обоснованные предсказания и делать более обоснованные аналитические выводы.

Задача идентификации представленной здесь многосекторной модели региональной экономики решается на данных Нижегородской области. Такого рода динамические балансовые нормативные модели экономики были идентифицированы для нескольких регионов России: Кировской области [2], Республики Алтай [3], а также для ряда развивающихся стран: Таджикистана [4], Монголии [5]. Нормативные модели содержат огромное число неизвестных параметров (нормативов распределения продуктов и нормативов распределения финансовых средств). Для их идентификации использовались эвристические методы сокращения размерности задачи, основанные на нахождении связи между параметрами на характерных режимах развития экономики с помощью разработанной в ВЦ РАН системы поддержки моделирования экономических систем ЭКОМОД [6], параллельные вычисления на

* Работа выполнена при поддержке совета по грантам Президента Российской Федерации (грант № НШ-64729.2010.9), при поддержке Российского фонда фундаментальных исследований (номера проектов 11-07-97017-р_поволжье_а, 11-01-92204-Монг_а, 11-01-00970-а).

кластерных суперкомпьютерах, позволяющие за разумное время сделать перебор возможных значений параметров при равномерном разбиении заданных интервалов их изменения. Эта методика здесь обобщается на использование эффективных методов глобальной оптимизации, скорость сходимости которых на несколько порядков лучше равномерного перебора.

Следует заметить, что дальнейшее развитие такого типа моделей экономики с большим числом неизвестных параметров невозможно без применения эффективных параллельных методов глобальной оптимизации и высокопроизводительных вычислительных систем [7-12], которое позволят повысить уровень сложности математических моделей региональной экономики, выраженный числом внешних параметров. Проведение сценарных расчётов на основе идентифицированных моделей региональной экономики даст возможность более точного прогнозирования экономических последствий тех или иных стратегических решений.

Актуальным является решение задач идентификации моделей экономики с использованием эффективных численных методов глобальной оптимизации на высокопроизводительных многопроцессорных вычислительных системах (суперкомпьютерах).

2. Анализ исходных статистических данных

Экономику Нижегородской области можно сагрегировать в три основные сектора, примерно равной мощности:

1) отрасли инфраструктуры, производства и распределения сырья (сельское хозяйство, электроэнергетика, строительство, транспорт, госуправление, образование, здравоохранение);

2) обрабатывающие отрасли (машиностроение и металлообработка, химическая и нефтехимическая промышленность, топливная промышленность, черная и цветная металлургия, промышленность строительных материалов, лесная, деревообрабатывающая и целлюлозно-бумажная, легкая, пищевая, химико-фармацевтическая, микробиологическая, мукомольно-крупяная и комбикормовая, полиграфическая промышленность, научный комплекс);

3) отрасли услуг (торговля, операции с недвижимым имуществом, финансовая деятельность, предоставление услуг).

При построении модели выделим следующих экономических агентов: Правительство области, Производителей в лице трех выделенных секторов, Банковскую систему, Домашние хозяйства области, Внешних потребителей и поставщиков.

Несмотря на отсутствие на территории Нижегородской области богатых природных ресурсов, ее экономика является одним из наиболее развитых в промышленном отношении регионов Российской Федерации. Основу промышленности Нижегородской области составляют обрабатывающие отрасли, мощный военно-промышленный комплекс, развит фундаментальный научный комплекс. Здесь имеются хорошие возможности для развития наукоемких производств и инновационных отраслей.

3. Описание модели региональной экономики

В качестве основы при построении математической модели экономики региона возьмем трехсекторный вариант модели общего равновесия с запасами продуктов, факторов производства и денег при налогообложении и наличии теневого оборота [1].

Напомним, что производители в модели экономики Нижегородской области представлены тремя секторами: (1) инфраструктурный комплекс (X) , (2) комплекс обрабатывающих отраслей (Y), (3) комплекс отраслей услуг, торговли, недвижимости, финансов (Z). Производители (сектора региональной экономики X, Y, Z), используют в производстве труд, капитал и промежуточную продукцию смежных секторов. Производители поставляют продукцию на внутренние рынки и внешний рынок. Домашние хозяйства L предлагают труд и потребляют конечную продукцию. Торговый посредник T занят перераспределением материальных и финансовых потоков. Банковская система B выдает кредиты производителям с целью извлечения банковской прибыли. Правительство региона G собирает налоги с производителей и домашних хозяйств и регулирует расходы бюджета. Считаем, что свои цены формируются на

каждом рынке каждой продукции и изменение цен обратно пропорционально изменению запасов соответствующих продуктов.

Для учета теневого оборота предполагаем, что произведенный продукт производители делят на открытый и теневой, последний не облагается налогами. В результате у производителя оказывается два вида денег – «белые» и «черные». «Черные» деньги могут отмываться, а их запас подвергается штрафным санкциям. У потребителя все деньги считаются «белыми», а свой доход потребитель делит по заданным нормам потребления легальных и теневых продуктов всех секторов. Производственные сектора $m = X, Y, Z$ платят налог на прибыль n_1, налог на добавленную стоимость n_2, акцизы на валовой выпуск n_3^m, единый социальный налог на фонд заработной платы n_4, таможенные платежи на экспорт n_5. Домашние хозяйства L в модели оплачивают таможенные платежи с импорта n_6, подоходный налог с зарплаты n_7.

Показатели и параметры модели снабжены верхними и нижними индексами, причем верхние индексы используются для агентов, а нижние для благ. Распределение запаса каждого блага производится по нормативу: a_i^{nm} – доля запаса блага i, идущая от агента n к агенту m. Распределение денег производится также по некоторому нормативу: b_i^{nm} – доля запаса денег агента m, идущая агенту n за продукт i. Коэффициенты фондоемкости также задаются нормативами: c_i^m – норма затрат продукта i на создание единицы фондообразующего продукта агента m. Параметры производственных функций секторов заданы константами.

3.1 Описание производственного сектора *X*

Приведем описание производственного сектора X (инфраструктурного комплекса отраслей Нижегородской области), описание других производственных секторов опускаем, оно аналогично. Выпуск $Y^X(t)$ продукта X экономическим агентом X (сектором X) описан степенной производственной функцией от используемых факторов производства (запасов Q): труда L, капитала K и промежуточных продуктов из секторов Y и Z.

$$Y_X = \left(a_L^X Q_L^X\right)^{\delta_L^X} \cdot \left(a_K^X Q_K^X\right)^{\delta_K^X} \cdot (a_Y^X Q_Y^X)^{\delta_Y^X} \cdot (a_Z^X Q_Z^X)^{\delta_Z^X}, \quad (1)$$

где $\delta_L^X + \delta_K^X + \delta_Y^X + \delta_Z^X = 1$; а остальные параметры принадлежат интервалу (0, 1).

Производство открытого X и теневого V продуктов агент X осуществляет на общих фондах $Q_K^X(t)$ и общих трудовых ресурсах $Q_L^X(t)$. Доля теневого продукта q_X в общем выпуске определяет прирост запасов открытого $Q_X^X(t)$ и теневого $Q_V^X(t)$ продуктов.

$$\frac{dQ_X^X}{dt} = (1-q_X)Y_X - (a_X^{XL} + a_X^{XY} + a_X^{XZ} + a_X^{XO})Q_X^X - c_X^X I_X, \quad (2)$$

где I_X - инвестиции продукта X в сектора региональной экономики (предполагаем, что все инвестиции осуществляются за счет открытых денежных средств), c_X^X - коэффициент приростной фондоемкости продукта X при инвестициях в сектор X. Продукт сектора X, поступающий на внешний рынок (в другие регионы России и за рубеж) $X_X^{XO} = a_X^{XO} Q_X^X$.

$$I_X = \frac{b_K^X W^X}{p_X^X c_X^X + p_Y^X c_Y^X + p_Z^X c_Z^X}. \quad (3)$$

Здесь b_K^X - доля расходов открытых денежных средств $W^X(t)$ на инвестиции в сектор *X*, c_i^X - коэффициент приростной фондоемкости продукта *i* при инвестициях в сектор *X*, p_i^X - индекс цен на продукт *i* для промежуточного потребления в секторе *X* , (*i* = *X*, *Y*, *Z*). Для однозначного определения цены на свою продукцию в секторе *X* полагаем $p_X^X = \min\left\{p_X^Y, p_X^Z\right\}$.

Запас теневого $Q_V^X(t)$ продукта сектора *X* прирастает за счет части q_X его выпуска и расходуется на потребление населения и смежных секторов по заданным нормам:

$$\frac{dQ_V^X}{dt} = q_X Y_X - \left(a_V^{XL} + a_V^{XY} + a_V^{XZ}\right)Q_V^X . \tag{4}$$

Изменения запасов открытых («белых») $W^X(t)$ и скрытых («черных») $B^X(t)$ денежных средств у агента *X* также описываются балансовыми соотношениями:

$$\begin{aligned}\frac{dW^X}{dt} = {} & wp_X^O X_X^{XO} + C^{BX} + \left(p_X^L a_X^{XL} + p_X^Y a_X^{XY} + p_X^Z a_X^{XZ}\right)Q_X^X - \\ & - \left(b_Y^{XY} + b_Z^{XZ} + b_W^{XY} + b_U^{XZ} + b_L^{XL} + b_H^{XB}\right)W^X - \\ & - T^{XG} + T^{GX} + b_B^X B^X ,\end{aligned} \tag{5}$$

$$\frac{dB^X}{dt} = \left(p_V^L a_V^{XL} + p_V^Y a_V^{XY} + p_V^Z a_V^{XZ}\right)Q_V^X - \left(b_B^{XL} + b_B^X + b_B^{XG}\right)B^X . \tag{6}$$

Здесь *w*(*t*) – рублевый курс доллара, $p_i^m(t)$ – индексы цен на продукт *i* на рынке для агента *m* (в случае внешнего рынка *O* в модели используется долларовый индекс цен), $T^{GX}(t)$ – трансферты из бюджета, $T^{XG}(t)$ – налоговые отчисления в консолидированный бюджет, $C^{BX}(t)$ – объем новых кредитов, $b_B^X B^X$ – поступления «отмытых» денег из теневого оборота.

Отчисления в консолидированный бюджет агента *X* (инфраструктурного комплекса Нижегородской области) $T^{XG}(t)$ складываются из отчислений по налогу на прибыль $T_1^{XG}(t)$, налогу на добавленную стоимость $T_2^{XG}(t)$, акцизам $T_3^{XG}(t)$, единому социальному налогу $T_4^{XG}(t)$, таможенным платежам на экспорт $T_5^{XG}(t)$.

$$T^{XG} = T_1^{XG} + T_2^{XG} + T_3^{XG} + T_4^{XG} + T_5^{XG} , \tag{7}$$

$$T_5^{XG} = n_5 wp_X^O X_X^{XO} , \tag{8}$$

$$T_4^{XG} = n_4 b_L^{XL} W^X , \tag{9}$$

$$T_3^{XG} = n_3^X \left[wp_X^O X_X^{XO} + \left(p_X^L a_X^{XL} + p_X^Y a_X^{XY} + p_X^Z a_X^{XZ}\right)Q_X^X \right], \tag{10}$$

$$\begin{aligned}T_2^{XG} = {} & n_2 \left\{ wp_X^O X_X^{XO} + \left(p_X^L a_X^{XL} + p_X^Y a_X^{XY} + p_X^{XZ}\right)Q_X^X - \right. \\ & \left. - \left(b_Y^{XY} + b_Z^{XZ} + b_H^{XB}\right)W^X - T_3^{XG} - T_4^{XG} - T_5^{XG} \right\},\end{aligned} \tag{11}$$

$$T_1^{XG} = n_1\{wp_X^O X_X^{XO} + \left(p_X^L a_X^{XL} + p_X^Y a_X^{XY}\right)Q_X^X - \\ -\left(b_Y^{XY} + b_Z^{XZ} + b_H^{XB} + b_L^{XL}\right)W^X - T_2^{XG} - T_3^{XG} - T_4^{XG} - T_5^{XG}\}, \quad (12)$$

Запас промежуточного продукта *Y* у агента *X* растет за счет покупки открытого продукта *Y* у агента *Y* (сектора обрабатывающих отраслей Нижегородской области) по цене $p_Y^X(t)$ и теневого продукта *W* у агента *Y* по цене $p_W^X(t)$ и убывает за счет использования его в производстве в качестве сырья и инвестиций. Запас промежуточного продукта *Z* у агента *X* растет за счет покупки открытого продукта у агента *Z* (сектора услуг Нижегородской области) по цене $p_Z^X(t)$ и теневого продукта *U* у агента *Z* по цене $p_U^X(t)$ и убывает за счет использования его в производстве в качестве сырья и инвестиций

$$\frac{dQ_Y^X}{dt} = \frac{b_Y^{XY}W^X}{p_Y^X} + \frac{b_W^{XY}W^X}{p_W^X} - a_Y^X Q_Y^X - c_Y^X I_X, \quad (13)$$

$$\frac{dQ_Z^X}{dt} = \frac{b_Z^{XZ}W^X}{p_Z^X} + \frac{b_U^{XZ}W^X}{p_U^X} - a_Z^X Q_Z^X - c_Z^X I_X, \quad (14)$$

Запас труда в секторе *X* прирастает за счет покупки у агента *L* (домашних хозяйств Нижегородской области) открытого труда L по официальной ставке заработной платы $s_L^X(t)$ и теневого труда B по теневой ставке $s_B^X(t)$ и убывает в силу спроса на труд агента *X*.

$$\frac{dQ_L^X}{dt} = \frac{b_L^{XL}W^X}{s_L^X} + \frac{b_B^{XL}B^X}{s_B^X} - a_L^X Q_L^X. \quad (15)$$

Запас капитала в секторе *X* прирастает за счет инвестиций $b_K^X W^X(t)$, убывает при амортизации капитала агента *X* с темпом μ_K^X и при использования капитала в производстве сектора *X*.

$$\frac{dQ_K^X}{dt} = b_K^X W^X - \mu_K^X Q_K^X - a_K^X Q_K^X. \quad (15)$$

Считаем, что агент *X* (предприятия сектора *X*) берет весь предлагаемый агентом *B* (банковской системой Нижегородской области) кредит, однако объем предоставляемого кредита $C^{BX}(t)$ ограничен ликвидационной стоимостью производственных фондов.

$$C^{BX} = \sigma^X Q_K^X, \quad \sigma^X > 0. \quad (16)$$

Задолженность $Z^X(t)$ агента *X* банковской системе *B* прирастает за счет выдачи новых кредитов $C^{BX}(t)$ и начисления текущего процента по кредитам $r(t)$ на имеющуюся задолженность, а уменьшается в силу платежей погашения $H^{XB}(t)$.

$$\frac{dZ^X}{dt} = C^{BX} + rZ^X - H^{XB}, \quad H^{XB} = b_H^{XB}W^X. \quad (17)$$

3.2 Описание домашних хозяйств Нижегородской области (агента *L*)

Безработные в составе экономически активного населения Нижегородской области в модели разделены по секторам производства *X*, *Y*, *Z*, а в них подразделены на безработных в открытой части экономики и безработных в теневой части экономики. Безработица увеличивается, если предложения труда превышает спрос на него и уменьшается в противном случае. Для краткости приведем описание только домашних хозяйств, приписанных в модели к сектору *X*. Предложение труда в открытой и теневой части сектора *X*

$$\frac{dQ_L^{LX}}{dt} = a_L^{LX} Q_L^{LX} - \frac{b_L^{XL} W^X}{s_L^X}, \quad \frac{dQ_B^{LX}}{dt} = a_B^{LX} Q_B^{LX} - \frac{b_B^{XL} B^X}{s_B^X}. \tag{18}$$

Считаем, что рост открытой $s_L^X(t)$ и теневой $s_B^X(t)$ ставок заработной платы в секторе *X* может происходить как при нехватке кадров, так и при росте потребительских цен на продукцию сектора. Открытая ставка зарплаты

$$\frac{ds_L^X}{dt} = \left[\alpha_L^X \left(\frac{b_L^{XL} W^X}{s_L^X} - a_L^{LX} Q_L^{LX} \right) + \frac{\beta_L^X s_L^X}{p_X^L} \left(\frac{b_X^{LX} W^L}{p_X^L} - a_X^{XL} Q_X^X \right) \right]_+, \tag{19}$$

где $\beta_L^X = \delta \alpha_X^L$ и используется обозначение: $X_+ = X$, если $X > 0$ и $X_+ = 0$, если $X \le 0$. Считаем, что доля прироста цен, отражающаяся на росте заработной платы, $\delta \in (0,1)$. Теневая ставка зарплаты в секторе *X*

$$\frac{ds_B^X}{dt} = \left[\alpha_B^X \left(\frac{b_B^{XL} B^X}{s_B^X} - a_B^{LX} Q_B^{LX} \right) + \frac{\beta_B^X s_B^X}{p_V^L} \left(\frac{b_V^{LX} W^L}{p_V^L} - a_V^{XL} Q_V^X \right) \right]_+, \tag{20}$$

где $\beta_B^X = \delta \alpha_V^L$. Считаем, что все деньги у домашних хозяйств являются открытыми (чистыми) независимо от источника поступления.

$$\begin{aligned} \frac{dW^L}{dt} &= d^{BL} + b_L^{XL} W^X + b_L^{YL} W^Y + b_L^{ZL} W^Z + b_B^{XL} B^X + b_B^{YL} B^Y + b_B^{ZL} B^Z - \\ &- \left(b_X^{LX} + b_V^{LX} + b_Y^{LY} + b_W^{LY} + b_Z^{LZ} + b_U^{LZ} + b_M^{LO} \right) W^L - T^{LG} + T^{GL}. \end{aligned} \tag{21}$$

Здесь $T^{LG}(t)$ - отчисления в консолидированный бюджет агента *L* (домашних хозяйств) - складываются из отчислений по таможенным платежам на импорт $T_6^{LG}(t)$ и отчислений по подоходному налогу $T_7^{LG}(t)$ с открытой части дохода.

$$T^{LG} = T_6^{LG} + T_7^{LG}, T_6^{LG} = n_6 b_M^{LO} W^L, \quad T_7^{LG} = n_7 \left(d^{BL} + b_L^{XL} W^X + b_L^{YL} W^Y + b_L^{ZL} W^Z \right). \tag{22}$$

3.3 Описание консолидированного бюджета Правительства Нижегородской области (агента *G*)

Запас денег на счетах консолидированного бюджета прирастает от налоговых поступлений и убывает при трансфертах секторам экономики и домашним хозяйствам. Может принимать отрицательные значения, если область является дотационной. Область является дотационной,

когда трансферты из федерального бюджета превышают налоговые отчисления в него. В модели все налоги остаются на счетах консолидированного бюджета.

$$\frac{dW^G}{dt} = b_V^{XG} B^X + b_W^{YG} B^Y + b_U^{ZG} B^Z + T^{XG} + T^{YG} + T^{ZG} + T^{LG} - T^{GX} - T^{GY} - T^{GZ} - T^{GL}, \quad (23)$$

где $T^{GX} = b_X^{GX} W^G$, $T^{GY} = b_Y^{GY} W^G$, $T^{GZ} = b_Z^{GZ} W^G$, $T^{GL} = b_L^{GL} W^G$.

Текущий дефицит (а если < 0 – профицит) консолидированного бюджета Нижегородской области $D^G(t)$ определяется правой частью (24) с обратным знаком.

$$D^G = -b_V^{XG} B^X - b_W^{YG} B^Y - b_U^{ZG} B^Z - T^{XG} - T^{YG} - T^{ZG} - T^{LG} + T^{GX} + T^{GY} + T^{GZ} + T^{GL}. \quad (24)$$

3.4 Описание банковской системы Нижегородской области (агента *B*)

Банковская система Нижегородской области не является замкнутой, большую роль в инвестиционных решениях играют филиалы Российских банков других регионов. Считаем, что часть золотовалютных резервов России обеспечивает резервирование активов области. Банковские активы областной банковской системы состоят из золото-валютных резервов $R(t)$ и суммарной задолженности секторов $Z(t)$, а пассивы – из суммарных запасов денег у контрагентов банковской системы $W(t)$, которые подчиняются финансовому балансу:

$$wR(t) + Z(t) = W(t), \quad (25)$$

где $w(t)$ – рублевый курс доллара, рублей за один доллар. Срочные депозиты, на которые начисляют процент, не учитываем. Золото-валютные резервы $R(t)$ определяются балансом

$$\frac{dR}{dt} = \sum_{i=X}^{Z} p_i^O X_i^{iO} - M_M^{OL}, \qquad M_M^{OL} = \frac{(1-n_6) b_M^{LO} W^L}{w}. \quad (26)$$

Здесь p_i^O – цена на экспортный товар X_i^{iO} сектора i на внешнем рынке O, в долларах за единицу. Рассматриваем весь импорт как отдельный товар, доля которого b_M^{LO} в потреблении населения фиксирована. Мы не учитываем внешние заимствования и перетоки капитала.

Суммарная задолженность агентов $Z(t)$ растет за счет выдачи новых кредитов $C^{Bi}(t)$, начисления процентов на остаток задолженност, снижается за счет погашения задолженности

$$\frac{dZ}{dt} = \sum_{i=X}^{Z} C^{Bi}(t) + r(t)Z(t) - \sum_{i=X}^{Z} H^{iB}(t), \quad (\text{или } Z = Z^X + Z^Y + Z^Z). \quad (27)$$

Банковская прибыль $d^{BL}(t) = r(t)Z(t)$ поступает в доходы населения. Суммарные запасы денег

$$W(t) = W^X + W^Y + W^Z + W^T + W^L + W^G. \quad (28)$$

Резервы банковской системы $wR(t)$ обеспечивают вклады контрагентов при законодательно установленной норме резервирования ξ.

$$wR(t) \geq \xi W(t). \quad (29)$$

Тогда банковская система стремится предоставить максимальный кредит, который допускают соотношения (25) и (29):

$$Z = \frac{1-\xi}{\xi} wR\,. \tag{30}$$

Считаем, что спрос на кредиты, обеспеченные ликвидационной стоимостью производственных фондов, полностью удовлетворяется банковской системой.
В результате, соотношения (26), (27) и (30) определяют процент по кредитам $r(t)$:

$$r(t) = \left(\frac{1-\xi}{\xi} w \left(\sum_{i=X}^{Z} p_i^O X_i^{iO} - M_M^{OL} \right) + \sum_{i=X}^{Z} H^{iB} - \sum_{i=X}^{Z} C^{Bi} \right) \Big/ Z(t)\,. \tag{31}$$

3.5 Описание посредника (агента *T*)

Агент T в модели играет роль чистого посредника, не получающего доходов и потому не имеющего ответственности по налогам. Описание этого агента введено для описания областных рынков продукции, на которых определяются внутренние цены на все продукты.

Изменение запаса $Q_X^L(t)$ конечного продукта *X* сектора *X*, предназначенного агенту *L* (домашним хозяйствам области), у посредника *T* определяет изменение индекса потребительских цен $p_X^L(t)$ на продукцию сектора *X*.

$$\frac{dQ_X^L}{dt} = a_X^{XL} Q_X^X - \frac{b_X^{LX} W^L}{p_X^L}\,, \tag{32}$$

$$\frac{dp_X^L}{dt} = \alpha_X^L \left(\frac{b_X^{LX} W^L}{p_X^L} - a_X^{XL} Q_X^X \right). \tag{33}$$

Общее число уравнений в модели почти 100, общее число параметров превышает 80.

4. Задача идентификации

В общем виде задача идентификации математической модели состоит в том, что её неизвестные параметры могут быть найдены из оптимальных соотношений, отражающих степень близости расчётных и экспериментальных (статистических) данных. Отсюда возникает задача поиска глобального оптимума многомерной нелинейной функции. Дальнейшее развитие математических моделей региональной экономики невозможно без применения эффективных параллельных методов глобальной оптимизации и высокопроизводительных вычислительных систем, которое позволит повысить уровень сложности математических моделей региональной экономики, выраженный числом входных параметров. Проведение сценарных расчётов на основе построенных моделей региональной экономики даст возможность более точного прогнозирования экономических последствий тех или иных стратегических решений. Большое количество неопределяемых напрямую из статистики параметров модели определяем при верификации, сравнивая выходные временные ряды переменных модели с доступными статистическими временными рядами 2000 – 2008 гг. Временные ряды считаются похожими, если они близки как функции времени. В качестве критериев близости расчетного и статистического временных рядов используем индекс несовпадения Тэйла, который измеряет несовпадение временных рядов *X* и *Y* и чем ближе он к нулю, тем ближе сравниваемые ряды.

Декомпозиция модели на отдельные блоки дает возможность за разумное время определить независимые параметры благодаря параллельным вычислениям для перебора параметров модели на заданных интервалах их изменения с последовательно уменьшающимся интервалом изменения параметров.

5 Постановка задачи многоэкстремальной оптимизации

Алгоритмы, развиваемые Нижегородской научной школой многоэкстремальной оптимизации, предполагают следующую постановку задачи[1]:

$$\varphi = \varphi(y^*) = \min\{\varphi(y):\ y \in D\},$$
$$D = \{y \in R^N:\ a_i \le y_i \le b_i,\ 1 \le i \le N\}, \tag{35}$$

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 D

$$D = \{y \in R^N:\ -2^{-1} \le y_i \le 2^{-1},\ 1 \le i \le N\} = \{y(x):\ 0 \le x \le 1\},$$

исходную задачу можно редуцировать к следующей одномерной задаче:

$$\varphi(y(x^*)) = \min\{\varphi(y(x)):\ x \in [0,1],\ g_j(y(x)) \le 0,\ 1 \le j \le m\}.$$

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.

$$|\varphi(y(x')) - \varphi(y(x''))| \le K|x' - x''|^{1/N},\ x', x'' \in [0,1],$$

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 [1], [8].

6. Индексный метод глобальной оптимизации

6.1 Редукция размерности и множественные отображения

Редукция многомерных задач к одномерным с помощью разверток имеет такие важные свойства, как непрерывность и сохранение равномерной ограниченности разностей функций при ограниченности вариации аргумента. Однако при этом происходит потеря части информации о близости точек в многомерном пространстве, так как точка $x \in [0,1]$ имеет лишь левых и правых соседей, а соответствующая ей точка $y(x) \in R^N$ имеет соседей по 2^N направлениям. А при использовании отображений типа кривой Пеано близким в N-мерном пространстве образам y', y'' могут соответствовать достаточно далекие прообразы x', x'' на отрезке [0,1]. Как результат, единственной точке глобального минимума в многомерной задаче соответствует несколько (не более 2^N) локальных экстремумов в одномерной задаче, что, естественно, ухудшает свойства одномерной задачи.

Сохранить часть информации о близости точек позволяет использование множества отображений

[1] Здесь применяются общепринятые обозначения для задач многоэкстремальной оптимизации. Следует помнить, что в приложении к задачам идентификации параметров модели региональной экономики роль вектора x играет вектор искомых параметров рассматриваемой модели экономики.

$$Y_L(x)=\{y^1(x),\ldots,\ y^L(x)\} \tag{36}$$

вместо применения единственной кривой Пеано $y(x)$ (см. [7], [10]). Каждая кривая Пеано $y^i(x)$ из $Y_L(x)$ может быть получена в результате поворота развертки вокруг начала координат. При этом найдется отображение $y^i(x)$, которое точкам многомерного пространства y', y'', которым при исходном отображении соответствовали достаточно далекие прообразы на отрезке [0,1], будет сопоставлять более близкие прообразы x', x''.

Максимальное число различных поворотов развертки, отображающей *N*-мерный гиперкуб на одномерный отрезок, составляет 2^N. Использование всех из них является избыточным. В используемой схеме (см. [10]) преобразование развертки осуществляется в виде поворота на угол $\pm\pi/2$ в каждой из координатных плоскостей. Число подобных пар поворотов определяется числом координатных плоскостей пространства, которое равно $C_N^2=\frac{N(N-1)}{2}$, а общее число преобразований будет равно $N(N-1)$. Учитывая исходное отображение, приходим к заключению, что данный способ позволяет строить до $N(N-1)+1$ развертки для отображения *N*-мерной области на соответствующие одномерные отрезки.

6.2 Параллельный индексный метод и локально-глобальная стратегия

Использование множества отображений $Y_L(x)=\{y^1(x),\ldots,\ y^L(x)\}$ приводит к формированию соответствующего множества одномерных многоэкстремальных задач

$$\min\{\varphi(y^l(x)):x\in[0,1],\ 1\le l\le L. \tag{37}$$

Каждая задача из данного набора может решаться независимо, при этом любое вычисленное значение $z=\varphi(y')$, $y'=y^i(x')$ функции $\varphi(y)$ в i-й задаче может интерпретироваться как вычисление значения $z=\varphi(y')$, $y'=y^s(x'')$ для любой другой s-й задачи без повторных трудоемких вычислений функции $\varphi(y)$. Подобное информационное единство позволяет решать исходную задачу (4) путем параллельного решения индексным методом L задач вида (6) на наборе отрезков [0,1]. Каждая одномерная задача решается на отдельном процессоре. Для организации взаимодействия на каждом процессоре создается L очередей, в которые процессоры помещают информацию о выполненных итерациях.

Подробное описание решающих правил параллельного индексного алгоритма глобальной оптимизации приведено в работе [9].

Локально-адаптивный алгоритм является модификацией индексного метода глобального поиска, состоящей в том, что, начиная с некоторого шага, при выборе точек итераций используется дополнительная информация – текущие оценки плотности вероятности для расположения точки искомого оптимума. Оценки плотности определяются по значениям функционалов задачи, вычисленных в точках выполненных итераций. Таким образом, плотность переоценивается после каждой итерации, причем максимумы плотности соответствуют окрестностям точек текущих оптимальных значений. Подробно решающие правила локально-адаптивного метода приведены, например, в [9]. Существенным параметром этого метода является целое число $0 \le \alpha \le 30$, влияющее на характер сходимости. При $\alpha=0$ поиск носит глобальный характер, при $\alpha=30$ – локальный.

Смешанный алгоритм является модификацией индексного метода глобального поиска, состоящей в том, что, начиная с некоторого шага итерации, определяемые правилами индексного метода, чередуются с итерациями, определяемыми правилами локально-адаптивного алгоритма. Частота чередования является параметром метода.

7. Результаты вычислительных экспериментов

Численные эксперименты с моделью проводились, чтобы найти работоспособный вариант, качественно верно отражающий процессы, происходящие в экономике Нижегородской области. Численные эксперименты показали работоспособность полной модели и отдельных ее частей. Это значит, что модель может использоваться в дальнейшей работе. Внешние параметры этого варианта можно взять за основу для более точной идентификации модели в будущем, а сам вариант использовать как базовый при проведении качественных сценарных расчетов.

Критерием качества идентификации параметров модели является количественное соответствие основных макроэкономических показателей статистическим показателям экономики региона за период с начала 2000 года и до конца 2008 г. При этом все реальные показатели выражены в постоянных ценах 2000 г. Решение системы получено с привлечением современных вычислительных и программных средств.

Изменения в сценарном расчете по сравнению с базовым сценарием будем представлять вариацией изменения макропоказателей, выраженной в процентах.

В сценарии 1 предполагается, что с 2007 года происходит увеличение трансфертов консолидированного бюджета Нижегородской области в сектор обрабатывающих отраслей на поддержку инноваций. Но при этом структура расходов сектора остается неизменной. А именно, пусть бюджетные трансферты в сектор биотехнологий возрастут с 3% до 20% консолидированного бюджета. Реальные трансферты в сектор *Y* в сценарии 1 увеличиваются немногим более чем в пять раз в сравнении с базовым сценарием, инвестиции сектора *Y* возрастают на 50%, выпуск сектора через 40 лет увеличивается на треть в сравнении с базовым вариантом. Прирост выпуска приводит к росту объемов продаж и по всем каналам. При этом запас «белых» денег прирастает в полтора раза, а «черных» увеличивается на четверть. Однако, при этом ставки заработной платы снижаются до 5%, а прирост «черных» денег в секторах *X* и *Z* увеличивается до 10%. Изменение структуры цен, сопровождающееся уменьшением уровня потребительских цен и ставок заработной платы, приводит падению номинальных доходов консолидированного бюджета от налогообложения секторов *Y* и *Z*, при малом росте поступлений от сектора *X*.

В оптимистическом сценарии 2 в результате мер по поддержке инновационных процессов по трансферу технологий в 2012 году происходит увеличение отдачи от всех факторов производства на 5%. Тогда выпуск комплекса инфраструктурных отраслей *X* удваивается через 5 лет, а секторов обрабатывающих отраслей *Y* и сектор услуг *Z* через 13 лет. При этом, несмотря на рост производительности труда, немного возрастает занятость в секторах экономики. Значительно увеличивается объем производственных фондов. Индексы цен меняются разнонаправлено: в секторе *X* индексы цен на продукцию, реализуемую по легальным каналам падают, а по теневым каналам растут; в секторах *Y* и *Z* все индексы цен растут. Все ставки заработной платы растут, за исключением легальной ставки сектора *X*. Запасы денег у всех экономических агентов возрастают, инвестиции секторов возрастают, доходы бюджета и домашних хозяйств возрастают.

8. Заключение

Улучшен предложенный ранее подход к распараллеливанию поисковых методов на многопроцессорных кластерных системах, не требующий синхронизации работы процессоров и характеризуемый высокими показателями масштабируемости и надежности, впервые позволяющий эффективно использовать сотни процессоров для методов глобальной оптимизации, имеющих серьёзные неявные зависимости по данным. Подход основан на новом оригинальном способе построения множества отображений типа кривых Пеано.

Оказалось, что поведение макропоказателей региональной экономики существенно зависит от политики, проводимой Правительством региона. При осуществлении задуманной политики повышения производительности факторов производства за счет инновационной деятельности выпуски продукции и доходы всех агентов растут, однако, наличие теневой составляющей в производстве сохраняет высокие темпы инфляции.

Построена динамическая балансовая нормативная модель региональной экономики Нижегородской области. Её идентификация с помощью эффективных параллельных методов

глобальной оптимизации на вычислительном кластере создают условия к проведению сценарных расчётов по прогнозированию экономического развития области.

С целью дальнейшего усложнения моделей будут разработаны новые способы ускорения сходимости параллельных методов глобальной оптимизации:

- внедрение методики локального уточнения рекордов в параллельный индексный метод с сохранением балансировки вычислительной нагрузки;
- построение новой адаптивной схемы редукции размерности на основе развёрток типа кривых Пеано с растущим уровнем детализации.

В результате на стыке двух научных направлений получена новая методика идентификации моделей региональной экономики с применением эффективных алгоритмов глобальной оптимизации.

Литература

1. Стронгин Р.Г. Поиск глобального оптимума. М.: Знание, 1990. 47 с. - (Новое в жизни, науке, технике. Сер. Математика, кибернетика; Вып.1990; 2)
2. Оленев Н.Н. Модель оценки инновационного потенциала региональной экономики // Экономика депрессивных регионов: Проблемы и перспективы развития региональных экономик: Труды международной научно-практической конференции / Под ред. Беляева В.И., Дубины И.Н., Мамченко О.П. Барнаул: Изд-во Алтайского гос. ун-та, 2007. С.178-188.
3. Оленев Н.Н., Стародубцева В.С. Исследование влияния теневого оборота на социально-экономическое положение в Республике Алтай // Региональная экономика: теория и практика. N 11(68) - 2008 апрель. С.32-37.
4. Оленев Н.Н., Солиев Х.Ю. Имитационная модель развивающейся экономики на примере республики Таджикистан //"Математика. Компьютер. Образование". Сб. трудов XVII международной конференции. Под общей редакцией Г.Ю. Ризниченко. Ижевск: Научно-издательский центр "Регулярная и хаотическая динамика", 2010. Том 2. - С.173-181.
5. Горбачев В.А., Оленев Н.Н. Идентификация модели добывающего сектора экономики Монголии //VI Московская международная конференция по исследованию операций (ORM2010): Москва, 19-23 октября 2010 г.: Труды/Отв.ред. П.С.Краснощеков, А.А.Васин. - М.: МАКС Пресс. 2010. С.97-98.
6. Завриев Н.К., Поспелов И.Г., Поспелова Л.Я. Исследование математических моделей экономики средствами системы ЭКОМОД // Матем. моделирование, 15:8. 2003. С.57–74.
7. Стронгин Р.Г. Параллельная многоэкстремальная оптимизация с использованием множества разверток // Ж. вычисл. матем. и матем. физ. Т.31. №8. 1991. С. 1173–1185.
8. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht, 2000.
9. Баркалов К.А. Ускорение сходимости в задачах условной глобальной оптимизации. Нижний Новгород: изд-во Нижегородского гос. ун-та, 2005.
10. Баркалов К.А., Рябов В.В., Сидоров С.В. Использование кривых Пеано в параллельной глобальной оптимизации // Материалы Девятой международной конференции-семинара "Высокопроизводительные параллельные вычисления на кластерных системах", Владимир, 2009. С. 44–47.
11. Стронгин Р.Г., Гергель В.П., Баркалов К.А. Параллельные методы решения задач глобальной оптимизации // Известия высших учебных заведений. Приборостроение. – Т. 52. №10. 2009. С. 25–32.
12. Баркалов К.А., Рябов В.В., Сидоров С.В. О некоторых способах балансировки локального и глобального поиска в параллельных алгоритмах глобальной оптимизации // Ж. Вычислительные методы и программирование. Т.11. 2010. С. 382–387.

Применение GPU в рамках гибридного двухуровневого распараллеливания MPI+OpenMP на гетерогенных вычислительных системах*

А.В. Горобец[1], С.А. Суков[1], А.О. Железняков[2], П.Б. Богданов[2], Б.Н. Четверушкин[1]

ИПМ РАН им М.В. Келдыша[1], НИИСИ РАН[2]

В работе рассматривается применение расширенного распараллеливания для расчетов задач газовой динамики и аэроакустики на гетерогенных кластерах с узлами, сочетающими вычислительные элементы принципиально разной архитектуры, CPU и GPGPU. Двухуровневая модель распараллеливания MPI+OpenMP дополняется применением OpenCL для загрузки GPGPU, таким образом, реализуется третий уровень параллелизма. Представлена параллельная модель алгоритма для неструктурированных сеток.

1. Введение

В настоящее время наблюдается определенная тенденция в развитии вычислительных систем. С одной стороны, как и прежде, продолжается рост производительности за счет увеличения числа процессорных ядер. С другой стороны, становятся всё более популярными гетерогенные системы, высокая производительность которых обусловлена применением вычислительных элементов принципиально иной архитектуры. Первое направление развития требует от алгоритма все большей степени параллелизма и мотивирует переход от MPI модели на двухуровневую параллельную модель MPI+OpenMP, которая лучше соответствует современной архитектуре суперкомпьютеров с многоядерными узлами. Второе направление требует адаптации алгоритмов к гетерогенной архитектуре и еще более сложной параллельной модели, сочетающей принципиально различные типы параллелизма.

В отличие от "обычного" кластера, на котором вычисления выполняются на однотипных процессорных ядрах, на гетерогенном кластере вычисления также выполняются на GPU, или более точно GPGPU (General-purpose graphics processing units — графические процессоры общего назначения), имеющих существенно отличающуюся архитектуру. GPU устройства имеют свою собственную оперативную память, доступ к которой осуществляется посредством шины PCI-Express. Программа для такой архитектуры состоит из кода для CPU, другими словами host-кода, который написан на обычном языке программирования C, C++, и кода для GPU – kernel кода, написанного на специальном языке, как правило, производном от C. Для выполнения вычислений на GPU необходимо передать входные данные в память GPU, выполнить набор kernel-подпрограмм (называемых ниже вычислительными ядрами) и получить результаты обратно в память CPU.

Для расчетов на гетерогенных вычислительных системах предлагается использовать многоуровневую параллельную модель. На первом уровне, как и на “обычном” кластере, используется MPI для объединения узлов в рамках модели с распределенной памятью на основе традиционного геометрического параллелизма. На втором уровне применяется OpenMP в рамках того же MIMD (multiple instruction multiple data - множественные потоки команд и данных) параллелизма, но в рамках модели с общей памятью для распараллеливания по CPU ядрам внутри узла. По сравнению с MPI, использование OpenMP имеет некоторые особенности, связанные с одновременным доступом к общей памяти, специфическими узкими местами многоядерных компьютеров и так далее. Более подробно типичные проблемы , возникающие в случае применения OpenMP, рассматриваются, например, в [1]. Гибридная двухуровневая модель распараллеливания MPI+OpenMP является достаточно хорошо известным подходом в вычислительной

* Работа выполнена при поддержке совета по грантам президента Российской Федерации, грант МК-7559.2010.1. Для расчетов использовался суперкомпьютер Ломоносов НИВЦ МГУ, суперкомпьютер К100 ИПМ РАН и тестовый стенд GPGPU НИИСИ РАН.

газовой динамике. Алгоритмы на основе MPI+OpenMP и сравнение с “плоским” MPI подходом приводятся, например, в [2–4]. Сравнительный анализ показывает, что гибридный подход повышает производительность вычислений, но, как правило, не намного. В [5], анализ производительности для MPI и MPI+OpenMP подходов к распараллеливанию приводит к вполне закономерному выводу, что второе имеет преимущество над первым только при расчетах на большом числе процессоров. Однако, основной мотивацией использования более сложной параллельной модели MPI+OpenMP является не столько выигрыш в производительности, сколько возможность задействовать существенно большее число процессоров.

Далее параллельная модель должна быть дополнена другим типом параллелизма для того, чтобы задействовать GPU, в которых, помимо MIMD параллелизма, также используется SIMD (single instruction multiple data – один поток команд на множество данных) параллелизм на уровне потоковых процессоров. Таким образом, появляется третий уровень, на котором из одной или нескольких OpenMP нитей CPU кода вызываются вычислительные ядра, выполняющиеся на одном или нескольких графических процессорах. Перенос вычислений на архитектуру GPU сам по себе представляет достаточно сложную задачу, требующую учета множества факторов. Примеры адаптации алгоритмов под графические процессоры представлены, например, в работах [6, 7].

В данной работе на упрощенном примере рассматривается проблема переноса газодинамического алгоритма для неструктурированных сеток на GPU. Далее в разделе 2 приводится краткое описание математической модели и двухуровневого распараллеливания MPI+OpenMP, в разделе 3 рассматривается адаптация упрощенного алгоритма к GPU архитектуре и в разделе 4 приводятся результаты измерения производительности вычислений на графических процессорах. Раздел 5 посвящен общему подходу к гетерогенным вычислениям.

2. Математические модели и параллельная реализация алгоритмов на CPU

Рассматривается моделирование сжимаемых течений с использованием неструктурированных сеток. Класс моделей аэроакустики [8], построенных на основе уравнений Эйлера, включает в себя три основные модели: линейная (линеаризованные уравнения Эйлера), нелинейная для пульсационных компонент течения, описываемая различными формами NLDE (NonLinear Disturbance Equations) уравнений, нелинейная для всего течения, описываемая полными уравнениями Эйлера. Действие молекулярной вязкости и теплопроводности реализовано в рамках моделей на основе уравнений Навье–Стокса и их линейных аналогов.

Для пространственной дискретизации в рамках конечно-объемного подхода используются неструктурированные тетраэдральные и гибридные сетки (которые также могут содержать шестигранники, четырехугольные пирамиды и треугольные призмы).

Конвективный поток через грани контрольных объемов вычисляется с использованием схемы Роу. Аппроксимация потоков имеет повышенный порядок точности (до шестого включительно) и реализуется на расширенном шаблоне, включающем два противопотоковых тетраэдра и все соседние узлы их вершин [9]. Для интегрирования по времени используются явные схемы Рунге-Кутты до 4-го порядка и неявные схемы до 2-го порядка на основе линеаризации по Ньютону.

Для параллельной реализации численных алгоритмов используется гибридный двухуровневый подход с MPI на первом уровне и OpenMP на втором. Применение OpenMP позволяет повысить параллельную эффективность на вычислительных системах с многоядерными узлами при использовании большого числа процессоров, поскольку в T раз сокращается число MPI процессов, где T – число OpenMP нитей. Это дает следующие преимущества:

1) MPI-процессу доступно примерно в T раз больше памяти.
2) Сокращается количество обменов данными, поскольку в T раз уменьшается количество участвующих в обмене данными MPI процессов
3) Примерно во столько же раз уменьшается общий объем пересылки, поскольку уменьшается протяженность границ между подобластями.

4) Не происходит простаивание процессов в ожидании своей очереди на доступ к разделяемым коммуникационным ресурсам узла.

Однако OpenMP также имеет некоторые особенности. Одна из основных проблем, это пересечение по данным между параллельными нитями. Присутствие критических секций и атомарных операций, которые ограничивают одновременный доступ нитей к данным, существенно снижает производительность. Избежать пересечения можно простым способом, реплицируя данные для каждой нити, однако это ведет к росту затрат оперативной памяти. Более универсальный и эффективный способ, который был реализован, это двухуровневое разбиение расчетной области, когда подобласть MPI-процесса разбивается далее на подобласти OpenMP нитей. Элементы сетки переупорядочиваются таким образом, чтобы внутренние элементы подобластей были сгруппированы в памяти и отделены от интерфейсных элементов (т. е. элементов сетки, принадлежащих более чем одной OpenMP подобласти). Это позволяет локализовать пересечение по данным. Нити OpenMP параллельно обрабатывают данные, соответствующие внутренним элементам, а затем последовательно (или параллельно, но с наложением вычислений) обрабатываются интерфейсные элементы. При этом, поскольку количество интерфейсных элементов, как правило, намного меньше, чем внутренних, достигается высокая параллельная эффективность.

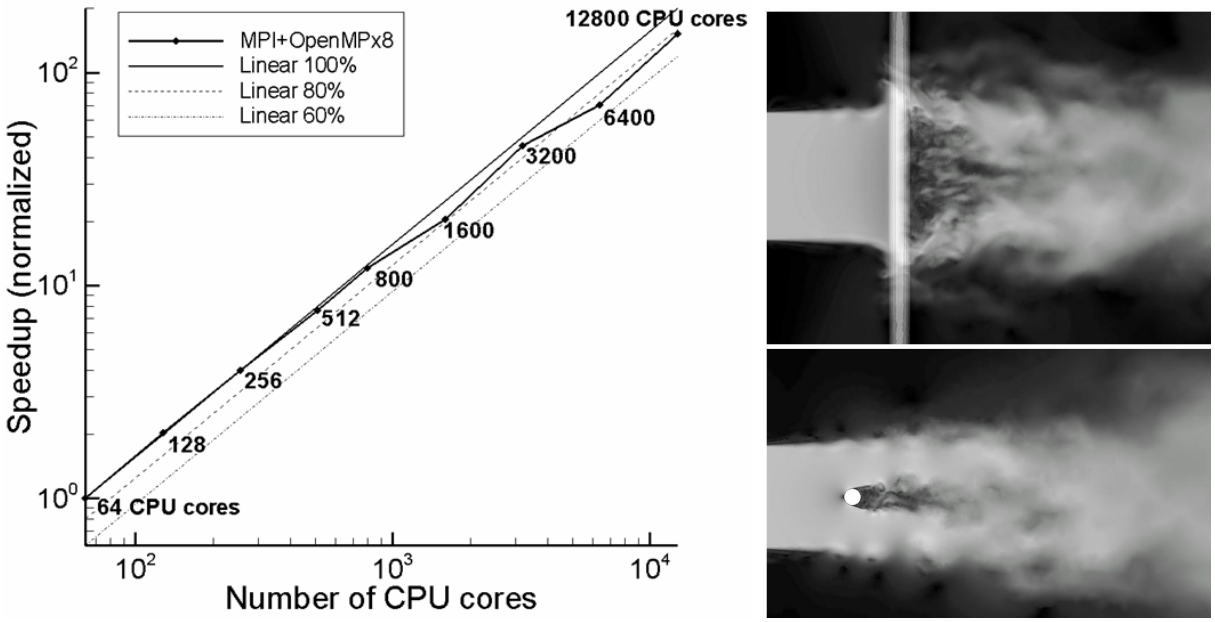


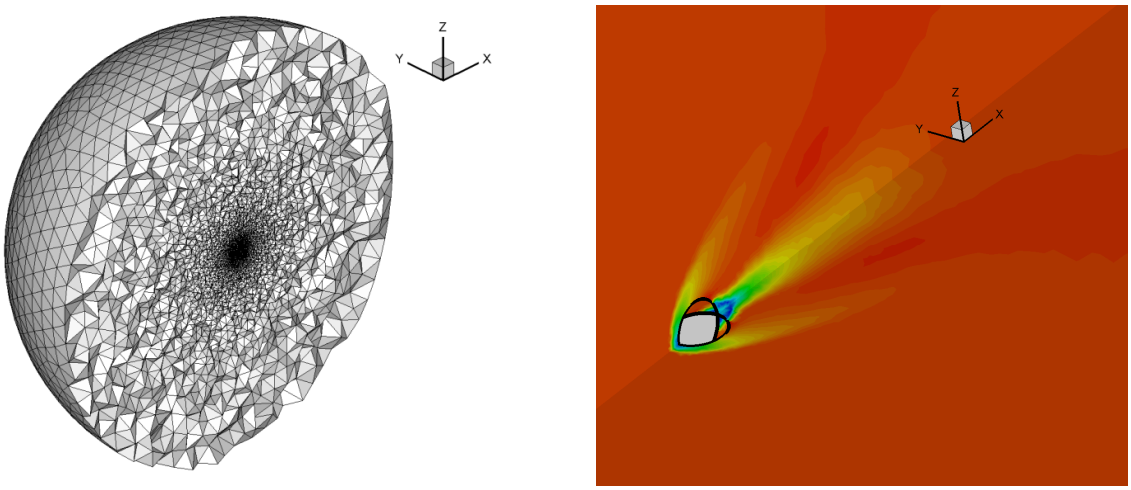
Рис. 1 Ускорение на суперкомпьютере Ломоносов, нормированное по времени на 64 ядрах (слева), полученное на расчете обтекания цилиндра дозвуковой струей (справа).

Тест на ускорение с двухуровневым распараллеливанием MPI+OpenMP для схемы повышенного порядка с центрами в узлах был выполнен на неструктурированной тетраэдральной сетке, содержащей 16 миллионов узлов и 100 миллионов тетраэдров, для задачи об обтекании цилиндра дозвуковой струей. Моментальная картина течения показана для наглядности на рис. 1 справа. Ускорение, нормированное по времени на 64 ядрах, составило на 12800 ядрах 152. Результаты показаны на рис. 1 слева. Один шаг по времени 4-шаговой схемы Рунге-Кутты занимал 26.8 и 0.18 секунды на 64 и 12800 ядрах соответственно.

Детальное таймирование по различным составляющим алгоритма показывает, что имеется достаточный запас параллелизма, и для расчетов на более подробных сетках могут быть эффективно задействованы несколько десятков тысяч процессорных ядер. Теперь основная проблема, которую требуется решить, это переход к гетерогенной архитектуре с графическими процессорами.

Применение GPU исследуется на упрощенном, но репрезентативном примере, воспроизводящем характерную вычислительную нагрузку. В качестве математической модели взята система безразмерных уравнений Эйлера, в качестве модельной тестовой задачи - сверхзвуковое обтекание сферы (число Маха равно 2.75). Расчетная область представляет собой сферу

(D = 50), внутри которой расположено обтекаемой тело - сфера единичного диаметра. Центр обтекаемой сферы совпадает с началом координат. Для дискретизации по пространству применяется конечно-объемный метод с определением значений сеточных функций в центрах элементов сетки, то есть контрольные объемы (расчетные ячейки) совпадают с элементами сетки. Используется неструктурированная тетраэдральная сетка, равномерно сгущающаяся к поверхности обтекаемого тела (рисунок 2 слева). Конвективный поток через грани контрольных объемов вычисляется с использованием схемы Роу. Интегрирование по времени – схема Рунге-Кутты первого порядка точности.



а) поверхностная сетка б) срединное сечение Y = 0

Рис. 2. Тетраэдральная сетка: 79 608 узлов, 472 114 элементов, 942 088 внутренних граней расчетных ячеек (слева) и картина течения (справа).

На рисунке 2 справа показана построенная по результатам проведенных вычислительных экспериментов картина распределения модуля скорости в плоскостях Y = 0 и Z = 0 вблизи поверхности сферы. Далее для данного примера подробно рассматривается реализация упрощенного алгоритма на GPU.

3. Использование графических процессоров

3.1 Выбор средства разработки

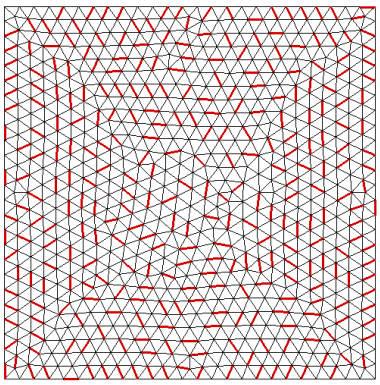
Существуют несколько методик и средств разработки для написания вычислительных ядер и компиляции программ для выполнения на GPU. Это в частности CUDA (Compute Unified Device Architecture) для графических процессоров NVidia, поддерживающих технологию GPGPU (произвольных вычислений на видеокартах) и OpenCL (Open Computing Language - открытый язык вычислений) для GPU различных производителей [10–12]. CUDA является наиболее популярным средством в настоящее время. Однако комплексы программ, использующие CUDA, не переносимы и могут использоваться только на оборудовании NVidia. Это может являться существенным недостатком, поскольку часто необходимо, чтобы расчеты можно было выполнять на любых доступных вычислительных системах. Поэтому выбор был сделан в пользу OpenCL, который обеспечивает переносимость программы и позволяет использовать графические процессоры как производства NVidia, так и ATI (AMD). Для сравнения на начальном этапе были выполнены реализации алгоритма с использованием обоих средств, CUDA и OpenCL, для того чтобы убедиться, что реализация на OpenCL не уступает в производительности реализации на CUDA. Измерение производительности было выполнено на GPU NVidia C1060 и показало отсутствие заметных различий в скорости работы программы.

3.2 Адаптация программных алгоритмов к GPU

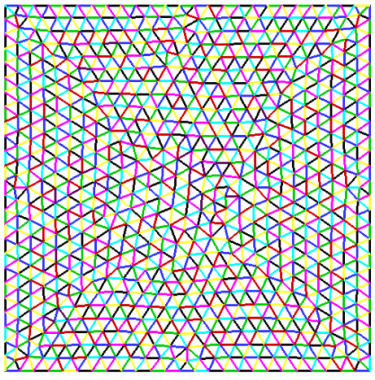
С точки зрения программной реализации, численный алгоритм, по сути, представляет собой множество требующих обработки однотипных заданий по вычислению потоков через грани расчетных ячеек. Для вычисления потока через внутреннюю грань необходимо знать значения газодинамических переменных в ячейках, которым эта грань принадлежит, а так же ее геометрические параметры (площадь, вектор нормали и т.д.). Для вычисления потока через граничную грань помимо ее геометрических параметров необходимо знать значения газодинамических переменных только в одной ячейке. Поскольку значения газодинамических переменных считаются постоянными для одного шага интегрирования по времени, то задания по вычислению потоков независимы по входным данным. То есть они могут обрабатываться параллельно, что идеально подходит для перехода на GPU. Но, как и в случае распараллеливания таких алгоритмов в рамках модели общей памяти, остается проблема суммирования потоков. Определенный с использованием схемы Роу поток через общую грань двух расчетных ячеек i и j суммируется в соответствующие массивы потоков по ячейкам, которые далее используются для нахождения значений газодинамических переменных на новом слое по времени. Тогда, если, например, параллельно вычислять потоки через грани между ячейками i → j и i → k, то существует большая вероятность возникновения ошибки при одновременном суммировании потоков для ячейки i двумя разными нитями. В данном случае при разработке алгоритма для GPU эта проблема была решена двумя способами: вычислением потоков с репликацией данных и потактовым вычислением потоков.

В первом случае, с целью исключения конфликтов по доступу к памяти на этапе вычисления потоков создается дополнительный массив для записи потоков по граням ячеек, что несколько увеличивает потребление оперативной памяти: для каждой грани записывается два набора из 5 значений (плотность, 3 компоненты скорости и давление). Для использованной тетраэдральной сетки, имеющей 942 088 внутренних граней, это около 75 МБ для чисел двойной точности. Кроме того, после вычисления потоков через грани, необходимо просуммировать потоки, чтобы получить конечные значения в ячейках. Для этого необходимо хранить в памяти так называемый дуальный граф связей контрольных объемов, который содержит $(2 \cdot N_S + N_E + 1)$ целочисленных значений. Здесь N_E – число ячеек в сетке, а N_S – число внутренних граней контрольных объемов. Для рассматриваемого случая объем памяти для хранения дуального графа сетки составляет порядка 9 МБ, а общее увеличение затрат оперативной памяти равно 84 МБ, что в общем случае не является критичным. Плюс этого подхода заключается в том, что все задания по обработке внутренних граней контрольных объемов могут быть запущены одновременно.

Во втором случае к дуальному графу сетки применяется алгоритм раскраски ребер. Идею данного алгоритма покажем на примере дуального графа сетки, изображенного на рисунке 3. Узлы графа – ячейки сетки, ребра – связи ячеек через общие грани контрольных объемов.



а) пример выделения подмножества



б) декомпозиция на восемь тактов

Рис.3. Декомпозиция дуального графа для реализации потактового алгоритма.

Попытаемся выделить из множества ребер графа такое подмножество ребер, чтобы каждый из узлов встречался не более одного раза (рисунок 3 а). Обработка запущенными на GPU нитями такого подмножество ребер может происходить параллельно и по определению исключает возможность конфликтов нитей по модификации одних и тех же ячеек памяти. Далее разобьем дуальный граф на такие подмножества (рисунок 3 б), а саму процедуру вычисления потоков на такты. На каждом из таких тактов обрабатывается одно из выделенных подмножеств ребер сетки. Число тактов по вычислению потоков не может быть меньше максимального числа граней для контрольных объемов, то есть, например, для гексаэдральной сетки не будет меньше 6. При этом число одновременно обрабатываемых на каждом из тактов граней не может быть больше, чем $N_E/2$. Необходимость многократного запуска процедуры вычисления потоков для каждого из тактов и снижение числа обрабатываемых одновременно граней – отрицательные стороны данного метода. Но по сравнению с репликацией данных в потактовом алгоритме нет необходимости в выделении дополнительных объемов оперативной памяти и лишнем суммировании вычисленных по граням потоков в ячейки.

Производительность обоих способов была сравнена на GPU NVidia C1060, первый вариант оказался быстрее на 17% и был принят в качестве основного. Было также оценено влияние на производительность способа размещения блочных векторов в памяти. Например, каждому контрольному объему соответствует 5 значений физических переменных. Таким образом, данные могут быть развернуты в линейный массив путем записи N_E блоков по 5 значений. В этом случае адресация j-й переменной для i-го контрольного объема будет иметь вид V[i*5+j]. Такой вариант больше подходит для архитектуры CPU, но для GPU архитектуры намного эффективнее разворачивать данные в 5 блоков по N_E значений, что соответствует адресации $V[i+j*N_E]$. Такое размещение соответствует слиянию доступа к памяти (coalescing), что существенно увеличивает производительность (в данном случае более чем в полтора раза).

4. Производительность вычислений на GPU

Для тестовой реализации алгоритма было выполнено измерение производительности на GPU и CPU с помощью таймирования с высоким разрешением временных затрат на выполнение как всего шага по времени, так и отдельных операций. В частности, замерялся расчет потоков через грани контрольных объемов, суммирование потоков с граней в ячейки, граничные условия (Г. У.), переход на новый временной слой по схеме Рунге-Кутты (Р. - К.). Измерения выполнялись для последовательной host-программы, работающей на одном CPU ядре, и программы, использующей одно GPU устройство. Измерения усреднялись на интервале порядка 100 шагов по времени, чтобы повысить точность таймирования и избежать случайных отклонений в результатах измерений. Все вычисления выполняются для чисел с плавающей точкой **двойной точности** (double, 64 бита). Результаты приведены в таблице 1 для нескольких моделей GPU и CPU.

Таблица 1. Время вычислений (сек.) на GPU и CPU.

Операция	**Intel Xeon E5504 2.0GHz**	**Intel Xeon X5670 2.93GHz**	**NVidia C1060**	**NVidia GTX470**	**NVidia C2050**	**AMD Radeon 5870**	**AMD Radeon 6970**
Расчет потоков	0.255	0.172	0.0095	0.0053	0. 0056	0.0041	0.0031
Суммирование	---[1]	---	0.0043	0.0033	0.0026	0.0030	0.0022
Г. У.	0.0011	0.0008	0.00045	0.00021	0.0003	0.0004	0.00028
Р. - К .	0.029	0.022	0.0015	0.00103	0.0014	0.0016	0.0019
Всего	0.285	0.195	0.0157	0.00984	0.0099	0.0091	0.0075

В данном случае алгоритм полностью выполнялся на GPU, при интегрировании по времени не требовалось передачи данных между CPU и GPU. Если бы на шаге по времени присутст-

[1] На CPU суммирование не выполняется, потоки сразу рассчитываются в узлах.

вовали этапы вычислений, выполняющиеся только на CPU, что возможно будет иметь место в перспективе, то обмен данными составил бы две операции (запись и чтение) по $5 \cdot N_E$ значений, которые бы заняли примерно 0.01 секунды, что могло бы увеличить время вычислений на GPU.

Для проверки корректности вычислений на GPU, как корректности реализации алгоритма, так и корректности работы устройства, вычисления дублировались на CPU, и через заданное число шагов выполнялось сравнение значений величин во всех контрольных объемах. На одной из тестируемых аппаратных конфигураций, а именно на системе, имеющей 4 GPU NVidia Tesla C1060, была отмечена некорректная работа 3-х устройств из 4-х: возникали расхождения с результатами CPU в произвольные моменты, меняющиеся от запуска к запуску. При интенсивном доступе к памяти GPU в процессе вычислений некорректно срабатывала выборка данных из памяти, что было проверенно на отдельном тесте с копированием массивов. При уменьшении числа нитей проблема пропадала, но при этом производительность снижалась в полтора раза. На других аналогичных устройствах такой проблемы не возникло, однако обнаруженная ненадежность GPU в случае выполнения интенсивных вычислений приводит к выводу о том, что необходима разработка дополнительного механизма проверки результатов вычислений на графических процессорах. К примеру, в данном случае, при отсутствии сравнения с CPU кодом, проблему невозможно было бы обнаружить, так как ошибки выборки из памяти не приводили к возникновению некорректных значений (NaN, inf, отрицательная плотность или давление, и т.д.). Однако, в реальных расчетах сравнение вычислений GPU и CPU на каждом шаге по времени естественно не имеет смысла и нужно искать другие подходы.

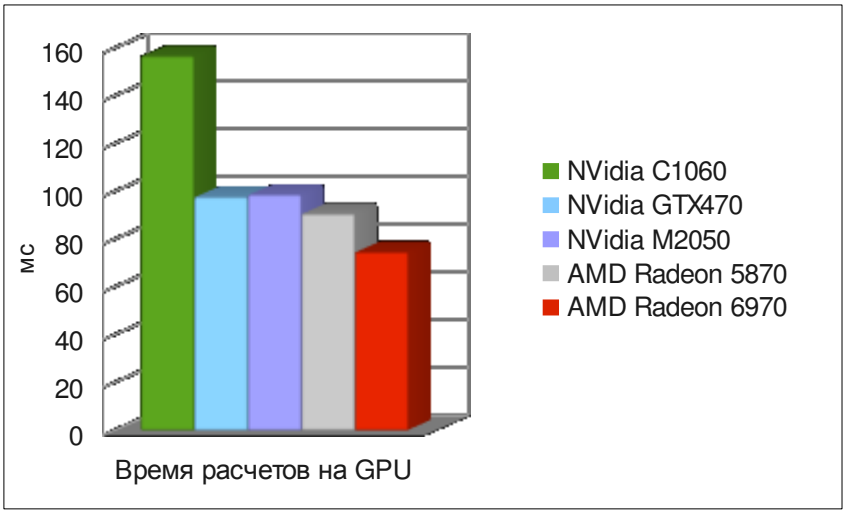


Рис.4. Сравнительная диаграмма времен исполнения алгоритма для разных моделей GPU

Работа тестовой реализации алгоритма была проверена на графических процессорах ATI (AMD) и NVidia. Здесь следует отметить, что на ATI 5870 и ATI 6970 была получена более высокая производительность, чем на NVidia C2050, при том, что последнее имеет примерно в 10 раз большую стоимость (на московском рынке декабря 2010 года). В то же время, действия по оптимизации при написании кода вычислительных ядер имели на ATI намного больший эффект, чем на NVidia. Из этого можно сделать вывод, что ATI имеет более слабый компилятор, в котором не реализованы основные оптимизационные алгоритмы, тогда как на NVidia компилятор сам выполняет оптимизацию кода достаточно хорошо. На ATI доработка кода повысила производительность примерно в два раза, в то время как на NVidia только на 10%. Также на AMD Radeon 6970 подпрограммы Г. У. и Р. - К. работали медленнее, чем на более старой модели, что тоже говорит о недостаточной эффективности компилятора AMD (ATI) для 6970. К особенностям модели 6970 надо также отнести энергосберегающий режим, в котором понижаются частоты процессора и памяти, поэтому необходимо «прогревать» карту перед началом расчетов для достижения максимальной производительности. Действия по оптимизации кодов вычислительных ядер, которые были выполнены, включают в себя:

1. Замена всех переменных, известных до выполнения ядра, на константы. Поскольку компиляция ядра выполняется в процессе работы CPU программы, на момент компиляции, например, уже известны размерности массивов, число контрольных объемов и граней, различные параметры (число Маха, параметры численной схемы и т.д.). Поэтому эти значения дописываются в виде #define в текст исходного кода ядра. Это действие на NVidia дало выигрыш 5%.
2. Минимизация числа операций деления. На центральном процессоре разница в скорости между выполнением операций умножения и деления сказывается намного меньше, в то время как на GPGPU эта разница может достигать двух порядков
3. Как уже было сказано, существенное, в полтора раза, увеличение скорости произошло при переходе к слиянию доступа к памяти (coalescing), за счет переупорядочивания массивов.
4. Существенным для ATI стало добавление директивы #pragma unroll <N> перед циклами внутри ядер.
5. Ядро Г. У. для ATI удалось ускорить в 50 раз после анализа результатов профайлера и сведения к нулю количества фальшивых регистров (scratch register).

5. Общий подход к гетерогенным вычислениям

Как уже было сказано, во многих реальных задачах потребуется эффективное использование ресурсов одновременно и CPU и GPU. Поэтому, необходима проработка общей методологии программирования гетерогенных вычислительных систем, то есть систем, в состав которых входят вычислители принципиально отличных архитектур. Современный вычислительный узел может одновременно нести на борту несколько универсальных многоядерных процессоров (SMP CPU), потоковые ускорители на базе графических процессоров (GPGPU), модули программируемой логики (FPGA), гибридные процессоры-ускорители (CELL) и т.д. Каждое устройство имеет свою модель программирования. В этой связи, эффективное программирование неоднородного вычислительного узла представляет собой нетривиальную задачу, сопряженную с глубоким знанием архитектуры каждого вычислителя и сбалансированным распределением аппаратных ресурсов под конкретную задачу. Проблему зоопарка программных моделей призван решить открытый стандарт языка для вычислений OpenCL, разработанный консорциумом Khronos[10]. Стандарт OpenCL оперирует с обобщенной моделью вычислительного устройства, под которую подходит большинство известных на текущий момент архитектур.

Вопрос планирования задач, пересылки данных, балансировки нагрузки между вычислителями, синхронизации этапов вычислений и т.п. решает модель StarPU - пакет, разработанный в национальном исследовательском институте информатики и автоматизации Франции[13]. На базе StarPU реализовано новое поколение эффективных библиотек численного анализа для гетерогенных систем [14].

Новые стандарты и подходы к программированию гетерогенных систем должны лечь в основу будущих кодов и пакетов прикладных программ. Отдельной задачей стоит перенос кодов, написанных в старой парадигме программирования многоядерных систем.

6. Заключение

Реализованное двухуровневое распараллеливание MPI+OpenMP позволяет выполнять расчеты на сетках с числом узлов до 200 миллионов и числом тетраэдров до 1.2 миллиарда с использованием более 10 тысяч процессорных ядер. На примере упрощенной математической модели был выполнен перенос вычислений на графические процессоры. Численный алгоритм для моделирования внешнего обтекания невязким сжимаемым газом на основе явной схемы первого порядка с центрами в контрольных объемах для неструктурированных сеток был адаптирован к вычислениям на GPU и реализован в виде вычислительных ядер на OpenCL. Сравне-

ние производительности вычислений с двойной точностью на одном ядре CPU и на одном GPU устройстве показало 20 ~ 38-кратное ускорение всего алгоритма.

Литература

1. Aubry R., Houzeaux G., Vazquez M., Cela J. M., Some useful strategies for unstructured edge-based solvers on shared memory machines, International Journal for Numerical Methods in Engineering (2010) n/a. doi:10.1002/nme.2973.
2. Itakura K., Uno A., Yokokawa M., Ishihara T., KanedaY., Scalability of hybrid programming for a CFD code on the Earth Simulator, Parallel Computing 30 (12) (2004) 1329–1343
3. Nakajima K., Three-level hybrid vs. flat MPI on the Earth Simulator: Parallel iterative solvers for finite-element method, Applied Numerical Mathematics 54 (2) (2005) 237–255.
4. Heuveline V., Krause M.J., Latt J., Towards a hybrid parallelization of lattice Boltzmann methods, Computers and Mathematics with Applications 58 (5) (2009) 1071–1080.
5. Chorley M.J., Walker D.W., Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters, Journal of Computational Science 1 (3) (2010) 168–174.
6. Monakov A., Lokhmotov A., Avetisyan A., Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, High Performance Embedded Architectures and Compilers, Lecture Notes in Computer Science, 2010, Volume 5952/2010, 111-125, DOI: 10.1007/978-3-642-11515-8_10
7. Buatois, Luc and Caumon, Guillaume and Levy, Bruno, Concurrent number cruncher: a GPU implementation of a general sparse linear solver, Int. J. Parallel Emerg. Distrib. Syst., 24 (3) (2009) 205—223, DOI: 10.1080/17445760802337010
8. Abalakin I., Dervieux A., Kozubskaya T., Computational Study of Mathematical Models for Noise DNS. – AIAA-2002-2585 paper.
9. Abalakin I., Dervieux A., Kozubskaya T., Ouvrard H., Accuracy Improvement for Finite-Volume Vertex-Centered Schemes Solving Aeroacoustics Problems on Unstructured Meshes, – AIAA paper 2010-3933 (2010)
10. Khronos OpenCL Working Group, The OpenCL Specification, Version: 1.1, http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (2010)
11. Advanced Micro Devices, Inc, AMD Accelerated Parallel Processing OpenCL Programming Guide, http://developer.amd.com/gpu/AMDAPPSDK/assets/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf (2011).
12. NVIDIA, OpenCL Programming Guide for the CUDA Architecture Version 2.3 , http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_OpenCL_ProgrammingGuide.pdf, (2011)
13. INRIA RUNTIME team, A Unified Runtime System for Heterogeneous Multicore Architectureshttp://runtime.bordeaux.inria.fr/StarPU/ (2010)
14. Dongarra J., Tomov S., Agullo I., Ltaief H., Augonnet C., Namyst R., Thibault S., Faster, Cheaper, Better – a Hybridization Methodology to Develop Linear Algebra Software for GPUs, (2010)

Разработка высокоэффективных тканевых защитных преград с использованием суперкомпьютерных вычислений*

Н.Ю. Долганина, С.Б. Сапожников

Южно-Уральский государственный университет

Работа посвящена разработке высокоэффективных тканевых защитных преград с использованием суперкомпьютерных технологий. Проведены численные эксперименты по исследованию масштабируемости задач динамического взаимодействия индентора с тканевыми защитными преградами, расположенными на регистрирующей среде с применением пакета программ LS-DYNA. Даны рекомендации по использованию градиентных структур при проектировании тканевых преград для увеличения их защитных свойств.

1. Введение

Бронежилеты по стойкости к воздействию средств поражения подразделяют на классы. В ГОСТ Р 50744-95 представлено 10 классов различных по конструкции бронежилетов: легкие (1 и 2 класса), представляющие собой слоистые тканевые бронепластины (защитные преграды) различной толщины; и комбинированные (от 3 до 6а класса), в которых слоистая тканевая бронепластина усилена с лицевой стороны жесткими элементами из металла или керамики [1]. Современные тенденции проектирования комбинированных бронежилетов высоких уровней защиты требуют, чтобы металл или керамика пробивались, притупляя или разрушая сердечники пуль, снижали их скорость до уровня, соответствующего надежной работе тыльной тканевой бронепластины. Таким образом, тканевые бронепластины, как важный элемент конструкции, должны обеспечить одинаково допустимое травмирование тела человека в бронежилетах любых уровней защиты. Так как объектом исследования в работе является многослойная тканевая бронепластина, то среди уровней угроз в 1-ом и 2-м классах самым опасным средством поражения является пистолет ТТ с пулями массой 5,5 г, калибром 7,62, стальным сердечником и максимальной скоростью 445 м/с.

Согласно американскому стандарту NIJ Standard-0101.06 сертификационные испытания бронежилетов проводят в 2 этапа [2]. На первом этапе определяют баллистический предел V_{50} – это скорость пули, при которой бронежилет пробивается с вероятностью 50 %. На втором этапе бронежилет располагают на регистрирующей среде (техническом пластилине: Roma Plastilina No.1) проводят обстрел и определяют глубину вмятины в пластилине, которая не должна превышать 44 мм (рис. 1). Всего при сертификационных испытаниях требуется произвести 168 выстрелов в 18 бронепластин.

Тканевые бронепластины могут содержать несколько десятков слоев высокопрочных тканей из арамидных нитей Кевлар (США) или Русар® (Россия) различного типа переплетения (саржа, сатин, полотно и др.) (рис. 2). При ударе пулей в тканевой бронепластине возникают сложные физические явления: динамическое деформирование с распространением ударных волн, большие прогибы, образование и исчезновение множественных фрикционных контактов, вытягивание и разрушение нитей и др. Все это существенно затрудняет теоретический анализ проблемы локального ударного взаимодействия тканевой бронепластины с индентором. Поэтому в настоящее время при разработке новых конструкций бронежилетов, отличающихся меньшей массой и стоимостью, высокой надежностью, опираются, в основном, на натурный многофакторный эксперимент, что приводит к удлинению сроков проектирования и увеличению стоимости этапа доводки (и изделия в целом), не позволяет выявить влияние различных факторов на прочность и уровень травмирования.

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 10-07-96007-р_урал_а).

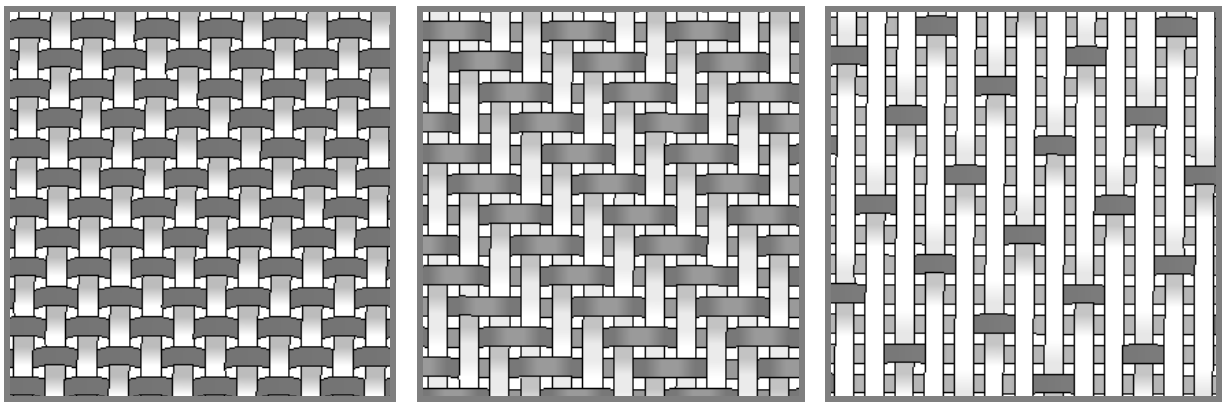
Определение баллистического предела V_{50}

Определение глубины вмятины в регистрирующей среде

1 – многослойная тканевая бронепластина; 2 – регистрирующая среда; W – фактический прогиб (max 44 мм); V_0 – начальная скорость пули; V_k – скорость пули после пробоя.

Рис. 1. Сертификационные испытания бронежилетов

Моделирование тканевых структур прошло несколько стадий развития. Фундаментальной работой в области исследования прочности при интенсивных кратковременных нагрузках является работа Х.А. Рахматулина 1961 года, где рассмотрен удар по прямой гибкой деформируемой незакрепленной нити бесконечной длины (аналитическое решение) [3]. Интенсивное развитие моделирования тканевых структур началось с развитием вычислительной техники и соответствующих прикладных программ. Вначале ткань заменяли ортотропной пластиной [4], затем сетками из ортотропных нитей со связанными узлами [5]. В последнее десятилетие развиваются модели, где моделируется каждая нить [6]. Они ориентированы на пакет конечно-элементного анализа LS-DYNA.



Полотняное переплетение Саржевое переплетение Сатиновое переплетение

Рис. 2. Типы переплетения нитей

В ряде работ используют оболочечную дискретизацию отдельных нитей [7]. Линзообразное поперечное сечение отдельной нити (рис. 3) моделируют несколькими смежными оболочечными элементами с различными толщинами (рис. 4). Однако в таких моделях оболочечные элементы с различными толщинами имеют разрыв в соответствующих граничных условиях и затруднена формулировка условий контакта нитей.

Нити в ткани разбивают также и на объемные конечные элементы (рис. 4) [8]. Преимущество дискретизации объемными элементами перед оболочечными с разными толщинами – это сглаживание плоскости поперечного сечения. Однако такие модели тканей при реальных размерах бронежилетов порядка 30х30 см имеют чрезвычайно большую размерность, что не позволяет рассчитывать слоистые тканевые бронепластины с немногими слоями ткани даже с использованием современных суперкомпьютеров.

Таким образом, задачей исследования было создать малопараметрическую модель ткани, которая бы позволила рассчитать на суперкомпьютере бронежилеты реальных размеров

(30x30 см) с реальным количеством слоев (десятки) для замены большего числа натурных экспериментов виртуальными.

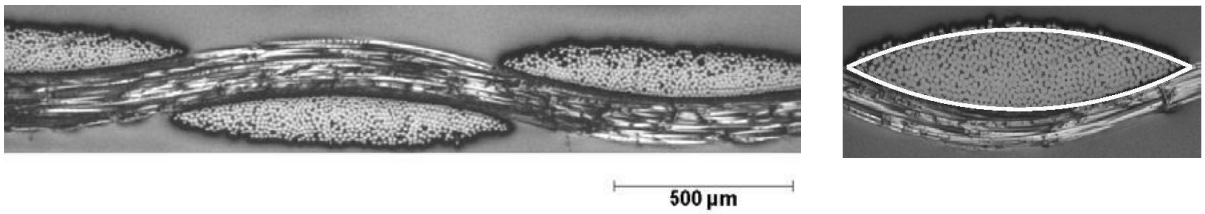
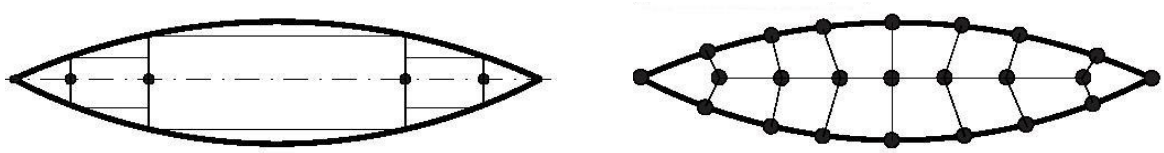


Рис. 3. Фотография баллистической ткани, разрезанной в плоскости



Дискретизация тремя оболочечными элементами с разными толщинами

Дискретизация объемными элементами (два элемента по толщине)

Рис. 4. Дискретизация нитей

В настоящей статье мы рассматриваем моделирование динамического взаимодействия индентора с тканевыми преградами размером 30x30 см с разным количеством слоев на вычислительном кластере «СКИФ Урал». Статья организована следующим образом. В разделе 2 приведена постановка задачи. В разделе 3 описываются методы исследования, и приводится описание задачи. В разделе 4 обсуждаются результаты проведенных экспериментов на вычислительном кластере. В заключении суммируются основные результаты, полученные в данной работе.

2. Постановка задачи

Проведено суперкомпьютерное моделирование натурных экспериментов, которые проводятся при сертификации бронежилетов:

– согласно американскому стандарту NIJ Standard-0101.06 часть выстрелов должно быть сделано под углом 30° к нормали поверхности ткани [2];

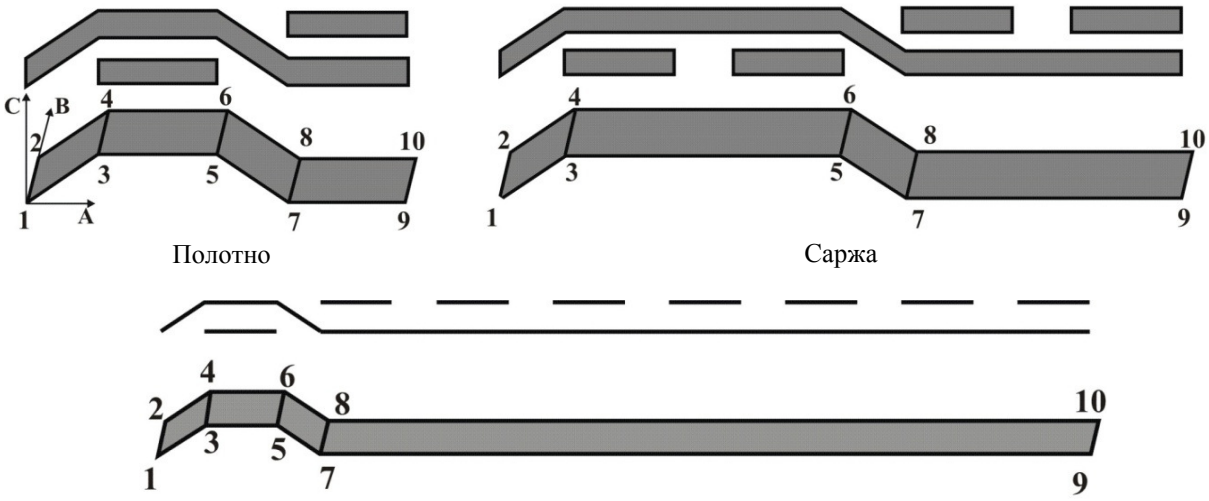
– определение баллистического предела;

– удар пулей в тканевую преграду, расположенную на пластилиновом основании.

При этом были рассмотрены тканевые преграды трех типов переплетения (рис. 2) состоящих из разного количества слоев ткани размером 30x30 см. В работе использована арамидная ткань Русар®. В расчетной модели нити имеют относительную свободу перемещения с возможностью вытягивания с учетом сухого трения. Рассматривали нити, которые имеют прямоугольное поперечное сечение и были представлены одним оболочечным элементом по ширине с одной точкой интегрирования по толщине и выполнены из ортотропного материала с малыми поперечно-сдвиговыми свойствами. Нити в расчетной модели могли разрушаться. В одних расчетах края ткани не были закреплены, в других тканевая преграда располагалась на неподвижном пластилиновом основании. В расчете индентор имел форму цилиндра с полусферическим основанием диаметром 7 мм, массой 5,5 г, с начальной скоростью 445 м/с (имитация пули пистолета ТТ) и был выполнен из абсолютно жесткого материала.

3. Методы исследования

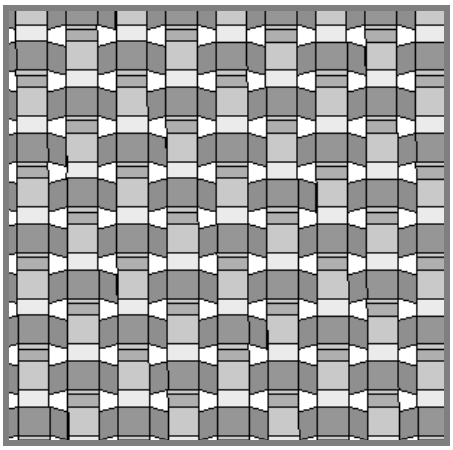
Чтобы снизить размерность задачи при расчете, геометрия нитей была предельно упрощена. Криволинейная ось нити была заменена ломаной, с прямолинейными горизонтальными и наклонными участками (рис. 5). Повторяющиеся элементы тканей показаны на рис. 5, где отмечены номера ключевых точек, координаты которых были введены в программу ANSYS, после чего по ключевым точкам были заданы соответствующие поверхности. Далее набор поверхностей был размножен до получения необходимых размеров модели, после чего была построена сетка конечных элементов (рис.6).



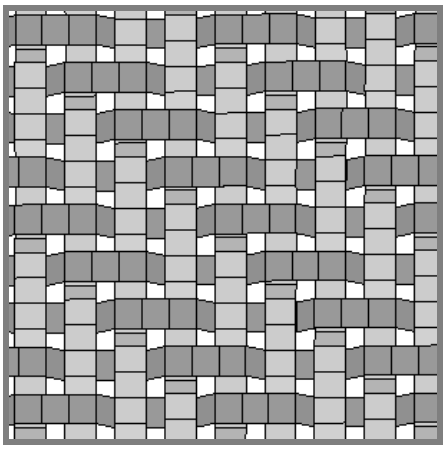
Полотно Саржа

Сатин

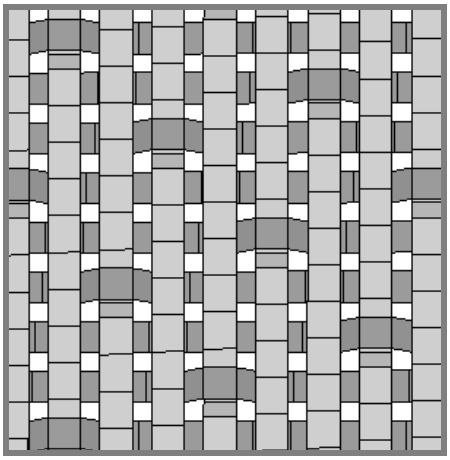
Рис. 5. Повторяющиеся элементы



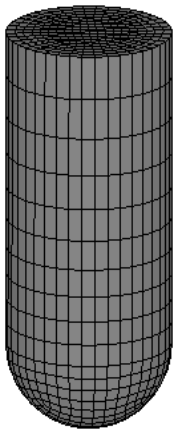
Полотно



Саржа



Сатин



Индентор (имитация пули пистолета ТТ)

Рис. 6. Сетка конечных элементов

В пакете программ LS-DYNA представлено более сотни моделей материала [9], наиболее интересными для нас были малопараметрические модели, их анализ показал, что для нитей наиболее подходит материал *MAT_ENHANCED_COMPOSITE_DAMAGE, позволяющий задать ортотропные свойства нитей и учесть разрушение при достижении первого главного на-

пряжения пределу прочности нитей (3 ГПа). Характеристики нитей для ввода в программу LS-DYNA приведены в табл. 1.

Таблица 1. Характеристики нитей

Параметр	Обозначение	Величина
Толщина нити, мкм	T	100
Ширина нити, мкм	D	500
Модули упругости, МПа	E_x	$1{,}4\cdot10^5$
	E_y	$1{,}4\cdot10^3$
Плотность, кг/м^3	ρ	1 440
Коэффициент Пуассона	μ_{xy}	0,3
Модули сдвига, МПа	G_{xy}	$1{,}4\cdot10^3$
	G_{yz}	$1{,}4\cdot10^3$
	G_{zx}	$1{,}4\cdot10^3$

В конечно-элементной модели ткани нити имеют относительную свободу перемещения и возможность вытягивания с учетом сухого трения. Контакт объектов моделировали командой *CONTACT_AUTOMATIC_ SURFACE_TO_SURFACE с коэффициентом сухого трения 0,4, характерным для арамидных нитей [10].

Материал технического пластилина считали упругопластическим с зависимостью предела текучести от скорости деформирования. Из списка материалов, заложенных в библиотеку пакета LS-DYNA, для технического пластилина был выбран *MAT_STRAIN_RATE_ DEPENDENT_PLASTICITY, который позволяет учесть зависимость предела текучести от скорости деформирования в табличном виде.

Механические свойства нитей и тканей определены экспериментально на универсальной испытательной машине Instron 5882. Из экспериментов на низкоскоростной удар были получены механические свойства технического пластилина [11].

Стальной индентор при взаимодействии с тканевой пластиной при скоростях до 600 м/с не разрушается и не имеет пластических деформаций, поэтому для сокращения времени расчетов для него был выбран материал *MAT_RIGID (жесткое тело) с плотностью ρ = 7800 кг/м^3 и модулем упругости $E = 2{,}1\cdot10^{11}$ Па.

Верификация моделей ткани и технического пластилина была проведена ранее в работе [11], результаты численных расчетов отлично согласуются с экспериментальными исследованиями. Способы декомпозиции модели тканевой преграды при расчете на суперкомпьютере рассмотрены в нашей работе [12], минимальное время расчета получается при разбиении модели на прямоугольные области, проходящие через всю толщину бронепластины.

4. Результаты исследований и их анализ

Расчеты были проведены на высокопроизводительном вычислительном кластере «СКИФ Урал» [13], оснащенном 166 вычислительными узлами с 2 процессорами Intel Xeon E5472 (4 ядра по 3.0 ГГц) и 8 ГБ оперативной памяти на каждом узле.

Вначале было проведено моделирование динамического взаимодействия одного слоя ткани размером 30х30 см сатинового переплетения с индентором, имитирующим пулю пистолета ТТ и ударяющим в центр ткани под углами 0-60° к нормали (при этом нити не разрушались) (рис. 7).

Было получено, что наиболее опасным случаем является удар под углом 30°. Были также проведены расчеты при отсутствии трения между индентором и тканью, получено, что при этом самым опасным случаем является удар под 0° к нормали.

Для определения баллистического предела V_{50} были проведены экспериментальные исследования динамического нагружения 10 слоев ткани сатинового переплетения размером 30х30 см шариком диаметром 8 мм, массой 2 г. Скорость шарика варьировалась от 200 до 600 м/с. Были измерены начальная скорость шарика и скорость после пробоя.

После проведения натурных испытаний были проведены расчетные исследования по замеру скорости после пробоя 10 слоев ткани сатинового переплетения размером 30х30 см инден-

тором сферической формы диаметром 8 мм, массой 2 г. В этих расчетах для экономии машинных ресурсов предложена концепция замены многослойной тканевой бронепластины (10 слоев) на эквивалентную по массе двух- или трехслойную бронепластину (рис. 8). Модельные слои имели увеличенную, по сравнению с реальными, толщину (и массу) нитей. Различие расчетных и экспериментальных данных баллистического предела и остаточных скоростей для всех случаев не превысило 4%.

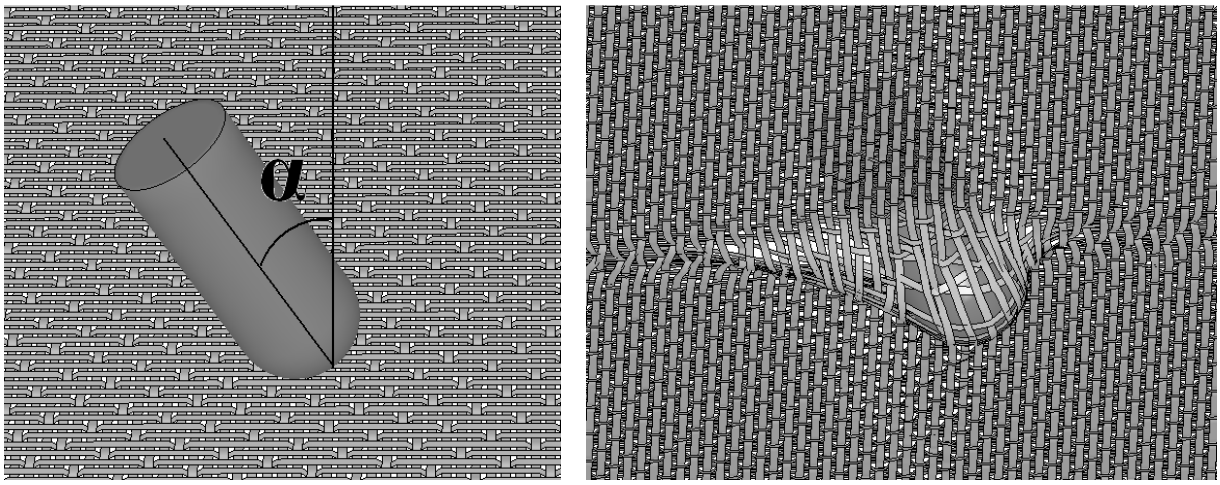


Рис. 7. Удар индентором в ткань под углом 30° к нормали

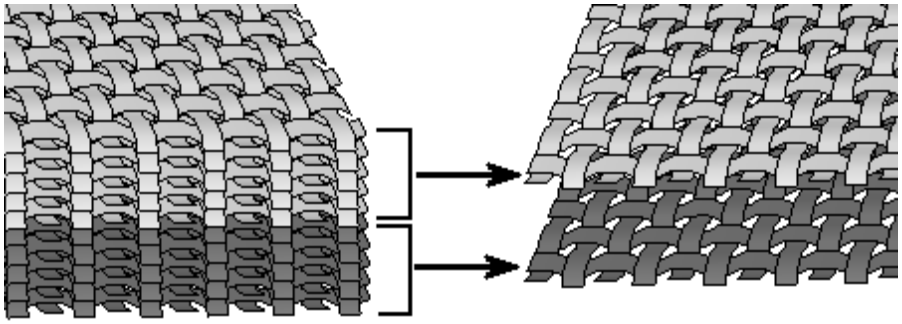


Рис. 8. Замена многослойной бронепластины эквивалентной по массе двухслойной бронепластиной

На рис. 9 показан пробой 2-х эквивалентных слоев ткани сатинового переплетения.

На кластере «СКИФ Урал» были проведены расчеты по обстрелу инденторами (имитация пули пистолета ТТ) четырех различных тканевых бронепластин размером 30х30 см с удельной массой 10 кг/м^2, расположенных на основании из технического пластилина. Скорость удара 445 м/с. Реальная многослойная (60 слоев) бронепластина была заменена 5-ю эквивалентными по массе слоями (это максимальное количество слоев, которое удалось рассчитать).

- В первом случае была рассмотрена бронепластина, состоящая только из тканей полотняного переплетения.
- Во втором – из тканей саржевого переплетения.
- Третья бронепластина имела 1/3 (по массе) наружных слоев полотняного переплетения, 1/3 средних слоев саржевого и 1/3 нижних – сатинового переплетения.
- Четвертая – 1/3 наружных слоев сатинового переплетения, 1/3 средних слоев саржевого и 1/3 нижних полотняного переплетения (рис. 10).

Было проведено измерение глубины вмятины оставленной четырьмя различными бронепластинами в основании из технического пластилина. В результате было получено, что минимальный размер вмятины получается при использовании 4-го типа бронепластины (рис. 11).

Затем был проведен расчет пятой бронепластины (аналог четвертой), в которой 1/3 верхних слоев сатинового переплетения располагались с зазором в 5 мм (выше остальных слоев на 5 мм). Было получено, что глубина вмятины меньше на 15%, чем при использовании этого же

пакета без зазора. Очевидно, силы трения нитей в наружном пакете совершили бóльшую работу.

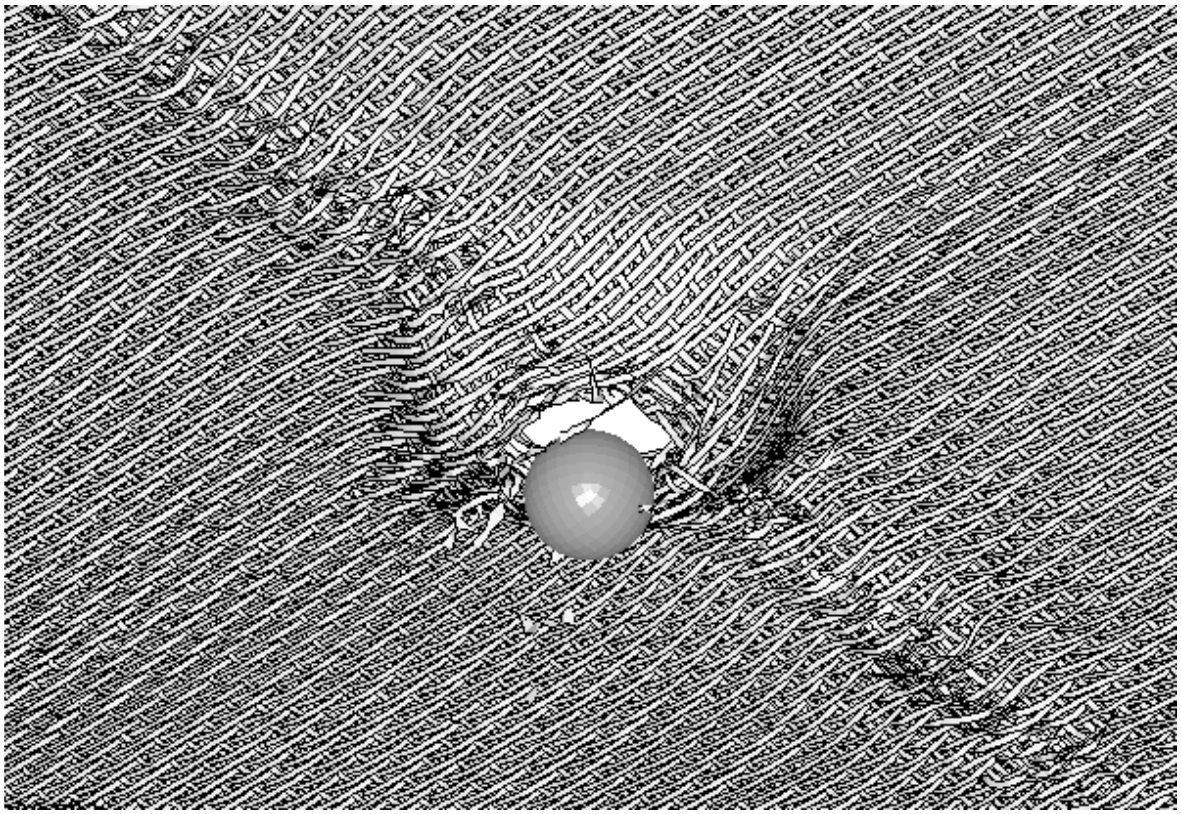


Рис. 9. Картина пробоя двух эквивалентных слоев ткани размером 30х30 см сатинового переплетения

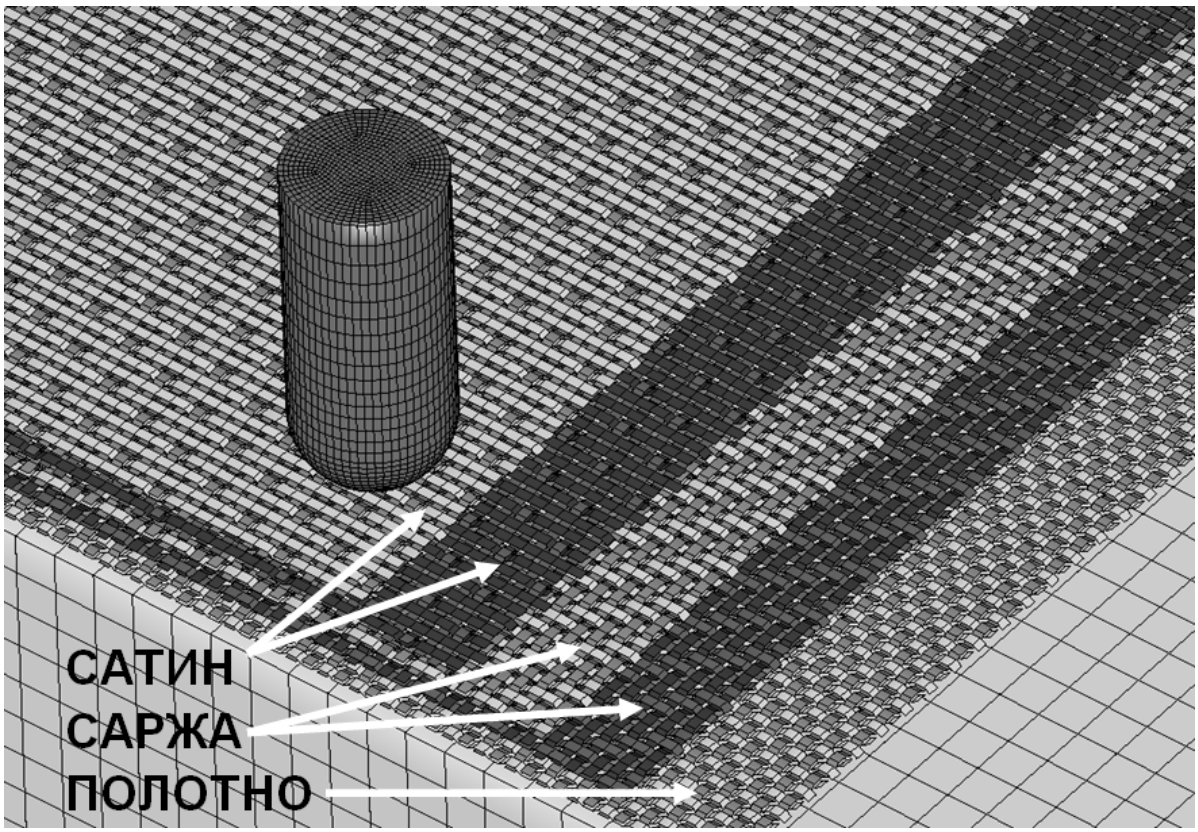


Рис. 10. Бронепластина: сатин-саржа-полотно

Максимальное количество конечных элементов было в задаче, где вся бронепластина состояла из тканей полотняного переплетения и оно равнялось 2 973 960. График ускорения для данной задачи представлен на рис. 12, а время расчета в табл. 2. Данную задачу удалось рассчитать максимум на 80 ядрах.

Общая рекомендация такова: для повышения эффективности работы бронежилетов необходимо использовать градиентные структуры. В них верхние слои должны иметь меньшую искривленность нитей (например, сатин), напряжения в нитях тканей такого переплетения меньше, чем в тканях с саржевым и полотняным переплетениями; коэффициент трения в верхних слоях должен быть снижен, чтобы уменьшить влияние на прочность нитей сверхзвукового удара в начальной фазе контакта с индентором; тыльные слои должны быть выполнены из тканей с максимальной искривленностью, например, с полотняным переплетением и высоким коэффициентом трения между нитями, чтобы увеличить энергию на вытягивание нитей; между наружными и тыльными слоями бронежилета, рационально выполнить зазор, чтобы дополнительно снизить скорость пули за счет потерь на трение при вытягивании нитей в наружных слоях.

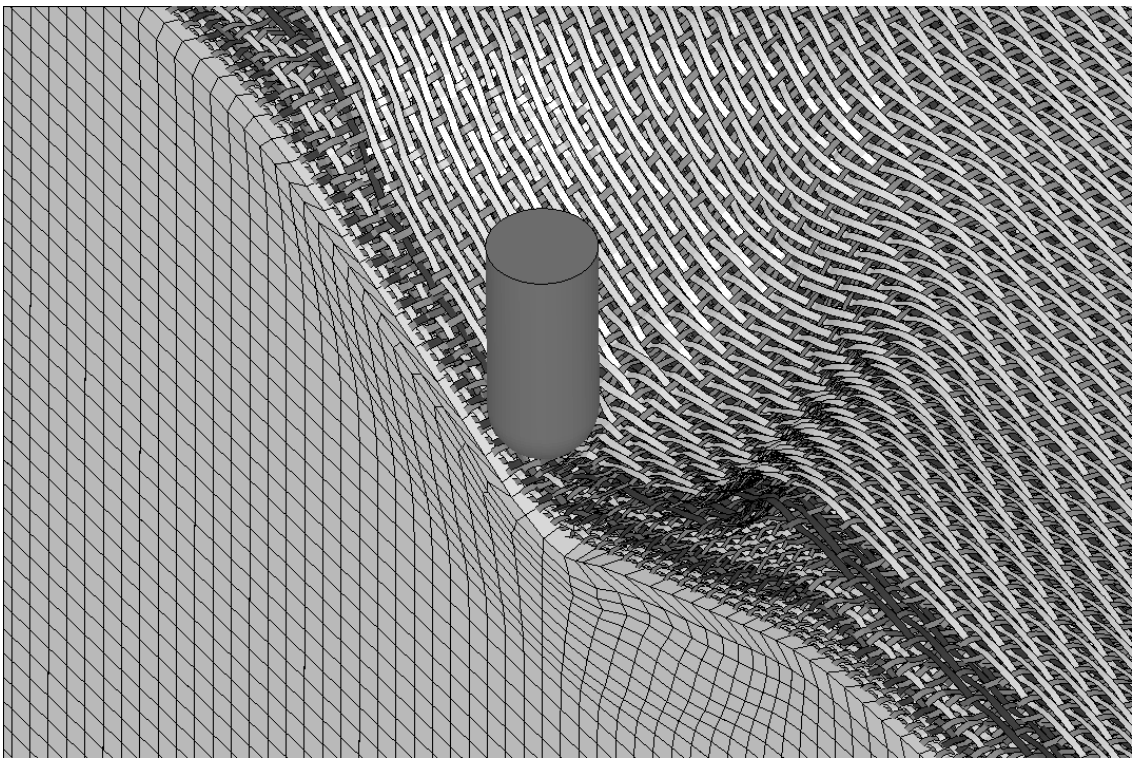


Рис. 11. Бронепластина: сатин-саржа-полотно после удара

Таблица 2. Время расчета

№	Количество ядер	Время расчета, сек
1	24	69 735
2	32	63 637
3	40	55 811
4	48	44 176
5	56	38 402
6	64	31 965
7	72	29 432
8	80	27 846

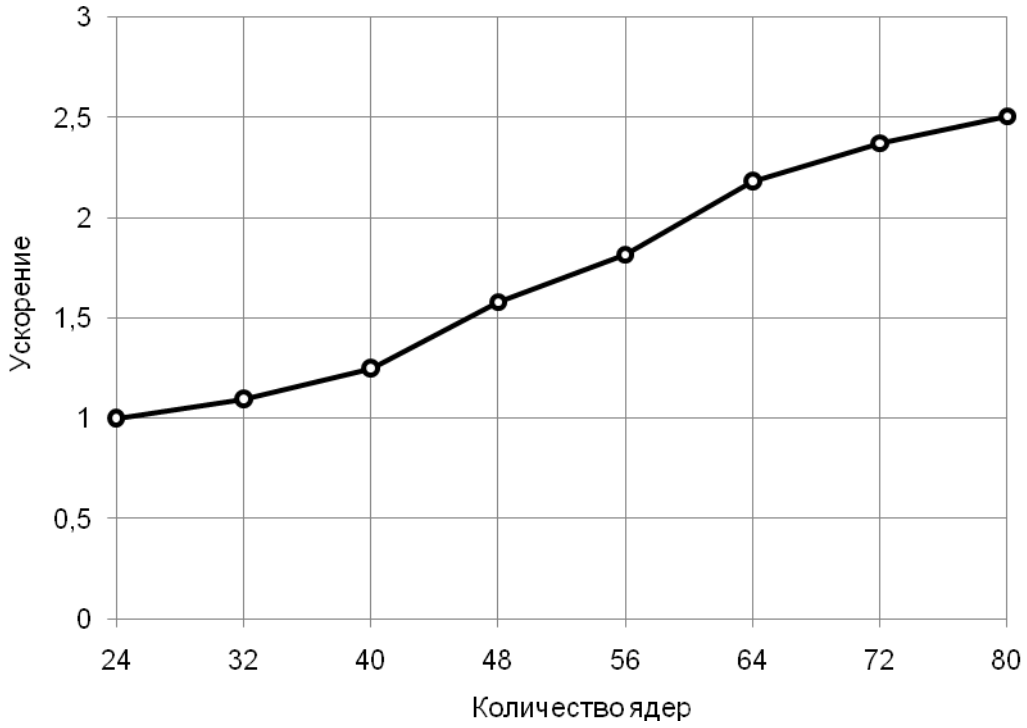


Рис. 12. Ускорение

5. Заключение

1. Разработаны малопараметрические модели тканевых структур, позволившие построить простейшую адекватную модель ткани с плоским переплетением (полотно, саржа, сатин).

2. Рассмотрены несколько случаев динамического нагружения индентором одного слоя ткани сатинового переплетения размером 30x30 см под разными углами к нормали. Самым опасным оказался случай в 30° к нормали, но если снизить до нуля коэффициент трения между тканью и индентором, то самым опасным будет случай нагружения по нормали.

3. Для более эффективного использования компьютерных ресурсов впервые была предложена концепция замены группы слоев в многослойной тканевой бронепластине эквивалентной по массе двух- или трехслойной бронепластиной, в которой слои имеют возможность разрушения.

4. Численные и экспериментальные данные при определении баллистического предела хорошо согласуются между собой (разница не превышает 4% для всех рассмотренных случаев).

5. При разработке новых более эффективных по запреградной энергии бронежилетов необходимо использовать градиентные структуры: в верхних слоях нити должны быть минимально искривлены, т.к. напряжения в более прямых нитях ниже, чем в более искривленных; коэффициент трения в верхних слоях должен быть снижен, чтобы уменьшить влияние сверхзвукового удара; тыльные слои должны быть выполнены из тканей с сильно искривленными нитями и высоким коэффициентом трения, чтобы увеличить энергию на вытягивание нитей; между наружными и тыльными слоями бронежилета нужно сделать зазор, чтобы снизить скорость пули за счет вытягивания нитей в наружных слоях.

6. Результаты данных исследований были внедрены в ЗАО «ФОРТ Технология» (г. Москва). Экономический эффект реализации расчетного проекта для предприятия выражается в одновременном снижении расходов на экспериментальную отработку за счет переноса центра тяжести на виртуальное прототипирование, а в итоге – на получение снижения массы бронежилета и его себестоимости.

Литература

1. ГОСТ Р 50744 – 95. Бронеодежда. Классификация и общие технические требования. Прин. Постановлением Госстандарта России от 09.09.98 № 345. Введ. с изм. № 1 (01.01.1999). – М., 1995.
2. National Institute of Technology Standard, NIJ Standard 0101.06 Ballistic Resistance of Body Armor, July 2008.
3. Рахматуллин Х.А., Демьянов Ю.А. Прочность при интенсивных кратковременных нагрузках. – М.: ГИФМЛ, 1961. – 339 с.
4. Lim C.T., Shim V.P.W., Ng Y.H. Finite-element modeling of the ballistic impact of fabric armor // International Journal of Impact Engineering. 2003. Vol. 28. P. 13-31.
5. Tan V.B.C., Shim V.P.W., Tay T.E. Experimental and numerical study of the response of flexible laminates to impact loading // International Journal of Solids and Structures. 2003. Vol. 40. P. 6245-6266.
6. Chocron S., Figueroa E., King N., Kirchdoerfer T., Nicholls A.E., Sagebiel E., Weiss C., Freitas C.J. Modeling and validation of full fabric targets under ballistic impact // Composites Science and Technology. 2010.
7. Blankenhorn G., Schweizerhof K., Finckh H. Improved Numerical Investigations of a Projectile Impact on a Textile Structure // 4th European LS-DYNA Users Conference: Proceedings of the European Users Conference (22-23 May 2003, Ulm). 2003. P. G-I-07 – G-I-14.
8. Talebi H., Wong S.V., Hamouda A.M.S. Finite element evaluation of projectile nose angle effects in ballistic perforation of high strength fabric // Composite Structures. 2009. Vol. 87. No. 4. P. 314-320.
9. LS-DYNA Keyword user's manual. v.970. LSTC, 2003. - 1564p.
10. Martinez M.A., Navarro C., Cortes R., Rodriguez J., Sanchez-Galvez V. Friction and wear behaviour of Kevlar fabrics // Journal of materials science. – 1993. – Vol. 28. – P. 1305 – 1311.
11. Долганина, Н.Ю. Оценка баллистического предела и прогиба многослойных тканевых пластин при ударе индентором // Вестник ЮУрГУ. Серия «Машиностроение». – 2010. – Вып. 15. – № 10(186). – С. 17 – 23.
12. Долганина Н.Ю., Сапожников С.Б., Маричева А.А. Моделирование ударных процессов в тканевых бронежилетах и теле человека на вычислительном кластере «СКИФ Урал» // Вычислительные методы и программирование: Новые вычислительные технологии. – 2010. – Т. 11. – С. 117 – 126.
13. Высокопроизводительный вычислительный кластер «СКИФ Урал»: [http://supercomputer.susu.ru/computers/ckif_ural/].

Параллельная реализация алгоритма предсказания с помощью модели градиентного бустинга деревьев решений*

П.Н. Дружков, Н.Ю. Золотых, А.Н. Половинкин

Нижегородский государственный университет им. Н.И. Лобачевского

Приводится описание реализации одного из алгоритмов обучения с учителем – градиентного бустинга деревьев решений (Gradient Boosting Trees). Представлены различные варианты параллельной реализации алгоритма предсказания с использованием библиотеки Intel Threading Building Blocks. Приводятся результаты экспериментального сравнения и анализ производительности различных подходов 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 [13], метод *K* ближайших соседей [9], нейронные сети [9], AdaBoost [9], деревья решений [2] и их различные ансамбли (случайные деревья [1], полностью случайные деревья [8], градиентный бустинг деревьев решений [6, 7]). 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 [12], 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 [11].

Необходимо отметить, что многие из решаемых в настоящее время практических задач машинного обучения и компьютерного зрения требуют обработки значительного объема входных данных. В частности, 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. Наглядным примером такой задачи может служить детектирование пешеходов [4], где в зависимости от выбираемых параметров необходимо классифицировать от 10000 до 185000 объектов на одно изображение. В связи с этим, наряду с качеством предсказания на одно из первых мест встает вопрос производительности используемого алгоритма. В работе рассматриваются аспекты параллельной реализации алгоритма обучения модели, а также предлагаются и анализируются различные подходы к распараллеливанию алгоритма предсказания на новых данных.

* Авторы благодарят И.Б. Меерова (ННГУ) и В. Л. Ерухимова (компания ITSeez) за ценные замечания и полезные обсуждения. Работа выполнена при поддержке федеральной целевой программы «Научные и научно-педагогические кадры инновационной России», госконтракт 02.740.11.5131.

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*. Каждому объекту $x \in X$ 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* 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* [9]. 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*) [5]. 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.

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 *k*-м шаге алгоритма обучения называется значение $\tilde{y}_i = y_i - T_0(x_i) - \nu \cdot \sum_{m=1}^{k-1} T_m(x_i)$, что соответствует разности между истинным значением ответа и значением, предсказанным ансамблем деревьев решений, построенным на $(k-1)$-м шаге алгоритма обучения, *M* – общее число деревьев в ансамбле. Тогда общая схема тренировки модели 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 $(x_i, \tilde{y}_i), i = 1,...,n$

Детальное описание алгоритма обучения, а также подобности, 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 [6].

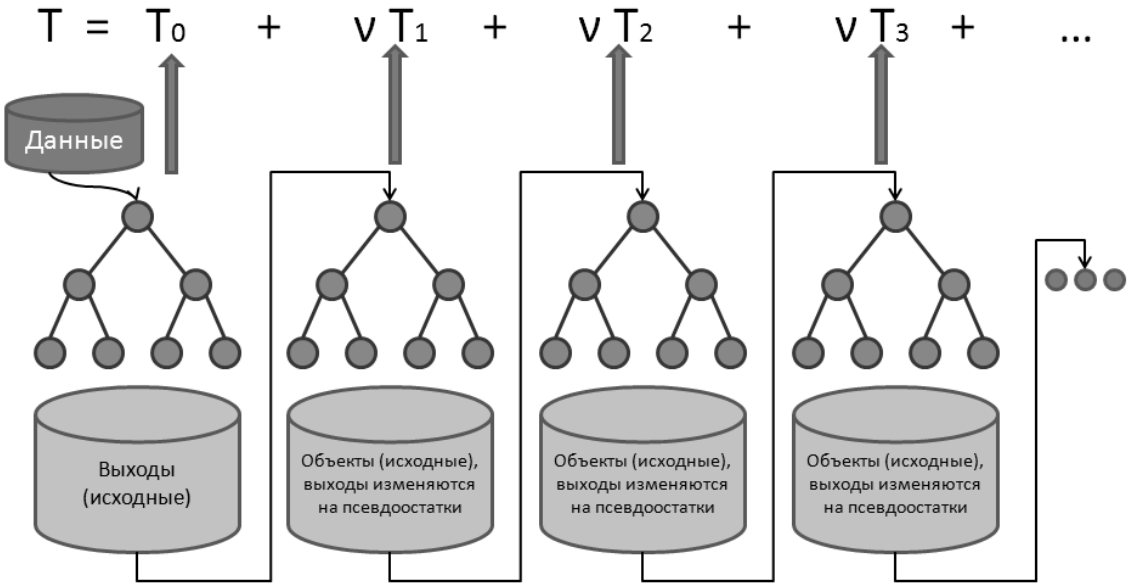


Рис. 1. Алгоритм обучения модели градиентного бустинга деревьев решений

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 $y = T_0(x) + \nu \cdot \sum_{m=1}^{M} T_m(x)$.

2.3 Реализация алгоритма градиентного бустинга деревьев решений

Авторами данной работы была выполнена программная реализация алгоритма градиентного бустинга деревьев решений, 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], машина опорных векторов [13]. 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.

Таблица 1. Результаты экспериментального сравнения алгоритмов обучения с учителем

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

3. Параллельная реализация алгоритма градиентного бустинга деревьев решений

3.1 Алгоритм обучения

Обучение модели представляет собой достаточно трудоемкий процесс. Более того, для осуществления подбора наилучших параметров модели (количество деревьев в ансамбле, масштабный параметр, ограничение на размер деревьев) требуется неоднократное повторение процесса, что обуславливает необходимость оптимизации по скорости, в том числе за счет применения техник параллельных вычислений. Согласно результатам профилировки последовательной версии алгоритма обучения (использовался набор данных spambase из библиотеки UCI: размерность пространства признаков равна 56, число прецедентов в обучающей выборке – 2300), приведенным на рис. 2, основное время работы тратится на обучение одиночных деревьев решений.

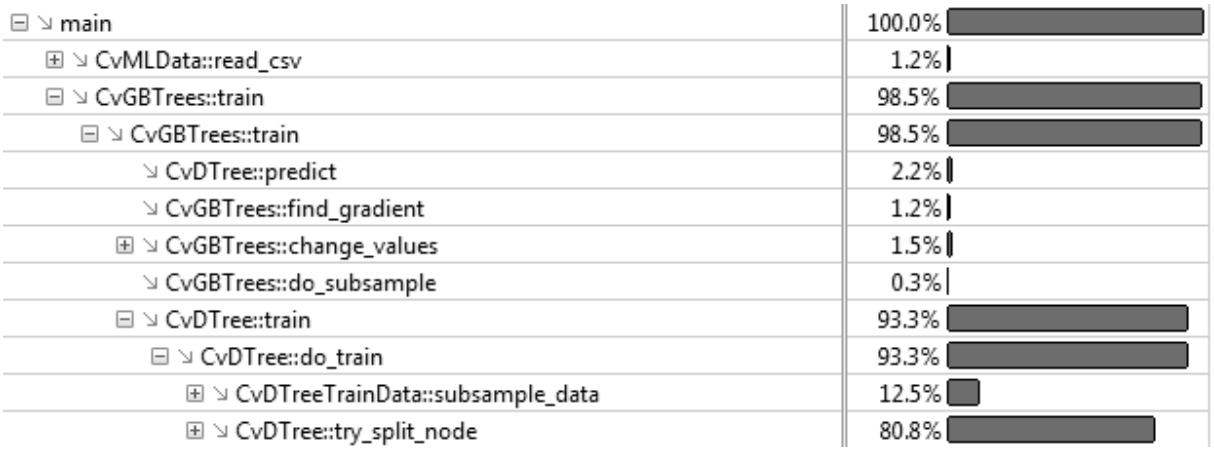


Рис. 2. Результаты профилировки алгоритма обучения модели градиентного бустинга деревьев решений

К сожалению, бустинг-алгоритмы 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. Однако некоторые этапы построения одиночного дерева решений могут быть выполнены независимо друг от друга: например, нахождение разбиения во внутреннем узле дерева, которое осуществляется путем вычисления для каждой переменной уменьшения значения функции неоднородности [2] и выбора оптимального значения среди всех переменных. Именно на этом уровне выполнено распараллеливание обучения деревьев решений в библиотеке OpenCV.

Для изучения влияния параллельного подхода к отысканию оптимальной переменной каждого разбиения на скорость работы всего алгоритма обучения модели градиентного бустинга была проведена серия экспериментов.

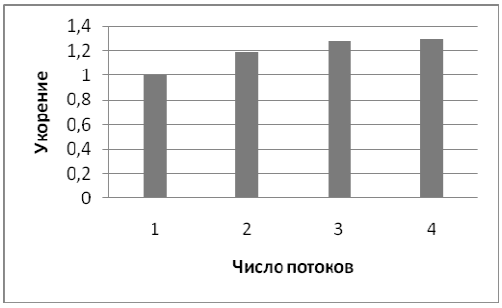


Рис. 3. Эффективность распараллеливания алгоритма обучения модели градиентного бустинга (на наборе данных spambase)

Как видно из рис. 3, полученное ускорение значительно ниже линейного. Это связано с тем, что суммарное время работы алгоритма поиска оптимального разбиения составляет (по результатам профилировки) лишь около 40% времени работы всего алгоритма обучения. Таким образом, максимальное ускорение, которое может быть получено, согласно закону Амдала равно 1.67.

3.2 Алгоритм предсказания

Несмотря на то, что предсказание в алгоритме градиентного бустинга деревьев решений не столь трудоемкий процесс по сравнению с построением модели, время работы этого алгоритма зачастую также является критичным. На это есть несколько причин. Обучение выполняется на «оффлайн» этапе, в то время как дальнейшее (и основное) применение модели заключается в осуществлении с ее помощью предсказаний на новых данных. Более того, на практике часто приходится выполнять не единичные предсказания, а целые серии: например, в некоторых алгоритмах, решающих задачу детектирования объектов на изображении, может потребоваться выполнение десятков тысяч предсказаний на одно изображение. Все это накладывает достаточно жесткие временные рамки на время вычисления предсказания – кроме того, в некоторых прикладных задачах требуется работа в режиме реального времени.

В рамках данной статьи рассматривается два различных подхода к распараллеливанию алгоритма предсказания 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. В отличие от процесса обучения, здесь значение каждого слагаемого может быть получено независимо от остальных – отсюда возникает первая параллельная схема 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 рассматриваемый 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. Программная реализация предложенных подходов к распараллеливанию выполнена с использованием технологии Intel Threading Building Blocks [10].

Как видно из рис. 4–5, оба рассматриваемых подхода дают существенное ускорение процесса предсказания. Однако метод распараллеливания по данным обладает лучшей масштабируемостью, так как число объектов, для которых требуется выполнить предсказание, как правило, значительно превосходит количество деревьев в модели градиентного бустинга.

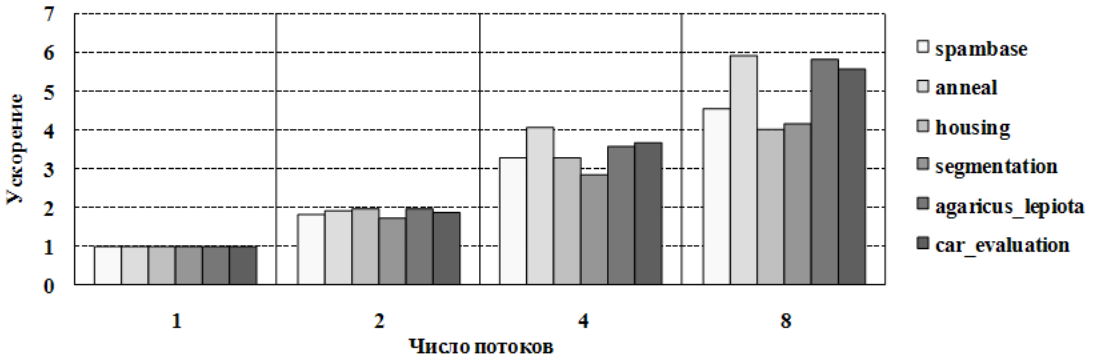


Рис. 4. Эффективность распараллеливания алгоритма предсказания по деревьям в ансамбле с использованием модели градиентного бустинга деревьев решений

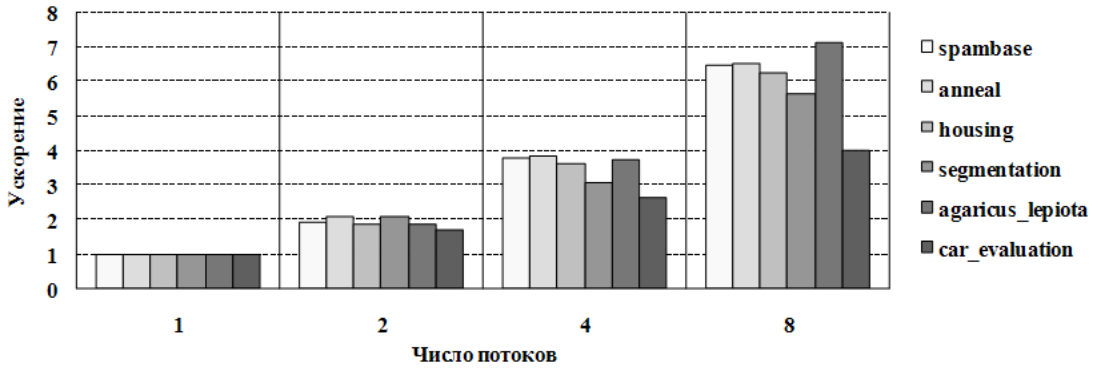


Рис. 5. Эффективность распараллеливания алгоритма предсказания по данным с использованием модели градиентного бустинга деревьев решений

4. Заключение

В статье рассмотрены аспекты параллельной реализации одного из наиболее перспективных алгоритмов обучения с учителем – градиентного бустинга деревьев решений. Проанализирована программная реализация обучения рассматриваемой модели, основанная на параллельной версии алгоритма тренировки отдельных деревьев решений, входящей в состав библиотеки OpenCV. Также в рамках данной работы были предложены и реализованы с использованием технологии Intel Threading Building Blocks две схемы распараллеливания алгоритма предсказания с помощью построенной модели.

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. 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.
5. Freund Y., Schapire R. Experiments with a New Boosting Algorithm // Machine Learning: Proceedings of the Thirteenth International Conference. 1996.
6. Friedman J. H. Greedy Function Approximation: a Gradient Boosting Machine. Technical report. Dept. of Statistics, Stanford University, 1999.
7. Friedman J. H. Stochastic Gradient Boosting. Technical report. Dept. of Statistics, Stanford University, 1999.
8. Geurts P., Ernst D., Wehenkel L. Extremely Randomized Trees // Machine Learning. 2006. V. 36, № 1. P. 3–42.
9. Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning. Springer, 2008.
10. Intel Threading Building Blocks.
URL: http://www.threadingbuildingblocks.org (дата обращения: 07.12.2010).
11. OpenCV Wiki.
URL: http://opencv.willowgarage.com/wiki (дата обращения: 07.12.2010).
12. UCI Machine Learning Repository.
URL: http://archive.ics.uci.edu/ml (дата обращения: 07.12.2010).

13. Вапник В. Н. Восстановление зависимостей по эмпирическим данным. М.: Наука, 1979. 448 с.

Имитационное стохастическое моделирование процессов высокоскоростной механической обработки (на примере шлифования)

А.А. Дьяконов[1], А.В. Лепихов[2]

ГОУ ВПО НИУ Южно-Уральский государственный университет[1],
ОАО ГРЦ им. В.П. Макеева[2]

В работе приведены результаты стохастического имитационного моделирования процесса шлифования на основе технологий параллельных вычислений. Структурно программа разбита на три блока, каждый из данных блоков представляет собой набор вложенных циклов (глубина вложенности от 2 до 4). При этом циклы, начиная с глубины 2 допускают эквивалентное преобразование к виду, содержащему независимые итерации. Цикл верхнего уровня допускает распараллеливание с условием синхронизации входных данных в начале каждой итерации. В параллельной реализации программного комплекса использована комбинация технологий MPI и OpenMP. Технология OpenMP используется при декомпозиции циклов нижнего уровня. Эффективность распараллеливания на данном уровне определяется размером кэш-памяти ядра и скорости доступа к оперативной памяти. Технология MPI используется для распараллеливания циклов верхнего уровня и применяется для расчетов с высокой степенью точности.

1. Введение

В настоящее время в машиностроении (станкостроении) происходят коренные преобразования, связанные с выпуском совершенно нового оборудования для абразивной обработки и интенсификации скоростей приводов рабочих движений станка, а следовательно, происходит переход от обычных традиционных режимов резания к скоростным и сверхскоростным режимам обработки. С другой стороны, для современного машиностроения характерен переход к использованию большой номенклатуры новых материалов (композиционные, полимерные, наномодифицированные, градиентные и сложноструктурные дисперсные материалы и т. д.), являющихся эффективными заменителями дорогостоящих.

В связи с этим сложилась парадоксальная ситуация – существует огромная номенклатура современных станков и новых материалов при отсутствии каких-либо режимно-инструментальных рекомендаций по проектированию технологического процесса.

С другой стороны, т.к. многие новые материалы, как обрабатываемые, так и абразивные, создаются на основе высокотехнологичных производств, то в принципе их механические, теплофизические и другие свойства известны. Однако остается открытым вопрос о их контактном силовом взаимодействии, т.е. явления, происходящие, в нашем случае, при резании, что является фундаментальной основой для разработки различных режимно-инструментальных рекомендаций. Традиционные эмпирические подходы к разработке таких рекомендаций в настоящее время являются неработоспособными, т.к. отсутствует производственный статистический материал, существенное изменение технологических задач при проектировании процесса обработки и т. д.

Как показывает отечественный и зарубежный опыт, самым эффективным решением подобных задач в свете стремительного увеличения производительности современных вычислительных систем, является разработка математической модели или системы моделей рассматриваемого процесса механической обработки и реализации посредством имитационного моделирования. Однако для получения репрезентативного результата необходима модель, максимально отражающая физику рассматриваемого процесса, т.е. возникает необходимость систематики и разработки обобщенной модели процесса, справедливой для каждого из более чем 50 видов абразивной обработки. Особенностью процессов абразивной обработки является быстропротекаемость (время единичного воздействия составляет порядка 10^{-5}–10^{-8} с) и наличие внутренней

существенной нелинейности прочностных свойств обрабатываемого материала от температурно-скоростных характеристик процесса. При этом реализацию существенно усложняет главная особенность процесса абразивной обработки – стохастический характер взаимодействия режущего профиля абразивного инструмента с обрабатываемой поверхностью заготовки [1, 2]. Поэтому реализация данного проекта возможна лишь на основе применения технологий параллельного программирования и применения высокопроизводительных кластеров.

2. Математическая постановка задачи

Решение задачи режимно-инструментального оснащения современного оборудования и разработки рекомендаций по обработке новых материалов проводится посредством имитационного стохастического моделирования, базирующегося на теплофизической модели процесса абразивной обработки.

Расчетная схема теплофизической модели представлена на **Рис. 1**. На детали, являющейся полупространством, зафиксирована неподвижная система координат XYZ (система координат станка). Активизация тепловых источников – случайно расположенных абразивных зерен происходит в области пятна контакта, ограниченного по длине отрезком OL и ширине отрезком OB. Развертка круга с расположенными на ней абразивными зернами движется со скоростью $V_к$, а деталь (полупространство) с соответствующей скоростью $V_д$. При этом для учета физики взаимодействия видов с продольной подачей введены две зоны резания.

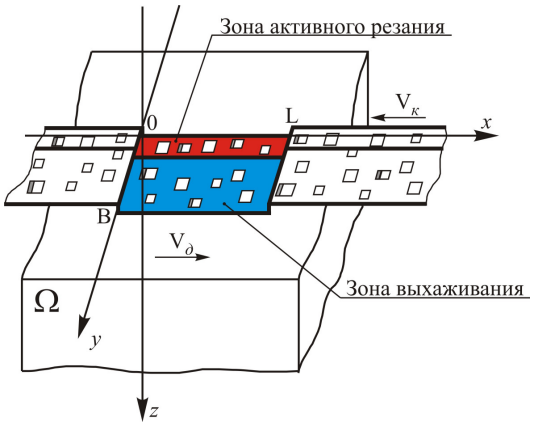


Рис. 1. Расчетная схема теплофизической модели процессов абразивной обработки

Пологая, что каждое зерно на пятне контакта является источником тепла интенсивности *q*, придем к следующей математической формулировке задачи:

$$\begin{cases} c(U)U_t + \vec{V}_д\,\overrightarrow{grad U} = div\left(\lambda(U)\,\overrightarrow{grad U}\right) \\ \lambda(U)U_z\big|_{z=0} = -q(x,y,t), \end{cases} \quad \begin{Bmatrix} -\infty < x < \infty \\ -\infty < y < \infty \\ z > 0 \end{Bmatrix}, \tag{1}$$

где *c(U)* – весовая теплоемкость материала; $\vec{V}_д$ – вектор скорости детали; λ – теплопроводность; *q(x,y,t)* – интенсивность теплового источника с координатами *x*, *y* в момент времени *t*.

Это вторая краевая задача для уравнения теплопроводности в полупространстве с конвективным членом.

Принимая во внимание то, что существенное изменение температурного поля происходит на разгонном участке, который составляет порядка 10–15% от общей длины контакта, можно принять допущения, что теплофизические свойства материала являются постоянными внутри из каждой рассматриваемой группы.

В итоге получим систему (2), являющейся линейной постановкой системы уравнений (1).

$$\begin{cases} \dfrac{\partial U}{\partial t} + V_o \dfrac{\partial U}{\partial x} = \chi\left(\dfrac{\partial^2 U}{\partial x^2} + \dfrac{\partial^2 U}{\partial y^2} + \dfrac{\partial^2 U}{\partial z^2} \right) & \begin{Bmatrix} -\infty < x < \infty \\ -\infty < y < \infty \\ z > 0 \end{Bmatrix}, \\ \dfrac{\partial U}{\partial z} = -\dfrac{1}{\lambda} q(x, y, t) & \begin{Bmatrix} 0 < x < L \\ 0 < y < B \end{Bmatrix}. \end{cases} \quad (2)$$

где χ – коэффициент температуропроводности.

Таким образом, имеем модель, обобщающую идеи двух разноплановых теплофизических направлений в абразивной обработке, т.е. рассматривается сплошная область контакта, по которой перемещаются тепловые источники, интенсивность тепловыделения которых определяется исходя из температурно-скоростных прочностных свойств материала.

2.1 Интегральное решение теплофизической модели

Математическое описание трехмерного температурного поля в зоне шлифования при полном учете кинематики процесса – скорости круга, заготовки, скоростей подачи, а также представлении процесса абразивной обработки, как процесса множественного микрорезания абразивными зернами приводит ко второй краевой задаче для уравнения теплопроводности в среде, движущейся со скоростью детали, и стохастическим множеством тепловых источников, пробегающих зону контакта круга с заготовкой со скоростью круга.

Рассмотрим действие одного теплового источника – абразивного зерна в начальный момент времени t=0 (**Рис. 2**). Положение зерна в пространстве будет описываться соответствующими координатами по оси абсцисс и ординат.

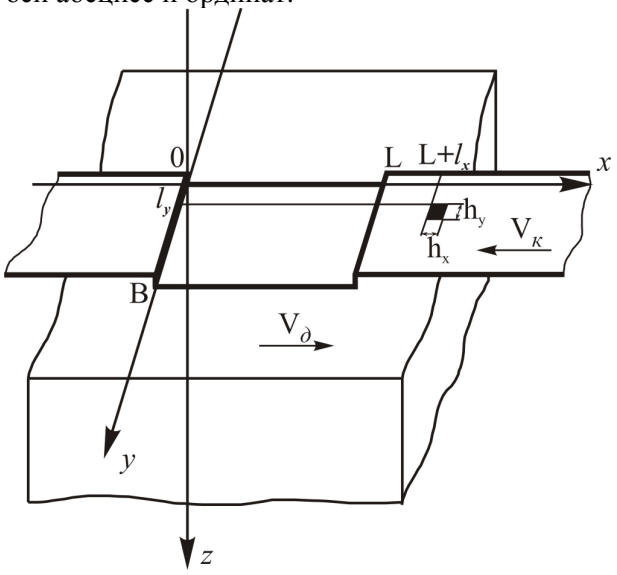


Рис. 2. Расчетная схема теплофизической модели от действия единичного источника тепла – абразивного зерна

В первом приближении форма зоны шлифования принята в виде прямоугольника с соответствующими границами. Такой вид она имеет для схем: круглое наружное врезное шлифование, шлифование отверстий, бесцентровое, плоское шлифование периферией круга.

В результате имеем вторую краевую задачу для уравнения теплопроводности в полупространстве с конвективным членом, для которой отсутствуют известные решения.

Из специальных разделов уравнений математической физики известны разнообразные методы, позволяющие отойти от конвективного члена в уравнении теплопроводности. Это методы интегрального и итерационного преобразования, применение функций Грина, Фурье и т. д.

Однако анализ предложенной расчетной схемы показал, что перевод задачи в подвижную систему координат, связанную с заготовкой, позволяет построить интегральное решение задачи в виде свертки функции интенсивности теплового источника с функцией Грина для полупространства. Функция интенсивности источника является кусочно-аналитической (**Рис. 3**):

$$q(x,y,t)=\begin{cases} \text{q} & \text{– в заштрихованных областях;} \\ 0 & \text{– вне областей.} \end{cases}$$

Вследствие такой составной структуры функции q (x, y, t) целесообразно введение элемента решения – функции влияния отдельного теплового источника, которая формируется как тройной интеграл по отдельной заштрихованной области, трансформирующийся после описания границ области в трехкратный (4).

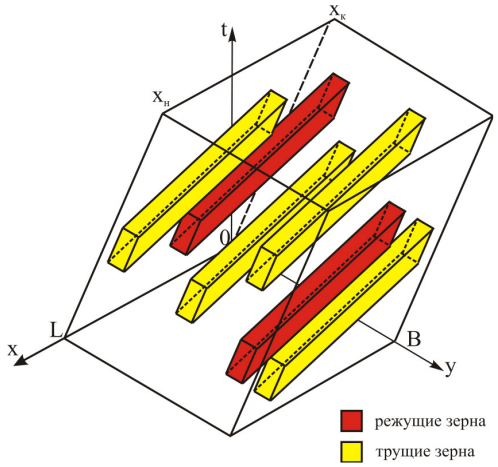


Рис. 3. Фазовый портрет множественного источника

$$U=\frac{q}{\sqrt{(4\pi\chi)^3}}\iiint_{x,y,z} e^{-\frac{z^2+(y-y`)^2+[(x-x`)-V_o(t-t`)]^2}{4\chi(t-t`)}}\frac{dt`dx`dy`}{\sqrt{(t-t`)^3}}. \qquad (4)$$

В зоне шлифования тепловой источник перемещается со скоростью V_k (см. **Рис. 2**), тогда закон его перемещения можно записать следующим образом:

– начало действия $x_{\text{п}}=L-h_x+l_x-(V_k+V_{\text{д}})t,$

– окончание действия $x_{\text{з}}=L+l_x-(V_k+V_{\text{д}})t.$

Структура подинтегральной функции и функций в пределах интегрирования в решении (4) такова, что получается естественный порядок рассмотрения однократных интегралов. Переменные $x`$ и $y`$ независимы друг от друга, и интегрирование по ним можно вести в первую очередь, затем рассматривается объединяющий все переменные интеграл по t`.

Пределы интегрирования по y` постоянны, поэтому интеграл по этой переменной выражается непосредственно через известные функции $\text{erf}(\varsigma)$.

Сложнее обстоит дело в интеграле по переменной $x`$. Здесь пределы интегрирования зависят от времени t и поскольку интеграл рассматривается для текущего времени, область интегрирования может принимать три разных формы (**Рис. 4**):

– $t_{\text{н}}^{\text{п}} \le t \le t_{\text{н}}^{3}$ – вход зерна-источника в зону шлифования, область интегрирования – треугольник AB^1C^1 – зона I;

– $t_{\text{н}}^{3} \le t \le t_{\text{к}}^{\text{п}}$ – работа зерна-источника в зоне шлифования, область – параллелограмм $BCD^{11}E^{11}$ – зона II;

– $t_{\text{к}}^{\text{п}} \le t \le t_{\text{к}}^{\text{з}}$ – выход зерна-источника из зоны шлифования, область интегрирования – трапеция $\text{DED}^{111}\text{E}^{111}$ – зона III.

Области I и III более сложны для интегрирования, поскольку ширина области переменна и является функцией времени. В области II ширина постоянна и не зависит от времени, т.е. интеграл существенно упрощается. Учитывая, что зона входа (I) и выхода (III) зерна на порядок меньше зоны его работы (II), предлагается пренебречь этими нестационарными зонами и ограничить рассмотрение интеграла (2) зоной I. Механически это допущение означает, что зерно мгновенно попадает в зону шлифования и также мгновенно из нее исчезает. Это математическое допущение даже более реалистично с точки зрения теплофизики процесса. При такой трактовке закономерностей работы теплового источника его фазовый портрет описывается лишь областью с двойной штриховкой – параллелепипед BCDE (см. **Рис. 4**).

Для модифицированного фазового портрета теплового источника интегрирование по $x`$ проводится аналогично интегрированию по $y`$.

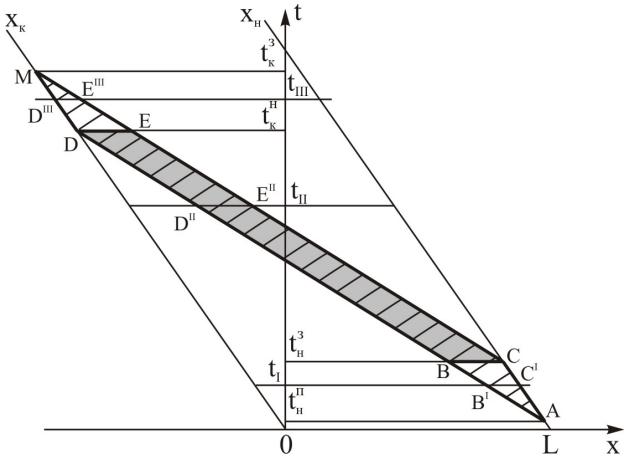


Рис. 4. Фазовый портрет единичного теплового источника в системе координат заготовки

В итоге трехкратный интеграл (4) сводится к однократному по переменной $t`$:

$$U = 2q\int_{t_{\text{н}}}^{t} \text{e}^{-\frac{z^2}{4\chi(t-t`)}} \left[ \text{erf}\,\frac{|y-l_y|}{\sqrt{4\chi(t-t`)}} - \text{erf}\,\frac{|y-l_y-h_y|}{\sqrt{4\chi(t-t`)}} \right] \times$$

$$\times \left[\text{erf}\,\frac{|X-L-l_{\text{x}}+h_{\text{x}}+wt|}{\sqrt{4\chi(t-t`)}} - \text{erf}\,\frac{|X-L-l_{\text{x}}+wt|}{\sqrt{4\chi(t-t`)}} \right] \times \tag{5}$$

$$\frac{4\pi\chi(t-t`)}{4} \frac{\text{d}t`}{\left(\sqrt{4\pi\chi(t-t`)}\right)^3} = \frac{1}{2} q\int_{t_{\text{н}}}^{t} \text{e}^{-\frac{z^2}{4\chi(t-t`)}} \left[\text{erf}\,\frac{|y-l_y|}{\sqrt{4\chi(t-t`)}} - \text{erf}\,\frac{|y-l_y-h_y|}{\sqrt{4\chi(t-t`)}} \right] \times$$

$$\times \left[\text{erf}\,\frac{|X-L-l_{\text{x}}+h_{\text{x}}+wt|}{\sqrt{4\chi(t-t`)}} - \text{erf}\,\frac{|X-L-l_{\text{x}}+wt|}{\sqrt{4\chi(t-t`)}} \right] \frac{\text{d}t`}{\sqrt{4\pi\chi(t-t`)}}.$$

Этот интеграл является существенно несобственным – верхний предел $t`=t$ образует особую точку для всех составляющих подинтегральной функции. Однако функциональное преобразование:

$$\sqrt{t-t`} = \xi$$

трансформирует его к виду (6), где особенность сохраняется лишь в аргументе функции $\mathrm{erf}(\xi)$, в нижнем пределе.

$$U = \frac{2q}{\sqrt{\pi\chi}} \int_0^{\sqrt{t-t_0}} \mathrm{e}^{-\frac{z^2\xi^2}{4\chi}} \left[\mathrm{erf} \frac{X\text{-}L\text{-}l_x + V_д \left(t\text{-}\xi^2\right)}{\sqrt{4\chi}\xi} - \mathrm{erf} \frac{X\text{-}L\text{-}l_x + h_x + V_д \left(t\text{-}\xi^2\right)}{\sqrt{4\chi}\xi} \right] \times$$

$$\times \left[\mathrm{erf} \frac{y\text{-}l_y}{\sqrt{4\chi}\xi} - \mathrm{erf} \frac{y\text{-}l_y\text{-}h_y}{\sqrt{4\chi}\xi} \right] \mathrm{d}\xi, \qquad (6)$$

Суммирование (6) по всем источникам позволяет сформировать многокритериальную трехмерную теплофизическую модель процессов абразивной обработки:

$$\mathrm{U} = \sum_{\mathrm{i}} \frac{2q\left(\mathrm{U}_{\mathrm{i-1}}\right)}{\sqrt{\pi\chi}} \int_0^{\sqrt{t-t_0}} e^{-\frac{z^2\xi^2}{4\chi}} \left[\mathrm{erf} \frac{X\text{-}L\text{-}l_{xi} + V_д \left(t_i\text{-}\xi^2\right)}{\sqrt{4\chi}\xi} - \right.$$

$$\left. - \mathrm{erf} \frac{X\text{-}L\text{-}l_{xi} + h_{xi} + V_д \left(t_i\text{-}\xi^2\right)}{\sqrt{4\chi}\xi} \right] \left[\mathrm{erf} \frac{y\text{-}l_{yi}}{\sqrt{4\chi}\xi} - \mathrm{erf} \frac{y\text{-}l_{yi}\text{-}h_{yi}}{\sqrt{4\chi}\xi} \right] \mathrm{d}\xi.$$

3. Структура программного комплекса

Разработанная модель, численно реализованная в виде программы «Пространственная многокритериальная теплофизическая модель процессов обработки» [3], позволяет расчетным способом спрогнозировать основные выходные показатели – точность, качество обработки, оценить влияние характеристики абразивного инструмента и т. д.

Отработка этой модели показала, что для полученных результатов расчета характерно наличие значительной дисперсии. В силу центральной предельной теоремы теории вероятности для наиболее значимой оценки генеральной совокупности в 6σ интервале необходимо рассматривать порядка 200 значений случайной величины. А, принимая во внимание, что результатом расчета является случайная функция, а не величина, то необходимо рассматривать еще большие выборки.

Решение этой задачи на обычных персональных компьютерах является практически не выполнимой, т.к. требует колоссальных затрат времени – расчет одной точки эксперимента может занимать от нескольких часов до двух десятков дней.

Такую трудоемкую и вычислительно-сложную задачу в настоящее время можно решить только посредством мультипроцессорных кластеров и технологии параллельного программирования [4]. В настоящее время совместно с кафедрой «Системного программирования» разработана тестовая версия распараллеленной программы, обобщенный алгоритм которой представлен на **Рис. 5**.

Одну реализацию расчета (редукцию) можно представить следующим образом.

Стандартный алгоритм программы состоит из 7 основных подсистем. По исходным данным (подсистема 1) формируется вероятностное строение рабочей поверхности инструмента (подсистема 2), далее проводится статистическая проверка распределения размеров зерен и их пространственного расположения (подсистема 3), при этом, если результат проверки статистически незначим, алгоритм программы возвращается в подсистему 1и производится аналогичная процедура. Сформированная развертка передается в подсистему 4, где на основе нелинейной теплофизической модели, базы данных теплофизических констант и температурно-скоростных прочностных характеристик обрабатываемого материала производится расчет температурных полей.

Данные по температурам передаются в подсистему 5, и на основе математической модели радиальной составляющей силы резания, рассчитывается суммарное усилие в зоне абразивной обработки для каждого рассматриваемого момента времени.

Полученные массивы температур и радиальной составляющей силы резания передаются в подсистему 7, где производится их оценка по двум центральным моментам математической статистики – математическому ожиданию и дисперсии.

Анализ структуры алгоритма с позиций технологии параллельного программирования показал, что его иерархически-модульная структура и последовательность работы естественным образом вписывается в данную технологию.

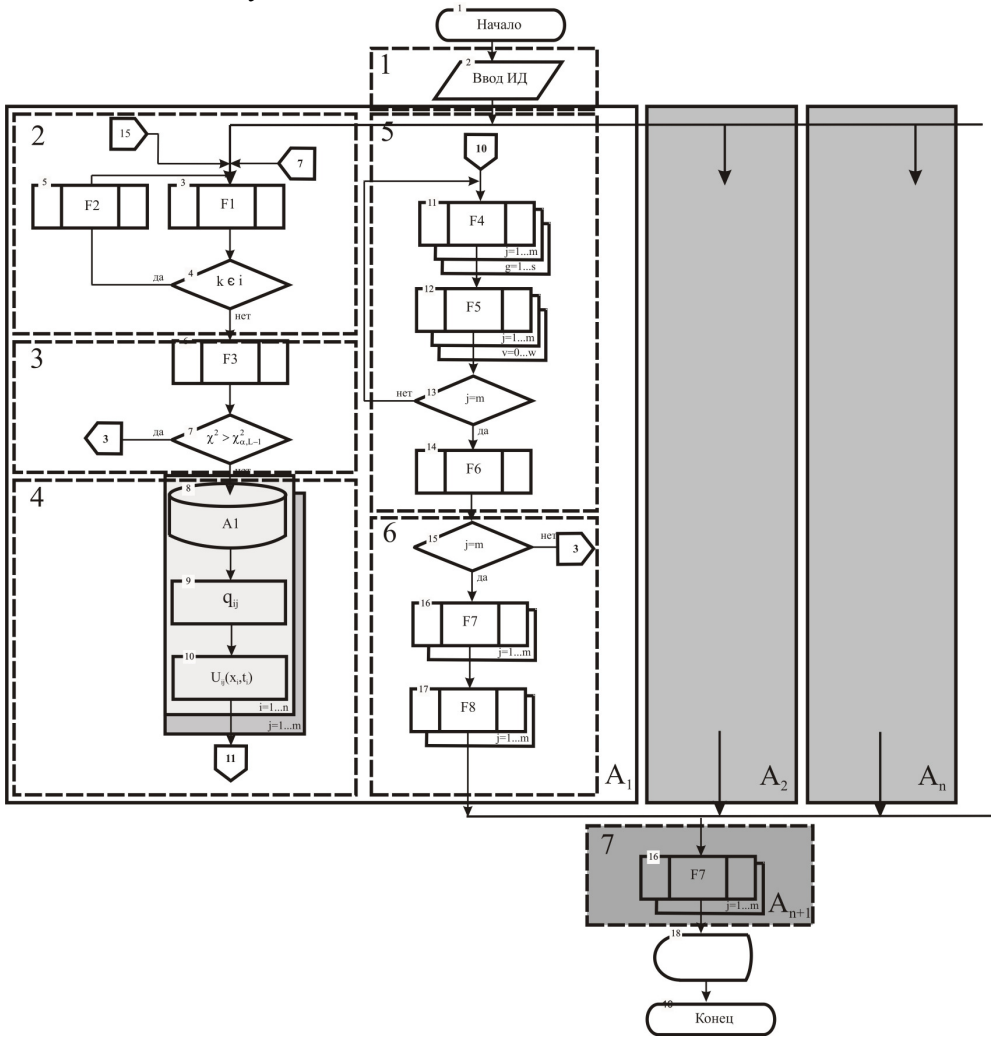


Рис. 5. Обобщенный алгоритм имитационной стохастической модели процессов абразивной обработки

4. Параллельные алгоритмы

Процесс анализа исходного кода последовательной версии программного комплекса проходил в два этапа:

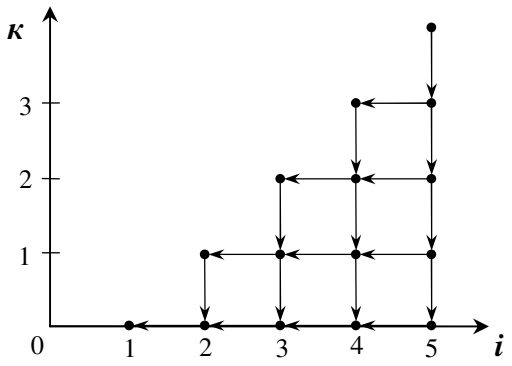
1. Выявление наиболее ресурсоемких участков кода программы.
2. Редуцирование исходного кода.

Под *редуцированием* в данной работе понимается создание модели информационных зависимостей участка кода (обычно цикла) следующим образом:

- группировкой последовательно выполняемых участков кода в блоки.
- удалением операторов, не имеющих зависимостей в редуцируемом коде.

Анализ исходного кода и практика использования программного комплекса показали, что наибольшее время работы программного комплекса (до 99%) затрачивается на выполнение следующих операций (**Рис. 6**).

```
for (k = 0; k < N; k++) {
  for (i = k+1; i < N; i++) {
    x = F1(A[k], A[i]);
    B[k] += F2(x);
    B[i] += F3(x);
  }
  A[k] = F4(B[k]);
}
```



а *б*

Рис. 6. Схема информационных зависимостей алгоритма формирования режущей части инструмента

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. После первичного построения режущей части возникает необходимость в корректировке – вторичное распределение зерен с учетом их случайных размеров, т.к. в некоторых случаях возникает наложение зерен вследствие первоначального распределения центров зерен, что противоречит технологии изготовления абразивного инструмента. 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.

На первом этапе распараллеливания программного комплекса был выполнен процесс редукции алгоритма формирования режущей части инструмента, который привел к схеме информационных зависимостей, показанной на **Рис. 6а**. Здесь и далее *A[]* и *B[]* – массивы, состоящие из вещественных элементов, $F_1,...,F_4$ – ресурсоемкие преобразования, зависящие по данным от значения переменной *x*. На **Рис. 6б** пространство итераций алгоритма представлено в соответствии с методом координат [5]. В результате анализа зависимостей пространства итераций, в соответствии с [4] было выполнено преобразование алгоритма путем замены счетчиков циклов следующим образом:

$$\begin{cases} \boldsymbol{i} = \dfrac{\boldsymbol{N}}{2} + \boldsymbol{s} + \boldsymbol{j} + 1 \\ \boldsymbol{k} = \dfrac{\boldsymbol{N}}{2} + \boldsymbol{s} - \boldsymbol{j} \end{cases}, \boldsymbol{s} \in \left[-\frac{\boldsymbol{N}}{2}, \ldots, \frac{\boldsymbol{N}}{2} \right]; \boldsymbol{j} \in \left[0, \ldots, \frac{\boldsymbol{N}}{2} - \boldsymbol{s} \right].$$

Схема полученного в результате преобразований алгоритма показана на **Рис. 7**. Данный алгоритм может быть эффективно распараллелен на одном вычислительном узле при помощи директив OpenMP.

На втором этапе распараллеливания была выполнена редукция алгоритма расчета интенсивности тепловыделения режущих зерен. На **Рис. 8** показана схема информационных зависимостей данного алгоритма. Как можно заметить, итерации внешнего цикла сильно связаны между собой по данным. Однако итерация состоит из двух независимых блоков кода. Распараллеливание выполняется путем расщепления итерации внешнего цикла на две OpenMP-секции.

На последнем, третьем этапе выполнен анализ и распараллеливание наиболее трудоемкой операции расчета распределения поля температур по глубине поверхностного слоя. Время выполнения данной операции на персональном компьютере существенно зависит от постановки задачи и варьируется в диапазоне от нескольких минут – для простых задач – до недель, для задач с большим количеством зерен и мелким шагом по времени. Редуцированный алгоритм данной операции показан на **Рис. 9**.

```
for (s = -N/2; s < N/2; s++) {
  for (j = 0; j < N/2 - s; j++) {
    x = F1(A[N/2 + s - j k], A[N/2 + s + j + 1]);
    B[N/2 + s - j] += F2(x);
    B[N/2 + s + j + 1] += F3(x);
  }
}
for (k = 0; k < N; k++)
  A[k] = F4(B[k]);
```

Рис. 7. Схема модифицированного алгоритма формирования режущей части инструмента

```
for (k = 0; k < N; k++) {
  for (i = 0; i < k; i++)
    x += F1(A[i]);
  A[j] = F2(x);

  for (i = 0; i < k; i++)
    u += F3(B[i]);
  B[j] = F4(u);
}
```

Рис. 8. Схема информационных зависимостей алгоритма расчета интенсивности тепловыделения режущих зерен

```
for (z = 0; z < LAYERS_NUM; z++)
  for (i = 0; i < Y_POINTS; i++)
    for (j = 0; j < X_POINTS; j++)
      for (k = 0; k < N; k++) {
        u += F1(A[z], B[i], C[j], D[k]);
        saveToDisk(F2(u));
}
```

Рис. 9. Схема информационных зависимостей алгоритма расчета распределения поля температур по глубине поверхностного слоя

Как можно заметить, распараллеливание алгоритма может быть выполнено на произвольном количестве вычислительных узлов. Существенным фактором, влияющим на эффективность распараллеливания, является необходимость дублирования массивов *A, B, C* и *D* на вычислительных узлах и необходимость их синхронизации после завершения этапа расчета температур по глубине изделия.

Параллельный алгоритм расчета поля температур на базе технологии MPI можно описать следующей последовательностью шагов. Пусть для расчета выделено *n* вычислительных узлов. На каждом узле выполняется отдельный процесс программного комплекса. На первом шаге процесс P_0, расположенный на нулевом узле, выполняет формирование режущей поверхности инструмента. На втором шаге процесс P_0 выполняет доставку подготовленной на шаге 1 режущей поверхности каждому процессу. На третьем шаге каждый процесс выполняет расчет интенсивности тепловыделения зерен. На четвертом шаге процессу назначается набор итераций для расчета распределения поля температур в соответствии с правилом:

$$m = (LAYERS\_NUM*Y\_POINTS*X\_POINTS)/n.$$

где *m* – количество итераций алгоритма. Процесс выполняет расчет поля температур по глубине. На пятом шаге процесс P_0 выполняет синхронизацию результатов.

Технология MPI используется для распараллеливания обработки итераций циклов трех верхних уровней. Для параллельного выполнения цикла нижнего уровня используется технология OpenMP. Особенностью предложенного параллельного решения является дублирование операции расчета интенсивности тепловыделения зерен, выполняемой на третьем шаге, что позволяет существенно снизить накладные расходы на обмен данными по сети и уменьшить общее время работы программного комплекса.

В настоящее время проводится исследование характеристик ускорения и расширяемости программного комплекса на базе вычислительного кластера Южно-Уральского государственного университета «Скиф Урал». Получаемые результаты показывают значительное ускорение времени счета. Например, реализация 500 редукций при средних технологических входных параметрах круглого центрового шлифования с радиальной подачей ориентировочно занимает в

среднем 0,5–1,2 минуты, против 520 минут для 50 редукций на персональном компьютере высокой производительности.

5. Область применения результатов

Массовая реализация готового продукта «Рекомендации по режимно-инструментальному оснащению современных абразивных станков и обработки новых материалов» планируется в трех направлениях:

1. Разработка справочника нового поколения для режимно-инструментального оснащения существующих в настоящее время 80 видов операций абразивной обработки. Учитывая, что последнее издание подобного материала – «Общемашиностроительные нормативы режимов резания и норм времени для шлифовальных работ» было выпущено в 1976 г., настоящий справочник представляет особую практическую ценность и имеет большие коммерческие перспективы. Подтверждением этому является то, что по заказам машиностроительных предприятий Российской Федерации авторами проекта совместно с ОАО «Уральский НИИ абразивов и шлифования» в 2005–2007 году разработаны:

– руководящий технический материал «Проектирование операций шлифования. Выбор характеристики шлифовального круга, назначение режимов резания, определение норм времени»;

– укрупненный справочник «Режимы резания на работы, выполняемые на шлифовальных и доводочных станках с ручным управлением и полуавтоматах».

Данные материалы внедрены в действующее производство на более чем 200 предприятиях, в числе которых флагманы аэрокосмической, автомобилестроительной и машиностроительной отрасли – ФГУП ГНПРКЦ «ЦСКБ-ПРОГРЕСС», ОАО «ВОСТСИБМАШ», группа «ГАЗ» и др.

2. Разработка программной оболочки для интеграции в CAD\CAM\CAPP системы. Основное направление данной разработки – реализация автоматизированного выбора характеристик абразивного инструмента, расчета режимов резания и норм времени. Необходимость реализации данной задачи вызвана тем, что в настоящее время в современных САПР ТП данные процедуры выполняются на основе чтения баз данных, занесенных из общемашиностроительных нормативов резания, которые в свою очередь являются укрупненными. Проведенный анализ показал, что такая методика приводит зачастую к существенному занижению режимов резания, что в итоге приводит к потерям в производительности от 20 до 65%.

В этом направлении также имеются большие коммерческие перспективы в виде сотрудничества с отечественным лидером в области САПР ТП – группы компаний «Omega-Technologics» – программный продукт ADEM CAD\CAM\CAPP.

3. Реализация программы для научно-исследовательских работ и решения проектных технологических задач режимно-инструментального оснащения операций абразивной обработки. Основное направление – проведение хоздоговорных работ, связанных с решением проблем внедрения и применения новых абразивных и обрабатываемых материалов, а также разработки режимных рекомендаций для сверхскоростных условий обработки (>150 м/с).

Планируется проводить также проектные работы по освоению и внедрению комбинированных методов механической обработки – обработка с предварительным нагревом, принужденным силовым и электрохимическим воздействием и т. д.

Заключение

Описан программный комплекс для решения задачи имитационного стохастического моделирования процессов высокоскоростной механической обработки. Представлена архитектура и основные подсистемы программного комплекса. Выполнен анализ возможности параллельной реализации программного комплекса. Представлены параллельные алгоритмы, разработанные в процессе его реализации. Выполнен анализ сфер применения программного комплекса и дальнейшие направления его развития.

В настоящее время выполняется анализ характеристик ускорения и расширяемости разработанной параллельной реализации, проводятся тестовые запуски программного комплекса на кластере ЮУрГУ «Скиф Урал».

В качестве направлений дальнейших исследований можно выделить решение следующих основных задач:

1. Анализ и модификация исходного (последовательного) алгоритма расчета интенсивности тепловыделения зерен с целью увеличения ресурса параллелизма.
2. Разработка системы структурирования и буферизации данных, выдаваемых программным комплексом в параллельном режиме, с целью снижения накладных расходов на взаимодействие с диском.
3. Разработка методики редуцирования исходного программы кода до схемы информационных зависимостей.

Литература

1. Дьяконов А.А. Стохастический подход к решению теплофизических и силовых задач теории шлифования // Металлообработка, 2008. №2(44). С.2–6.
2. Dyakonov A.A. Application area of fast-moving sources theory in thermophysics of abrasive machining tasks // Proceedings of the Chelyabinsk Scientific Center, 2010. Vol. 1 (47). P. 31–34.
3. Свидетельство о государственной регистрации программ для ЭВМ 2010610052 Российская Федерация. Пространственная многокритериальная теплофизическая модель процессов абразивной обработки / А.А. Дьяконов, А.В. Геренштейн, А.А. Кошин. – № 2009616027; заявл. 28.10.2009; зарегистр. 11.05.2010.
4. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2004. 608 с.
5. *Lamport L.* The coordinate method for the parallel execution of DO-loops. // Commun. ACM, Vol. 17, Issue 2. 1974. P. 83-93.

Компетентностный подход в обучении суперкомпьютерным технологиям

А.С. Евдокимова, Н.С. Силкина

Южно-Уральский Государственный университет

В настоящее время активно осуществляется переход системы российского образования на государственные образовательные стандарты третьего поколения, в которых требования к дисциплинам определяются с помощью компетенций. В данной статье представлена модель эГОС-3 и ее использование на примере обучения современным параллельным вычислительным технологиям.

1. Введение

В изменяющихся условиях рынка труда важно обеспечить мобильность выпускника. Это обеспечивает переход от квалификационной к компетентностной модели специалиста, ориентированной на сферу профессиональной деятельности, менее жестко привязанной к конкретному объекту и предмету труда. Европейские университеты уже более 30 лет осваивают компетентностный подход в образовании в рамках Болонского процесса [1]. На сегодняшний день разработаны международные стандарты, спецификации и рабочие соглашения. Наиболее популярными являются:

– спецификация «Reusable Definition of Competency or Educational Objective» [2], разработанная промышленным консорциумом IMS GLS (Instructional Management System Global Learning Consortium);
– стандарт IEEE 1484.20 «Standard for Learning Technology – Data Model for Reusable Competency Definitions» [3], разработанный комитетом Learning Technology Standards Committee IEEE;
– спецификация «Competencies (Measurable Characteristics) Recommendation» [4], разработанная консорциумом HR-XML Consortium;
– рабочее соглашение «A European Model for Learner Competencies» [5], принятое комитетом CEN (The European Committee for Standardization).

Россия присоединилась к Болонскому процессу в сентябре 2003. По поручению коллегии Министерства образования и науки от 1 февраля 2007 года «О разработке нового поколения государственных образовательных стандартов и поэтапном переходе на уровневое высшее профессиональное образование с учетом требований рынка труда и международных тенденций развития высшего образования» [6] Федеральным институтом развития образования (ФИРО) разработаны в пакете с законопроектом «Макет Федерального государственного образовательного стандарта (ФГОС) третьего поколения» [7]. В настоящее время уже утверждено достаточно много государственных образовательных стандартов (ГОС ВПО) третьего поколения, в высших профессиональных учреждениях осуществляется переход на стандарты нового поколения. Основой ГОС ВПО третьего поколения является понятие компетенции. Компетенция описывает знания, умения и навыки студентов, определяя требования к дисциплинам ГОС ВПО.

Целью данного исследования является разработка модели компетенции для использования в системе электронного обучения UniCST (Universal System for E-learning) [8]. Система UniCST реализует структурно-иерархическую дидактическую модель содержания электронного учебного комплекса, разработанную на кафедре системного программирования ЮУрГУ. Система предназначена для создания электронных учебных энциклопедий различных областей знаний и электронных учебных курсов по различным учебным дисциплинам, а также для организации и проведения компьютерного тестирования. Данный программный продукт может быть использован при чтении лекций, организации семинаров, зачетов, экзаменов и других учебных или контрольных мероприятий.

2. Структурно-иерархическая дидактическая модель

Структура модели дидактического содержания электронного учебного комплекса подробно описана в работе [8]. В данном разделе приведем лишь краткое описание. На Рис. 1 схематично изображена данная модель, в соответствии с которой учебно-методический материал делится на четыре уровня (слоя). На верхнем (четвертом) уровне находится ГОС ВПО, на базе которого разрабатываются рабочие программы учебного комплекса. Все используемые образовательные стандарты хранятся в базе данных в структурированном виде. Рабочие учебные программы (РУП), входящие в учебный комплекс, образуют третий уровень. Они представляются в виде иерархических структур, называемых граф-планами. На нижнем (первом) уровне находятся электронные учебные энциклопедии (ЭУЭ) по различным областям знаний. Эти электронные энциклопедии имеют стандартную структуру, которая описана в работе [8]. Электронные учебные энциклопедии создаются известными специалистами в соответствующей области знаний и могут использоваться одновременно в нескольких электронных учебных комплексах. Электронные учебные курсы (ЭУК) по различным дисциплинам образуют второй уровень электронного учебного комплекса. Они создаются на основе соответствующих РУП путем связывания в иерархию модулей (статей) из энциклопедий, описывающих изучаемые понятия [9].

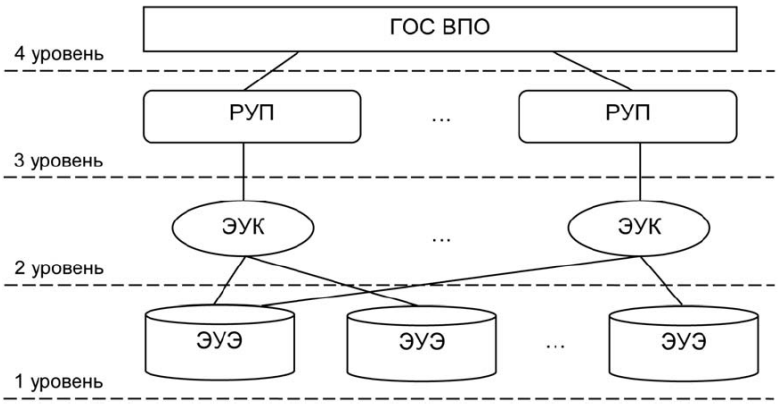


Рис. 1. Структура модели

Для данного исследования наибольший интерес представляет четвертый уровень модели. Рассмотрим его подробно. Каждый ГОС ВПО содержит общую характеристику направления (специальности), требования к условиям реализации образовательной программы, требования к подготовке выпускника, требования к результатам освоения основных образовательных программ и другую информацию. С точки зрения организации электронного обучения, наиболее интересным является то, как определяются требования к основным образовательным программам. ГОС ВПО устанавливает учебные циклы дисциплин. Например, ГСЭ (гуманитарные и социально-экономические дисциплины), ЕН (математические и естественнонаучные дисциплины), ОПД (общепрофессиональные дисциплины направления или специальности), СД (специальные дисциплины), ФТД (факультативные дисциплины). Количество циклов, включаемых в основную образовательную программу, может быть различным в разных стандартах. Каждый учебный цикл имеет базовую (обязательную) часть и вариативную (профильную), устанавливаемую ВУЗом.

Каждая дисциплина идентифицируется уникальным индексом, состоящим из шифра цикла по ГОС ВПО, шифра части (если применимо) и двузначного порядкового номера дисциплины внутри цикла. Для профильных дисциплин может вводиться многоуровневая нумерация. В качестве атрибутов дисциплин базовой части фигурируют требования к обязательному минимуму содержания и общее количество часов, отводимое ГОС ВПО для изучения дисциплины. Требования к обязательному содержанию дисциплины представляется в виде древовидной структуры, узлами которой являются элементы обязательного содержания [8].

3. Модель эГОС-3

На основе стандарта IEEE 1484.20 [3] и макета ГОС ВПО третьего поколения нами была разработана модель ГОС ВПО третьего поколения эГОС-3 (электронный Государственный Образовательный Стандарт третьего поколения). На Рис. 3 изображена структура модели эГОС-3. Данная модель детализирует четвертый и третий уровни структурно-иерахической дидактической модели, описанной во втором разделе.

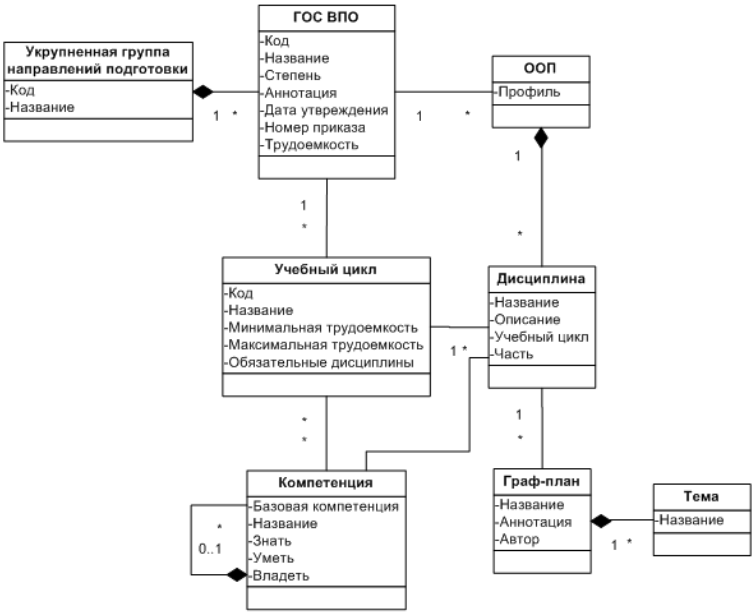


Рис. 2. Модель эГОС-3

Все имеющиеся специальности и направления подготовки разбиваются на *укрупненные группы*. Укрупненные группы представляют собой агрегацию ГОС ВПО для схожих направлений подготовки. Любой стандарт может принадлежать только к одной укрупненной группе, в каждой укрупненной группе может быть неограниченное количество стандартов.

Для каждого направления подготовки определяется и утверждается *ГОС ВПО*. Для стандартов определяется общая трудоемкость. Это значит, что суммарная трудоемкость входящих в образовательную программу дисциплин не должна превышать данного числа.

Каждый ГОС ВПО определяет *учебные циклы* дисциплин. Для каждого учебного цикла стандартом устанавливается набор компетенций, которые учащиеся должны освоить, после изучения всех дисциплин данного цикла. Так же устанавливается список обязательных дисциплин и допустимый диапазон трудоемкости (суммарная трудоемкость дисциплин должна принадлежать данному диапазону).

На основе ГОС ВПО высшими учебными заведениями составляются основные образовательные программы, которые представляют собой совокупность различных документов начиная от учебного плана, заканчивая описаниями учебных *дисциплин*. С точки зрения организации обучения в электронной системе наиболее интересны описания учебных дисциплин. Каждая учебная дисциплина относится к определенному учебному циклу. Учебно-методическим отделом ВУЗа составляется матрица соответствия компетенций, составных частей основной образовательной программы и оценочных средств. На основе данной матрицы можно отследить для каждой дисциплины компетенции, которые они должны реализовывать.

Для каждой учебной дисциплины составляется рабочая программа, которая представляется в виде иерархической древовидной структуры, называемой *граф-планом*. Граф-план представляет собой совокупность упорядоченных *тем*, которые будут пройдены в ходе изучения дисциплины. Электронные учебные курсы составляются путем связывания модулей электронных учебных энциклопедий и тем граф-планов.

Компетенции определяются в терминах знать-уметь-владеть. Нами было принято решение предоставить возможность определения составных компетенций, что не исключает существо-

вания простых компетенций. Если компетенция является составной, то для ее усвоения необходимо овладеть входящими в ее состав компетенциями.

4. Реализация

На основе модели эГОС-3 планируется реализовать подсистему «Редактор стандартов» системы электронного обучения UniCST. Архитектура системы UniCST [8] реализована на основе трехзвенной архитектуры клиент – web-сервер – сервер баз данных. В качестве клиента может использоваться любой стандартный интернет-обозреватель («тонкий» клиент). Для реализации системы UniCST были выбраны технология ASP.Net, ADO.Net, язык программирования C#, СУБД Microsoft SQL Server.

Редактор стандартов должен обеспечивать работу с тремя основными сущностями: ГОС ВПО, основная образовательная программа (ООП) и компетенция. На **Ошибка! Источник ссылки не найден.** изображена архитектура подсистемы «Редактор стандартов». «Редактор стандартов» включает в себя три модуля: «Редактор ГОС ВПО», «Редактор ООП» и «Редактор компетенций».

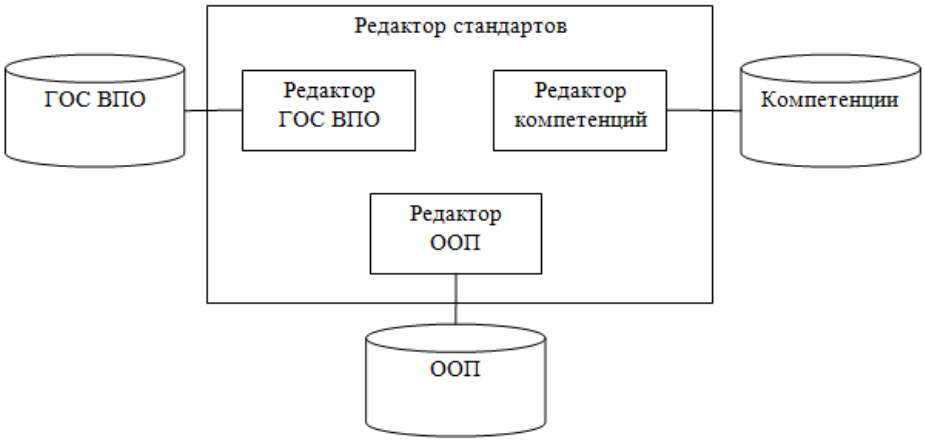


Рис. 3. Архитектура подсистемы «Редактор стандартов»

«Редактор компетенций» предназначен для управления компетенциями. В связи с тем, что большое число компетенций для укрупненных групп специальностей и направлений подготовки будет совпадать, нами было принято решение организовать хранилище компетенций. Для его управления используется «Редактор компетенций». Данный редактор позволяет создавать и редактировать компетенции, на которые впоследствии могут ссылаться различные ГОС ВПО. Так как компетенции UniCST соответствуют стандарту IEEE 1484.20, комплекс компетенций может быть получен извне (передан Министерством образования и науки, Учебно-методическим управлением университета и др.) и импортирован в систему UniCST.

«Редактор ГОС ВПО» позволяет создавать электронные представления ГОС ВПО третьего поколения. Для обеспечения учебного процесса внутри редактора должна предоставляться возможность создания и редактирования электронных представлений ГОС ВПО третьего поколения. Важной составляющей ГОС ВПО являются учебные циклы, поэтому в редактор включена возможность создания и редактирования учебных циклов, входящих в ГОС ВПО. Так же было решено предоставить возможность импорт и экспорт электронных представлений ГОС ВПО.

«Редактор ООП» предназначен для управления основными образовательными программами (создание и редактирование). В состав основных образовательных программ входят учебный план подготовки и рабочие учебные программы дисциплин, входящих в учебный план. Управление учебным планом и граф-планами дисциплин также осуществляет «Редактор ООП».

«Редактор стандартов» также позволяет проводить проверку на соответствие основной образовательной программы государственному образовательному стандарту. Основная образовательная программа должна удовлетворять всем требованиям ГОС ВПО: общая трудоемкость, трудоемкость учебных циклов, реализация компетенций в нужных дисциплинах и др. На основе прошедших проверку основных образовательных программ разрабатываются электронные

учебные курсы. В дальнейшем планируется реализовать возможность проверки электронного учебного курса на соответствие рабочей программе учебной дисциплины, для которой составлялся данный учебный курс.

5. Организация обучения суперкомпьютерным технологиям

Описанная модель позволяет с успехом проводить обучение суперкомпьютерным технологиям. Например, по курсу «Параллельные системы баз данных» (ПСБД). Для организации обучения по данному курсу можно сформировать компетенции, представленные в таблице 1. Столбец «Базовая компетенция» содержит в себе номер компетенции, в состав которой входит компетенция строки. Если в данном столбце значение не заполнено, то компетенция не входит в состав никакой другой компетенции. Пример компетенций составлен на основе программы дисциплины «Параллельные системы баз данных» [11].

Таблица 1. Пример компетенций курса «Параллельные системы баз данных»

№	Базовая компетенция	Название компетенции	Знать	Уметь	Владеть
1	-	Понимание теоретических основ и принципов использования параллельных систем баз данных	Основы технологии построения ПСБД, классификацию архитектур ПСБД, методы параллелизации запросов, различные формы параллелизма	Проектировать ПСБВ, строить параллельный план исполнения SQL-запроса, реализовывать Исполнитель запросов	Средствами и технологиями разработки ПСБД
2	1	Способность понимать и реализовывать на практике формы параллельной обработки транзакций	Межтранзакционный и внутритранзакционный параллелизм, межзапросный и внутризапросный параллелизм, межоперационный и внутриоперационный параллелизм, виды межоперационного параллелизма	Определять степень параллелизма, составлять функцию фрагментации	Терминологическим аппаратом, навыками определения подходящей формы параллелизма и функции фрагментации
3	1	Способность понимать и реализовывать на практике архитектуры многопроцессорных платформ параллельных систем баз данных	Симметричные мультипроцессорные архитектуры (SMP), архитектуры с неоднородным доступом к памяти (NUMA), архитектуры с массовым параллелизмом (MPP) и кластерные архитектуры	Проектировать параллельную систему баз данных	Методами построения параллельных систем баз данных
4	1	Понимание требований параллельным системам баз данных	Понятия масштабируемости, производительности, доступности данных	Оценивать различные конфигурации ПСБД	Технологией оценки ПСБД
5	1	Способность понимать и реализовывать на практике выполнение запросов в ПСБД	Организацию конвейерного параллелизма, организация раздельного (фрагментного) параллелизма, понятие параллельный агент, способ преобразования последовательного плана выполнения запроса в ПСБД	Строить параллельный план исполнения SQL-запроса	Технологией выполнения параллельных запросов в ПСБД

6. Заключение

В данной работе представлена модель ГОС ВПО третьего поколения эГОС-3. Данная модель конкретизирует третий и четвертые уровни структурно-иерархической дидактической модели электронного учебного комплекса. Модель позволяет создавать электронные представления ГОС ВПО третьего поколения, основные образовательные программы и рабочие программы учебных дисциплин.

В настоящее время на основе данной модели ведется работа по реализации подсистемы «Редактор стандартов» системы электронного обучения UniCST. Одним из достоинств подсистемы является возможность автоматической проверки основной образовательной программы на соответствие государственному образовательному стандарту.

Модель эГОС-3 позволит внедрять курсы по суперкомпьютерным технологиям в учебный процесс на базе системы электронного обучения UniCST. Для этого авторами предложен примерный список компетенций для дисциплины «Параллельные системы баз данных».

Дальнейшее развитие работы предполагает разработку и реализацию методов и алгоритмов автоматической проверки электронного учебного курса на соответствие рабочей программе дисциплины, для которой разрабатывался данный курс.

Литература

1. Кривов В.Д., Мамедова Н.А., Крупенков В.В., Мельников А.А. Разработка инновационных подходов к обучению в сфере информационно-аналитического обеспечения деятельности органов государственного управления. М.: Академия Естествознания, 2010.
2. IMS Reusable Definition of Competency or Educational Objective - Information Model // IMS Global Learning Consortium.
3. IEEE P1484.20.1/D8. Draft Standard for Learning Technology – Data Model for Reusable Competency Definitions // IEEE LTSC.
4. Competencies (Measurable Characteristics) Recommendation // HR-XML Consortium, 2006.
5. European Model for Learner Competencies // Comité Européen de Normalisation / Information Society Standardization System (CEN/ISSS), 2006.
6. Решение коллегии Минобрнауки России по вопросу «О разработке нового поколения государственных образовательных стандартов и поэтапном переходе на уровневое высшее профессиональное образование с учетом требований рынка труда и международных тенденций развития высшего образования», 31.01.2007 г.
   URL: http://www.edu.ru/db/portal/spe/3v/310107s.htm (дата обращения: 14.12.2010).
7. Макет Федерального государственного образовательного стандарта (ФГОС) третьего поколения, 22.02.2007 г. URL: http://www.edu.ru/db/portal/spe/3v/220207m.doc (дата обращения: 14.12.2010).
8. Силкина Н.С., Соколинский Л.Б. Система UniCST - универсальная среда электронного обучения // Системы управления и информационные технологии. 2010. № 2. С. 81-86.
9. Жигальская Н.С. Моделирование дидактической структуры электронных учебных комплексов // Вестник Южно-Уральского государственного университета. 2008. № 27. Вып. 2. С. 4-9.
10. Coi J.L., Herder E., Koesling A., Lofi C., Olmedilla D., Papapetrou O., Siberski W. A Model for competence gap analysis // Web Interface and Applications. 2007.
11. Соколинский Л.Б. Программа дисциплины «Параллельные системы баз данных».
    URL: http://hpc-education.ru/files/Sokolinsky01.pdf (дата обращения: 14.12.2010).

Архитектура системы разграничения доступа к ресурсам гетерогенной вычислительной среды на основе контроля виртуальных соединений

В.С. Заборовский, А.А. Лукашин, С.В. Купреенко, В.А. Мулюха

Санкт-Петербургский государственный политехнический университет

В статье представлена модель разграничения доступа на основе контроля виртуальных соединений. Предложена идея осуществлять обработку виртуальных соединений при помощи скрытой фильтрации. Отмечен новый уровень сложности задач защиты информации при обеспечении информационной безопасности в распределенных виртуализированных вычислительных средах. Рассмотрена архитектура разработанной защищенной гетерогенной распределенной вычислительной среды для решения научно-технических задач. Предложено использование специализированных межсетевых экранов для разграничения доступа между вычислительными ресурсами, как в рамках гипервизора, так и в рамках вычислительной среды в целом.

1. Введение

Виртуализация распределенных вычислительных ресурсов и формирование гетерогенных сред виртуальных вычислительных машин является перспективным направлением развития информационных технологий. В настоящее время различные компоненты данного направления принято объединять термином «облачные вычисления» (cloud computing), которые развиваются как технология, предоставляющая вычислительную услугу в виде сервиса. Обеспечение информационной безопасности (ИБ) таких вычислительных сред является важнейшей задачей. В статье вводится понятие виртуального соединения как эмерджентной сущности, на основе анализа которой осуществляется разграничение доступа. Сетевой трафик рассматривается как совокупность виртуальных соединений. Благодаря тому, что распределенная виртуализированная среда предоставляет гетерогенные вычислительные ресурсы, целесообразно использовать их для обеспечения её информационной безопасности. Так как виртуальные соединения функционируют независимо друг от друга, можно организовать параллельную обработку сетевого трафика при помощи организации «домена безопасности», функционирующего в рамках гипервизора и использующего то количество ресурсов (ядра, память), которое требуется для решения текущих задач ИБ.

2. Специфика обеспечения ИБ в виртуализированных средах

На сегодняшний день многие компании переводят вычислительные ресурсы в виртуальную инфраструктуру, в том числе и ведущие университеты России, используя для этого как открытые системы (Eucalyptus, OpenNebula), так и коммерческие решения (VmWare, Citrix, IBM). В связи с этим, остро встает проблема обеспечения информационной безопасности в виртуальных вычислительных системах такого рода [1]. Такая виртуальная среда обладает определенными специфическими особенностями, хотя многие ее характеристики аналогичны тем, которые встречаются в сетях распределенных вычислительных ресурсов и ГРИД приложениях. Среди важных отличий можно выделить следующие:

1. Обработка информации происходит в виртуальных машинах, которые находятся под полным контролем гипервизора, способного контролировать все данные, обрабатываемые виртуальными вычислительными ресурсами;
2. Средства управления виртуальной инфраструктурой (например, Eucalyptus) осуществляют распределение нагрузки между гипервизорами и являются новой сущностью в информационной среде, требующей защиты;

3. Традиционные средства защиты информации, такие как программно-аппаратные межсетевые экраны не могут контролировать трафик внутри узла виртуализации, таким образом, сетевое взаимодействие между виртуальными машинами в рамках одного гипервизора оказывается вне контроля;
4. В виртуальной среде в качестве устройств хранения данных выступают файлы, которые размещаются в сетевых хранилищах, а не аппаратные жесткие диски;
5. При миграции виртуальных машин между гипервизорами возникает передача фрагментов оперативной памяти, которая может содержать конфиденциальную информацию.

Таким образом, благодаря вышеперечисленным особенностям, возникают новые угрозы информационной безопасности, среди которых:

1. Атака на средства управления виртуальными машинами, контроллеры вычислительной среды (контроллер облака), кластера и на хранилище данных, на котором располагаются виртуальные образы машин и пользовательские данные;
2. Неавторизованный доступ к узлу виртуализации;
3. Использование виртуальной сети для передачи данных, не предусмотренных политикой информационной безопасности.

Важной особенностью виртуальной инфраструктуры является то, что атака или неавторизованный доступ могут быть произведены из виртуальной сети, где отсутствуют такие устройства как коммутаторы, аппаратные межсетевые экраны и физические линии связи, что существенно затрудняет применение существующих методов и средств защиты информации в компьютерных сетях и ГРИД системах.

Распределенные и виртуальные вычислительные среды в настоящее время не имеют эффективных средств защиты информации. Одна из проблем заключается в отсутствии межсетевых экранов, способных функционировать в среде виртуальных машин так же эффективно, как существующие на рынке программно-аппаратные средства защиты информационных ресурсов и отражения компьютерных атак. Отсутствие межсетевых экранов, сертифицированных по требованиям РД ФСТЭК, сдерживает применение новых технологий облачных вычислений в государственных учреждениях, научных и образовательных центрах [2]. Для ряда решений, например, свободно распространяемой и открытой облачной среды Eucalyptus, построенной на гипервизорах XEN или KVM, отсутствуют эффективные решения по защите виртуальных машин, несмотря на быстро растущую популярность данной среды, которая обеспечивается благодаря совместимости с интерфейсами продуктов от компании Amazon (Amazon EC2, Amazon S3).

3. Разграничение доступа как задача обработки виртуальных соединений

Виртуального соединение (ВС) [3] **–** это любой логически упорядоченный обмен сообщениями между узлами сети. Компьютерная сеть – это совокупность виртуальных соединений. Виртуальные соединения классифицируются на два уровня – технологические виртуальные соединения (ТВС) и информационные виртуальные соединения (ИВС) (рис. 1). Для реализации политики разграничения доступа правила фильтрации, которые могут быть заданы для различных уровней описания потоков данных, основанных на полях и заголовках канальных, транспортных и прикладных протоколов декомпозируются в форму ИВС и ТВС. В терминах разграничения доступа модель ТВС можно определить как поток пакетов, формируемый сетевыми приложениями в рамках информационного взаимодействия. Модель ТВС представлена в виде потенциально счетного подмножества декартова произведения множества пакетов P и временных меток T:

$$TBC = \{ p_{ti} \}, i = \overline{1, N}, N \in [1, \infty) \subset P \times T .$$

Представленная модель характеризуется конечным набором параметров, характеризующих субъект и объект доступа, а также действие в форме потока пакетов между ними в рамках межсетевого взаимодействия. Параметрами модели являются идентификаторы субъекта и объекта, такие как адреса, порты, и другие характеристики сетевых протоколов. Для оперативной классификации трафика, наряду с моделью ТВС, используется модель ИВС для описания взаимо-

действие между объектом и субъектом на уровне прикладных сервисов. Модель ИВС представляет собой совокупность ТВС, число и характеристики которой определяются декартовым произведением информационных моделей взаимодействия (ИМД), субъекта (ИМС) и объекта (ИМО) доступа:

$$ИВС = \{ТВС_i\}, i = \overline{1, N} \subset (ИМС \times ИМД \times ИМО).$$

Представленная формализация позволяет представить ИМС доступа как конечное подмножество, объем которого определяется на основе описания разрешенных субъектов межсетевого взаимодействия в рамках заданной политики разграничения доступа. ИМО характеризуется конечным подмножеством информационно-сетевых ресурсов, доступ к которым разрешен в соответствии с политикой РД. ИМД характеризует операции, совершаемые субъектом в рамках ИМО.

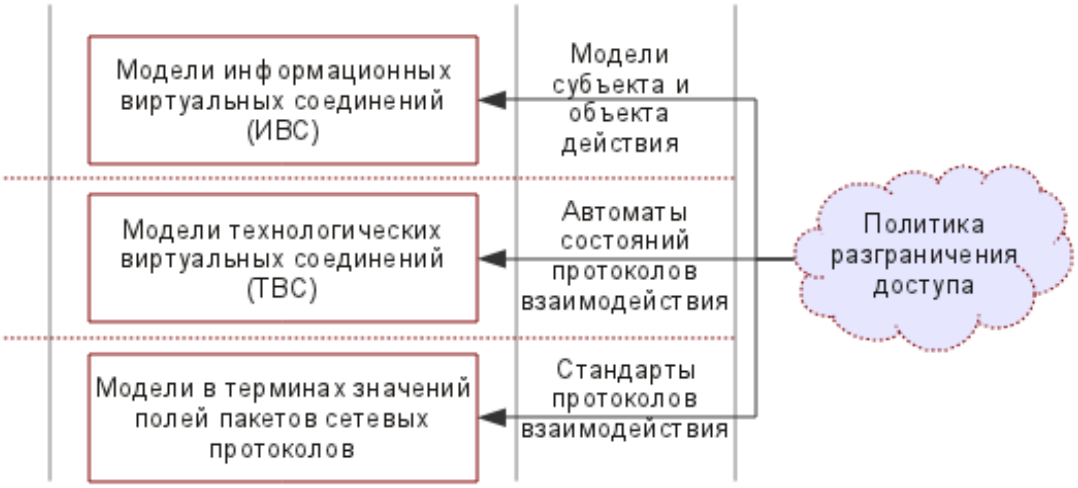


Рис. 1. Иерархическая структура формирования требований РД при использовании моделей ИВС и ТВС

4. Параллельная обработка ВС разных классов

Виртуальное соединение, как некоторая абстракция, существует параллельно и независимо от других виртуальных соединений, при этом виртуальные соединения не имеют между собой разделяемых ресурсов, что позволяет осуществлять их параллельную обработку [4]. Предложенный подход к фильтрации сетевого трафика, построенный на понятии виртуального соединения, позволяет выделить контекст соединения, который можно представить в виде вектора Y_i, описывающего его параметры, например, такие как порты и адреса источника и приемника, состояние соединения (на примере протокола TCP). Контроль виртуального соединения есть вычисление индикаторной функции *F*, для выполнения которого требуются такие ресурсы, как вычислительные процессоры и оперативная память: $F(Y_i\} = \{1,0,*\}$. Индикаторная функция F принимает значения: 1 – если ВС разрешено, 0 – если ВС запрещено, * - если на текущий момент невозможно однозначно определить запрещенное оно или нет, решение откладывается и ВС временно разрешается.

Вычислительные задачи можно разбить на две группы. Задачи первой группы носят потоковый характер и могут быть вычислены с помощью SIMD вычислителей (например, используя графические процессоры и технологию CUDA). Задачи второй группы решаются на стандартных многоядерных вычислителях MIMD. Так как описываемая распределенная среда является гетерогенной по отношению к доступным вычислителям, то в задачах межсетевого экранирования могут быть задействованы как потоковые SIMD вычислители, так и классические многоядерные MIMD процессоры. Благодаря тому, что межсетевые экраны, обеспечивающие защиту гипервизора, функционируют в виртуализированной среде, можно менять конфигурацию (вычислительные ядра, память, потоковые вычислители) устройства защиты в зависимости от параметров загрузки, политик доступа и количества имеющихся ресурсов.

Вычисление индикаторной функции F декомпозируется на множество вычислительных процессов – $\{F_i\}$. В таком случае, задачу контроля ВС можно описать в форме графа G(Q,X), называемого информационным графом контроля ВС (графовое описание потоковых задач подробно представлено в [5]). Граф контроля ВС состоит из множества вершин $F_i \in Q$, каждой из которых приписана операция F_i. Дуги $x(f_i, f_{i+1}) \in X$ определяют последовательность операций, приписанных вершинам графа G(Q,X), причем если две вершины q_i и q_{i+1} соединены дугой, то это означает, что результат операции F_i является входным для операции F_{i+1}. Каждый узел имеет дугу $x(f_i, F) \in X$, которая характеризует ситуацию, когда $F_i = 0$. В этом случае ВС считается запрещенным и дальнейший анализ не производится.

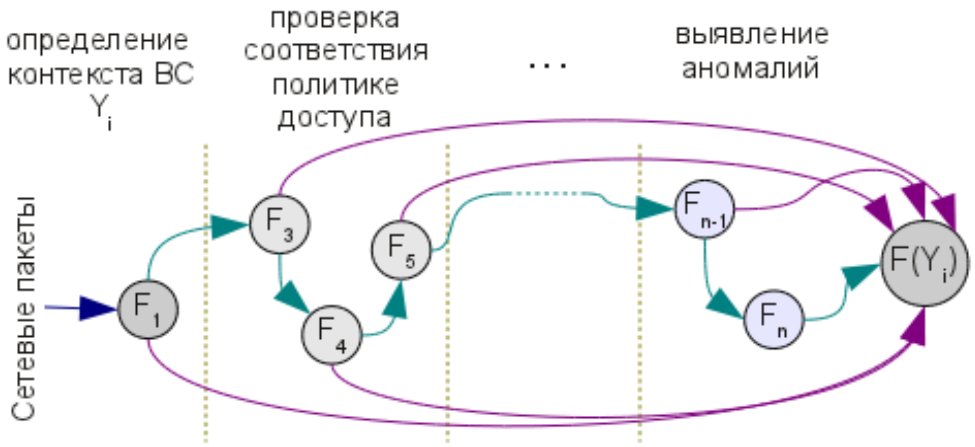


Рис. 2. Информационный граф контроля ВС с использованием вычислителей разных типов

Многопроцессорную вычислительную систему, решающую задачи межсетевого экранирования можно также представить в виде полносвязного графа $G^*(Q_m^*, Q_s^*)$, вершинами которого являются MIMD вычислители q_{mi}^* и потоковые вычислители q_{si}^*. Граф является полносвязным, так как коммуникации между вычислителями обеспечиваются аппаратными средствами и операционной системой и нет заранее заданного пути между вычислителями, данные могут попасть напрямую с одного узла на другой. Обычно граф вычислительной системы и информационный графы не соответствуют друг другу в силу того, что количество вычислительных ресурсов ограничено и меньше количества вычислительных процессов.

Граф контроля ВС можно разбить на N непересекающихся подграфов и построить конвейер обработки ВС, а также благодаря тому, что виртуальные соединения существуют независимо друг от друга возможна их параллельная обработка. При наличии C вычислителей MIMD типа время обработки ВС будет ограничено соотношением:

$$T_{BC} \le \frac{\max(z(f_i)) \cdot \max(\tau_j)}{C},$$

$z(f_i)$ - количество тактов требуемых для вычисления f_i, τ_j - продолжительность такта. Неравенство возникает благодаря тому что решение о классификации ВС может быть принято до окончания прохода всех узлов графа.

Благодаря гетерогенности и реконфигурируемости вычислительной среды в ряде случаев можно адаптировать конфигурацию межсетевого экрана под решаемые в данный момент задачи разграничения доступа. Достичь этого можно при помощи графовых моделей обработки сетевого трафика и использования технологии Netgraph[6], позволяющей организовать обработку сетевого трафика в контексте ядра операционной системы [4]. На рис. 2 представлен пример информационного графа контроля виртуальных соединений с декомпозицией индикаторной функции контроля на составляющие. Представленный подход вместе с использованием технологии виртуализации ресурсов позволяет повысить производительность контроля сетевого трафика и использовать только те вычислительные компоненты, которые нужны для решения текущих задач разграничения доступа.

5. Архитектура защищенной гетерогенной вычислительной среды

Распределенная вычислительная среда для решения научно-технических задач представляет разнородное множество вычислительных ресурсов в виде виртуальных машин и имеет следующие особенности [7]:

1. Среду используют широкий круг пользователей, решающих задачи разных классов;
2. Виртуальные машины разных групп пользователей могут функционировать в рамках одного гипервизора;
3. Используется широкий спектр программных компонентов (CAD/CAE приложения, средства разработки) и операционных систем (Linux, Windows);
4. Различные аппаратные конфигурации, в том числе виртуальные многоядерные вычислительные машины и виртуальные машины, позволяющие проводить потоковые вычисления с использованием технологии CUDA.

Гипервизор вычислительной среды является мощным многоядерным узлом, на котором функционирует домен уровня 0 (dom0 в терминах гипервизора XEN или сервисная консоль в терминах других гипервизоров) и виртуальные вычислительные машины (домен уровня U, domU). Для обеспечения информационной безопасности и разграничения доступа (РД) между виртуальными машинами, функционирующими в рамках одного гипервизора необходимо осуществлять контроль внутреннего («виртуального» трафика) и внешнего (поступающего с других гипервизоров и из внешних сетей). Решить задачу разграничения доступа можно путем интеграции в гипервизор виртуального межсетевого экрана, функционирующего в рамках гипервизора, но отдельно от пользовательских виртуальных машин. Домен виртуального межсетевого экрана можно определить как «домен безопасности» (security domain, domS). Важным аспектом при осуществлении контроля сетевого трафика является скрытое функционирование средства фильтрации, межсетевой экран не должен изменять топологию сетевой подсистемы гипервизора. Достичь этого можно путем использования технологии «Стелс» – осуществления невидимого для других сетевых компонентов контроля пакетного трафика [8].

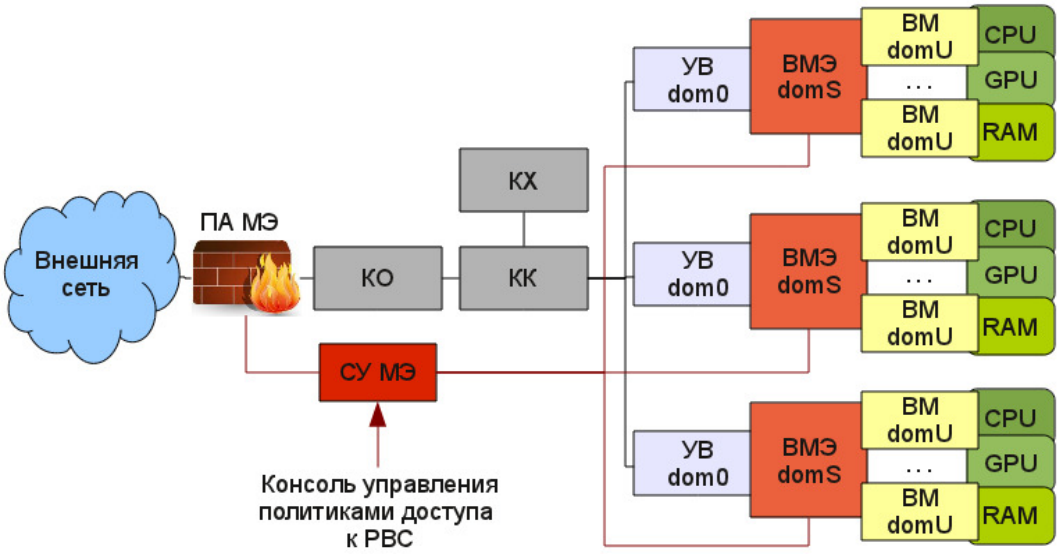


Рис. 3. Архитектура защищенной вычислительной среды

На рис 3. представлена обобщенная архитектура распределенной вычислительной среды с внедренными средствами разграничения доступа. Используются следующие сокращения: ПА МЭ – программно-аппаратный межсетевой экран, ВМЭ – виртуальный межсетевой экран, СУ МЭ – система управления межсетевыми экранами, ВМ – виртуальная машина, КО – контроллер облака, КК – контроллер кластера, КХ – контроллер хранилища. СУ МЭ решает задачи синхронизации и согласования политик безопасности между компонентами средств защиты информации (ПА МЭ, ВМЭ). При изменении политики доступа, новые правила реплицируются на все компоненты защиты РВС. Предложенный подход позволяет защитить распределенную вычислительную среду как от внешних так и от внутренних угроз. Внедрение прозрачного домена

безопасности domS изолирует гипервизор от виртуальных вычислительных машин, что исключает возможность атаки гипервизора из внутренней сети.

6. Заключение

Представленная архитектура распределенной гетерогенной вычислительной среды предоставляет вычислительные ресурсы разных конфигураций, что позволяет организовать в ней средства защиты в виде выделенного домена безопасности (domS), обладающего свойством реконфигурирумости. Реконфигурация средств защиты происходит в соответствии с решаемой в данный момент задачей, таким образом вычислительная среда защищает сама себя и адаптируется под текущую ситуацию. Проведенные исследования зависимости времени обработки ВС от количества доступных ядер показали линейную масшабируемость до восьми вычислителей включительно. На базе кафедры телематики СПбГПУ разработан и функционирует прототип вычислительной среды [7], предоставляющий гетерогенные вычислительные ресурсы, что позволяет проводить дальнейшие исследования и разработки в области обеспечения информационной безопасности распределенных вычислительных сред.

Литература

1. Cloud Security Alliance, Top Threats to Cloud Computing, Март 2010. [Электронный ресурс]. URL: http://www.cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf (дата обращения: 5.11.2010).
2. В.Мулюха, А.Г.Новопашенный, Ю.Е. Подгурский, В.С. Заборовский Методы и средства защиты компьютерной информации. Межсетевое экранирование. Издательство СПб Государственный политехнический университет, СПб, 2010, 90 с.
3. Силиненко А.В. Разграничение доступа в IP-сетях на основе моделей состояния виртуальных соединений: дис. канд. тех. Наук : 05.13.19: / Силиненко Александр Витальевич. – СПб, 2010. – 144 с.
4. Заборовский В.С., Лукашин А.А., Купреенко С.В. Многоядерная вычислительная платформа для высокопроизводительных межсетевых экранов. Высокопроизводительные вычислительные системы // Материалы Седьмой Международной научной молодежной школы. – Таганрог: Изд-во ТТИ ЮФУ, 2010. - 336 с.
5. Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И. Реконфигурируемые мультиконвейерные вычислительные структуры. - Ростов-на-Дону: Изд-во ЮНЦ РАН, 2008. - 320 с.
6. Cobbs A. All about Netgraph. [Электронный ресурс]. URL: http://www.daemonnews.org/200003/netgraph.html (дата обращения: 5.11.2010).
7. Лукашин А.А. Рощупкин И.А. Методы и средства построения распределенной вычислительной среды для решения наукоемких задач // XXXIX неделя науки СПбГПУ, Материалы Всероссийской межвузовской научно-технической конференции студентов и аспирантов, 6 - 11 декабря 2010 года, Часть XV, факультет при ЦНИИ робототехники и технической кибернетики. – СПб.: Изд-во Политехнического университета. – 2010. – с. 13-15
8. V. Zaborovsky, A. Titov. Specialized Solutions for Improvement of Firewall Performance and Conformity to Security Policy. Proceedings of the 2009 International Conference on Security & Management. v. 2. p. 603-608. July 13-16, 2009.

Параллельные алгоритмы решения SAT в применении к оптимизационным задачам с булевыми ограничениями*

О.С. Заикин, И.В. Отпущенников, А.А. Семенов

Институт динамики систем и теории управления СО РАН

Предложена параллельная технология, применимая к целому ряду задач дискретной оптимизации, и использующая концепцию крупноблочного параллелизма. Технология основана на эффективных процедурах сведения задач комбинаторной оптимизации к SAT-задачам. Процесс решения исходной оптимизационной задачи реализован в виде итерационной схемы, каждый этап которой – это решение некоторой SAT-задачи. Получаемые SAT-задачи решаются при помощи крупноблочных параллельных алгоритмов. Для учета информации, накопленной в предыдущих итерациях, реализована техника «Incremental SAT», применяемая в задачах верификации многих дискретных систем. Разработанная технология была протестирована на решении задач 0-1-ЦЛП в распределенных вычислительных средах.

1. Введение

Как известно, многие NP-трудные задачи возникли из совершенно конкретных практических постановок. В ряде направлений без умения решать данные задачи невозможно обойтись. Речь идет о проблемах синтеза и верификации дискретных управляющих/управляемых систем, проблемах экономики, производственного планирования, логистики и многих других. В этих областях вопросы построения практически эффективных алгоритмов решения соответствующих комбинаторных задач чрезвычайно актуальны. Большое число исследований посвящено построению приближенных алгоритмов решения NP-трудных проблем. Однако в некоторых приложениях, например, в задачах верификации микросхем приближенные алгоритмы (даже обладающие малой погрешностью) совершенно бесполезны. С другой стороны, необходимость находить точные решения существенно сужает класс методов – методы непрерывной математики, генетические алгоритмы, разнообразные «эволюционные эвристики» в общем случае не дают точных решений.

В настоящей работе приведены основы «пропозиционального подхода» к решению комбинаторных задач из весьма широкого класса. Данный подход предполагает нахождение точных решений и включает две составляющих. Во-первых, это алгоритмы сведения комбинаторных задач к булевым уравнениям, и, во-вторых, это символьные алгоритмы поиска решений получаемых уравнений. В последние годы интерес именно к такому рассмотрению комбинаторных задач заметно усилился в связи с существенным прогрессом в алгоритмике булевых решателей, а также в связи с бурным развитием параллельных вычислительных технологий, поскольку булевы задачи допускают весьма естественные формы параллелизма.

Библиография по решению булевых уравнений весьма обширна – от фундаментальной монографии С. Рудяну [1] до многочисленных в последние годы работ по SAT-задачам. Под SAT-задачами (от англ. «Satisfiability», т.е. «Выполнимость») понимаются задачи поиска выполняющих наборов булевых формул, как правило, приведенных к конъюнктивным нормальным формам (КНФ).

Статей, специально посвященных сведению комбинаторных проблем к булевым уравнениям, сравнительно мало (можно сослаться на обзорную статью [2] и список литературы к ней). Удивительно то, что в подавляющем большинстве эти результаты имеют характер наглядных примеров и правдоподобных рассуждений – самой строгой в этом смысле продолжает оставаться процедура, фигурирующая в оригинальном и последующих доказательствах теоремы Кука [3], [4]. Следует отметить, что реализовать на основе перечисленных результатов кон-

* Работа выполнена при поддержке гранта «Лаврентьевский конкурс молодежных проектов СО РАН» 2010-2011.

кретные процедуры сведения, применимые к достаточно широким классам комбинаторных задач, крайне затруднительно: сведения, описанные в [2], слишком специфичны, а известные варианты теоремы Кука доказаны в отношении машины Тьюринга – модели, которая крайне далека от современных ЭВМ в плане языка и организации вычислений.

В работе [5] описаны механизмы пропозиционального кодирования программ, вычисляющих дискретные функции на машинах с неограниченными регистрами (МНР). Язык данной модели очень естествен – фактически это аналог ассемблера современных ЭВМ. Там же в общих чертах описаны основные принципы высокоуровневой трансляции алгоритмов, вычисляющих дискретные функции, в булевы уравнения. Реализацией этих идей стал программный комплекс Transalg, архитектура которого и функциональные возможности в применении к решению задач обращения дискретных функций описаны в следующем пункте.

Комплекс Transalg применяется для сведения различных дискретных задач к SAT-задачам, в отношении которых возможно использование различных техник распараллеливания [6], [7]. В настоящей работе Transalg используется для сведения к SAT-задачам задач 0-1-целочисленного линейного программирования (0-1-ЦЛП).

В заключительной части работы описаны общие принципы распараллеливания SAT-задач, кодирующих задачи дискретной оптимизации. Здесь же приведены результаты программной реализации соответствующей технологии и результаты численных экспериментов на задачах 0-1-ЦЛП.

2. Сведение задач комбинаторной оптимизации к задаче о пропозициональной выполнимости (SAT)

Программный комплекс Transalg предназначен для сведения к булевым уравнениям (и в том числе к SAT-задачам) задач обращения полиномиально вычислимых дискретных функций. С этой целью алгоритм вычисления функции записывается на специальном С-подобном языке (ТА-язык), после чего происходит трансляция полученной ТА-программы в систему булевых уравнений. На заключительном этапе трансляции система приводится к одной из возможных нормальных форм («КНФ=1», «ДНФ=0», полиномиальные уравнения над полем GF(2)). Также предусмотрена возможность построения И-НЕ-графа [8], представляющего рассматриваемый алгоритм.

Схематично процесс трансляции ТА-программ представлен на **Рис. 1**.

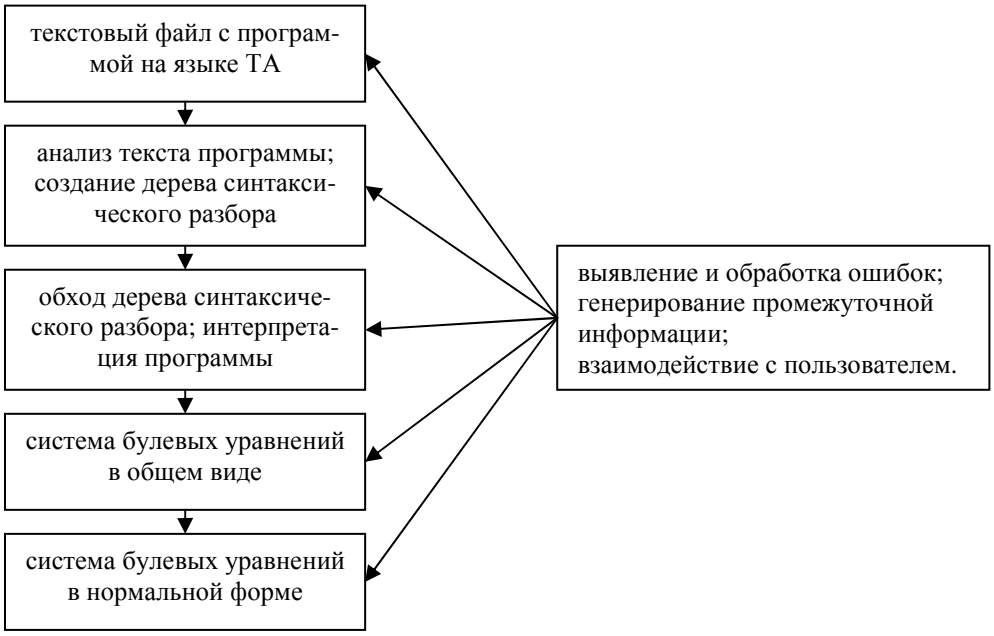


Рис. 1. Общая схема работы программного комплекса Transalg

Фазы анализа текста ТА-программы, построения дерева синтаксического разбора и обход полученного дерева с целью интерпретации реализованы стандартным способом (см., например [9]).

Язык ТА представляет собой процедурный язык программирования с блочной структурой и С-подобным синтаксисом. Каждый блок – это конечный список инструкций ТА-программы. Программа на языке ТА представляет собой набор определений функций, а также объявлений и определений глобальных переменных и констант. В языке ТА реализованы все основные примитивные конструкции, характерные для процедурных языков программирования:

- объявление/определение переменной или массива переменных;
- определение именованных констант;
- оператор присваивания;
- составной оператор;
- условный переход;
- цикл;
- определение пользовательской функции;
- возврат из функции;
- вызов функции.

В языке ТА поддерживается два основных типа данных. Переменные целочисленных типов хранят параметры транслируемой программы, не зависящие от данных, подаваемых ей на вход. Например, это могут быть длины входного и выходного слов, количество итераций в циклах, целочисленные константы, используемые при вычислении значения дискретной функции. Тип данных bit используется для объявления булевых переменных, кодирующих входную информацию транслируемой программы, а также информацию, возникающую в процессе работы этой программы.

Особо подчеркнем, что переменные транслируемой ТА-программы и переменные пропозиционального кода этой программы представляют, вообще говоря, разные сущности. Переменные, фигурирующие в тексте транслируемой ТА-программы (далее «переменные программы»), понимаются в традиционном смысле – это идентификаторы областей памяти. Переменные, попадающие в пропозициональный код (далее «переменные кода»), понимаются как символы некоторого конечного алфавита. По своему смыслу переменные кода – это переменные итоговой системы булевых уравнений. Тем не менее, эти два вида переменных тесно связаны. Каждой переменной программы соответствует специальная структура данных «var_object». Эта структура позволяет связывать переменные программы типа bit с переменными кода. При интерпретации инструкций, содержащих переменные программы, транслятор проверяет соответствующие структуры var_object на наличие в них связи с переменными кода.

Кроме перечисленных, в тексте ТА-программы могут встречаться переменные, необходимые для хранения результатов промежуточных вычислений. Структура var_object таких переменных не связывает их с переменными кода. Далее такие переменные называются фиктивными. Проиллюстрируем все сказанное на следующем примере.

Пример 1. Рассмотрим ТА-программу, которая реализует регистр сдвига с линейной обратной связью (РСЛОС, [10]), заданной полиномом (над GF(2)) $P(x) = x^{19} + x^{18} + x^{17} + x^{14} + 1$.

```
__in bit reg[19];
__out bit output[100];
bit shiftReg(){
   bit x = reg[18];
   bit y = reg[18]^reg[17]^reg[16]^reg[13];
   for(int j = 18; j > 0; j = j - 1){
         reg[j] = reg[j - 1];
   }
   reg[0] = y;
   return x;
}
void main(){
   for(int i = 0; i < 100; i = i + 1){
         output[i] = shiftReg();
```

```
    }
}
```

Массив булевых переменных reg описывает в каждый фиксированный момент времени текущее состояние регистра сдвига. Его содержимое на начальном шаге соответствует входной информации (данный факт отмечен атрибутом __in), которая описывается переменными кода, образующими множество входных переменных $X^0 = \{x_1,...,x_{19}\}$.

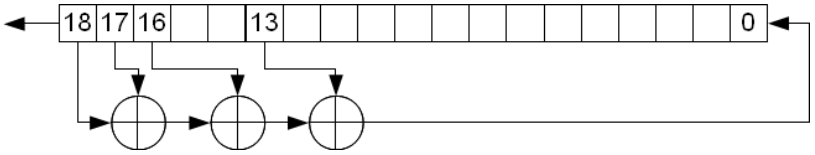


Рис. 2. Схема РСЛОС, реализуемого функцией shiftReg()

Представленная программа реализует 100 тактов работы регистра сдвига. В теле основной функции main() организован цикл, в котором вызывается функция сдвига регистра shiftReg(). Значения, возвращаемые функцией shiftReg(), определяют биты выходного слова, которое представлено в программе массивом output.

Сдвиг регистра обновляет значения всех элементов массива reg. На первом такте данная операция приводит к вводу новых переменных кода, образующих множество $X^1 = \{x_{20},...,x_{38}\}$. Новые переменные связаны с переменными из множества X^0 следующей системой булевых уравнений:

$$\begin{cases} (x_{20} \equiv x_{19} \oplus x_{18} \oplus x_{17} \oplus x_{14}) = 1 \\ (x_{21} \equiv x_1) = 1 \\ (x_{22} \equiv x_2) = 1 \\ ..... \\ (x_{38} \equiv x_{18}) = 1 \end{cases}$$

Переменная x_{19} кодирует первый бит ключевого потока, полученный в результате первого сдвига рассматриваемого РСЛОС. Отметим, что локальные переменные x и y функции shiftReg() необходимы лишь для корректной организации вычислений и не связаны с переменными кода программы. В контексте вышесказанного x и y – фиктивные переменные.

Аналогичным образом можно описывать и преобразовывать в булевы уравнения алгоритмы вычисления дискретных функций из весьма широкого класса. Далее мы описываем процесс сведения к SAT оптимизационной задачи из класса 0-1-ЦЛП. Специализированные для данной задачи процедуры трансляции в SAT были описаны, например, в работе [11]. В нашем случае для трансляции 0-1-ЦЛП в SAT использовался комплекс Transalg.

Рассматривается система неравенств

$$A \cdot x \le b, \qquad (1)$$

где A – $m \times n$-матрица с целочисленными компонентами, b – вектор длины m, состоящий из целых чисел. Предполагаем, что переменные $x_i, i \in \{1,...,n\}$, могут принимать значения в множестве целых чисел $\{0,1\}$. Требуется при ограничениях (1) минимизировать линейную форму $\langle c, x \rangle$, где c – целочисленный вектор длины n.

Процесс сведения данной задачи к SAT состоит в преобразовании линейных неравенств, образующих систему (1), в конъюнкции дизъюнктов. При этом можно использовать эквивалентные преобразования исходных ограничений, приводящие к ограничениям вида

$$\langle a', x^{\sigma} \rangle \le b_0, \qquad (2)$$

где a' – вектор длины n с целыми неотрицательными компонентами, x^{σ} – вектор, образованный литералами над переменными из множества $X = \{x_1,...,x_n\}$, а b_0 – неотрицательное целое число.

Пример 2. Рассмотрим ограничение $3 \cdot x_1 - 2 \cdot x_2 + 5 \cdot x_3 \le 3$. Результатом замены $x_2 = 1 - \bar{x}_2$ является ограничение $3 \cdot x_1 + 2 \cdot \bar{x}_2 + 5 \cdot x_3 \le 5$.

Сказанное означает, что от исходной задачи возможен эффективный переход к задаче минимизации линейной формы $\left\langle c', x^{\sigma} \right\rangle$ при ограничениях

$$A' \cdot x^{\sigma} \le b',$$

где A' – $m \times n$-матрица с неотрицательными целыми компонентами, b', c' – векторы, состоящие из неотрицательных целых чисел, x^{σ} – вектор литералов над X.

Наиболее простой метод трансляции состоит в том, что псевдобулеву ограничению вида (2) посредством таблицы истинности сопоставляется булева функция $f(x_1,...,x_n)$, принимающая значение 1 тогда и только тогда, когда выполняется ограничение (2). Такой подход возможен при условии, что число n невелико (например, $n \le 20$), т.е. таблица истинности может быть легко построена. В этом случае в систему булевых уравнений добавляется уравнение

$$\Phi_f(x_1,...,x_n) = 1,$$

где Φ_f – формула (например, в КНФ), реализующая функцию $f(x_1,...,x_n)$.

В прикладных задачах ЦЛП часто встречаются ограничения, линейная часть которых зависит от сотен переменных, что делает непосредственный переход к таблице истинности невозможным. В этом случае используется представление функции $f(x_1,...,x_n)$ в виде вычисляющего ее алгоритма. Пропозициональный код данного алгоритма – система булевых уравнений, совместная тогда и только тогда, когда выполняется ограничение (2). Данная система строится при помощи комплекса Transalg.

Пример 3. Рассмотрим ограничение $3 \cdot x_1 + 2 \cdot \bar{x}_2 \le 5$. На первом шаге для термов $3 \cdot x_1$ и $2 \cdot \bar{x}_2$ строятся слова $(x_1 x_1)$ и $(\bar{x}_2 0)$ (используется тот факт, что числа 3 и 2 представляются двоичными векторами (11) и (10)). Линейной форме $3 \cdot x_1 + 2 \cdot \bar{x}_2$ сопоставляется система булевых уравнений

$$\begin{cases} (x_4 \equiv x_1) = 1 \\ (x_5 \equiv x_1 \oplus \bar{x}_2) = 1 \\ (x_6 \equiv x_1 \cdot \bar{x}_2) = 1. \end{cases} \qquad (3)$$

Данная система описывает процесс вычисления дискретной функции, которая, получая на входе произвольный двоичный вектор $(x_1 x_2)$, на выходе выдает число $3 \cdot x_1 + 2 \cdot \bar{x}_2$. Данное число кодируется двоичным вектором $(x_4 x_5 x_6)$. Тот факт, что $3 \cdot x_1 + 2 \cdot \bar{x}_2$ не превосходит числа 5, справедлив тогда и только тогда, когда формула $(\bar{x}_6 \vee x_6 \cdot \bar{x}_5)$ принимает значение «истина». Таким образом, итоговой системой булевых уравнений, кодирующей ограничение $3 \cdot x_1 + 2 \cdot \bar{x}_2 \le 5$, является система (3), дополненная уравнением $(\bar{x}_6 \vee x_6 \cdot \bar{x}_5) = 1$.

Построенные таким образом системы булевых уравнений приводятся к уравнениям вида «КНФ=1» при помощи преобразований Цейтина [12]. При этом множество переменных разрастается (не более чем полиномиально), однако между множеством решений исходной системы булевых уравнений и множеством решений получаемого уравнения вида «КНФ=1» существует биекция.

В комплексе Transalg при построении уравнений вида «КНФ=1» применяются разнообразные приемы оптимизации итогового пропозиционального кода. Было проведено сравнение пропозиционального кода задач 0-1-ЦЛП, полученного с помощью комплекса Transalg, с кодом, полученным известным псевдобулевым решателем MiniSat+ [13]. Сравнение проводилось на серии тестов из библиотеки 0-1-ЦЛП задач [14]. Число переменных в КНФ, получаемых на выходе Transalg, было в среднем меньше на 20%, чем в КНФ, выдаваемых MiniSat+. По числу дизъюнктов, тем не менее, Transalg на данный момент проигрывает MiniSat+ в среднем на те же 20%.

На основе описанных выше механизмов трансляции задач 0-1-ЦЛП в SAT и известного SAT-решателя MiniSat2.0 был создан новый решатель псевдобулевых задач «PBSolver». На вход решателю поступает файл с задачей 0-1-ЦЛП на минимум в формате «LP». Процедуры разбора данного файла, а также процедуры трансляции ограничений задачи и целевой функции в КНФ встроены в решатель. Пусть $f(x)$ – целевая функция исходной задачи 0-1-ЦЛП. Поиск оптимального решения осуществляется итеративным вызовом SAT-решателя. На нулевой итерации SAT-решатель запускается на КНФ, которая кодирует только ограничения исходной задачи 0-1-ЦЛП. Если ответ «UNSAT», то допустимое множество пусто. В противном случае находится некоторая допустимая точка x_0 и строится первое приближение $(x_0, f(x_0))$. Затем к имеющейся системе добавляется ограничение $f(x) < f(x_0)$, и на основе полученной системы формируется новая SAT-задача. Тем самым, такой процесс решения 0-1-ЦЛП-задачи – это фактически схема последовательных приближений, итогом которой является гарантированное нахождение оптимального решения.

3. Процедуры распараллеливания задач комбинаторной оптимизации, представленных в форме SAT-задач

Одной из причин построения нового решателя псевдобулевых задач была проблема разработки параллельных технологий решения таких задач. Ниже описываются общие принципы распараллеливания задач комбинаторной оптимизации, представленных в форме SAT-задач. Исходная задача рассматривается при этом как задача минимизации некоторой функции, принимающей целые неотрицательные значения, на допустимом множестве, определяемом системой ограничений. Полагаем, что построена SAT-задача, кодирующая систему ограничений, определяющую допустимое множество. Если полученная КНФ невыполнима, то допустимое множество пусто и процесс останавливается. В противном случае решается серия SAT-задач – выполняющий набор каждой очередной SAT-задачи определяет точку в допустимом множестве, значение целевой функции в которой меньше, чем в точке, полученной на предыдущей итерации. При этом возможны различные техники распараллеливания. Наиболее очевидная схема состоит в разбиении интервала $[0, f(x_0))$ на непересекающиеся (но покрывающие весь этот интервал) интервалы меньшей длины. Каждому такому интервалу соответствует некоторая SAT-задача. Полученное семейство SAT-задач обрабатывается как параллельный список.

На настоящий момент реализована другая схема распараллеливания, которая близка в идейном плане схемам, использованным при обращении дискретных функций [6], [7]. В соответствии с данной схемой выбирается некоторое множество булевых переменных, варьирование всевозможных значений которых позволяет построить декомпозиционное семейство, образованное SAT-задачами меньшей размерности (в сравнении с исходной). Полученное декомпозиционное семейство обрабатывается как параллельный список. Далее в терминах стандарта MPI (управляющий процесс/вычислительные процессы, см., например, [15]) описывается собственно процесс обработки списка заданий параллельным решателем PD-SAT [7].

Обрабатываемыми заданиями являются SAT-задачи из декомпозиционного семейства с дополнительными ограничениями, определяющими текущее рекордное значение целевой функции. Все задания обрабатываются псевдобулевым решателем PBSolver, который был описан в разделе 2. Далее используется следующая классификация заданий (см. [7]): свободные задания – задания, процесс решения которых на текущий момент не был запущен; связанные незавершенные задания – задания, которые решаются на текущий момент; связанные завершенные задания – задания, которые на текущий момент уже решены. Вычисления разделены на три этапа.

Этап 1. PD-SAT запущен на n процессах: процесс номер 1 управляющий, процессы с номерами $2,...,n$ – вычислительные. На управляющем процессе решается SAT-задача для КНФ C_{constr}, кодирующей только ограничения исходной 0-1-ЦЛП задачи. Если КНФ C_{constr} невыполнима, то допустимое множество пусто, вычисления прекращаются и выдается ответ «исходная 0-1-ЦЛП задача не имеет решений». Если C_{constr} выполнима, то допустимое множество

не пусто. В этом случае из выполняющего набора КНФ C_{constr} выделяется вектор x_0 – допустимая точка исходной 0-1-ЦЛП задачи и формируется начальное приближение $(x_0, f(x_0))$.

Этап 2. Управляющий процесс по входным данным формирует список заданий. Число заданий D равно ближайшей справа степени 2 от числа $(n-1)\cdot C$. Здесь C – константа, влияющая на загрузку вычислительных процессов. Данная константа определяется эмпирически. С управляющего процесса отсылаются первые $n-1$ свободных заданий из списка: i-е задание ($i \in \{1,...,n-1\}$) отсылается на вычислительный процесс с номером $i+1$ (каждое такое задание становится связанным незавершенным). Каждый вычислительный процесс приступает к обработке полученного задания.

Этап 3. После выполнения этапов 1–2 управляющий процесс переходит в состояние ожидания решений заданий с вычислительных процессов. Если на управляющий процесс приходит ответ, то соответствующее задание становится связанным завершенным. На приславший данный ответ вычислительный процесс отправляется очередное свободное задание из списка. Данное задание – это некоторая КНФ из декомпозиционного семейства и текущее рекордное значение целевой функции.

Помимо передачи заданий предусмотрена возможность передавать с управляющего процесса на вычислительные найденные рекордные значения целевой функции. Каждое новое рекордное значение отправляется на вычислительные процессы, даже если на данный момент на них не передается очередное задание. Вычислительные процессы периодически проверяют наличие сообщений с управляющего процесса с обновленными рекордными значениями. В используемые SAT-решатели были внесены изменения, позволяющие осуществлять такую проверку за счет применения асинхронных обменов. Если сообщение с обновленным рекордным значением получено и оно меньше, чем текущее значение, найденное в процессе работы псевдобулевого решателя, то решение текущей SAT-задачи прерывается и псевдобулев решатель формирует и решает новую SAT-задачу с учетом нового рекордного значения. Таким образом, информация, полученная в ходе решения одного задания, может ускорить процесс решения других заданий. Вычисление останавливается после обработки всего параллельного списка заданий.

Также при обработке списка заданий используется техника «Incremental SAT» [16], состоящая в том, что часть конфликтных дизъюнктов, накопленных SAT-решателем при обработке предыдущего задания, конъюнктивно приписывается к КНФ, которая является последующим заданием.

На данный момент объем численных экспериментов невелик. Проведенные эксперименты использовали примеры из библиотек тестов [17, 18]. На ряде примеров при распараллеливании было получено ускорение, близкое к линейному, однако эффекта сверхлинейного ускорения, в отличие от задач обращения некоторых дискретных функций [19], ни на одном тесте добиться не удалось.

Заключение

В статье предложена технология распараллеливания, применимая к обширному классу задач дискретной оптимизации. Предварительно рассматриваемая оптимизационная задача сводится к SAT-задаче с дополнительными (оптимизационными) условиями на выполняющий набор. Процесс сводимости осуществляется при помощи специального программного комплекса, краткому описанию которого посвящен второй пункт работы. В заключительной части описана технология распараллеливания SAT-задач, кодирующих задачи дискретной оптимизации. В основе технологии лежат идеи, предложенные в более ранних работах авторов. Особенность технологии в применении к оптимизационным постановкам состоит в необходимости мониторинга процесса обновления рекордных значений целевой функции. Перспективность описанной технологии в возможности ее применения к решению оптимизационных задач с различными типами ограничений (в том числе с нелинейными ограничениями типа равенств и неравенств) и с различными типами целевых функций. В качестве модельной задачи дискретной

оптимизации, на примере которой демонстрировались характерные черты предложенной технологии, была выбрана задача 0-1-целочисленного линейного программирования.

Литература

1. Rudeanu S. Boolean functions and equations, Amsterdam-London: North-Holland Publishing Company, 1974. 442 p.
2. Prestwich S. CNF encodings. In Handbook of Satisfiability (editors: A. Biere, M.Heule, H. van Maaren, T. Walsh). IOS Press, 2009. pp. 75-97.
3. Cook S.A. The complexity of theorem-proving procedures // Proc. 3rd Ann. ACM Symp. on Theory of Computing (STOC 71). ACM. 1971. pp. 151-159.
4. M.R. Garey and S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman, 1979, 340 p.
5. Семенов А.А. Трансляция алгоритмов вычисления дискретных функций в выражения пропозициональной логики // Прикладные алгоритмы в дискретном анализе. Серия: Дискретный анализ и информатика, Вып. 2. 2008. Иркутск: Изд-во ИГУ. С. 70-98.
6. Заикин О.С., Семенов А.А. Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. №1. С. 43–50.
7. Заикин О.С. Реализация процедур прогнозирования трудоемкости параллельного решения SAT-задач // Вестник УГАТУ. 2010. Т. 14, №4(39) С. 210-220.
8. Формат AIG.
   URL: http://fmv.jku.at/aiger/ (дата обращения: 11.12.2010).
9. Ахо А., Сети Р., Ульман Дж. Компиляторы. Принципы, технологии, инструменты. – М.: СПб, Киев, «Вильямс», 2001. 768 с.
10. Menezes A., Oorschot P., Vanstone S. Handbook of Applied Cryptography. CRC Press. 1996. 657 p.
11. Een N., Sorensson N. Translating Pseudo-Boolean Constraints into SAT // Journal on Satisfiability, Boolean Modeling and Computation. 2006. Vol. 2. pp. 1-25.
12. Цейтин Г.С. О сложности вывода в исчислении высказываний // Записки научных семинаров ЛОМИ АН СССР. 1968. Т.8. С. 234-259.
13. MiniSat+ solver.
    URL: http://minisat.se/MiniSat.html (дата обращения: 11.12.2010).
14. Beasley J.E. Or-library: distributing test problems by electronic mail // J. Oper. Res. Soc. 1990. Vol. 41(11). pp. 1069–1072.
15. Гришагин В.А., Свистунов А.Н. Параллельное программирование на основе MPI. Учебное пособие. – Нижний Новгород: издательство ННГУ им. Н.И. Лобачевского, 2005.
16. Stefan Disch, Christoph Scholl: Combinational Equivalence Checking Using Incremental SAT Solving, Output Ordering, and Resets. ASP-DAC 2007. pp. 938-943.
17. Pseudo-Boolean Competition 2010.
    http://www.cril.univ-artois.fr/PB10/ (дата обращения: 11.12.2010).
18. MIPLIB - Mixed Integer Problem Library.
    URL: http://miplib.zib.de (дата обращения: 11.12.2010).
19. Семенов А.А., Заикин О.С. Неполные алгоритмы в крупноблочном параллелизме комбинаторных задач // Вычислительные методы и программирование. 2008. Т. 9. №1. С. 112–122.

Распараллеливание алгоритмов умножения чисел многократной точности

Е.Г. Качко

Харьковский национальный университет радиоэлектроники

Операция умножения для длинных чисел остается объектом исследования математиков и программистов с точки зрения минимизации вычислительной сложности. Для оценки вычислительной сложности этой операции традиционно используется количество элементарных операций. При этом не учитываются свойства современных процессоров, такие как суперскалярность и многоядерность. В работе рассмотрены современные алгоритмы умножения с точки зрения возможности их распараллеливания, сделана теоретическая оценка вычислительной сложности и экспериментальная проверка с помощью Open MP и TBB. Выполнен анализ полученных результатов

1. Введение

Практически все современные несимметричные криптографические алгоритмы используют операцию умножения для длинных чисел. При этом длина сомножителя все время увеличивается. Так, для алгоритма RSA в качестве сомножителей могут использоваться числа длиной до 4096 бит, в дальнейшем и этот диапазон может быть расширен. Так как операцию умножения необходимо выполнять многократно при возведении в степень, а последняя операция используется и в алгоритмах RSA, и в алгоритмах на основе эллиптических кривых, то даже незначительное уменьшение вычислительной сложности для этой операции приведет к существенному уменьшению времени для формирования и проверки цифровой подписи.

Многие математики и программисты неоднократно возвращались к теме минимизации вычислительной сложности операции умножения. Из наиболее распространенных алгоритмов следует назвать алгоритмы: умножения в столбик (так называемый школьный метод), Карацубы [1], Шёнхаге и Штрассена [2], Фюрера [3]. В таблице 1 указаны значения вычислительной сложности для каждого из выше названных алгоритмов

Таблица 1. Вычислительная сложность основных алгоритмов умножения

Алгоритм	**Вычислительная сложность**
Вычисление в «Столбик»	$O(n^2)$
Карацубы и Офмана алгоритм	$O(n^{\log 3})$
Шёнхаге и Штрассена алгоритм	$O(n\log n\log\log n)$ $(2^{15}..2^{17})$
Фюрера алгоритм	$O(n\log n 2^{O(\log^* n)})$[1]

Следует заметить, что последний алгоритм предполагает представление сомножителей как полиномы с системой счисления 2^{16} для длин чисел до 65536 бит включительно. Для 16 битных процессоров это удачное представление, для 32-битных и 64 битных – такое представление приведет к существенной потере производительности. В дальнейшем будет рассмотрен вопрос о возможности перехода на систему счисления 2^{32}, в данной работе этот алгоритм не рассматривается.

Для оценки вычислительной сложности алгоритмов все авторы используют число элементарных операций, которые необходимо выполнить. При этом не учитывается ни возможность выполнения самих операций параллельно за счет суперскалярности процессора, ни возможность выполнения отдельных функций параллельно, ни свойства кэша. Учет этих особенностей

[1] Обозначение log*n означает loglog…log n раз по количеству рекурсий, которые необходимо выполнить

может существенно влиять на вычислительную сложность, как в сторону уменьшения, так и в сторону увеличения.

Вопросы параллельного выполнения операции умножения рассматривались и ранее. В связи со спецификой вычислительных средств этот вопрос рассматривался в условиях использования распределенных вычислений для кластерных систем [4]. В работе [5] предложен параллельный алгоритм умножения, основанный на предвычислениях, который приводит к значительному увеличению количества циклов.

Цель данной работы – исследование основных методов умножения для чисел многократной точности с точки зрения возможности их параллельного выполнения.

2. Математическая постановка задачи

Даны 2 числа:

$X = \sum_{i=0}^{n-1} x_i * B^i$ $\qquad$ $Y = \sum_{i=0}^{n-1} y_i * B^i$,

где

B - основание системы счисления: x_i, y_i – "цифры" сомножителей.

Для современных процессоров $B=2^{32}$ или 2^{64}. Все экспериментальные данные получены при $B=2^{32}$.

Необходимо вычислить $Z=X * Y$, обеспечив минимальную вычислительную сложность за счет параллельного выполнения на многоядерном процессоре.

3. Определение показателей эффективности для алгоритмов умножения

3.1 Вычисление в «столбик»

3.1.1 Алгоритм 1 (A1)

1. Обнуление произведения.
2. Для всех «цифр» второго сомножителя
    a. Вычисление произведения на первый сомножитель
    b. Сложение с текущим значением произведения.

Для простоты будем считать, что количество "цифр" для обоих сомножителей одинаково и равно n. В этом случае потребуется n операций умножения n-разядного числа на одно разрядное и сложение полученных результатов. Обозначим время выполнения операции умножения t_m, операции сложения t_a. Время выполнения алгоритма вычисления в столбик в последовательном режиме составляет:

$$T_{A1} = n^2 * (t_m + t_a) \qquad (1)$$

3.1.2 Алгоритм 2 (A2). «Быстрый столбик» [6]

Для всех цифр произведения вычисляем значение цифры по формуле:

$$Z_s = \sum_{i=0}^{i=s} x_{s-i} * y_i; \qquad s = 0..n-1;$$
$$Z_s = \sum_{i=s-n+1;k=n-1}^{i=n-1,k=n-i} x_k * y_i; \quad s = n..2n-1; \qquad (2)$$

Фактически «быстрый» столбик не что иное, как вычисление значения свертки для всех цифр результата, если «цифры» - это коэффициенты многочлена.

Для этого алгоритма необходимо выполнить такое же количество операций сложения и умножения, как для Алгоритма 1, но в другом порядке поэтому $T_{A2} = T_{A1}$.

Так как количество ядер для современных процессоров небольшое, а цель данной работы – дать практические рекомендации по использованию параллельных вычислений при умножении длинных чисел, в работе не рассматривается вариант неограниченного параллелизма, т.е. бесконечного числа параллельных ветвей.

Пусть число параллельных ветвей p, причем $p < n/2$ и n кратно p.

3.1.3 Параллельный алгоритм 1 (PA1).

Для обеспечения максимальной грануляции и равномерного распределения нагрузки разделим один из сомножителей на порции равной длины. В этом случае каждый поток может выполняться параллельно. Если не учитывать накладных расходов, связанных с использованием потоков, то общее время выполнения PA1 равно:

$$T_{PA1} = n^2 * (t_m + t_a) / p \quad (3)$$

Значения ускорения и эффективности соответственно равны:

$$S_{PA1} = \frac{T_{A1}}{T_{PA1}} = p; \quad E_{PA1} = \frac{S_{PA1}}{p} = 1. \quad (4)$$

3.1.4 Параллельный алгоритм 2 (PA2)

Так как сложность вычисления «цифры» зависит от ее номера, будем использовать динамический режим распределения итераций цикла. В этом случае потребуется специальный механизм учета переносов.

3.3 Алгоритм Карацубы и Офмана (KO)

3.3.1 Последовательный алгоритм (KO)

Для определения числа операций запишем формулы для вычисления: значения произведения по этому методу. Пусть XL - младшая часть первого сомножителя, а XH - его старшая часть, т.е $X = XH * B^{n/2} + XL.$. Пусть YL - младшая часть первого сомножителя, а YH - его старшая часть, т.е $Y = YH * B^{n/2} + YL.$. Обозначим ZL, ZH младшую и старшую часть произведения *Z*. Тогда:

$$Z = C * B^n + E * B^{n/2} + A \quad (5)$$

где:

$$A = XL * YL; \quad C = XH * YH; D = (XL + XH) * (YL + YH); \quad E = D - (A + C). \quad (6)$$

Заметим, что значение коэффициента E не отрицательно. В работе [7] предложена формула вычисления значения произведения:

$$Z = (B^n + B^{n/2})XH*YH + B^{n/2}* (XH-XL)*(YL-YH) + (B^{n/2}+1)*XL*YL, \quad (7)$$

Формула (7) эквивалентна (5), но второе слагаемое в (7) может быть как положительным, так и отрицательным, первое и третье слагаемое вычисляется сложнее (дополнительные операции сложения), поэтому в дальнейшем рассматривается формула (5). Для исключения рекурсии для вычисления *A, C, E* используем умножение «в столбик» (*A1*). Общее время выполнения в последовательном режиме равно:

$$T_{KO} = (3n^2/4 + n + 1)t_m + (3n^2/4 + 11n/2 + 6)t_a. \quad (8)$$

3.3.1 Параллельный алгоритм (PKO)

Схема параллельного вычисления для $p \geq 4$ представлена в таблице 2.

Таблица 2. Параллельное вычисление произведения. Алгоритм Карацубы Хофмана

Ядро процессора			
1	2	3	4
$ZL = XL * YL$	$ZH = XH * YH$	$XL + XH$	$YL + YH$
$ZL + ZH$		$D = (XL + XH) * (YL + YH)$	
$E = D - (ZL + ZH)$			
$Z += E$			

Операции, которые выполняются параллельно, определены в одной строке таблицы. В этом случае время выполнения параллельного алгоритма равно:

$$T_{PKO}^4 = (n^2/4 + n + 1)t_m + (n^2/4 + 5n + 5)t_a \quad . \qquad (9)$$

Ускорение и эффективность для алгоритма Карацубы и Офмана для $p \geq 4$:

$$S_{PKO}^4 = \frac{T_{KO}}{T_{PKO}^4} = \frac{(3n^2/4 + n + 1)t_m + (3n^2/4 + 11n/2 + 6)t_a}{(n^2/4 + n + 1)t_m + (n^2/4 + 5n + 5)t_a}, E_{PKO}^4 = S_{PKO}^4/4; \qquad (10)$$

Для 2-х ядер соответствующие формулы для времени выполнения и эффективности:

$$T_{PKO}^2 = (n^2/2 + n + 1)t_m + (n^2/2 + 4n + 4)t_a$$

$$S_{PKO}^2 = \frac{T_{KO}}{T_{PKO}^2} = \frac{(3n^2/4 + n + 1)t_m + (3n^2/4 + 11n/2 + 6)t_a}{(n^2/2 + n + 1)t_m + (n^2/2 + 4n + 4)t_a}, E_{PKO}^2 = S_{PKO}^2/2 \qquad (11)$$

3.4 Алгоритм Шёнхаге и Штрассена (SS). Последовательный и параллельный режимы

Этот метод основан на замене умножения больших чисел на умножение полиномов, для вычисления которых используется дискретное преобразование Фурье. Для вычисления каждой цифры полинома достаточно в качестве модуля использовать число, больше или равное $2nB^2$. Для того чтобы не выполнять вычисления с числами, превосходящими 2^{64}, будем использовать простые числа, произведения которых превосходят заданный модуль. Пусть максимальное число бит сомножителя $n_{Bits} = 16384$[2]. Такая длина выбрана из практических соображений. В криптографии числа большей длины не используются. Минимальный размер модуля, достаточный для таких сомножителей, равен 2^{74}. Этот модуль гарантируется произведением трех простых чисел, меньших, чем 2^{32}, но близких к этому значению. Выбор простых чисел гарантирует наличие корней и обратного элемента. Так как значение модуля зависит только от длины сомножителей, эти числа могут быть заранее вычислены для каждой из используемых длин.

3.4.1 Алгоритм SS (Последовательный режим)[7]

1. Предвычисления (вычисление простых составляющих модуля, их инверсий, корней и их степеней, и коэффициентов для обратного преобразования числа из смешанной системы счисления).
2. Для всех простых чисел
   a. приведение первого сомножителя по модулю;
   b. формирование преобразования Фурье для 1 сомножителя;
   c. приведение второго сомножителя по модулю;
   d. формирование преобразования Фурье для 2 сомножителя;
   e. покомпонентное умножение по модулю;
   f. формирование обратного преобразования Фурье.
3. Преобразование для получения результата

[2] Современные криптографические алгоритмы не используют больших длин

В виду громоздкости общей формулы для определения времени выполнения, она не приведена. Рассмотрим способы параллельного вычисления для алгоритма SS. При наличии не менее трех ядер каждое ядро может выполнить полный цикл обработки для одного простого числа. Так как предварительная обработка данных не учитывается, а последовательная часть программы в этом случае невелика, то ожидаемое ускорение приблизительно равно 3. Для двух ядерного процессора первое ядро выполняет преобразования для первого и второго простого числа, а второе – для третьего. В этом случае ожидаемое ускорение примерно равно 1.5

4. Реализация алгоритмов умножения

Для реализации описанных выше алгоритмов использовался процессор Intel (R) Core (TM)2 Duo CPU E6850 @3.00GHz. Среда разработки: Microsoft Visual C++ 2008.Компиляторы: Intel(R) C++ Compiler Professional Edition 11.1, Microsoft Visual C++ 2008. Для обоих компиляторов использовался режим оптимизации по времени. Среды для разработки параллельных программ: Open MP, версия 3.0 (май 2008), TBB 3.0. Были реализованы функции для всех описанных выше алгоритмов в последовательном и параллельном режимах. Для сравнения временных характеристик в качестве эталонной библиотеки использовалась библиотека Miracl[3] (версия 5.4.3 от 29.10.10). Обе библиотеки откомпилированы в режиме без использования ассемблерных вставок. Все функции реализованы для сомножителей одинаковой длины в диапазоне от 512 до 16384 бита. Длина сомножителей в таблицах задается в 32-битных словах. В таблицах результаты определяют время выполнения функций (секунды). Результаты приведены в таблицах 3-6.

Таблица 3. Вычисление «в столбик»

Len	Intel(R) C++ Compiler				Visual C++			
	Miracl	A1	PA1, Open MP	PA1, TBB	Miracl	A1	PA1, Open MP	PA1, TBB
16	2.79E-06	2.23E-06	2.23E-06	6.12E-05	3.91E-06	3.07E-06	3.35E-06	7.29E-05
32	6.94E-06	4.75E-06	5.31E-06	7.57E-05	8.94E-06	7.82E-06	7.26E-06	6.03E-05
64	2.43E-05	1.53E-05	1.68E-05	8.41E-05	3.10E-05	2.74E-05	2.23E-05	9.92E-05
128	9.33E-05	5.64E-05	6.22E-05	0.00012	0.000119	0.000109	8.18E-05	0.000151
256	0.00037	0.00022	0.00024	0.00033	0.000472	0.000415	0.00032	0.000318
512	0.00154	0.00085	0.00098	0.00113	0.00187	0.0017	0.00126	0.001037

Таблица 4. «Быстрый столбик»

Len	Intel(R) C++ Compiler			Visual C++		
	A1	A2	PA2,Open MP	A1	A2	PA2,Open MP
16	2.23E-06	4.75E-06	1.54E-05	3.07E-06	2.79E-06	8.10E-06
32	4.75E-06	5.59E-06	1.62E-05	7.82E-06	6.42E-06	1.03E-05
64	1.53E-05	5.03E-06	1.20E-05	2.74E-05	1.87E-05	1.93E-05
128	5.64E-05	5.59E-05	3.88E-05	0.000109	6.42E-05	4.55 E-05
256	0.00022	0.000209	0.000142	0.000415	0.00024	0.00015
512	0.00085	0.000814	0.00055	0.0017	0.00094	0.00057

Таблица 5. Алгоритм Карацубы и Офмана

Len	Intel(R) C++ Compiler			Visual C++		
	A1	KO	PKO,Open MP	A1	KO	PKO,Open MP
16	2.23E-06	2.79E-06	4.47E-06	3.07E-06	3.63E-06	9.77E-06
32	4.75E-06	5.59E-05	5.59E-06	7.82E-06	6.42E-06	1.42E-05
64	1.53E-05	1.54E-05	1.23E-05	2.74E-05	1.65E-05	1.98E-05
128	5.64E-05	5.17E-05	3.72 E-05	0.000109	5.59 E-05	4.61E-05
256	0.00022	0.000208	0.000137	0.000415	0.000208	0.000150
512	0.00085	0.000751	0.000510	0.0017	0.000802	0.000542

[3] http://www.shamus.ie/index.php?page=Downloads

Таблица 6. Алгоритм Шёнхаге и Штрассена

Len	Intel(R) C++ Compiler			Visual C++		
	Miracl (SS)	SS	PSS, Open MP	Miracl (SS)	SS	PSS, Open MP
16	3.88E-05	3.57E-05	3.24E-05	3.94E-05	3.6E-05	3.18E-05
32	8.46E-05	7.23E-05	5.11E-05	8.57E-05	7.96E-06	5.89E-05
64	0.000185	0.000160	0.000109	0.000194	0.000176	0.000125
128	0.000409	0.000357	0.000241	0.000433	0.000392	0.000267
256	0.000915	0.000786	0.000531	0.000957	0.000866	0.000587
512	0.00202	0.00174	0.00116	0.00210	0.0014	0.00127

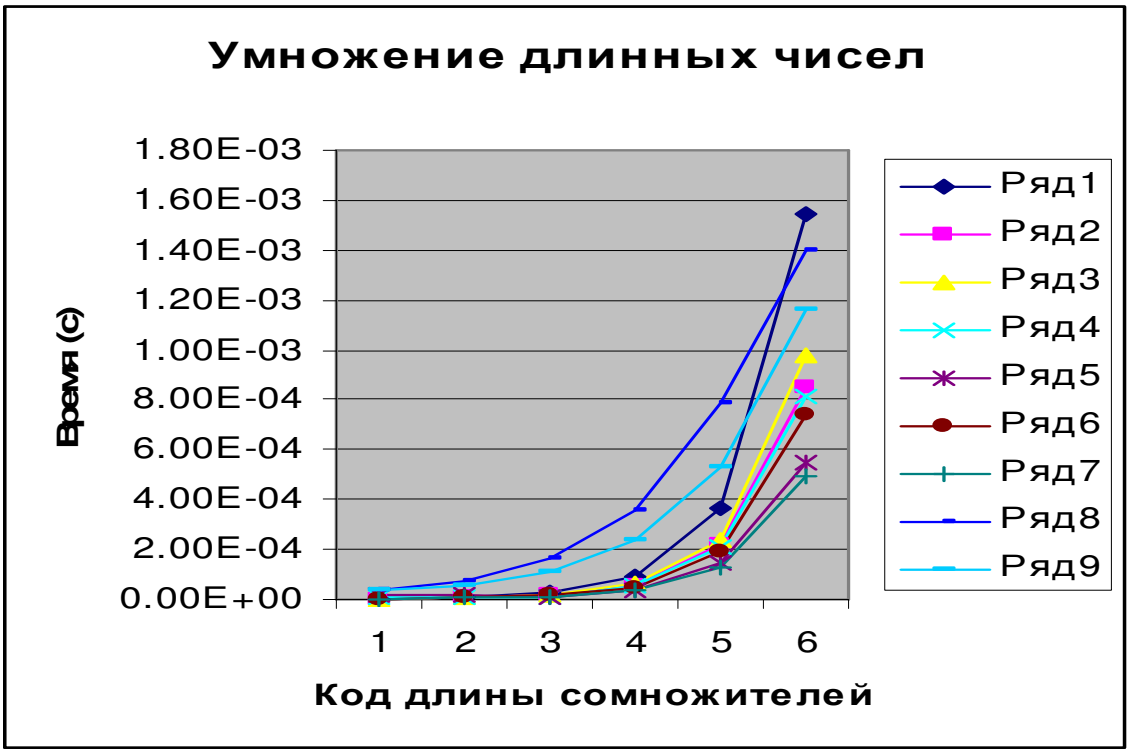
Как следует из таблицы 3, разработанная библиотека (алгоритм A1) более эффективна, чем эталонная (Miracl). Ускорение составляет от 32 до 62% для компилятора Intel C++ и от 13 до 78% для компилятора Visual C++. Сравнение эффективности кодов, построенных с помощью двух компиляторов, показывает, что первый компилятор строит более эффективный код (ускорение составляет от 9 до 63%). Параллельное выполнение кода, реализованное с помощью Open MP и TBB, не дает выигрыша. Так как использование TBB не дает выигрыша по сравнению с Open MP для этого типа задач, в дальнейшем результаты использования этой технологии для рассматриваемого класса алгоритмов не приводятся.

Как следует из таблицы 4, Алгоритм A2 работает медленнее алгоритма A1 для всех длин. И, хотя параллельный режим (алгоритм PA2), позволяет уменьшить время выполнения, этот метод продолжает быть медленнее, чем A1.

Метод Карацубы и Офмана (таблица 5) опережает обычный метод на длине 64 слова (2048 бит). Параллельный режим позволяет получить ускорение до 50%.

Алгоритм Шёнхаге и Штрассена (Таблица 6) проигрывает даже простому столбику на всех рассмотренных длинах. Наилучшим сточки рения вычислительной сложности является первый алгоритм для длин сомножителей меньше 2048 и параллельный алгоритм Карацубы и Офмана для всех остальных длин.

Для всех алгоритмов компилятор Intel C++ создает более эффективный код. На рисунке 1 представлены графики зависимостей времени вычислений от длины сомножителей для компилятора Intel C++ для всех рассмотренных выше алгоритмов.



Ряд 1 соответствует функции умножения («в столбик») базовой библиотеки (Миракл). Эта функция менее эффективна по сравнению со всеми функциями, кроме варианта использования функции для алгоритма Шёнхаге и Штрассена для длины сомножителей не более 8192.

Ряды 2 и 3 соответствуют функции умножения («в столбик»), последовательному и параллельному вариантам соответственно. Эффективности обоих методов практически совпадают и выше эффективности функции умножения (ряд 1)..

Ряды 4 и 5 соответствуют функции умножения («быстрый столбик»), для последовательного и параллельного режимов. Эти графики находятся ниже графиков для рядов 2, 3 для всех длин.

Ряды 6 и 7 соответствуют алгоритму Карацубы и Офмана (последовательный и параллельный режим). Этому алгоритму соответствует минимальная вычислительная сложность для длин сомножителей не менее 2048 бит.

Ряды 8 и 9 – графики для алгоритма Шёнхаге и Штрассена. Эти графики пересекают график для функции умножения (Миракл) для длины сомножителей 16384 для последовательного режима и для длины 8192 для параллельного режима.

Ожидаемые зависимости скорости вычислений от длин не совпадают с полученными. Так, для последовательного «столбика» ожидается зависимость (1), а получается близкая к (3). Это связано с суперскалярностью современных процессоров, когда одновременно выполняется несколько команд. Тоже для других алгоритмов.

Литература

1. Карацуба А., Офман Ю. Умножение многозначных чисел на автоматах // Доклады Академии Наук СССР. — 1962. — Т. 145. — № 2
2. Arnold Schönhage and Volker Strassen, Schnelle Multiplikation grosser Zahlen, Computing 7 (1971), 281{292
3. Martin Förer. Faster integer multiplication. Proceedings of the 39th ACM Symposium on Theory of Computing, pages 57–66, 2007.
4. *Fagin B. S.* Large integers multiplication on massively parallel processors. http://barryfagin.name/Papers/FMPC90.htm.
5. Efficient Parallel Multiplication Algorithm for Large Integers. http://www.springerlink.com/content/hqup5uxp5np2dhmy/fulltext.pdf
6. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии. http://window.edu.ru/window_catalog/files/r23845/book.pdf
7. Г. Нуссбаумер. Быстрое преобразование Фурье и алгоритмы вычисления сверток http://www.toroid.ru/nussbaymerG.html
8. Кнут Д. Искусство программирования для ЭВМ. т. 2. Получисленные алгоритмы. Мир, М. 1977., с. 314.

Модель и технология интеграции online-сервисов эксперимента ATLAS на Большом Адронном Коллайдере (БАК) и сервисов ГРИД-инфраструктуры

В.В. Кореньков, В.М. Котов, Н.А. Русакович, А.В. Яковлев

Объединенный институт ядерных исследований

В статье рассмотрены модель и технология интеграции сервисов реального времени из набора online-сервисов в ATLAS Control Room и сервисов передачи данных FTS (File Transfer System) ГРИД-инфраструктуры. Прототип системы удаленного доступа реального времени (СУДРВ), разработанный в ОИЯИ, обеспечивает доступ к данным, поступающим в ATLAS Control Room для проведения экспресс-анализа качества данных в ГРИД-системе обработки для экспериментов на Большом Адронном Коллайдере (БАК). Описаны структура и алгоритмы работы интегрированного комплекса, методы организации и управления распределенными потоками данных.

1. Введение

Основной проблемой современных крупномасштабных интернациональных проектов в области фундаментальной науки является географическая распределённость участников проектов, которым необходимо обеспечить удаленный доступ к экспериментальным установкам и информационно-вычислительным ресурсам для обработки экспериментальных данных.

Важной составляющей такого участия в обработке и анализе данных, получаемых в экспериментах на БАК является создание в Объединённом институте ядерных исследований (ОИЯИ) г. Дубна системы удалённого доступа реального времени (СУДРВ) и интеграция ее в глобальную сервис-ориентированную архитектуру ГРИД-системы сбора и обработки данных экспериментов на БАК.

Функционал СУДРВ должен обеспечивать не только удаленный мониторинг к процессам сбора и обработки данных, но и предоставлять условия для участия физиков ОИЯИ в географически распределенной системе анализа качества данных, полученных в ходе экспериментов на БАК.

СУДРВ должен удовлетворять следующим критериям:

- предоставлять безопасный и защищенный доступ к информации как в ATLAS Control Room (ACR), так и в других операционных центрах ATLAS в ЦЕРНе (Satellite Control Room (SCR));
- обеспечивать контроль доступа в режиме реального времени и управления ресурсами, для того, чтобы предотвратить любое вмешательство удаленного доступа в операционную деятельность системы сбора и обработки данных эксперимента ATLAS в ЦЕРНе;
- предоставлять инструментарий для совместной работы, как удаленных операторов в ACR, так для участия в распределенной системе анализа качества данных, полученных в ходе эксперимента ATLAS;

Кроме того, данный сервис может быть использован при обучении и предварительной подготовке сотрудников ОИЯИ для работы в эксперименте ATLAS в ЦЕРНе.

2. Модель композитного сервиса

СУДРВ как композитный сервис использует технологии и сервисы, входящие в состав мониторинга и управления обработкой на разных стадиях сбора и обработки данных эксперимента ATLAS, а также ГРИД сервисы доступа к данным и их передачи в СУДРВ ОИЯИ.

Модель композитного сервиса представлена на рис.1.

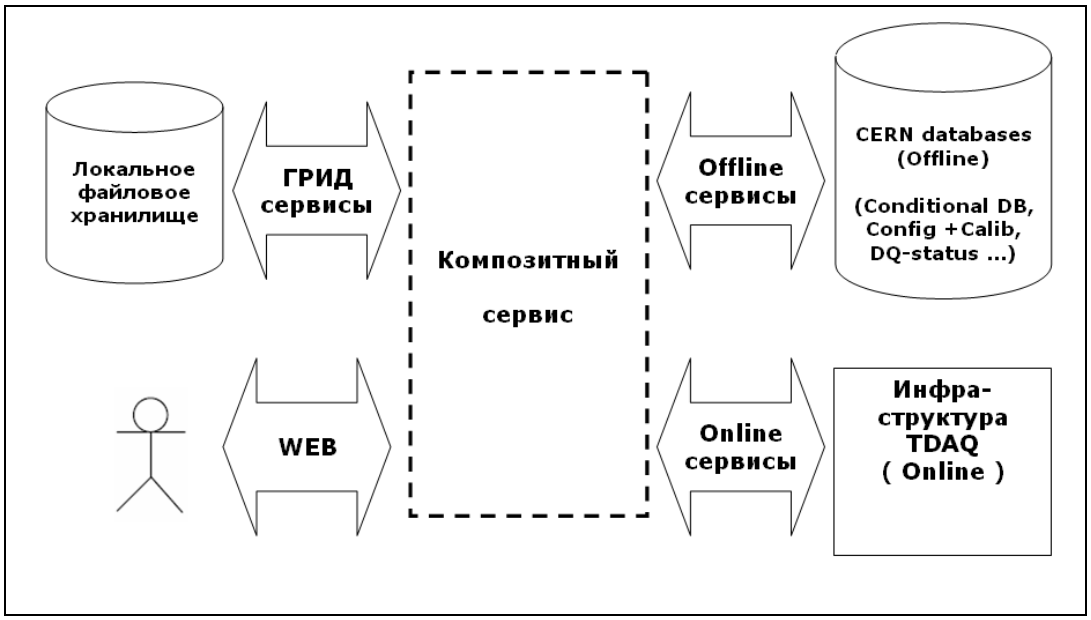


Рис. 1. Модель композитного сервиса

В качестве источников экспериментальных данных, получаемых в ходе текущего сеанса, поступающих в ACR в режиме реального времени выступают online - сервисы системы сбора и обработки данных TDAQ ATLAS, и прежде всего сервис WebIS, который обеспечивает удаленный доступ в режиме реального времени. Для доступа к данным об условиях проведения сеанса: конфигурации детекторов и подсистем, поправок магнитного поля, данные калибровки используются сервисы баз данных эксперимента ATLAS (ConfDB, ConditionDB),

Для передачи данных измерения в СУДРВ ОИЯИ для анализа их качества используются сервисы ГРИД (FTS), поэтому интеграция сервисов реального времени TDAQ и сервисов ГРИД является одной из основных функций и особенностью СУДРВ как композитного (составного) сервиса и позволяет организовать объединение вычислительных и программных ресурсов для решения задач в форме композитных приложений, необходимых для дальнейшей обработки и анализа качества данных измерения в оперативном режиме, близком к режиму реального времени.

3. Технологии композитного сервиса Remote Data Aggregator (RDA)

Основным содержательным элементом композитного сервиса представленной модели является Remote Data Aggregator(RDA). Технологии и сервисы, используемые при создании RDA, можно условно разделить на внешние и внутренние.

3.1 Внешние сервисы

В качестве внешних сервисов RDA используются сервисы системы сбора и обработки данных TDAQ ATLAS и сервисы ГРИД инфраструктуры экспериментов на БАК ЦЕРН.

File Transfer Service (FTS) – сервис передачи данных, реализованный в рамках проекта WLCG (ГРИД). Сервис FTS обеспечивает надёжность передачи данных и взаимодействие между остальными сервисами управления передачей данных ГРИД – инфраструктуры экспериментов, необходимых для передачи данных экспресс-анализа качества из ЦЕРНа в локальное файловое хранилище СУДРВ ОИЯИ.

Схема взаимодействия RDA с сервисами FTS приведена на рис.2.

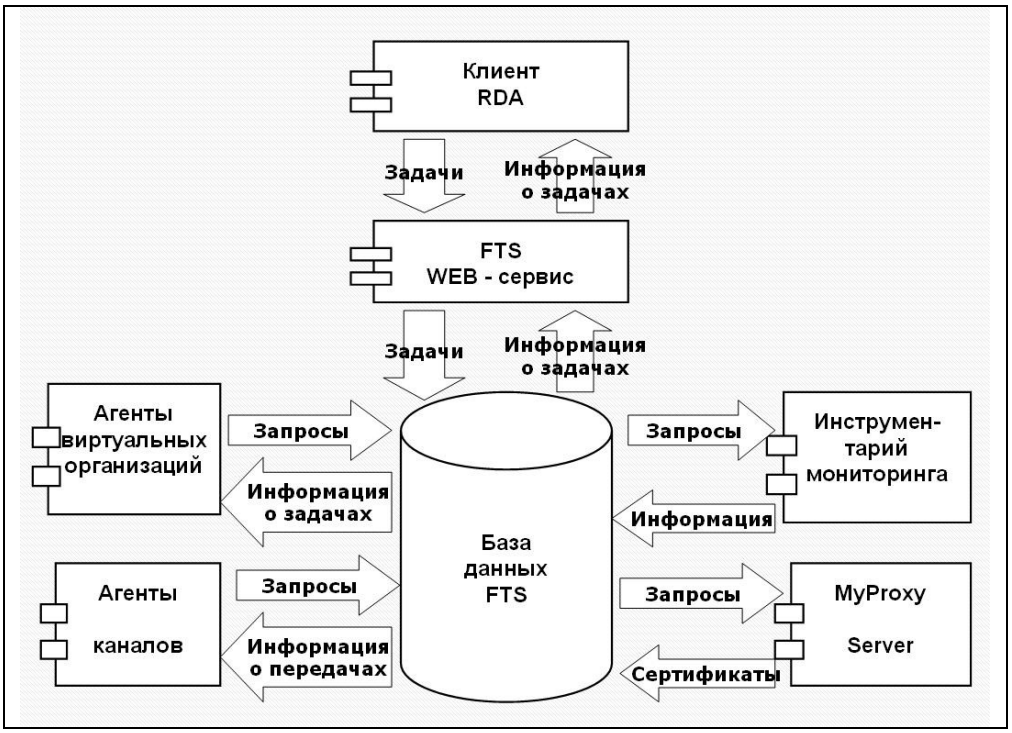


Рис. 2. Схема взаимодействия RDA с сервисами FTS

WebIS - сервис удаленного мониторинга обеспечивает доступ к сервису информационного обмена (Information service (IS)) online-сервисов системы сбора и обработки данных TDAQ ATLAS и предназначен для передачи в режиме реального времени информации о процессах сбора и обработки данных. IS является информационной шиной межпроцессорного обмена специализированного промежуточного программного обеспечения TDAQ.

Сервис WebIS позволяет удаленному пользователю осуществлять интерактивные запросы для доступа к информации, поступающей с внутренних IS-серверов TDAQ, и является эффективным средством при разработке композитных приложений удаленного мониторинга.

Data Quality Monitoring Framework (DQMF) - является инструментарием для разработки композитных приложений по анализу качества данных в рамках TDAQ ATLAS. DQMF взаимодействует с сервисами online мониторинга и сервисами конфигурирования и управления инфраструктуры TDAQ ATLAS.

RDA использует DQMF для формирования запросов на выборку и получение данных необходимых удаленному пользователю для проведения анализа качества данных, обеспечивая их полноту и непротиворечивость.

Взаимодействие компонентов DQMF с сервисами TDAQ приведено на рис.3.

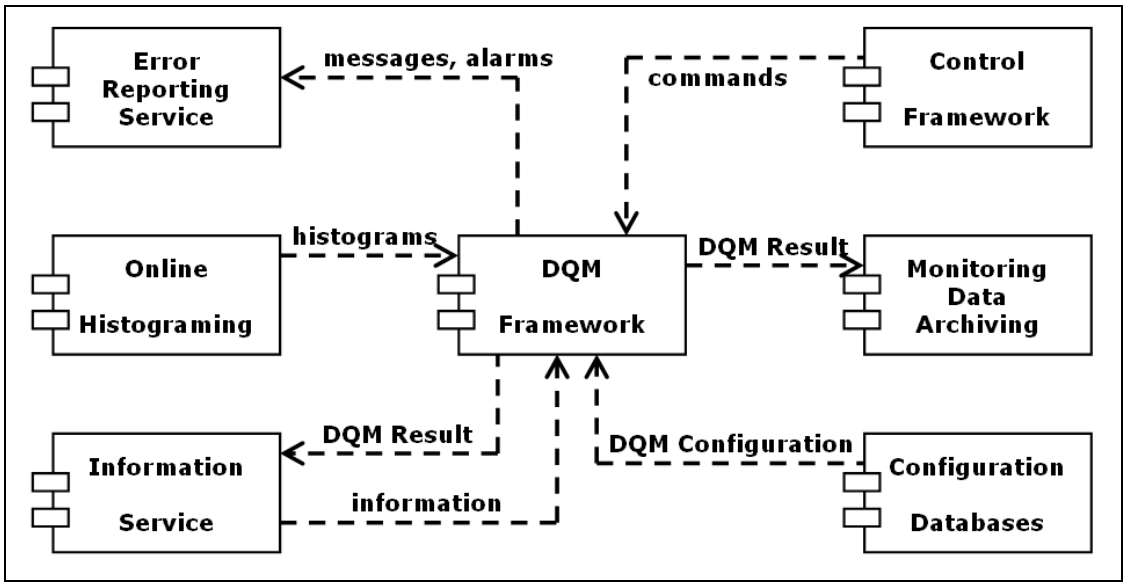


Рис. 3. Взаимодействие компонентов DQMF с сервисами TDAQ

3.2 Внутренние модули (сервисы)

Внутренними модулями (сервисами) являются программные компоненты, разработанные непосредственно для RDA.

Планировщик заданий

Планировщик заданий является центральным модулем сервиса RDA и обеспечивает координацию взаимодействия модулей и сервисов RDA. Планировщик осуществляет маршрутизацию заданий (запросов), полученных от пользователя и сопровождение их в течение всего хода обработки запросов. Планировщик установлен на аппаратном обеспечении инфраструктуры СУДРВ в ОИЯИ.

Сервис обработки запросов

Задачей данного сервиса является взаимодействие с пользователем. Сервис предоставляет пользователю web-интерфейс, позволяющий осуществлять запрос (выбор) требуемых данных для последующей передачи. Сервис установлен на аппаратном обеспечении инфраструктуры СУДРВ в ОИЯИ.

Сервис сбора данных

Обеспечивает сбор данных из различных источников инфраструктуры эксперимента на БАК ЦЕРН, предварительную обработку и их упаковку для передачи при помощи сервисов ГРИД в СУДРВ ОИЯИ. Сервис сбора данных установлен на аппаратном обеспечении, находящемся в общедоступной сети в ЦЕРНе (CERN Public Network).

Сервис обработки данных

Задачей данного сервиса является взаимодействие с сервисом FTS распаковка данных и запись локальное в файловое хранилище СУДРВ, для дальнейшей обработки. Сервис обработки данных установлен на аппаратном обеспечении инфраструктуры СУДРВ в ОИЯИ.

Сервис-ориентированная архитектура компонентов RDA, интегрированная в структуру СУДРВ приведена на рис.4.

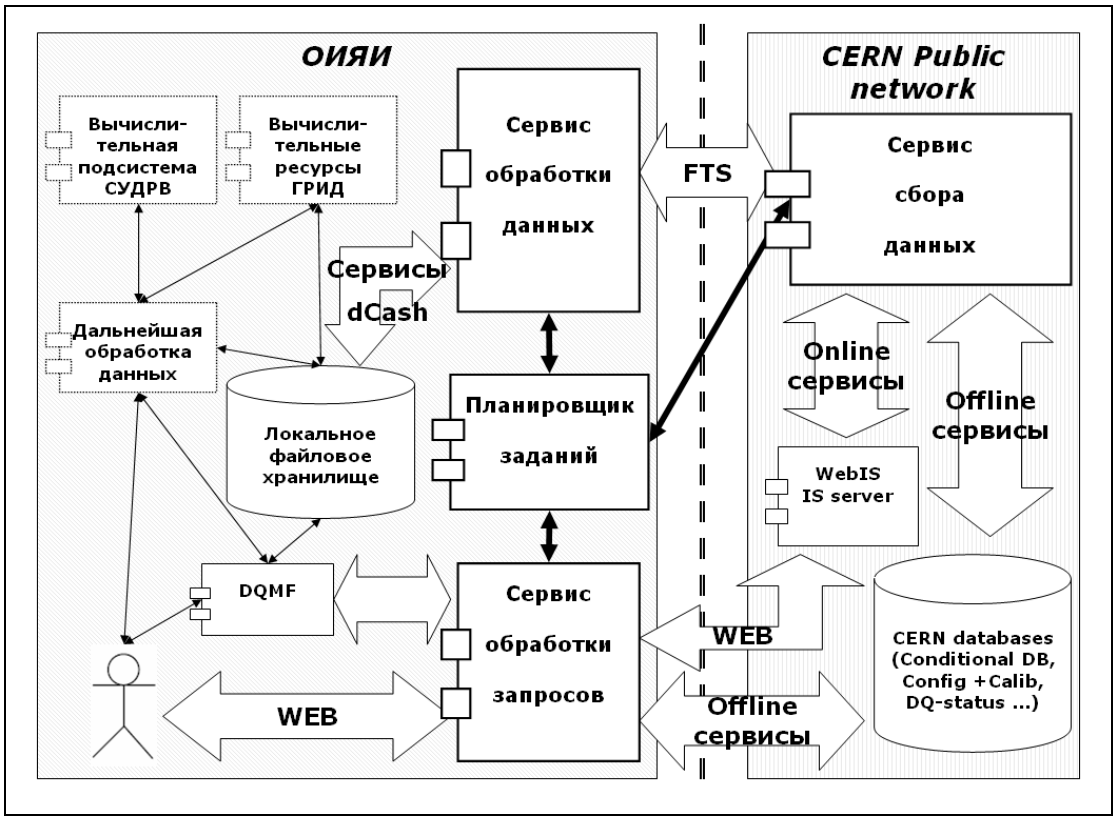


Рис. 4. Сервис-ориентированная архитектура компонентов RDA, интегрированная в структуру СУДРВ

3.3 Алгоритм работы композитного сервиса RDA

Пользователь через web-интерфейс обращается к сервису обработки запросов для выбора необходимых данных. При формировании запроса задается конфигурация набора данных (Dataset configuration) – список всех данных, которые соответствуют определенному событию или группе событий эксперимента ATLAS, необходимых для последующей обработки в СУДРВ ОИЯИ. Каждый вид обработки требует определенного набора данных, и при этом возможна ситуация, когда одному событию данных измерения будут соответствовать несколько разных наборов данных. Необходимость тех или иных дополнительных данных в наборе зависит от типа обработки, для которой этот набор подготовлен. Поэтому конфигурация содержит также и описание параметров источников данных, форматов, местонахождения данных, а также способов доступа к ним. При формировании запроса, данные, включенные в конфигурацию, проверяются с помощью сервисов DQMF на полноту и непротиворечивость.

Сформированный запрос поступает в очередь заданий Планировщика и передается в Сервис сбора данных. Пользователь через web-интерфейс может контролировать ход его выполнения.

Сервис сбора данных, в соответствии с заданной в запросе конфигурацией, осуществляет сбор данных из различных источников, а также обработку и упаковку полученных данных. Для доступа к данным, поступающим в режиме реального времени в ACR, используется сервис WebIS, который обеспечивает интерактивный доступ к этим данным в режиме реального времени. Для доступа к offline -данным (данные о конфигурации детекторов и подсистем, данные об условиях проведения сеанса) используются сервисы баз данных (ConfDB, ConditionDB). Упакованные данные передаются при помощи сервисов ГРИД (FTS) в Сервис обработки данных.

Сервис обработки данных осуществляет обработку данных, распаковку и подготовку к записи в локальное файловое хранилище. Для записи полученных данных в локальное файловое хранилище используются сервисы dCash.

4. Заключение

Сервис-ориентированная архитектура композитного сервиса интегрирует основные сервисы инфраструктуры СУДРВ (online - сервисы TDAQ Software и сервисы ГРИД), обеспечивая удаленного пользователя эффективным инструментом для дальнейшей обработки и анализа качества данных, получаемых в эксперименте ATLAS. Такой подход полностью соответствует концепции удаленного мониторинга эксперимента ATLAS, и обеспечивает участие физиков ОИЯИ в географически распределенной системе мониторинга и анализа качества данных в режиме реального времени.

Успешное выполнение данной работы было бы невозможно без поддержки и помощи: Livio Mapelli CERN Serguei Kolos, CERN, Petersburg Nuclear Physics Institute (PNPI); Igor Soloviev, CERN, Petersburg Nuclear Physics Institute (PNPI), и сотрудников ATLAS DAQ Group CERN Physics Dept.

Литература

1. Mapelli L. The ATLAS Data Acquisition system: Why and How we did it like that. // XXII International Symposium on Nuclear Electronics & Computing NEC'2009. –P.44.
2. V. Kotov, N. Russakovich. Development of the SYSTEM REMOTE ACCESS REAL TIME (SRART) at JINR for monitoring and quality assessment of data from the ATLAS LHC experiment // XXII International Symposium on Nuclear Electronics & Computing NEC'2009. –P.37.
3. А. Ужинский, В. Кореньков. Архитектура сервиса передачи данных в grid // Журнал "Открытые системы", 2008 г. N2, URL: http://www.osp.ru/os/2008/02/4926522/
4. 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.

Автоматизация создания вычислительных программ для современных кластеров*

В.А. Крюков

ИПМ им. М.В. Келдыша РАН

В докладе будут рассмотрены три направления автоматизации параллельного программирования:
1. Использование гибридных языков, объединяющих разные модели параллельного программирования;
2. Использование языков с неявным параллелизмом;
3. Автоматизация преобразования имеющихся последовательных программ или параллельных MPI-программ в эффективные параллельные программы на гибридных языках или языках с неявным параллелизмом.

1. Введение

При разработке программ для многоядерных кластеров программисту приходится использовать две сильно различающиеся модели программирования (MPI, OpenMP) и соответствующие инструментальные средства. Появление в узлах кластера графических процессоров серьезно усложнило разработку программ, поскольку потребовало дополнительно использовать еще и низкоуровневую технологию CUDA [1]. На подходе новые процессоры с большим количеством ядер (например, 48-ядерный Intel SCC процессор, который стал недавно доступен для широкого круга исследователей [2]), для эффективного использования которых потребуются новые модели программирования.

Разработчики новых языков параллельного программирования явно не успевают отслеживать архитектурное разнообразие многоядерных процессоров. Очень маловероятно, что в течение 5-10 лет появится и широко распространится новый высокоуровневый язык. Поэтому в ближайшие годы программистам придется использовать гибридные языки, объединяющие разные модели параллельного программирования.

Однако такой подход может рассматриваться только в качестве временной меры - нельзя и дальше усложнять модели параллельного программирования. Если в долгосрочном плане можно надеяться на появление и стандартизацию новых языков программирования высокого уровня, то в среднесрочном плане наиболее привлекательным подходом представляется использование для создания параллельных программ языков с неявным параллелизмом (Фортран, Си, Си++). Вполне вероятно, что такие языки надолго останутся наиболее удобными языками для создания вычислительных программ, алгоритмы которых, как правило, не требуют для своего выражения привлечения механизмов параллельных процессов и средств их взаимодействия.

Независимо от того, какой из упомянутых подходов будет использоваться программистами, разработка параллельных программ намного бы упростилась и ускорилась, если бы им были предоставлены средства автоматизации преобразования имеющихся у них последовательных программ (и уже широко распространенных параллельных MPI-программ) в эффективные параллельные программы для современных кластеров.

2. Использование гибридных моделей и языков

В настоящее время при разработке программ для высокопроизводительных вычислений на современных кластерах широко используются три модели программирования – MPI, OpenMP и CUDA. При этом пока вполне можно обходиться комбинацией MPI/OpenMP или MPI/CUDA, поскольку GPU используют для тех программ, для которых они на порядок эффективнее, чем

* Работа поддержана программой Союзного государства СКИФ-ГРИД, программами Президиума РАН №14 и №15, грантом РФФИ № 10-07-00211, грантом Президента РФ МК-3324.2010.9.

многоядерные процессоры, и, следовательно, неполной загрузкой многоядерных процессоров можно пренебречь. Если будут появляться программы, в которых только часть вычислений выгодно производить на GPU, а оставшиеся вычисления лучше оставить на универсальном многоядерном процессоре, то придется использовать и комбинацию из трех перечисленных моделей. Технически объединять низкоуровневые модели программирования, реализованные через библиотеки, проще, чем высокоуровневые модели, реализуемые посредством языков и соответствующих компиляторов. Но 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 [3]. Поэтому важно создавать высокоуровневые гибридные модели и языки программирования.

Примером такой модели является гибридная модель DVM/OpenMP [4] разработанная в ИПМ им. М.В.Келдыша РАН. Она объединяет две модели (DVM [5] и OpenMP), реализуемые через расширения стандартных языков директивами компилятору. Недавно аналогичная модель (PGI_APM [6]) появилась и для GPU, ориентированная на ускорители разной архитектуры (в частности, для GPU NVIDIA и AMD). Все эти три модели (DVM, OpenMP, PGI_APM) объединяет и то, что программист может полностью контролировать отображение данных и вычислений на аппаратуру. Поэтому вполне естественно создать гибридные модели DVM/PGI_APM и DVM/OpenMP/PGI_APM и соответствующие языки и компиляторы. Это не только упростит дальнейшую разработку автоматически распараллеливающих компиляторов, но и предоставит возможность использовать эти гибридные языки для некоторого промежуточного представления распараллеленной программы, на котором при необходимости программист сможет проводить дополнительную ручную оптимизацию программы.

3. Использование языков с неявным параллелизмом

В настоящее время многие специалисты предлагают использовать для разработки параллельных программ языки с неявным параллелизмом, при программировании на которых не требуется знать архитектуру параллельной ЭВМ, и автоматически отображать такие программы на параллельные машины. Ярким примером такого языка является разработанный под руководством И.Б. Задыхайло в 1985 году декларативный язык НОРМА [7]. В качестве таких языков особенно привлекательно использовать Фортран и Си/Си++, поскольку их в основном и используют программисты при решении задач, наиболее остро требующих распараллеливания.

Таким образом, снова ставится вопрос об автоматическом распараллеливании последовательных программ. Что же изменилось за последние 20 лет, чтобы снова возвращаться к проблеме, уже давно (с появлением многопроцессорных ЭВМ с распределенной памятью) считающейся непреодолимой?

Во-первых, для обеспечения возможности использования языков с неявным параллелизмом задача автоматического распараллеливания теперь имеет другую постановку – автоматически распараллеливать надо не уже имеющиеся программы, а новые программы, предназначенные для параллельного выполнения. Такие препятствия для распараллеливания, как применение сугубо последовательного алгоритма или неудачные технические решения (например, вместо двумерного массива в программе с целью экономии памяти используется список векторов разной длины) должны быть устранены программистом.

Во-вторых, проблемы статического анализа, неспособного справиться в случаях использования в программе косвенной индексации или зависимости индексов и параметров циклов от динамически вводимых данных, могут быть решены с помощью динамического анализа или дополнительных аннотаций, вставляемых программистом в программу. Да и техника статического анализа за прошедшие два десятилетия существенно продвинулась – межпроцедурный анализ хорошо изложен в учебной литературе [8], появились доступные программные средства [9], существенно упрощающие его реализацию. Да и многократно возросшие возможности ЭВМ позволяют справляться в приемлемое время с анализом даже очень больших программ.

В-третьих, в работах [10, 11] показано, что для достаточно широкого класса программ удается их эффективно распараллелить для многопроцессорных ЭВМ с распределенной памятью. Распараллеливание для таких ЭВМ принципиально отличается от распараллеливания для мультипроцессоров и GPU тем, что невозможно инкрементальное (постепенное) распараллелива-

ние – надо находить глобальные решения по распределению данных и вычислений, что влечет за собой необходимость прогнозирования времени выполнения программы для разных возможных вариантов ее распараллеливания. Несомненно, что построение упомянутого выше автоматического распараллеливателя для многопроцессорных ЭВМ с распределенной памятью было бы проблематичным, если бы не был использован многолетний опыт ручного распараллеливания последовательных программ, существенный задел по реализации высокоуровневых языков параллельного программирования, развитые средства функциональной отладки и отладки эффективности параллельных программ [12].

Полученный нами опыт автоматического распараллеливания для многоядерных кластеров [13] и публикации об успешных экспериментах с автоматическим распараллеливанием для GPU [14, 15], позволяет нам надеяться, что языки с неявным параллелизмом очень скоро станут широко использоваться для создания программ для кластеров с многоядерными и графическими процессорами.

Необходимо отметить, что методы автоматического распараллеливания могут быть применены не только к последовательным программам, но и к имеющимся параллельным MPI-программам для адаптации их к многоядерным и графическим процессорам, используемым в узлах современных кластеров.

Тем не менее, надо учитывать, что автоматическое распараллеливание для многопроцессорных ЭВМ с распределенной памятью принципиально отличается от привычной оптимизации, широко применяемой в компиляторах. Вполне вероятно, что такое автоматическое распараллеливание не сможет рассматриваться как отдельная фаза компиляции, а будет представлять собой отдельный шаг преобразования последовательной программы в параллельную программу и потребует создания специальных методов сокращения требуемых ресурсов. Возможно, эти методы будут очень схожи с когда-то популярным методом инкрементальной компиляции, в котором отслеживались модификации программы, сделанные после ее предыдущей компиляции.

4. Автоматизация преобразования имеющихся программ

Изменение постановки задачи автоматического распараллеливания вызывает изменение подхода к автоматизации распараллеливания программ: использовать диалог с программистом прежде всего для уточнения свойств последовательной программы, а выбор наилучших решений по ее распараллеливанию осуществлять полностью автоматически.

Преобразование последовательной программы в параллельную программу можно представить состоящим из двух основных этапов.

На первом этапе проводится автоматизированное исследование и преобразование программистом последовательной программы с целью получить такую последовательную программу, которую автоматически распараллеливающий компилятор может преобразовать в эффективную параллельную программу. При этом от программиста может потребоваться описание свойств программы (через диалог или в виде специальных аннотаций в тексте программы), необходимых или очень существенных для ее автоматического распараллеливания. Например, в случае косвенной индексации элементов массива статический анализатор может сообщить о возможной зависимости между витками цикла, но программист может указать, что этой зависимости нет.

На втором этапе автоматически распараллеливающий компилятор преобразует потенциально параллельную программу в параллельную программу для заданной ЭВМ.

Достижение приемлемой эффективности выполнения параллельной программы может потребовать многократного повторения этих двух этапов, но система автоматизации распараллеливания должна быть ориентирована на сокращение таких итераций. Прежде всего, она должна позволить на инструментальной машине и еще до появления текста параллельной программы оценить эффективность ее работы на разных ЭВМ.

Эта способность быстрой оценки влияния разных решений по распараллеливанию на эффективность программы, которая сопровождается детальной разъясняющей информацией, очень важна для ускорения разработки параллельных программ, а также при обучении параллельному программированию.

Тут опять следует отметить принципиальное отличие распараллеливания для кластеров от распараллеливания для мультипроцессоров, вызванное невозможностью инкрементального распараллеливания. При распараллеливании на мультипроцессор программисту достаточно показать причины, по которым не удается распараллелить те циклы, выполнение которые требует больше всего времени. При распараллеливании же на кластер возможна ситуация, когда абсолютно все циклы могут быть распараллелены, но не удается найти такой вариант распределения данных, чтобы время выполнения параллельной программы было бы меньше, чем время выполнения исходной последовательной программы. Как убедить программиста, что такого варианта распределения данных не существует? Наверное, можно ему предоставить возможность задать самому распределение всех массивов, а система должна показать, что заданный им вариант распределения не лучше тех, которые были найдены ею.

5. Заключение

При разработке программ для многоядерных кластеров программисту сейчас приходится использовать две сильно различающиеся модели программирования (MPI, OpenMP) и соответствующие инструментальные средства. Появление в узлах кластера графических процессоров серьезно усложнило разработку программ, поскольку потребовало дополнительно использовать еще и низкоуровневую технологию CUDA. На подходе новые процессоры с большим количеством ядер, для эффективного использования которых потребуются новые модели программирования.

Разработчики новых языков параллельного программирования явно не успевают отслеживать архитектурное разнообразие многоядерных процессоров. Новые языки (Chapel [16], X10 [17], Fortress [18]) еще не доказали свою эффективность для современных кластеров, а уже выглядят слишком сложными и непривычными для программистов.

Использование гибридных языков, объединяющих разные модели параллельного программирования может рассматриваться только в качестве временной меры – нельзя и дальше усложнять модели параллельного программирования.

Поэтому наиболее привлекательным подходом представляется использование для создания параллельных программ вычислительного характера языков с неявным параллелизмом (Фортран, Си, Си++) и автоматически распараллеливающих компиляторов. За последние годы получены убедительные доказательства, что такие компиляторы для современных кластеров могут быть созданы.

Появление автоматически распараллеливающих компиляторов позволит создавать качественно новые системы автоматизации преобразования имеющихся последовательных программ (и уже широко распространенных параллельных MPI-программ) в эффективные параллельные программы для современных кластеров. Их способность быстрой оценки влияния разных решений по распараллеливанию на эффективность программы, которая сопровождается детальной разъясняющей информацией, очень важна для ускорения разработки параллельных программ, а также при обучении параллельному программированию.

Литература

1. NVIDIA CUDA, http://developer.nvidia.com/object/cuda.html
2. Timothy G. Mattson, Rob F. Van der Wijngaart, and other. ”The 48-core SCC processor: the programmer’s View”/ Proceedings of the 2010 ACM/IEEE conference on Supercomputing, SC10, New Orleans, Louisiana, Nov. 2010
3. Nunshi, A., Ed. The OpenCL Specification. The Kronos Group, Oct. 2009
4. В.А. Бахтин, Н.А.Коновалов, В.А. Крюков. Расширение языка OpenMP Fortran для распределенных систем. Вопросы атомной науки и техники. сер. Математическое моделирование физических процессов. 2002 г. Вып.4. стр.65-70

5. Konovalov N.A., Krukov V.A., Mihailov S.N. and Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development. Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994
6. The Portland Group, Inc. The PGI Fortran and C Accelerator Programming Model, Nov. 2009. available at www.pgroup.com/accelerate
7. А.Н.Андрианов, К.Н.Ефимкин, И.Б.Задыхайло, Н.В.Поддерюгина. Язык Норма. Препринт ИПМ им.М.В.Келдыша АН СССР, N165, 1985, 34 с
8. Альфред *В.* Ахо*,* Моника *С.* Лам*,* Рави Сети*,* Джеффри Д. Ульман. Компиляторы: Принципы, технологии и инструментарий, 2-е издание: Пер. с англ.- М.ООО "И.Д.Вильямс", 2008.-1184 с
9. Дроздов А.Ю. Компонентный подход к построению оптимизирующих компиляторов. // Программирование. 2009. № 5, с. 70-80
10. 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
11. М.С. Клинов. Алгоритмы автоматического отображения последовательных Фортран-программ на кластер. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность”, сентябрь 2009 г., г. Новороссийск. – М.: Изд-во МГУ, 2009, с. 298-299
12. DVM система: http://www.keldysh.ru/dvm
13. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина. Автоматическое распараллеливание последовательных программ для многоядерных кластеров. //Труды Международной научной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, сентябрь 2010 г., г. Новороссийск. – М.: Изд-во МГУ, 2010, с. 12-15
14. Muthu Manikandan Baskaran, J. Ramanujam, and P. Sadayappan/ Automatic C-to-CUDA Code Generation for Affine Programs, R. Gupta (Ed.): CC 2010, LNCS 6011, pp. 244–263, 2010
15. Lee, S., Min, S.-J., and Eigenmann, R. OpenMP to GPGPU: A compiler framework for automatic translation and optimization. In Proceedings of the 2009 ACM SIGPLAN Symposium in Principles and Practice of Parallel Programming (Raleigh, N.C., Feb. 2009), pp. 101–110
16. http://chapel.cray.com/
17. http://x10-lang.org/
18. E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, G. Steele, and S. Tobin-Hochstadt. The Fortress language specification. Available from http://research.sun.com/projects/plrg/

Отформатировано: русский
Отформатировано: русский
Отформатировано: русский
Отформатировано: русский
Отформатировано: русский

Построение кинетической модели реакции циклоалюминирования олефинов на основе параллельных вычислений

Ю.С. Лаврентьева

Институт нефтехимии и катализа РАН

Предложена методика распараллеливания вычислительного процесса при решении обратной задачи химической кинетики для реакции циклоалюминирования олефинов, реализующая сочетание трех уровней: распараллеливание по экспериментальной базе, использование внутреннего параллелизма задачи и распараллеливание численных методов решения задачи. На основе разработанной методики с использованием интерфейса MPI создан комплекс программ построения кинетической модели реакции. Произведен вычислительный эксперимент и сделаны физико-химические выводы о реакционной способности веществ, участвующих в реакции. Проведен анализ параллелизма и сделаны выводы об эффективности использования MPI для решения поставленной задачи.

1. Введение

При изучении механизмов сложных химических реакций для определения реакционной способности тех или иных веществ экспериментатору необходимо проводить лабораторный эксперимент при разных начальных данных об исходных реагентах и в разных условиях проведения эксперимента. В результате происходит накопление большого объема информации о реакции, которую необходимо в короткие сроки обработать, проанализировать и на ее основе построить кинетическую модель реакции. При этом большую роль играет скорость проведения расчетов, поскольку, по словам основоположника математического моделирования химической кинетики М.Г. Слинько, для того, чтобы кинетическая модель служила опорой для экспериментатора, она должна быть построена не более чем за 4 месяца [1]. При проведении вычислительного эксперимента стандартными методами с использованием персональных вычислительных машин на построение кинетических моделей подобных реакций в лабораториях Института нефтехимии и катализа РАН уходило более двух лет. Возникла необходимость оптимизации вычислительного процесса, для чего было предложено использование параллельных вычислений. Разработанный программный комплекс, реализующий расчет параметров кинетической модели реакции циклоалюминирования олефинов на основе использования технологии параллельных вычислений, планируется включить в информационно-аналитическую систему, разработанную в лаборатории математической химии ИНК РАН для расчета и построения кинетических моделей промышленно значимых химических реакций [2].

2. Реакция циклоалюминирования олефинов

В Институте нефтехимии и катализа РАН (ИНК РАН) ведется активное исследование сложных реакций с участием металлокомплексных катализаторов, которые являются безопасной и универсальной альтернативой распространенным в настоящее время термическим реакциям синтеза высших алюминийорганических соединений. Одной из ключевых реакций металлокомплексного катализа является реакция циклоалюминирования олефинов.

В лаборатории каталитического синтеза ИНК РАН, в группе под руководством к.х.н., доцента Рамазанова И.Р., ведутся исследования по определению реакционной способности олефиновых и ацетиленовых соединений в реакции циклоалюминирования [3]:

$$A_1 + ½A_2 \leftrightarrow ½A_3 + ½A_4;$$
$$A_1 + A_3 = A_4 + A_{12};$$
$$A_3 + A_4 = A_1 + ½A_{13} + ½A_6;$$
$$A_3 = A_5 + A_{13};$$

$$A_5 + A_1 = A_8 + A_{13};$$
$$A_5 + A_9 = A_{10};$$
$$A_1 + A_{10} = A_3 + A_{11};$$
$$A_6 + 2A_1 = 2A_4 + A_7;$$
$$A_7 = A_3 + A_5. \quad (1)$$

Построение кинетических моделей любых химических реакций требует решения двух фундаментальных задач:

1) экспериментальное исследование химического объекта;

2) математическая обработка экспериментальных данных с целью идентификации модели [4].

На этапе математической обработки экспериментального материала производится постановка и решение прямой и обратной кинетических задач.

Прямая кинетическая задача - это задача расчета состава реагирующей смеси по заданной кинетической модели. Математическое описание прямой задачи нестационарной химической кинетики состоит из системы обыкновенных нелинейных дифференциальных уравнений:

$$\frac{dX_i}{dt} = F_i, i = 1..M; \qquad F_i = \sum_{j=1}^{N} S_{ij} W_j;$$
$$W_j = P_j * \prod_{i=1}^{M} (X_i)^{|\delta_{ij}|} - Q_j * \prod_{i=1}^{M} (X_i)^{|\epsilon_{ij}|}; \quad (2)$$
$$P_j = P_j^O * \exp\left(-\frac{E_j^P}{RT}\right), \; Q_j = Q_j^O * \exp\left(-\frac{E_j^Q}{RT}\right)$$

с начальными условиями:

$$x_i(0) = x_i^0, \quad (3)$$

где x_i – концентрации (мольные доли) веществ, участвующих в реакции; M – количество веществ; N – количество стадий; α_{ij} и β_{ij} – соответственно, положительные и отрицательные элементы стехиометрической матрицы; ω_j – скорость j-ой стадии, 1/с; P_j и Q_j – приведенные константы скорости прямой и обратной реакции, соответственно, 1/с; E_j^P, E_j^Q – энергии активации прямой и обратной реакции, соответственно, кДж/моль; R – универсальная газовая постоянная, равная 8,31 кДж/(моль·К); T – температура, К.

Обратная кинетическая задача – это задача восстановления по экспериментальному материалу вида кинетической модели и ее параметров. Поиск кинетических констант осуществляется многократным решением прямой кинетической задачи и минимизацией критерия отклонения экспериментальных и расчетных данных**:**

$$F = \sum_{i=1}^{K} \sum_{j=1}^{M} \left| X_{ij}^{R} - X_{ij}^{E} \right|, \quad (4)$$

где X_i^R – расчетные значения концентраций наблюдаемых веществ; X_{ij}^E – экспериментальные данные по наблюдаемым веществам; K – количество точек эксперимента; M – количество наблюдаемых веществ, участвующих в реакции.

Скорости стадий реакции (1) на основе закона действующих масс имеют следующий вид:

$$\omega_1 = P_1 X_1 X_2^{0,5} - Q_1 X_3^{0,5} X_4^{0,5};$$
$$\omega_2 = P_2 X_1 X_3; \qquad \omega_3 = P_3 X_3 X_4;$$
$$\omega_4 = P_4 X_3; \qquad \omega_5 = P_5 X_5 X_1; \quad (5)$$
$$\omega_6 = P_6 X_5 X_9; \qquad \omega_7 = P_7 X_1 X_{10};$$
$$\omega_8 = P_8 X_6 X_1^2; \qquad \omega_9 = P_9 X_7.$$

Таким образом, математическая модель исследуемого процесса представляет собой систему обыкновенных нелинейных дифференциальных уравнений, включающую 10 уравнений для определения концентраций компонентов реакции вида (2) с 10 кинетическими параметрами $P_1, \ldots, P_9, Q_1$.

3. Параллельные вычисления при решении обратной задачи определения кинетических параметров реакции циклоалюминирования

В работе [5] предложена методология решения обратных задач химической кинетики с использованием технологии параллельных вычислений, согласно которой распараллеливание вычислительного процесса для рассматриваемых задач может быть осуществлено на трех уровнях:

1) использование внутреннего параллелизма задачи;
2) распараллеливание по экспериментальной базе;
3) распараллеливание численного метода решения обратной задачи.

Распараллеливание вычислительного процесса по экспериментальной базе для реакции циклоалюминирования олефинов представляет собой решение прямой и обратной кинетической задач параллельно для данных при разных температурах или по разным наборам экспериментальных данных при одинаковых температурах, при этом рассматривается 864 независимо протекающих реакций (рис. 1).

Рис. 1. Независимо протекающие стадии реакции циклоалюминирования олефинов

Распараллеливание вычислительного процесса по экспериментальной базе осуществляется по принципу процессорной фермы (или стратегии «главный – подчиненные»), при которой один процессор-главный считывает данные из базы и распределяет их между процессорами-подчиненными, а затем собирает посчитанные результаты и заносит их в базу данных.

Рассматриваемый процесс также обладает внутренним параллелизмом, который заключается в том, что общая схема реакции включает 9 протекающих параллельно стадий (см. схему реакции (4)). При осуществлении вычислительного процесса решение обратной задачи осуществляется независимо для параллельно протекающих стадий, что позволяет существенно сократить время расчета.

Наконец, распараллеливание численного метода решения обратной задачи осуществляется на основе принципа геометрического параллелизма, который заключается в декомпозиции расчетной области по числу работающих процессов (процессоров многопроцессорной вычислительной системы) (рис. 2). При этом область изменения кинетических параметров задается априорно (т.е. представляется экспериментаторами на основе лабораторных исследований).

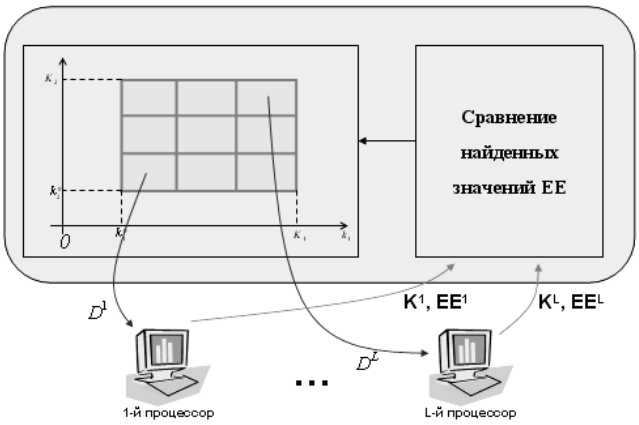


Рис. 2. Решение обратной задачи с использованием принципа геометрического параллелизма

4. Вычислительный эксперимент и основные результаты

На основе разработанных алгоритмов создан программный комплекс, реализующий с использованием технологии параллельных вычислений расчет кинетических параметров реакции циклоалюминирования олефинов (CYCLOAL). Программный комплекс написан на языке программирования C++ с использованием интерфейса передачи сообщений MPI и тестирован на суперкомпьютере МВС-100К Межведомственного суперкомпьютерного центра РАН. Структура разработанного программного комплекса представлена на рисунке 3.

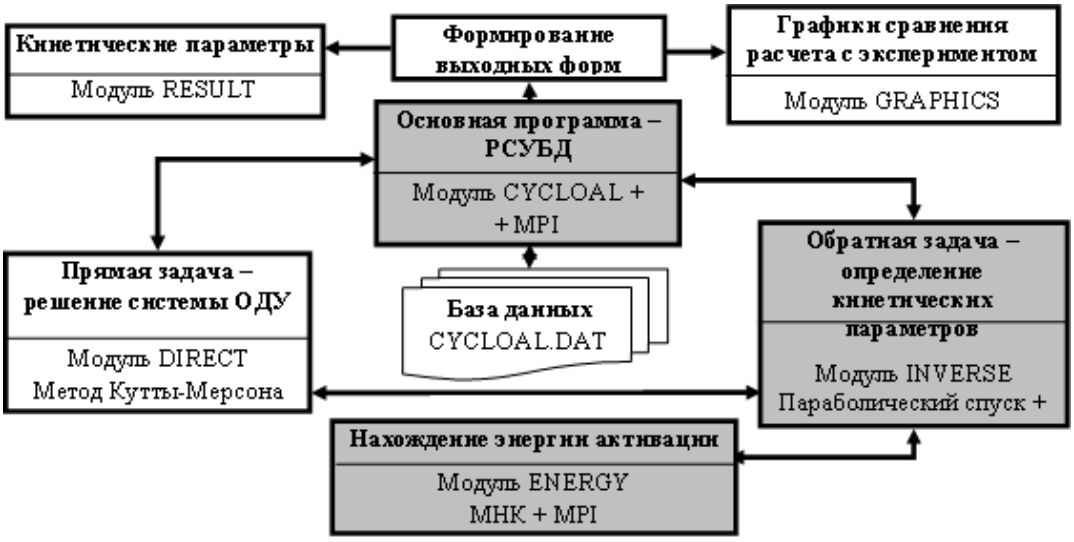


Рис. 3. Структура программного комплекса CYCLOAL

С использованием разработанного программного комплекса CYCLOAL найдены кинетические константы и энергии активации для всех параллельно протекающих стадий реакции циклоалюминирования олефинов (таблица 1).

Таблица 1. Кинетические параметры для реакции циклоалюминирования олефинов (олефин октен-1)

T, °C	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}
18	62,42	0,05	3,28	1,61	0,21	196,63	30,49	0,67	0,45	0,05
25	156,98	0,80	23,84	3,04	3,04	8750,0	963,00	3,42	0,87	1,17
30	263,86	0,05	35,65	26,85	5,41	844,37	361,95	23,24	9,71	1,06
40	97,15	3,91	67,26	215,65	10,00	1313125	1020,7	13,29	10,93	10,61
50	488,17	9,21	176,04	799,54	24,45	2991,7	1560,0	36,01	61,43	53,24
E_{ak}, kkal/mol	9,69	31,55	23,60	38,97	27,32	17,81	25,67	23,44	28,45	43,48

По результатам численного эксперимента построена кинетическая модель реакции и сделано несколько практически важных выводов относительно механизма реакции циклоалюминирования олефинов. Так, слабая тенденция роста константы скорости седьмой стадии в случае октина-4 с увеличением температуры может быть связана с большими стерическими (пространственными) затруднениями, возникающими на стадии переметаллирования. Кроме того, было показано, что реакционная способность октина-1 выше, чем у октина-4, что может быть вызвано большей поляризацией связей в октине-1.

Проведен анализ зависимости скорости стадий реакции циклоалюминирования олефинов от времени (рис. 4).

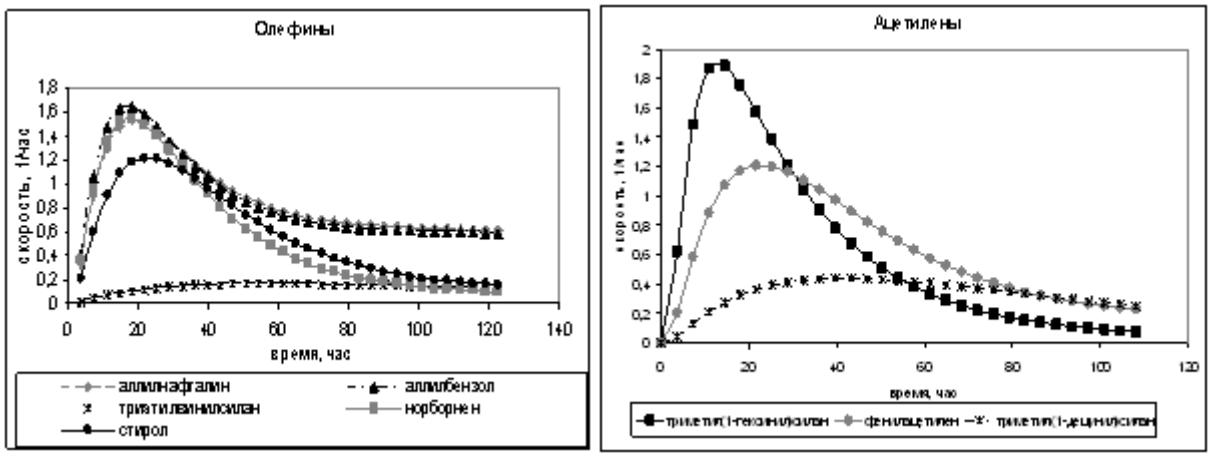


Рис. 4. Зависимость скоростей отдельных стадий реакции циклоалюминирования от времени

На основании полученных зависимостей можно сделать вывод о том, что в начальный момент реакции происходит накопление каталитически активных компонентов, которое приводит к увеличению скорости стадий реакции. По мере протекания реакции, концентрации реагентов уменьшаются, вследствие чего наблюдается максимум на графиках зависимости скорости реакции от времени.

Рассмотрена также зависимость скорости протекания реакции от температуры (рис. 5).

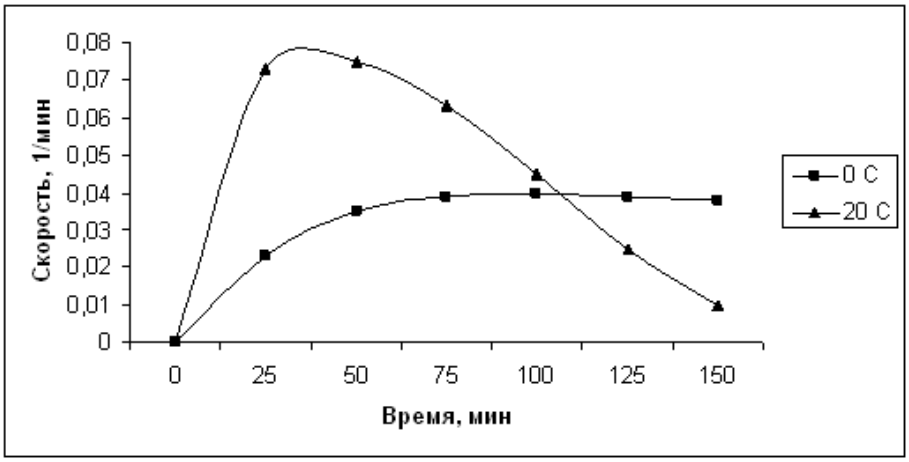


Рис. 5. Зависимость скорости реакции от температуры в случае октина-1

Таким образом, численным расчетом подтверждено экспериментально наблюдаемое явление смещения максимума к начальному моменту времени с увеличением температуры и увеличение скорости реакции, что подтверждает адекватность построенных математических моделей.

Анализ эффективности параллелизма для решения поставленной задачи показал, что оптимальным является использование 11-13 процессоров суперкомпьютера, а при увеличении числа работающих процессоров эффективность распараллеливания значительно снижается (рис. 6). По всей видимости, снижение эффективности работы программы вызвано тем, что при числе

процессоров <13 время межпроцессорных взаимодействий значительно меньше времени расчета каждым процессором, а при увеличении числа процессоров эти значения становятся сопоставимыми. Причиной этому по всей вероятности является то, что парадигма программирования для вычислительных систем с распределенной памятью (MPI) используется на системах гибридной архитектуры.

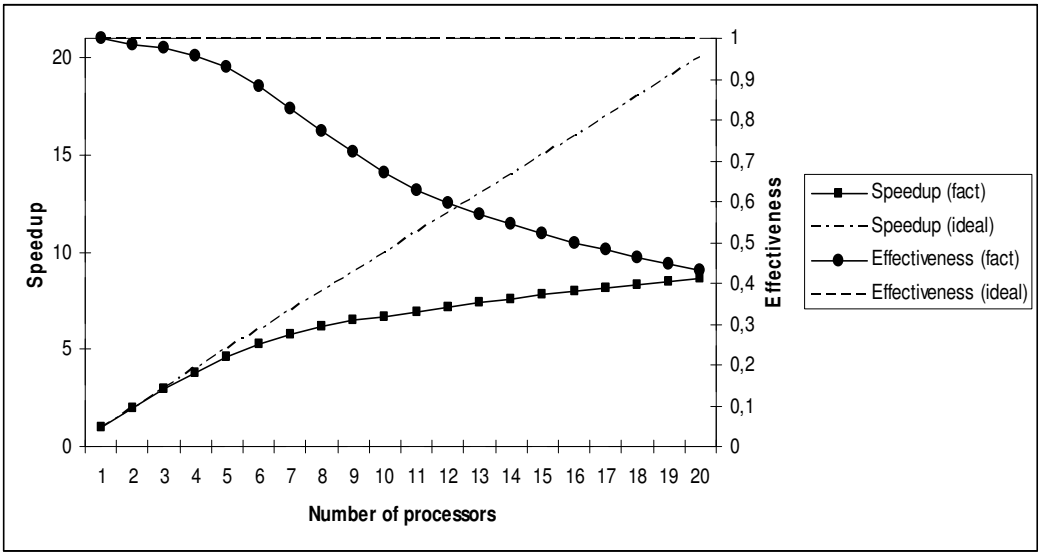


Рис.6. Ускорение и эффективность параллелизма программы

На следующем этапе исследования планируется провести вычислительный эксперимент по сравнению использования MPI и сочетания MPI+OpenMP для решения поставленной задачи с целью повышения эффективности работы программного комплекса.

5. Выводы

Реализована методология применения параллельных вычислений к решению обратной кинетической задачи для реакции каталитического циклоалюминирования олефинов. На основе вычислительного эксперимента построена кинетическая модель реакции и сделано несколько практически важных выводов о реакционной способности участвующих в реакции веществ. Проведен анализ эффективности параллелизма, на основе которого сделан вывод о необходимости оптимизации разработанного программного комплекса использованием сочетания парадигм MPI и OpenMP.

Литература

1. Слинько М.Г. Основы и принципы математического моделирования каталитических процессов: Новосибирск, Ин-т катализа им. Борескова СО РАН, 2004. 488 с.
2. 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.
3. Балаев А.В., Парфенова Л.В., Русаков С.В., Губайдуллин И.М., Спивак С.И., Халилов Л.М., Джемилев У.М. Механизм реакции циклоалюминирования алкенов триэтилалюминием в алюмациклопентанты, катализируемой Cp2ZrCl2: ДАН, 2001.Т. 381, №3.
4. Царева З.М., Орлова Е.А. Теоретические основы химтехнологии: Киев: ВШ, 1986. 271 с.
5. Линд Ю.Б. Математическое моделирование обратных задач физической химии на основе параллельных вычислений // Диссертация на соискание ученой степени кандидата физико-математических наук:Уфа, 2010. 179 с.

Параллельный алгоритм решения дробно-дифференциальных уравнений переноса на основе модифицированного метода Шварца

С.Ю. Лукащук

ГОУ ВПО Уфимский государственный авиационный технический университет

Предлагается модификация метода декомпозиции Шварца для дифференциальных уравнений переноса, содержащих производные дробного порядка. Доказывается сходимость метода. Приводятся описание численной схемы и схемы распараллеливания, а также оценка эффективности предлагаемого параллельного алгоритма.

1. Введение

Методы декомпозиции расчетной области являются основой для построения большинства современных параллельных алгоритмов, нацеленных на решение практических задач в геометрически сложных областях. Обширный класс таких методов образуют методы Шварца, относящиеся к группе итерационных методов подструктур [1]. В методах Шварца решение задачи строится независимо в пересекающихся или непересекающихся подобластях, а для согласования этих решений организуется итерационный процесс. При этом после каждой итерации происходит обмен внутренними граничными условиями между соседними подобластями.

В классическом методе Шварца на внутренних границах выставляются граничные условия первого рода (условия Дирихле). Недостатком такого подхода является низкая скорость сходимости итерационного процесса, а в случае неперекрывающихся областей сходимость может вообще отсутствовать [2]. Эти недостатки преодолены в оптимизированном варианте метода [3]. При этом граничные условия Дирихле заменены на искусственные граничные условия, содержащие линейные дифференциальные операторы специального вида.

Первоначально метод Шварца был разработан для решения уравнений эллиптического типа, однако впоследствии он был распространен и на уравнения гиперболического и параболического типов. В работе [4] показывается, что использование оптимизированного метода Шварца для решения классического уравнения диффузии приводит к тому, что дополнительный оператор, входящий во внутренние граничные условия, становится нелокальным по времени. Для преодоления этого недостатка в [4] предложены различные способы приближения нелокального оператора локальным, сохраняющим сходимость итерационного процесса.

В данной работе показывается, как методы Шварца могут быть адаптированы для решения уравнений переноса, содержащих частные производные дробного порядка по времени [5]. В настоящее время наиболее хорошо исследованным видом таких уравнений являются дробно-дифференциальные уравнения аномальной диффузии, позволяющие описывать как аномально медленные (субдиффузия), так и аномально быстрые (супердиффузия) процессы переноса. Такие процессы наиболее часто наблюдаются в неоднородных сложных средах, таких как пористые и трещиновато-пористые среды, перколяционные кластеры и самоподобные фрактальные среды, турбулентные потоки жидкости, газа и плазмы и многие другие [6]. Как правило, аномальный перенос обусловлен эффектами памяти, пространственной нелокальности и перемежаемости.

Следует отметить, что поскольку производная дробного порядка является нелокальным интегро-дифференциальным оператором, численное моделирование по дробно-дифференциальным моделям требует весьма значительных вычислительных ресурсов. При этом, если уравнение содержит дробную производную по времени, то объем вычислений будет линейно расти с ростом номера временного слоя. Указанная проблема является одним из факторов, сдерживающих внедрение дробно-дифференциальных моделей в практику современных инженерных расчетов. Поэтому разработка параллельных алгоритмов решения этих задач представляется весьма актуальной.

2. Модификация метода Шварца для уравнения диффузии дробного порядка

Рассмотрим задачу Коши для уравнения диффузии дробного порядка

$$\begin{aligned} &\frac{\partial y(x,t)}{\partial t} = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial^2 y(x,t)}{\partial x^2}\right) + f(x,t), \quad t>0, \quad x\in\Omega, \quad \alpha\in(0,1); \\ &y(x,0) = g(x), \quad x\in\Omega, \end{aligned} \tag{1}$$

где $\Omega = (-\infty,\infty)$, функция y ограничена в Ω, D – коэффициент аномальной диффузии,

$$\frac{\partial^{\alpha} y(x,t)}{\partial t^{\alpha}} = \frac{1}{\Gamma(1-\alpha)}\frac{\partial}{\partial t}\int_0^t \frac{y(x,\tau)}{(t-\tau)^{\alpha}}\,d\tau, \quad \alpha\in(0,1)$$

– левосторонняя частная производная дробного порядка α по t типа Римана-Лиувилля [5].

Разобьем область Ω на две, возможно неперекрывающиеся, подобласти $\Omega_1 = (-\infty, L]$ и $\Omega_2 = [0,\infty)$, $L\geq 0$ (см. **Рис. 1**). Обозначим приближение к искомой функции $y(x,t)$ в подобласти Ω_1 через $u(x,t)$, а в области Ω_2 – через $v(x,t)$. Запишем задачу (1) отдельно для каждой подобласти:

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial^2 u(x,t)}{\partial x^2}\right) + f(x,t), \quad t>0, \quad x\in\Omega_1, \quad \alpha\in(0,1); \\ &u(x,0) = g(x), \quad x\in\Omega_1; \\ &\frac{\partial v(x,t)}{\partial t} = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial^2 v(x,t)}{\partial x^2}\right) + f(x,t), \quad t>0, \quad x\in\Omega_2, \quad \alpha\in(0,1); \\ &v(x,0) = g(x), \quad x\in\Omega_2. \end{aligned} \tag{2}$$

Решение задачи (1) может быть получено композицией решений u и v. Следуя [4], поставим на внутренних границах областей $x=0$ и $x=L$ следующие граничные условия:

$$\begin{aligned} &(B+\Lambda_1)u(x,t)\big|_{x=L} = (B+\Lambda_1)v(x,t)\big|_{x=L}, \\ &(B+\Lambda_2)v(x,t)\big|_{x=0} = (B+\Lambda_2)u(x,t)\big|_{x=0}, \end{aligned} \tag{3}$$

где оператор B имеет вид, соответствующий аномальному диффузионному потоку: $B = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial}{\partial x}\right)$, а Λ_1 и Λ_2 – некоторые линейные операторы, действующие по переменной t и подлежащие определению.

Для системы (2),(3) можно записать следующий итерационный процесс:

$$\begin{aligned} &\frac{\partial u^n(x,t)}{\partial t} = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial^2 u^n(x,t)}{\partial x^2}\right) + f(x,t), \quad t>0, \quad x<L, \\ &u^n(x,0) = g(x), \quad x\leq L, \\ &(B+\Lambda_1)u^n(x,t)\Big|_{x=L} = (B+\Lambda_1)v^{n-1}(x,t)\Big|_{x=L}, \quad t>0; \\ &\frac{\partial v^n(x,t)}{\partial t} = D\frac{\partial^{\alpha}}{\partial t^{\alpha}}\left(\frac{\partial^2 v^n(x,t)}{\partial x^2}\right) + f(x,t), \quad t>0, \quad x>0, \\ &v^n(x,0) = g(x), \quad x\geq 0, \\ &(B+\Lambda_2)v^n(x,t)\Big|_{x=0} = (B+\Lambda_2)u^{n-1}(x,t)\Big|_{x=0}. \end{aligned} \tag{4}$$

Операторы Λ_1 и Λ_2 должны выбираться так, чтобы гарантировать сходимость итерационного процесса (4).

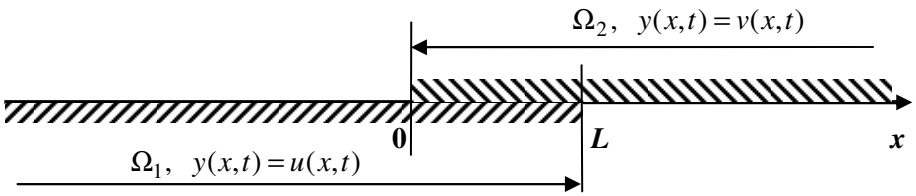


Рис. 1. Схема декомпозиции области

Обозначим через $\varphi^n(x,t)=u(x,t)-u^n(x,t)$ и $\psi^n(x,t)=v(x,t)-v^n(x,t)$ соответствующие ошибки на n-ой итерации. В силу линейности всех операторов, для определения ошибок из (2)-(4) получаем следующий итерационный процесс:

$$\begin{aligned}
&\frac{\partial\varphi^n(x,t)}{\partial t}=D\frac{\partial^\alpha}{\partial t^\alpha}\left(\frac{\partial^2\varphi^n(x,t)}{\partial x^2}\right),\quad t>0,\quad x<L,\\
&\varphi^n(x,0)=0,\quad x\le L,\\
&(B+\Lambda_1)\varphi^n(x,t)\Big|_{x=L}=(B+\Lambda_1)\psi^{n-1}(x,t)\Big|_{x=L},\quad t>0;\\
&\frac{\partial\psi^n(x,t)}{\partial t}=D\frac{\partial^\alpha}{\partial t^\alpha}\left(\frac{\partial^2\psi^n(x,t)}{\partial x^2}\right),\quad t>0,\quad x>0,\\
&\psi^n(x,0)=0,\quad x\ge 0,\\
&(B+\Lambda_2)\psi^n(x,t)\Big|_{x=0}=(B+\Lambda_2)\varphi^{n-1}(x,t)\Big|_{x=0}.
\end{aligned}\tag{5}$$

Сходимость итерационного процесса (5) к нулю независимо от начального приближения гарантирует сходимость исходного процесса (4). Из этого условия и могут быть найдены операторы Λ_1 и Λ_2.

Обозначим через $\tilde{y}(x,s)$ преобразование Лапласа от функции $y(x,t)$ по времени:

$$\tilde{y}(x,s)\equiv\int_0^\infty e^{-st}y(x,t)dt\ .$$

Следуя [4] будем полагать, что преобразование Лапласа от $\Lambda_i y$ есть $\lambda_i(s)\tilde{y}$, $i=1,2$. Применяя преобразование Лапласа по времени к задаче (5), получаем:

$$\begin{aligned}
&s\tilde{\varphi}^n(x,s)=Ds^\alpha\tilde{\varphi}^n_{xx}(x,s),\quad x<L,\\
&Ds^\alpha\tilde{\varphi}^n_x(L,s)+\lambda_1(s)\tilde{\varphi}^n(L,s)=Ds^\alpha\tilde{\psi}^{n-1}_x(L,s)+\lambda_1(s)\tilde{\psi}^{n-1}(L,s);\\
&s\tilde{\psi}^n(x,s)=Ds^\alpha\tilde{\psi}^n_{xx}(x,s),\quad x>0,\\
&Ds^\alpha\tilde{\psi}^n_x(0,s)+\lambda_2(s)\tilde{\psi}^n(0,s)=Ds^\alpha\tilde{\varphi}^{n-1}_x(0,s)+\lambda_2(s)\tilde{\varphi}^{n-1}(0,s).
\end{aligned}\tag{6}$$

Разрешая уравнения (6) с учетом ограниченности функций $\tilde{\varphi}$ и $\tilde{\psi}$, находим

$$\tilde{\varphi}^n(x,s)=C_1^n e^{\omega x},\quad x<L;\qquad \tilde{\psi}^n(x,s)=C_2^n e^{-\omega x},\quad x>0;\quad \omega=\sqrt{\frac{s^{1-\alpha}}{D}}\ .\tag{7}$$

Подставляя (7) в граничные условия из (6), приходим к соотношениям

$$\tilde{\varphi}^{n+1}(0,s)=\rho(s)\tilde{\varphi}^{n-1}(0,s),\quad \tilde{\psi}^{n+1}(0,s)=\rho(s)\tilde{\psi}^{n-1}(0,s)\,,\tag{8}$$

где

$$\rho(s)=\frac{\left(\lambda_1(s)-Ds^\alpha\omega\right)\left(\lambda_2(s)+Ds^\alpha\omega\right)}{\left(\lambda_1(s)+Ds^\alpha\omega\right)\left(\lambda_2(s)-Ds^\alpha\omega\right)}e^{-2\omega L}$$

– коэффициент, характеризующий сходимость итерационного процесса. При

$$\lambda_1(s)=\sqrt{D}s^\beta,\quad \lambda_2(s)=-\sqrt{D}s^\beta,\quad \beta=\frac{1+\alpha}{2}\tag{9}$$

$\rho(s)$ тождественно обращается в нуль. Из (8) следует, что в этом случае $\tilde{\varphi}^2(0,s)=\tilde{\psi}^2(0,s)=0$, что с учетом (7) дает $\tilde{\varphi}^2(x,s)=0$ $(x<L)$ и $\tilde{\psi}^2(x,s)=0$ $(x>0)$. Таким образом, условие (9) обеспечивает сходимость итерационного процесса (5) за две итерации независимо от начального приближения и глубины перекрытия *L*.

Применяя к (9) обратное преобразование Лапласа, находим искомые операторы:

$$\Lambda_1=\sqrt{D}\,\frac{\partial^\beta}{\partial t^\beta},\quad \Lambda_2=-\sqrt{D}\,\frac{\partial^\beta}{\partial t^\beta}. \tag{10}$$

Можно показать, что при разбиении исходной области на *N* подобластей такой способ выбора операторов гарантирует сходимость итерационного процесса метода Шварца за *N* шагов.

3. Численная схема алгоритма

Теперь рассмотрим некоторые особенности численной реализации описанного в предыдущем разделе метода декомпозиции для случая ограниченной области $\Omega=[-1,1]$.

В качестве модельной будем рассматривать следующую первую начально-краевую задачу для уравнения диффузии дробного порядка:

$$\begin{aligned}&\frac{\partial y(x,t)}{\partial t}=D\,\frac{\partial^\alpha}{\partial t^\alpha}\left(\frac{\partial^2 y(x,t)}{\partial x^2}\right)+f(x,t),\quad t>0,\quad x\in(-1,1),\quad \alpha\in(0,1);\\ &y(x,0)=g(x),\quad x\in[-1,1];\\ &y(-1,t)=\mu_1(t),\quad y(1,t)=\mu_2(t),\quad t>0.\end{aligned} \tag{11}$$

Для простоты будем полагать, что область Ω разделяется на две равные подобласти без перекрытия: $\Omega_1=[-1,0]$ и $\Omega_2=[0,1]$. Как и ранее, решение задачи в области Ω_1 обозначим через $u(x,t)$, а в области Ω_2 – через $v(x,t)$. На внутренней границе подобластей ставим стыковочные граничные условия вида (3) с операторами (10).

Каждую подобласть разобьем равномерной конечно-разностной сеткой с шагом Δx и числом узлов *M+1*. Будем использовать локальную нумерацию узлов в пределах каждой подобласти: $x_i=i\Delta x,\ i=0,1,...,M$. Шаг по времени Δt будем считать постоянным, временной слой обозначим через $t_j=j\Delta t,\ j=0,1,...$. Значения сеточных функций обозначим $z(x_i,t_j)=z_{i,j}$.

Производную дробного порядка по времени аппроксимируем на основе формулы Грюнвальда-Летникова [5]:

$$\left.\frac{\partial^\alpha z(t)}{\partial t^\alpha}\right|_{t=t_N}=\frac{1}{(\Delta t)^\alpha}\sum_{k=0}^{N}w_k^{(\alpha)}z(t_N-k\Delta t),\quad t_N=N\Delta t,$$

$$w_0^{(\alpha)}=1,\quad w_k^{(\alpha)}=\left(1-\frac{\alpha+1}{k}\right)w_{k-1}^{(\alpha)},\quad k=1,2,...\,.$$

Аналогично (4), в каждой подобласти организуем итерационный вычислительный процесс. С учетом конечно-разностной аппроксимации, для $x\in\Omega_1$ имеем:

$$\begin{aligned}&\frac{u_{i,j}^n-u_{i,j-1}}{\Delta t}=\frac{D}{(\Delta t)^\alpha(\Delta x)^2}\left[\Delta u_{i,j}^n+\sum_{k=1}^{j}w_k^{(\alpha)}\Delta u_{i,j-k}\right]+f_{i,j},\ i=1,2,...,M-1,\quad j=1,2,...;\\ &u_{i,0}^n=g_i,\ i=0,1,...,M;\quad u_{0,j}^n=\mu_{1,j},\ j=1,2,...;\\ &\frac{D}{(\Delta t)^\alpha\Delta x}\left[\nabla u_{M,j}^n+\sum_{k=1}^{j}w_k^{(\alpha)}\nabla u_{M,j-k}\right]+\frac{\sqrt{D}}{(\Delta t)^\beta}\left[u_{M,j}^n+\sum_{k=1}^{j}w_k^{(\beta)}u_{M,j-k}\right]=\\ &\qquad=\frac{D}{(\Delta t)^\alpha\Delta x}\left[\nabla v_{0,j}^{n-1}+\sum_{k=1}^{j}w_k^{(\alpha)}\nabla v_{0,j-k}\right]+\frac{\sqrt{D}}{(\Delta t)^\beta}\left[v_{0,j}^{n-1}+\sum_{k=1}^{j}w_k^{(\beta)}v_{0,j-k}\right],\ j=1,2,...\,.\end{aligned} \tag{12}$$

Здесь обозначено $\Delta u_{i,j}^n = u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n$, $\nabla u_{M,j}^n = u_{M,j}^n - u_{M-1,j}^n$, $\nabla v_{0,j}^n = v_{1,j}^n - v_{0,j}^n$. В качестве начального приближения используем значение с предыдущего временного слоя: $u_{i,j}^0 = u_{i,j-1}$, $j = 1,2,\ldots$. Отметим, что в (12) итерационный процесс организуется только для временного слоя t_j. На всех предыдущих временных слоях процесс считается сошедшимся, поэтому у сеточных функций на этих слоях индекс номера итерации отсутствует.

Аналогично (12), записывается итерационный процесс в области Ω_2 для функции $v(x,t)$.

Легко заметить, что (12) на каждом временном слое t_j приводит к системе линейных алгебраических уравнений $AU_j^n = F_j^n$ с трехдиагональной матрицей

$$A = \begin{pmatrix} 1 & 0 & & & & & \\ -a & 1+2a & -a & & & & \\ & -a & 1+2a & -a & & & \\ & & & \ldots & & & \\ & & & & -a & 1+2a & -a \\ & & & & & -b & 1+b \end{pmatrix}, \quad a = \frac{D(\Delta t)^{1-\alpha}}{\Delta x^2}, \quad b = \frac{\sqrt{D}(\Delta t)^{\beta-\alpha}}{\Delta x}, \quad (13)$$

векторами неизвестных $U_j^n = \left(u_{0,j}^n, u_{1,j}^n, \ldots, u_{M,j}^n\right)^T$ и правой части $F_j^n = \left(f_{0,j}, f_{1,j}, \ldots, f_{M,j}^n\right)^T$, где

$$\begin{aligned} &f_{0,j} = \mu_{1,j}; \quad f_{i,j} = u_{i,j-1} + a\sum_{k=1}^{j} w_k^{(\alpha)} \Delta u_{i,j-k} + f_{i,j},\ i = 1,2,\ldots,M-1; \\ &f_{M,j}^n = v_{0,j}^{n-1} + b\nabla v_{0,j}^{n-1} + \sum_{k=1}^{j}\left[w_k^{(\beta)}\left(v_{0,j-k} - u_{M,j-k}\right) + bw_k^{(\alpha)}\left(\nabla v_{0,j-k} - \nabla u_{M,j-k}\right)\right]. \end{aligned} \quad (14)$$

Данная система легко может быть решена методом прогонки.

Аналогичная система записывается в области Ω_2 для вектора неизвестных $V_j^n = \left(v_{0,j}^n, v_{1,j}^n, \ldots, v_{M,j}^n\right)^T$.

4. Параллельный алгоритм и оценка его эффективности

Теперь приведем формальное описание параллельного алгоритма, порождаемого численной схемой из предыдущего раздела.

Пусть имеется *p* вычислительных модулей (процессоров или ядер) одинаковой производительности. В этом случае расчетная область разбивается на *p* неперекрывающихся подобластей одинакового размера и каждый вычислительный модуль отвечает за расчет своей подобласти. Так как внутреннее граничное условие содержит производную по пространству, каждый вычислитель на каждом временном слое со стороны каждой внутренней границы должен иметь две вспомогательные переменные для хранения значений, передаваемых с соседней подобласти (например, для первой подобласти – это значения $v_{0,j}^{n-1}$ и $v_{1,j}^{n-1}$, см.(14)). Важно отметить, что в отличие от классических уравнений переноса, при расчете аномальной диффузии необходимо сохранять значения вектора неизвестных *на всех* временных слоях. Далее выполняются следующие действия.

1) На основании исходных данных задачи и параметров сетки все вычислители параллельно рассчитывают коэффициенты матрицы *A*.
2) На каждом вычислителе организуется глобальный цикл по временным шагам. На нулевом шаге вектор искомых значений заполняется в соответствии с начальным условием.
3) Для произвольного *j*-го временного шага выполняются следующие действия.
   а) Вычисляются новые коэффициенты $w_j^{(\alpha)}$ и $w_j^{(\beta)}$.
   б) На каждом вычислителе организуется внутренний итерационный процесс.

в) Вычислители, отвечающие за расчет соседних подобластей, обмениваются двумя приграничными значениями. При этом на первом итерационном шаге пересылаются значения с предыдущего временного слоя, а на последующих итерационных шагах – значения с предыдущей итерации.
г) На первом итерационном шаге каждый вычислитель по формулам вида (14) параллельно вычисляет вектор правой части системы. На последующих итерациях пересчитываются только первый и последний элементы вектора *F*, соответствующие внутренним границам.
д) Каждый вычислитель параллельно решает свою систему методом прогонки.
е) Если номер итерации меньше *p*, повтор с шага б), иначе переход на новый временной слой и повтор с шага 3).

Проведем теоретическую оценку эффективности этого алгоритма. Пусть, для простоты, $f(x,t)=0$. Если число узлов сетки в каждой подобласти равно $M+1$, то расчет элементов вектора *F* на *j*-ом временном слое на первом итерационном шаге требует для внутренней подобласти $2(2j+1)(M+3)$ операций, а на последующих итерациях – 8 операций. Трудоемкость решения системы методом прогонки составляет $9(M+1)$. Тогда трудоемкость расчета *j*-го временного слоя для *p* вычислителей за *p* итераций без учета затрат на передачу данных составит

$$T_p = 2(2j+1)(M+3)+8(p-1)+9p(M+1).$$

Трудоемкость реализации алгоритма на одном вычислителе (в этом случае внутренние итерации отсутствуют) составит

$$T_1 = 2(2j+1)[(M+1)p-2]+9p(M+1).$$

Для ускорения параллельного алгоритма имеем

$$S_p \equiv \frac{T_1}{T_p} = \frac{p-\dfrac{2}{M+1}+\dfrac{9p}{2(2j+1)}}{1+\dfrac{2}{M+1}+\dfrac{9p}{2(2j+1)}+\dfrac{8(p-1)}{2(2j+1)(M+1)}}. \qquad (15)$$

Как правило, $M>>1$ и оценка (15) упрощается:

$$S_p \approx \frac{p+\dfrac{9p}{2(2j+1)}}{1+\dfrac{9p}{2(2j+1)}}. \qquad (16)$$

Из (16) видно, что для больших временных слоев алгоритм имеет асимптотически линейное ускорение $S_p \sim p$ при $j\to\infty$.

Литература

1. Копысов С.П., Красноперов И.В., Рычков В.Н. Объектно-ориентированный метод декомпозиции области // Вычислительные методы и программирование. 2003. Т. 4. С.1-18.
2. Gander M.J., Golub G.H. A Non-Overlapping Optimized Schwarz Method which Converges with Arbitrarily Weak Dependence on h // Fourteenth International Conference on Domain Decomposition Methods, 2003. 8 p.
3. Gander M. J., Halpern L., Nataf F. Optimized Schwarz methods // Twelfth International Conference on Domain Decomposition Methods, Chiba, Japan, Bergen, 2001. P. 15-28.
4. Gander M. J., Halpern L., Nataf F. Optimal convergence for overlapping and nonoverlapping Schwarz waveform relaxation // Eleventh international Conference on Domain Decomposition Methods, 1999. P. 27-36.
5. Podlubny I. Fractional Differential Equations. Academic press, San Diego, 1999. 340 p.
6. Metzler R., Klafter J. The random walk's guide to anomalous diffusion: A fractional dynamic approach // Phys. Rep. 2000. Vol. 339. P. 1-77.

Прямое численное моделирование эффекта Ранка

Д.Ф. Марьин[1], К.И. Михайленко[1], Л.Х. Хазиев[2]

Институт механики УНЦ РАН[1],
Уфимский государственный авиационный технический университет[2]

Численное моделирование вихревого эффекта Ранка-Хилша в противоточной вихревой трубе проводится посредством прямого решения уравнений динамики вязкого газа методом крупных частиц. Крупномасштабные эффекты турбулентности учтены при помощи использования DNS. Математическая модель решалась в трехмерных цилиндрической и декартовой системах координат. Температурное распределение внутри вихревой трубы получено на основе анализа распределения внутренней энергии в рассматриваемой системе. Ускорение вычислительного процесса получено за счет использования технологий OpenMP и MPI. Представлены оценки эффективности распараллеливания.

1. Введение

В газовой динамике хорошо известно такое нетривиальное явление как вихревой эффект (эффект Ранка–Хилша). Этот эффект характеризуется разделением газа или жидкости при закручивании в цилиндрической или конической камере на две фракции. На периферии образуется поток с большей температурой, а в центре — охлажденный поток. Эффект был обнаружен французским инженером Жозефом Ранком в 1931 году [1], а первые подробные теоретические исследования провёл немецкий физик Хилш в 1946 году [2].

К настоящему времени предпринято множество попыток теоретически объяснить описываемый эффект (см., например, [3, 4]), но ни одно объяснение не является полностью неоспоримым.

В работе представлено первое приближение к прямому моделированию описываемого эффекта. За основу взяты уравнения газовой динамики для вязкого газа, численный эксперимент проводился с помощью метода крупных частиц. Турбулентные эффекты учитываются за счёт использования мелкомасштабной сетки узловых точек.

2. Математическая модель

Математическая модель рассматриваемого процесса записывается при следующих допущениях:

- среда представляет собой идеальный вязкий газ;
- вязкость газа постоянна и сравнима с вязкостью воздуха;
- газ представляет собой ньютоновскую среду, подчиненную обобщенному закону Ньютона о линейной связи между тензором вязких напряжений $\mathbf{P}$ и тензором скоростей деформаций $\dot{\mathbf{S}}$:

$$\mathbf{P} = 2\mu\dot{\mathbf{S}} - \left(p + \frac{2}{3}\mu \operatorname{div}\vec{W}\right)\mathbf{I}\,,$$

где $\mathbf{I}$ — единичный тензор; $\vec{W}$ — скорость среды.

Поток вязкого сжимаемого газа будем описывать в виде системы уравнений Навье–Стокса (уравнение неразрывности, импульса и энергии):

$$\frac{\partial \rho}{\partial t} + \operatorname{div}\left(\rho \vec{W}\right) = 0\,,$$

$$\frac{\partial \rho u}{\partial t} + \operatorname{div}\left(\rho u \vec{W} - \mathbf{P}\right) = 0\,,$$

$$\frac{\partial \rho v}{\partial t} + \operatorname{div}\left(\rho v \vec{W} - \mathbf{P}\right) = 0\,,$$

$$\frac{\partial \rho w}{\partial t} + \operatorname{div}\left(\rho w \vec{W} - \mathbf{P}\right) = 0\,,$$

$$\frac{\partial \rho E}{\partial t} + \operatorname{div}\left(\rho E \vec{W} - \mathbf{P}\vec{W}\right) = 0\,.$$

Система замыкается с помощью какого-либо уравнения состояния, например, уравнения состояния идеального газа:

$$J = \frac{p}{(\gamma - 1)\rho}\,,$$

где $J = E - \dfrac{W^2}{2}$.

В модели отсутствует явный учёт турбулентных эффектов. Однако, при использовании достаточно мелкой сетки, турбулентные вихри соответствующего масштаба могут проявляться (DNS-моделирование турбулентности). Также нами не рассматриваются эффекты теплопроводности.

3. Метод крупных частиц

Для численной реализации представленной модели был выбран метод крупных частиц, который позволяет получить характеристики нестационарных течений газа, а также их стационарные значения с помощью процесса установления. Использование такого подхода особенно целесообразно в задачах, когда имеет место разнохарактерное развитие по времени физического явления, например, изучение трансзвуковых потоков газа, в расчете обтекания конечных тел и др., где при довольно быстром установлении большей части поля структура течения в местных сверхзвуковых зонах, областях срыва и т.п. формируется сравнительно медленно.

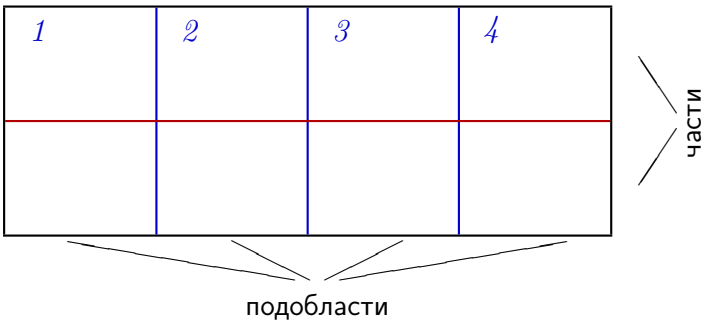
Суть метода состоит в расщеплении по физическим процессам нестационарной системы уравнений, записанной в форме законов сохранения. Среда моделируется системой из жидких (крупных) частиц, совпадающих в рассматриваемый момент времени с ячейками эйлеровой сетки. Расчет каждого временного шага разбивается на 3 этапа [5].

Эйлеров этап — пренебрегают всеми эффектами, связанными с перемещением элементарной ячейки (потока масс через границы ячеек нет), и учитываются эффекты ускорения жидкости лишь за счет давления; здесь для крупной частицы определяются промежуточные значения искомых параметров — скорость, энергия.

Лагранжев этап — используют промежуточные значения параметров потока, полученные на предыдущем этапе, вычисляются потоки массы через границы эйлеровых ячеек.

Заключительный этап — окончательные значения газодинамических параметров потока (скорость, энергия, плотность) определяются для нового момента времени на основе законов сохранения массы, импульса и энергии для каждой ячейки и всей системы в целом на фиксированной расчетной сетке.

a.



b.

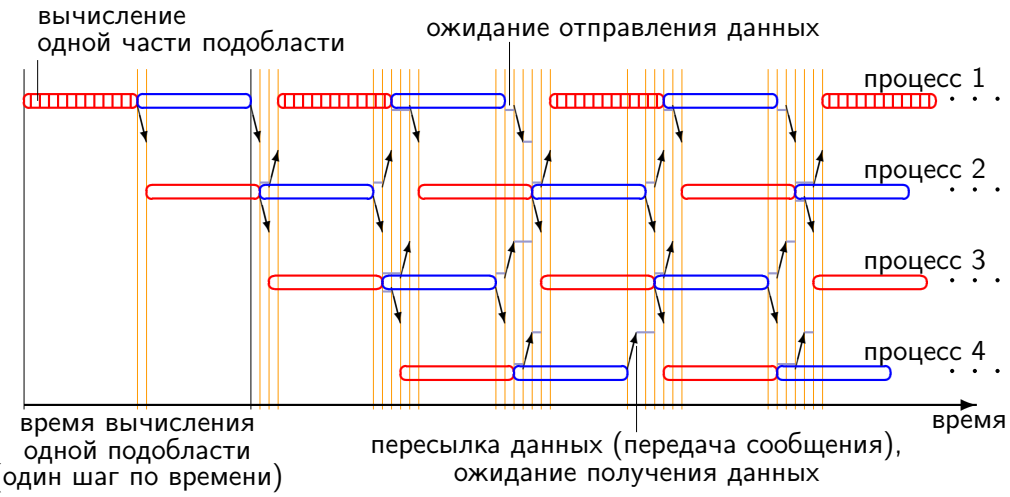


Рис. 1. Расчетная область (a) и пространственно-временная диаграмма (b) для случая декомпозиции по топологии «линейка» с дополнительным разбиением подобластей на две части.

4. Параллельный алгоритм

Первая подзадача является весьма ресурсоёмкой при расчёте в трёхмерной области и при учёте мелких деталей (большое количество узловых точек вычислительной сетки), в связи с чем возникла необходимость использования методов высокопроизводительных вычислений.

Однако прямое использование параллелизма по пространству для данной задачи представляется невыгодным. Это связано с тем, что в ряде случаев в указанной задаче возникают слишком протяжённые теневые границы и, соответственно, простой процессоров (низкая эффективность распараллеливания) в моменты обмена теневыми гранями.

Улучшить загрузку процессоров позволяет такая организация вычислительного процесса, когда подобласть рассчитывается не целиком, а по частям, с промежуточной пересылкой данных.

На Рис. 1(a) представлена схема разбиения расчётной области, предусматривающая выделение в каждой подобласти двух частей. При этом обмен теневыми гранями производится после расчёта каждой из частей подобласти. Такой подход позволяет заметно увеличить производительность, что отражено на пространственно-временной диаграмме на Рис. 1(b). З аштрихованные горизонтальные полоски на диаграмме описывают процесс вычисления процессом первой части своей подобласти, а незаштрихованные — вычисление второй части.

Для объяснения роста производительности рассмотрим ход вычислительного процесса.

После окончания расчета первой части подобласти, каждый процесс отсылает необходимые результаты соседям и, не дожидаясь получения данных, переходит к расчёту второй

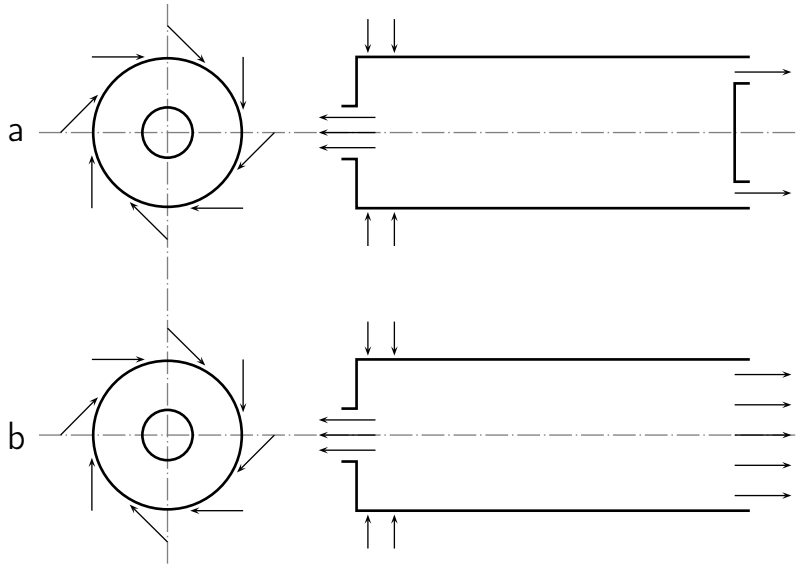


Рис. 2. Принципиальная схема вихревой трубы: (a) классическая потивоточная; (b) используемая в представленной модели

части своей подобласти. К моменту завершения этого расчёта данные, отправленные ранее уже получены, потому каждый процесс рассылает теневые грани после расчёта второй части подобласти и переходит к следующему временному шагу. Далее процесс повторяется.

В том случае, если время расчёта одной части подобласти значительно превосходит времена пересылок данных, эффективность представленного алгоритма оказывается весьма высока.

5. Вычислительный эксперимент

На основе представленной модели нами проведен ряд расчётов по численному моделированию вихревого эффекта. Рассматривалась модель противоточной вихревой трубы. В литературе [3] для конструкции вихревой трубы в качестве оптимальных приводятся следующие параметры: диаметр $D = 94$ мм; длина $L = 520$ мм (отношение длины к диаметру $L/D \approx 5,5$); диаметр диафрагмы холодного воздуха $d_c = 35$ мм ($d_c/D \approx 0,37$). Указанные параметры использовались при построении геометрии модели. Была выбрана модель, как это показано на рис. 2(a). Так же производились расчеты, когада диаметр диафрагмы горячего воздуха был взят совпадающим с диаметром самой трубы, как это показано на рис. 2(b).

Подача газа осуществляется при помощи тангенциального завихрителя, расположенного на одном из концов трубы (на схемах рис. 2 — слева), и характеризуется постоянным расходом. Горячий газ выходит через диафрагму на противоположном к завихрителю конце трубы, а холодный газ отбирается через центральную диафрагму, которая располагается на том же конце трубы, где и завихритель.

Параметры задачи для противоточной вихревой трубы:

- вязкий газ ($\rho_g \approx 1$ кг/м^3, $\mu \approx 10^{-5}$ Па·с);
- расчетная сетка с ≈ 100 тыс. ячеек;
- число Рейнольдса ($\mathrm{Re} \approx 5000$).

На рис. 3–5 представлены результаты моделирования течения вязкого газа в вихревой трубе, параметры которой описаны выше. На данных рисунках представлены распределения полной энергии, продольной и радиальной составляющих скорости в продольном сечении вихревой трубы.

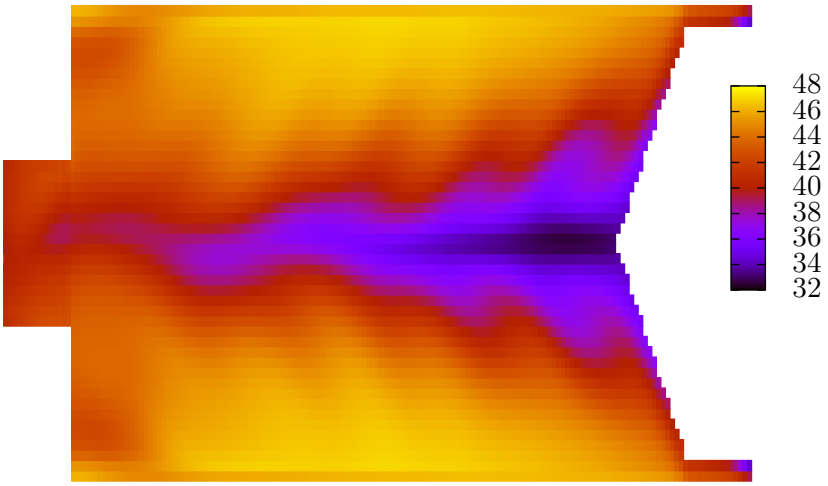


Рис. 3. Распределение температуры газа в продольном сечении вихревой трубы

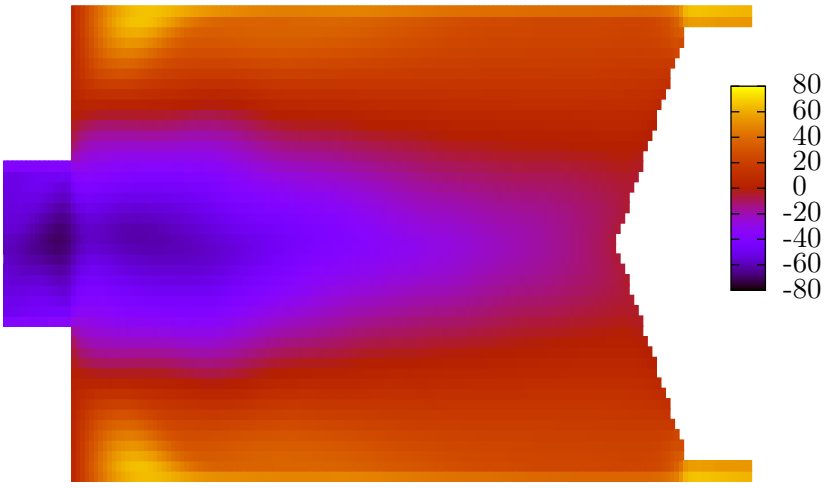


Рис. 4. Распределение продольной составляющей скорости газа в продольном сечении вихревой трубы

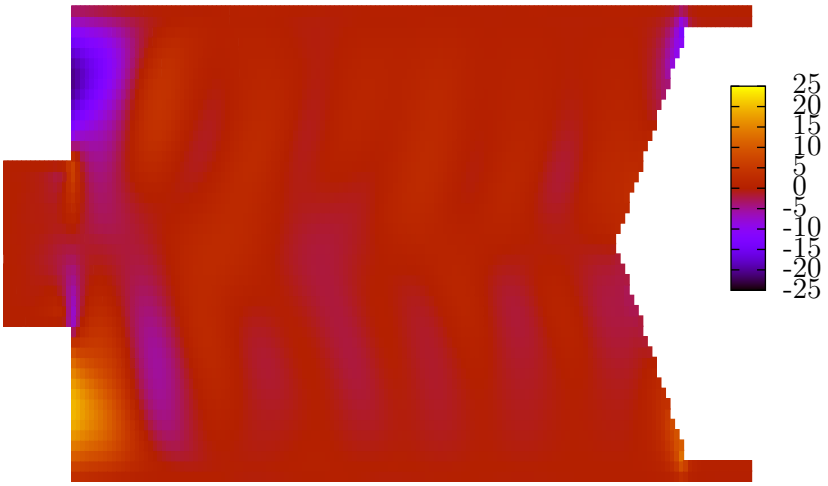


Рис. 5. Распределение поперечной составляющей скорости газа в продольном сечении вихревой трубы

На рис. 3 приведено распределение температуры газа в плоскости, расположенной вдоль канала вихревой трубы и проходящей через центральную ось. Из рисунка хорошо видно, что энергия газа имеет минимум в районе оси канала.

Эпюра продольной составляющей скорости приведена на рис. 4. Хорошо видно, что вдуваемый через тангенциальный завихритель газ имеет нулевую составляющую продольной скорости и постепенно разгоняется при движении в направлении диафрагмы горячего воздуха. Однако в центре канала, где формируется возвратный вихрь, продольная составляющая скорости газа ниже, а в районе диафрагмы холодного воздуха она отрицательна (то есть направлена из вихревой трубы наружу).

Наконец, на рис. 5 показана радиальная составляющая скорости газа в том же продольном сечении вихревой трубы. Здесь можно видеть характерный для вихревых труб высокий уровень турбулентности потока. Так, по мнению А. Ф. Гуцола [3], именно такие интенсивные турбулентные пульсации в радиальном направлении способствуют температурному разделению газа.

6. Результаты распараллеливания

На рис. 6-8 показаны графики времени выполения,ускорени, эффективности от числа узлов.

Для объяснения роста производительности рассмотрим ход вычислительного процесса.

После окончания расчета первой части подобласти каждый процесс отсылает необходимые результаты соседям и, не дожидаясь получения данных, переходит к расчету второй части своей подобласти. К моменту завершения этого расчета данные, о тправленные ранее, уже получены, потому каждый процесс рассылает т еневые грани после расчета второй части подобласти и переходит к следующему временному шагу. Далее процесс повторяется.

В том случае, если время расчета одной части подобласти значительно превосходит времена пересылок данных, эффективность представленного алгоритма оказывается весьма высока.

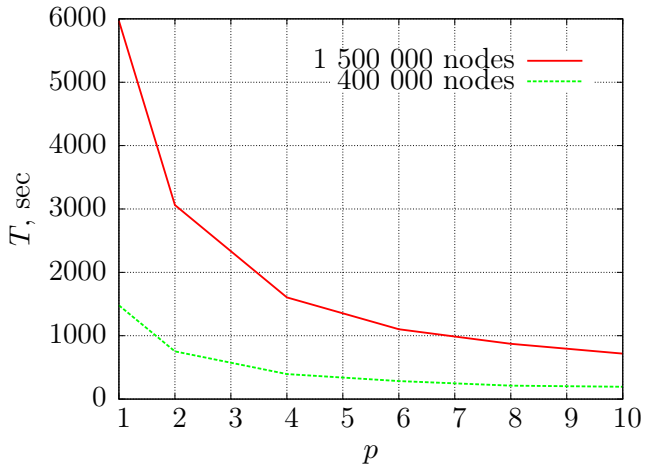


Рис. 6. Завивисимость вмемени расчетов от числа узлов

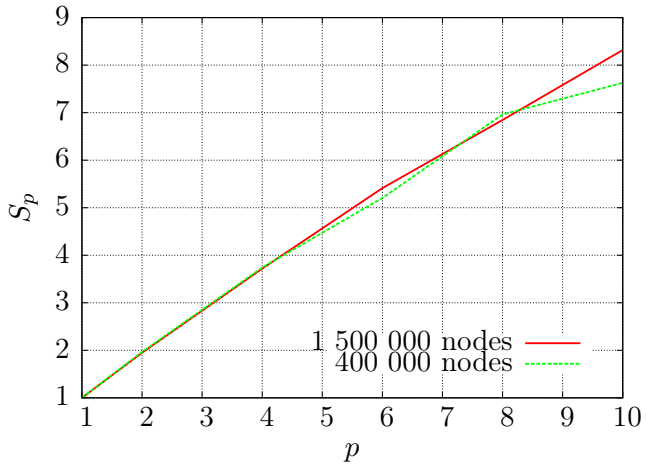


Рис. 7. Завивисимость ускорени от числа узлов

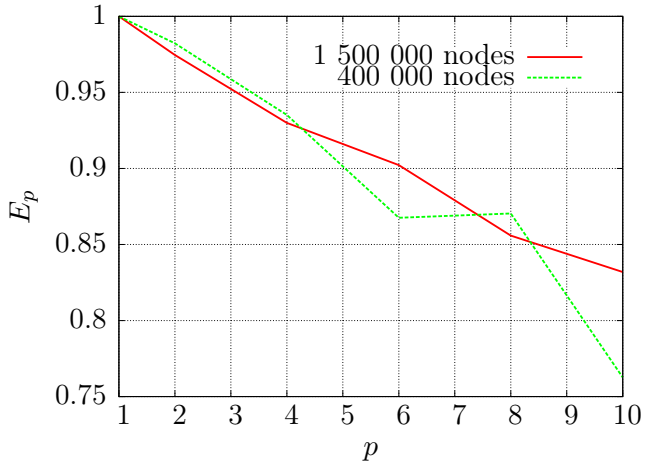


Рис. 8. Завивисимость эффективности от числа узлов

7. Заключение

Таким образом, показано, что прямое численное моделирование на основе уравнений Навье–Стокса позволяет рассчитывать эффекты, происходящие внутри вихревой трубы. Даже в представленном упрощённом подходе эффект «температурной» (в данном случае — энергетической) стратификации потока хорошо заметен. Более того, в представленной модели произведено ещё одно упрощение: диафрагмой горячего воздуха является весь торец вихревой трубы, противоположный тангенциальному завихрителю. Скорее всего данное упрощение привело к уменьшению эффекта температурной стратификации.

Возможно на величину эффекта влияет DNS турбулентности на недостаточно мелкой сетке. Однако значительные времена счёта даже по такой модели показывают необходимость вводить в математическую модель учет турбулентности.

Полученные результаты показывают, что предлагаемые математическая модель и численный подход адекватны рассматриваемому явлению и должны развиваться как для исследования самого эффекта Ранка–Хилша, так и для решения технических задач, ориентированных на использование данного эффекта.

Список литературы

1. Ranque, G. J. Expériences sur la Détente Giratoire avec Productions Simultanées d'un Echappement d'air Chaud et d'un Echappement d'air Froid // J. Phys. Radium., 4, 1933. Pp. 112–114.
2. Hilsch, R. Die Expansion von Gasen im Zentrifugalfeld als Kälteprozess // Zeitung für Naturforschung, 1, 1946. Pp. 208–214.
3. Гуцол А. Ф. Эффект Ранка //Успехи физических наук. 1997. Т. 167, № 6. С. 665–687.
4. Akhesmeh S. Numerical Study of the Temperature Separation in the Ranque-Hilsch Vortex Tube // American J. of Engineering and Applied Sciences, 1(3), 2008. Pp. 181–187.
5. Белоцерковский О. М., Давыдов Ю. М. Метод крупных частиц в газовой динамике. М.: Наука, 1982. 392 с.

COACCEL — конвейерная вычислительная среда для multi-GPU

Д.Н. Микушин[1], О.А. Левченко[2], А.П. Петров[3]

Институт Вычислительной Математики РАН[1], Геофизический Центр РАН[2],
Сибирский научно-исследовательский гидрометеорологический институт[3]

В данной работе представлена распределенная конвейерно-гибридная вычислительная среда COACCEL. Унификация программных интерфейсов взаимодействия составных частей гибридной системы в COACCEL позволяет масштабировать существующее GPU-приложение для массивно-параллельных систем. Управление потоками данных реализовано с помощью программного конвейера, стадии которого выполняют асинхронные операции на нескольких уровнях иерархии памяти. Анализ корректности выполнен на примере реализации разностных схем и функций BLAS. Результаты тестовых измерений приведены для распределенной среды, построенной из многоядерных процессоров и ускорителей GT200.

1. Введение

Доля гибридных суперкомпьютеров в последние годы существенно возросла. На конец 2010 г. в рейтинге top500 значится 28 систем гибридной архитектуры, когда как всего два года назад была лишь одна. В числе преимуществ подобных систем обычно называется большая вычислительная плотность в отношениях $/FLOPS и Ватт/FLOPS, когда как другая важная характеристика остаётся в тени: *программируемость*. Насколько трудоёмкой может оказаться разработка приложения с интенсивным обменом данными для такой системы, если потребовать чтобы эффективность реальных FLOPS приложения оставалась столь же привлекательной, как пиковые показатели и результаты на стандартных тестах?

Вокруг любой вычислительной технологии формируется экосистема средств разработки и вспомогательных библиотек. Наиболее развитая экосистема построена вокруг MPI и OpenMP, части которой перерабатываются для использования NVIDIA CUDA, OpenCL и Cell, формируя их собственные экосистемы. В результате гибридному суперкомпьютеру, основанному на комбинации каких-либо из перечисленных технологий, некоторые стандартные функции становятся доступными на всех уровнях. Однако сумма отдельных проработанных составляющих с точки зрения программируемости - это ещё не то же самое, что простая однородная система. Поэтому ответ на ранее поставленный вопрос зависит от того, насколько сочетаются друг с другом представления отдельных уровней о гибридной системе, и в какой мере их замкнутость возможно ослабить с помощью обобщений.

Рассмотрим гибридную систему, состоящую из многопроцессорных узлов с графическими ускорителями. Узлы соединены с помощью сети Infiniband, а массивы ускорителей подключены к параллельной PCI-Express шине узлов (рис. 1). Для облегчения восприятия карта Infiniband на схеме представлена ниже GPU, хотя она, вообще говоря, так же является PCI-E устройством[1]. Пунктирной линией на схеме показан путь, по которому выделенная, в качестве примера, пара GPU может обмениваться данными. Как видно, он состоит из звеньев (*интерфейсов*), соединяющих локальную память различных устройств.

$$GPU_1 \xrightarrow{cudaMemcpy} RAM_1 \xrightarrow{MPI\_Send/MPI\_Recv} RAM_2 \xrightarrow{cudaMemcpy} GPU_2 \qquad (1)$$

[1]Поскольку Infiniband - это PCI-E устройство, существует теоретическая возможность исключить из иерархии звенья, соответствующие памяти хостов и передавать данные напрямую по PCI-E шине, но такой режим в драйверах не релизован.

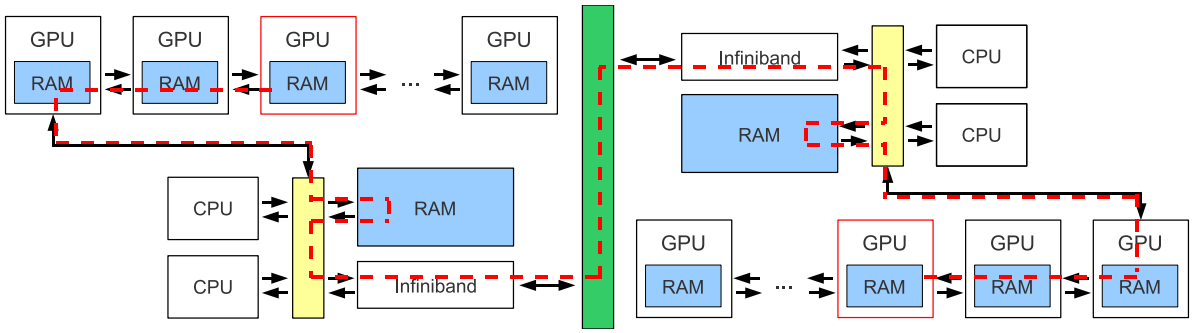


Рис. 1. Пара узлов гибридной вычислительной системы

Назовём совокупность интерфейсов и уровней памяти *иерархией памяти.*

Другой типичный пример — потоковая обработка данных, находящихся в памяти узла. Обычно для ускорения передачи данных из памяти узла по PCI-Express используется некешируемая (*nonpagable*) память, которая в случае большого объёма данных может присутствовать лишь в качестве дополнительного буфера. В этом случае полный цикл пересылки, обработки и возврата данных для каждого фрагмента будет состоять из четырёх стадий:

$$\begin{cases} RAM \xrightarrow{memcpy} RAM_{pinned} \xrightarrow{cudaMemcpyAsync} GPU, \\ GPU \xrightarrow{cudaMemcpyAsync} RAM_{pinned} \xrightarrow{memcpy} RAM \end{cases} \tag{2}$$

Из-за привязки контекстов устройств к потокам, каждая последовательность операций пересылки с участием GPU требует либо независимой обработки в отдельном потоке, либо переключения контекстов.

Ещё один аспект, усложняющий общую картину — асинхронность. Использование асинхронных обменов данными часто является единственным способом повышения эффективности вычислений, в случае если времена пересылок и расчётов соизмеримы. Это потребует в дополнение к многослойности функций обмена и потокам реализовывать в программе нити или конвейер.

Рассмотренные примеры показывают, что на каждом уровне памяти соответствующая технология задействует собственные механизмы обмена данными, которые, хоть и осложнены некоторыми особенностями, тем не менее функционально очень похожи. Из этих соображений получение общего единообразного интерфейса обмена данными действительно может позволить уменьшить сложность программирования гибридной системы.

Таким образом, целью настоящего исследования одновремено не является ни решение какой-либо узко специальной задачи, ни чрезмерно идеалистичное обобщение. Задача состоит в выработке базовых примитивов организации вычислений на гибридных системах, которые могут быть положены в основу промышленных библиотек и численных моделей, позволяя, к примеру, расширить действие библиотеки CUBLAS с одного GPU на несколько подобно тому, как в Intel MKL по сравнению с классическими BLAS/LAPACK реализована внутренняя многопоточность.

В следующих далее разделах статьи излагается функциональная суть некоторых базовых примитивов и алгоритмы их построения. Программная реализация представляет собой библиотеку COACCEL, для которой демонстрируются основные функции пользователя и пример производительности одного приложения. С целью получения большего числа экспертных оценок и привлечения сообщества к разработке тестовых приложений, приведены инструкции по загрузке актуальной версии исходного кода.

2. Постановка задачи

Логика выбора функциональных примитивов может быть основана на анализе некоторых свойств задач, эффективная параллельная реализация которых наиболее трудоёмка, таких как, например, случайный доступ к данным или разбаллансировка нагрузки, связанная с динамическим перестроением сеток. С другой стороны, нежелательно чрезмерно специализировать общий механизм в ущерб его применимости, например, передача библиотечной функции слишком большого контроля над управлением памятью или обменом данными потенциально ведёт к допущению лишь конкретного порядка следования элементов в памяти, т.е. привязке к типу сетки. Попытка полностью уйти от предметной специфики оставляет доступными для рассмотрения лишь несколько возможностей:

- Базовые свойства алгоритмов, например асинхронность
- Примитивы, на которых основана работа самой гибридной системы, например контекст или поток
- Синтетические конструкции — представление составных операций как новых базовых, например сведение (в представлении пользователя) рассмотренного выше обмена данными по иерархии памяти к простой команде асинхронного копирования

На этой основе сформулируем следующие задачи:

1. Предложить алгоритм и прототип системы одновременного исполнения GPU-программы на нескольких устройствах, управляемых в различных потоках хост-системы, каждый — эксклюзивно для конкретного GPU. Выделенной группе устройств создать хост-потоки для выполнения заданных пользователем функций инициализации, деинициализации и циклической обработки данных с полной или выборочной барьерной синхронизацией после каждой итерации.
2. Разработать алгоритмы и реализацию конвейера по иерархии памяти гибридной системы с поддержкой асинхронного копирования. Конвейер должен перемещать порции начального массива данных заданного размера по иерархии памяти, задаваемой последовательностью устройств. Если одно из устройств помечено как расчётное, то к данным, поступающим в его локальную память применяется заданная функция, и далее по иерархии отправляются изменённые данные. При копировании не должно делаться никаких допущений о структуре данных, вместо этого пользователю предлагается через обратный вызов самостоятельно формировать список сегментов данных, подлежащих пересылке, в зависимости от номера порции и уровня.
3. Используя предыдущие два примитива, построить общий явный метод разностного решения эволюционных уравнений на регулярной сетке с полуширинами разностного шаблона, задаваемыми с помощью параметров. Обобщение состоит вызове заданной функции пересчёта на каждом шаге по времени и автоматической синхронизации внутренних границ расчётной области (заданных ширин), образованных в результате декомпозиции.

Перечисленные задачи необходимо решить, предполагая, что в качестве ускорителей гибридной системы могут быть как устройства с поддержкой CUDA, так и OpenCL или других технологий.

3. Основные алгоритмы

3.1. Конвейерная пересылка данных

Аппаратные конвейеры присутствуют в процессорах и контроллерах, начиная едва ли не со времён самых ранних электронных систем 40-х годов. Принцип разделения составной операции на стадии асинхронного выполнения в длинном потоке команд позволяет минимизировать простой функциональных элементов. Если время *пролога* (начального насыщения функциональных элементов полезной работой) и *эпилога* много меньше времени конвейерной фазы (*pipeline steady state*), то можно считать, что общее время операции вместо бесконвейерного суммирования времён составляющих $\sum_i t_i$ стремится к $\max_i t_i$ — времени исполнения самой долгой операции.

Программные конвейеры (*software pipelining*) так же широко используются, в том числе для организации пересылки данных, например, в программных моделях CUDA [1] и Cell [2]. Однако чаще всего встречаются схемы, реализующие только самый простой вариант из трёх асинхронных операций — загрузка, расчёт и выгрузка. Для случая иерархической памяти данные требуется пропустить уже через несколько промежуточных уровней, общее число которых может изменяться.

Пусть данные длины L, поделенные на m порций равной длины l пересылаются с помощью конвейера по иерархии памяти из n уровней в прямом и обратном направлении. На каждом уровне памяти создадим пару из входящего и исходящего буфера длины l для обоих направлений движения данных. Тогда простой конвейер может быть реализован как цикл изменения состояний порций данных, проходящих стадии $LOAD$, $SAVE$, $SYNC$, $COMP$ на уровнях иерархии $\overline{0,n}$, в соответствии с правилами перехода (3).

3.2. Решение сеточных задач

Пусть в результате декомпозиции расчётой области на каждом из устройств получена часть результатов на очередном шаге по времени. Граничные полосы результирующих трёхмерных массивов подлежат синхронизации с результатами устройств, обрабатывающих соседние части расчётной области. Поскольку система гибридная, различные типы устройств могут использоваться для расчётов, например CPU и GPU. В таком случае для каждой пары устройств необходимо установить отдельный конвейер, а в случае двух GPU, работающих в разных потоках - два связанных конвейера для достижения ближайшей памяти хоста и обмена данными.

$$\begin{cases} & \to \quad LOAD^1, \\ LOAD^i & \to \quad SYNC^i, \\ SYNC^i & \to \quad LOAD^{i+1}, \qquad i = \overline{1, n-1}, \\ SYNC^n & \to \quad COMP^n, \\ COMP^n & \to \quad SAVE^n, \\ SAVE^i & \to \quad SYNC^i, \qquad i = \overline{n, 0} \\ SYNC^i & \to \quad SAVE^{i-1}, \qquad i = \overline{n, 1}. \end{cases} \tag{3}$$

- Если одно из устройств - это GPU, то конвейер должен работать в реверсивном режиме, т.е. стадии LOAD и SAVE меняются местами.

- Если устройства относятся к разным узлам, то на каждом узле может присутствовать

лишь одна часть конвейера, проходящая только по локальным узлам, но согласованная с удалённой идентификатором сообщений.

При динамическом построении конвейеров между заданными устройствами возникает задача поиска наилучшего пути по иерархии памяти, которую можно интерпретировать как поиск кратчайшего пути в заданном графе допустимых переходов между поддерживаемыми типамис устройств (может быть решена с помощью Алгоритма Дейкстры [3]).

4. Программная реализация

Для поддержки компонентов гибридной системы различного типа (в настоящий момент — CPU, CUDA и MPI в синхронных и асинхронных режимах) библиотека COACCEL реализует интерфейсы, представленные на Рис. 2. Чтобы добавить в библиотеку поддержку нового устройства, достаточно реализовать для него эти функции.

```
void* coaccel_memcpy_start(
        coaccel_device device, int count,
        coaccel_address* dst, coaccel_address* src,
        size_t* size, int dir);

void coaccel_memcpy_finish(coaccel_device device, void* desc);

void coaccel_device_lock(coaccel_device device);

void coaccel_device_unlock(coaccel_device device);

coaccel_address coaccel_device_malloc(
        coaccel_device desc, size_t size);
```

Рис. 2. Функции, реализуемые для устройств гибридной системы

Интерфейс *multi* (Рис. 3) реализует управление несколькими GPU в отдельных потоках, *pipeline* (Рис. 4) — конвейерную обработку с заданными пользователем функциями

```
typedef int (*coaccel_multi_func_t)(
        coaccel_device_group group, coaccel_device device,
        int idevice, int ithread, void* data);

coaccel_multi coaccel_multi_init(
        coaccel_device_group group, int nthreads,
        coaccel_multi_func_t thread_init,
        coaccel_multi_func_t thread_deinit,
        void* data);

void coaccel_multi_step_all(coaccel_multi desc,
        coaccel_multi_func_t process, void* data);

void coaccel_multi_finalize(coaccel_multi desc, void* data);
```

Рис. 3. Интерфейс *multi*, решение задачи №1

4.1. Тестовая задача

Примером работы реализованных интерфейсов является тест, основанный на функциях BLAS/CUBLAS, выполняющий для заданных матриц $L[m \times m]$, $K[m \times m]$, $R[m \times n]$, $Q[m \times n]$ последовательность матричных операций (4). Предполагается, что m достаточно мало, чтобы матрицы L и K помещались в локальную память каждого GPU, а n достаточно велико, чтобы матрицы R и Q можно было загрузить на GPU лишь частично. Расчёт выполняется по схеме поточной обработки (2), параллельно на заданном числе GPU.

```
typedef coaccel_memreq (*coaccel_pipeline_transfer_t)(
        int istage, size_t ihost, size_t idevice, size_t szchunk,
        void* data);

typedef int (*coaccel_pipeline_process_t)(
        int mode, size_t ihost, size_t idevice, void*);

coaccel_pipeline coaccel_pipeline_init(
        coaccel_device_group devices,
        size_t size, size_t szchunk, int reverse,
        coaccel_pipeline_transfer_t load_func,
        coaccel_pipeline_transfer_t save_func,
        coaccel_pipeline_process_t process_start_func,
        coaccel_pipeline_process_t process_sync_func, void* ptr);

void coaccel_pipeline_process(coaccel_pipeline desc);

void coaccel_pipeline_done(coaccel_pipeline desc);
```

Рис. 4. Интерфейс *pipeline*, решение задачи №2

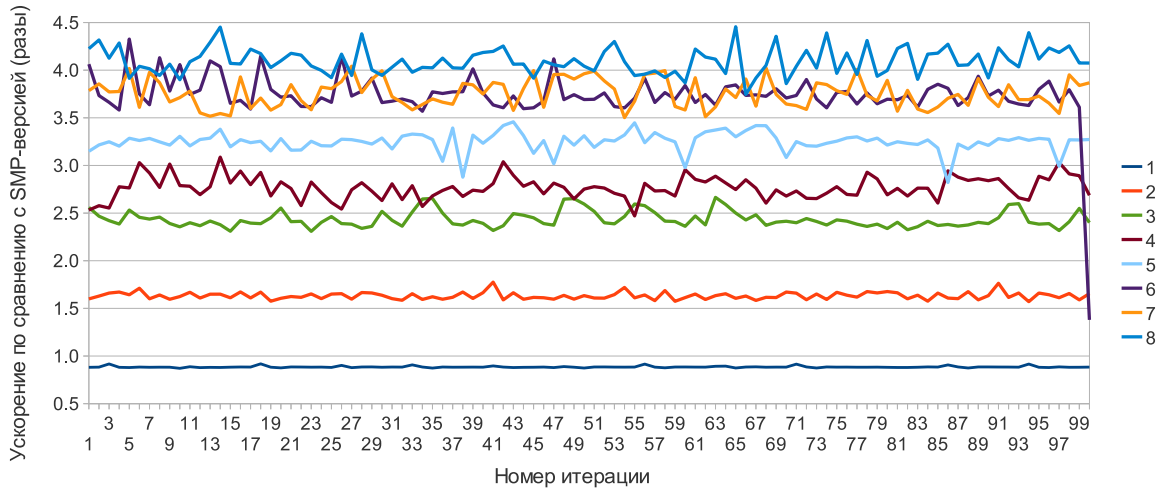


Рис. 5. Тестовая задача для гибридной SMP-системы Intel Xeon X5670 + 2×NVIDIA Tesla S1070 (8 GPUs)

$$\begin{aligned} R &:= 0.5(Q - R), & Q &:= 0.5(Q + R), \\ R &:= L \times R, & Q &:= K \times Q, \\ R &:= R + Q, & Q &:= R - Q. \end{aligned} \tag{4}$$

Тестирование проведено на сервере с двумя 6-ядерными процессорами Intel Xeon и двумя составными GPU S1070 (по 4 устройства архитектуры GT200 в каждом). На Рис. 5 представлена динамика изменения производительности теста в разах (ось Y) по сравнению с CPU-версией, использующей Intel MKL для каждой из последовательных 100 итераций метода (ось X). Каждая кривая f_{ngpus}, $ngpus = \overline{1,8}$ показывает наихудшую производительность GPU в соответствующей группе: $f_{ngpus}(t) = \min_i f_i(t)$, $i = \overline{1, ngpus}$ - номер GPU в группе, t - номер итерации. Проще говоря, время каждой итерации учитывает полную синхронизацию устройств.

Результаты данного теста получены для $m = 1024$, $n = 131072$, все вычисления проводились с двойной точностью. Примечательно, что обычная скорость пересылки данных до-

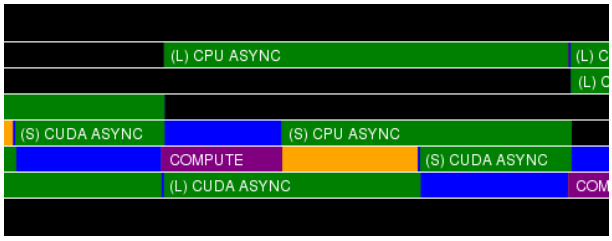


Рис. 6. Пример визуального представления работы программного конвейера (тестовая задача для гибридной SMP-системы).

стигается при задании в интерфейсе *pipeline* размера буфера $szchunk \geq 8$, что при четырёх таких буферах и дополнительных массивах в сумме составляет менее 64 Мбайт. Таким образом потоковая обработка оказывается совершенно не требовательной к размеру собственной памяти GPU. Однако необходимо учитывать, что в случае когда время пересылки данных становится сравнимым со временем вычислений, потери производительности с синхронным конвейером становятся значительными. На Рис. 6 визуализированы стадии обработки данных синхронного конвейера для того же самого теста, но на GPU Tesla C2050. Из графика видно, что если пересылки данных и вычисления не накладываются, то примерно половину времени устройство простаивает.

5. Заключение

В работе предложено несколько базовых конструкций, призванных сделать гибридный суперкомпьютер с графическими ускорителями («кластер из кластеров») более целостной системой с точки зрения программной модели обмена данными. По алгоритмам составлены программные ализации, корректность которых проверяется набором из 16 различных тестов, а быстродействие — тестовыми приложениями. Исходный код доступен по адресу [4].

Дальнейшее развитие проекта может быть связано с подготовкой подробной документации к примерам и разработкой большего числа демонстрационных приложений, в том числе для случаев асинхронного конвейера и сеточных задач, которые позволили бы лучше оценить применимость библиотеки в реальных условиях.

Литература

1. Shinta Nakagawa, Fumihiko Ino and Kenichi Hagihara. A middleware for efficient stream processing in CUDA // Computer Science - R&D, Vol. 25, Number 1-2, pp. 41-49, 2010.

2. Michael Kistler, Michael Perrone and Fabrizio Petrini. Cell Multiprocessor Communication Network: Built for Speed. // IEEE Micro, Vol. 26, pp. 10–23, 2006.

3. E. W. Dijkstra. A note on two problems in connection with graphs. // Numerische Mathematik. V. 1 (1959), Springer-Verlag, Berlin, P. 269-271

4. Микушин Д.Н. и др. COACCEL — the generalized framework for distributed computing on accelerators: URL: http://tesla.parallel.ru/trac/coaccel .

Некоторые модели многопроцессорных систем обслуживания с тяжелыми хвостами *

Е. В. Морозов, А. С. Румянцев

Учреждение Российской академии наук
Институт прикладных математических исследований
Карельского научного центра РАН

Работа посвящена изучению свойств процесса загрузки в многопроцессорных системах обслуживания с распределениями, обладающими тяжелым хвостом. Такие распределения могут приводить к появлению долговременной зависимости, что существенно затрудняет построение надежных статистических оценок. Построена модель вычислительного кластера, являющаяся модификацией классической многосерверной системы обслуживания. Приведены результаты статистических экспериментов.

1. Введение

Теоретический интерес последних двух десятилетий к моделям систем обслуживания, описываемых распределениями с так называемыми тяжелыми хвостами, обусловлен обнаружением этого эффекта в экспериментах по изучению компьютерных систем и сетей (см., напр., [4]). Более медленное, чем экспоненциальное, убывание хвостовой вероятности $\overline{F}(x) := P(X > x)$ для заданной случайной величины X (*«тяжелый хвост распределения»*), т.е.

$$\lim_{x\to\infty} e^{-ax}\overline{F}(x) = \infty, \quad \forall a > 0 \tag{1}$$

ограничило ранее повсеместное использование экспоненциальных распределений для моделирования процессов в современных компьютерных системах. Эмпирически подтверждено присутствие тяжелых хвостов в распределениях размеров файлов на жестких дисках, времен вычисления задач на процессоре, времен передачи файлов между сервером и клиентом в высокоскоростных сетях и т.д. [4]. Наиболее важными для аналитического исследования таких процессов подклассами распределений с тяжелыми хвостами являются *субэкспоненциальные распределения*, которые определяются как

$$\lim_{x\to\infty} \frac{P(X_1 + X_2 > x)}{P(X > x)} = \lim_{x\to\infty} \frac{\overline{F} * \overline{F}(x)}{\overline{F}(x)} = 2, \quad x \geqslant 0, \tag{2}$$

где $\overline{F} * \overline{F}(x) = \int_0^\infty \overline{F}(x-y)\overline{F}(y)\,dy$ означает свертку распределений (для неотрицательных с.в.), а также *правильно меняющиеся распределения*, характеризуемые соотношением

$$\lim_{x\to\infty} \frac{\overline{F}(xt)}{\overline{F}(x)} = t^{-\alpha}, \quad \alpha \geqslant 0. \tag{3}$$

(При $\alpha = 0$ функция F называется медленно меняющейся). Наиболее часто используемым является распределение Парето, имеющее (стандартный) вид

$$\overline{F}(x) = x^{-\alpha},\ x \geqslant 1, \quad \overline{F}(x) = 1,\ x \leqslant 1. \tag{4}$$

Отметим, что хороший обзор субэкспоненциальных распределений представлен в работе [7], свойства регулярно меняющихся функций подробно обсуждаются в [12], а работа [8] посвящена применению распределений с тяжелыми хвостами к сетевым моделям и системам обслуживания.

*Работа поддержана РФФИ, проект 10-07-00017.

Однако, обнаружение тяжелых хвостов в компьютерных системах поставило ряд проблем, недооценивание которых приводит к серьезным последствиям как для компаний-производителей сетевого оборудования, так и для потребителей. (Частично эти проблемы обсуждаются в [16].) Одна из проблем состоит в том, что многие алгоритмы управления сетевыми устройствами и процессами в вычислительных системах строились с учетом предположения об экспоненциальности соответствующих распределений [9].

Присутствие тяжелого хвоста на уровне одного клиента системы или сети (состоящей из нескольких обслуживающих устройств) в результате суперпозиции большого числа независимых источников приводит к тому, что сетевой процесс $\{X(t), t > 0\}$ ведет себя не регулярно [10]. Точнее говоря, имеет место следующее равенство по распределению

$$C^{-H}\{X(Ct), t > 0\} =_d \{X(t), t > 0\}, \quad H \in (0.5, 1]$$

для шкалированного процесса $C^{-H}\{X(Ct), t > 0\}$. Такой процесс называется *самоподобным*, а постоянная H называется параметром Херста.

В таком процессе, в частности, возникают длительные периоды пребывания процесса $\{X(t)\}$ выше или ниже своего среднего значения, которые можно трактовать или как отсутствие стационарности, или как *долговременную зависимость* (долгую память) («эффект Иосифа» [18]). Аналитически долгая память выражается в расходимости ряда автокорреляций процесса, что (в случае дискретного времени) выражается как

$$\lim_{t\to\infty} \sum_{k=1}^{t} corr(X_0, X_k) = \infty. \tag{5}$$

Расходимость ряда (5) приводит к сложностям, связанным с надежностью статистических оценок параметров системы, выражающих качество обслуживания (QoS), например задержки в системе [1]. В этой связи отметим классическую работу [14], посвященную самоподобию и долговременной зависимости сетевого трафика.

Практический интерес для исследователя часто представляет вероятность большого уклонения (например, вероятность переполнения большого буфера, вероятность большого времени ожидания в системе и т.д.), а также моментные свойства сетевых процессов, в частности, моментный индекс, т.е. максимальный конечный момент. Особенно важным является величина дисперсии длины (передачи) сообщения. Для пояснения заметим, что дисперсия длины сообщений, варьирующихся в современных сетях от коротких электронных сообщений до видеоконференций, часто при моделировании может быть принята равной бесконечности. Заметим, что свойства корреляции изучались при исследовании эффекта долгой памяти [5], а зависимость моментных свойств сетевых процессов от дисциплины обслуживания рассматривается в [3]. Однако такого рода результаты известны для достаточно узкого класса односерверных систем обслуживания, и в гораздо меньшей степени для многосерверных систем. Особенно сложен анализ в случае, когда заявки в систему обслуживания поступают в виде пачек/групп (см., напр., работы [13, 15, 24]), или в виде регенеративного процесса [11]. Вычислительные кластеры и Грид представляют в этом смысле особую сложность, т.к. обладают вышеперечисленными свойствами, такими как многопроцессорность, сложные дисциплины обслуживания потока заявок, возможность поступления групп, большие времена обслуживания на процессорах (тяжелые хвосты).

2. Модели многосерверных систем обслуживания

В этом разделе дан обзор ряда важных известных результатов для многосерверных систем обслуживания, в т.ч. с бесконечным числом серверов. Эти результаты в основном опираются на классические результаты для односерверных систем, которые также приводятся для сравнения.

2.1. Система G/G/s

Пусть в систему обслуживания, состоящую из идентичных s серверов (процессоров), в моменты $\{t_i, i \geqslant 1\}$, образующие процесс восстановления, поступают заявки с независимыми, одинаково распределенными временами обслуживания $\{S_i\}$. Обозначим интервалы между приходами заявок $T_i := t_{i+1} - t_i$, $i \geq 1$. Введем функции распределения интервала $A(x) = P(T \leqslant x)$, и времени обслуживания $B(x) = P(S \leqslant x)$ (индекс опущен, когда рассматривается типичный представитель последовательности). Заявка ожидает время $D_i \geq 0$, в очереди, формируемой в порядке поступления (FIFO), пока не освободится наименее загруженный сервер, на который она поступает на обслуживание. Можно определить вектор загрузки в системе W_i, состоящий из оставшегося времени обслуживания на каждом процессоре в момент прихода заявки i (вектор загрузки Кифера-Вольфовица [20]):

$$W_{i+1} = R\left(W_i(1) + S_i - T_i, W_i(2) - T_i, \ldots, W_i(s) - T_i\right)^+, \quad i \geqslant 1, \tag{6}$$

компоненты которого представляют незавершенные нагрузки на серверах, отсортированные в возрастающем порядке, $W_i(1) \leqslant \ldots \leqslant W_i(s)$. (Оператор $R(\cdot)$ располагает компоненты вектора в возрастающем порядке, $(x)^+ = \max(0, x)$.) Таким образом, время ожидания i-й заявки является первой компонентой вектора загрузки, $D_i = W_i(1)$. При условии $\rho := ES/ET < s$ имеет место слабая сходимость вектора загрузки к стационарному значению, $W_i \Longrightarrow W := (W(1), \ldots, W(s))$. Обозначим $U_i = W_i(2) - W_i(1)$ — время, оставшееся на втором наименее загруженном сервере в момент поступления заявка i на обслуживание. Обозначим $P_i = \min(U_i, S_i)$, тогда время ожидания D_i удовлетворяет рекурсии [20]:

$$D_{i+1} = (D_i + P_i - T_i)^+, \quad i \geqslant 1. \tag{7}$$

Заметим, что при $s = 1$ формула (7) определяет классическую рекурсию Линдли

$$D_{i+1} = (D_i + S_i - T_i)^+, \quad i \geqslant 1.$$

2.1.1. Моменты времени ожидания

Значительное развитие классических результатов о конечности моментов стационарного времени ожидания D в системе G/G/s (с дисциплиной FIFO) достигнуто в работе [23]. Именно, пусть для некоторого натурального k выполнено условие $k < \rho < k + 1 \leqslant s$. Тогда

$$E\left(S^{1+\frac{\alpha}{s-k}}\right) < \infty \quad \text{влечет} \quad E(D^\alpha) < \infty. \tag{8}$$

При дополнительных ограничениях на распределение B это условие является также необходимым.

Таким образом, ввиду (8) имеет место зависимость моментных свойств процесса загрузки от коэффициента загрузки ρ. Эта зависимость имеет ступенчатый характер, в отличие от классического результата для односерверной системы, где, при условии $ET < \infty$,

$$ES^{\alpha+1} < \infty \quad \text{тогда и только тогда, когда} \quad ED^\alpha < \infty. \tag{9}$$

Частный случай (8) при загрузке $\rho < 1$ в системе G/G/s приведен в работе [19]. Именно, для любого $2 \leqslant j \leqslant s$ в системе G/G/j имеет место следующее условие конечности момента порядка j/s времени ожидания $D(j)$:

$$ES^{1+s^{-1}} < \infty \quad \text{влечет } E\left[D(j)\right]^{j/s} < \infty. \tag{10}$$

В частности, для $j = s$

$$ES^{(s+1)/s} < \infty \quad \text{влечет} \quad ED < \infty. \tag{11}$$

В предельном случае при $s \to \infty$ получается классическое условие $ES < \infty$ конечности среднего времени пребывания в системе G/G/∞. В работе [21] приведена также верхняя граница для стационарной средней задержки в системе G/G/s.

Одним из вариантов практического применения формулы (8) может быть обоснование выбора между s «медленными» серверами (работающими со скоростью $1/s$) и одним «быстрым» (имеющим скорость 1). Действительно, если выбрать $s > 1/(1-\rho)$, то при одном и том же входном потоке задержки в s-серверной системе будут иметь конечный момент более высокого порядка, чем в односерверной. Действительно, обозначив $\beta = 1 + \alpha/(s-k)$, из (8) получим

$$ES^\beta < \infty \quad \text{влечет } E\left(D^{(s-k)(\beta-1)}\right) < \infty. \tag{12}$$

Таким образом, для улучшения моментных свойств задержки целесообразнее использовать несколько медленных серверов/процессоров, чем один быстрый [23].

Рассмотрим численный пример. Пусть система с процессором частотой 3 ГГц обслуживает процессы, времена выполнения которых имеют тяжелый хвост, $\overline{B}(x) = x^{-1.5}$, $x \geq 1$ (распределение Парето). Пусть загрузка системы составляет $\rho = 0.3$. Согласно (9), стационарное время ожидания имеет лишь конечный момент $ED^{0.5} < \infty$, т.е. бесконечное среднее. Заменим систему на двухпроцессорную, где каждый процессор имеет частоту 1.5 ГГц. Тогда загрузка системы возрастет до 0.6, и согласно (12), $ED^{2\cdot 0.5} = ED < \infty$, т.е. средняя задержка конечна. (Заметим, что поскольку $\rho < 1$ то здесь $k = 0$.) Если же заменить систему на восьмипроцессорную, то $2 < \rho < 3$ и $k = 2$, т.е. $ED^3 < \infty$ и стационарная задержка имеет конечный третий момент.

2.1.2. Моменты компонент вектора загрузки

Еще более неожиданные моментные свойства компонент вектора загрузки в многосерверной системе $(W(1), \ldots, W(s))$ получены в работе [22]. Так, если коэффициент загрузки $\rho = ES/ET < s$, то для всех компонент с индексами $i \leqslant \lceil \rho \rceil$ (наименьшее целое большее ρ) имеют место *одни и те же* достаточные условия конечности моментов порядка $\alpha \geqslant 1$:

$$ES^{(s-\lfloor\rho\rfloor+\alpha)/(s-\lfloor\rho\rfloor)} < \infty \quad \text{влечет} \quad E\left[W(i)\right]^\alpha < \infty. \tag{13}$$

Однако для компонент с индексами $i > \lceil \rho \rceil$ моментные свойства зависят от индекса i:

$$ES^{(s-i+\alpha)/(s-i)} < \infty \quad \text{влечет} \quad E\left[W(i)\right]^\alpha < \infty. \tag{14}$$

При дополнительных ограничениях на распределение $B(x)$ эти условия являются также необходимыми [22]. Таким образом, имеет место зависимость условий конечности моментов компонент вектора W от порядка компонент в векторе.

2.1.3. Двухсерверная система: случай тяжелого хвоста

Классический результат (9) имеет интуитивное объяснение. Определяющим параметром для времени ожидания в системе является оставшееся время обслуживания S_I клиента i в момент прихода клиента $i+1$. Известно, что стационарное незавершенное время обслуживания (при условии, что система занята) имеет распределение

$$\overline{B}_I(x) = \frac{1}{ES}\int_x^\infty \overline{B}(y)\,dy, \;\; x \geq 0. \tag{15}$$

При этом для выполнения $ES_I^{k-1} < \infty$ при некотором $k > 1$ требуется $ES^k < \infty$, т. е. моментные свойства S_I хуже, чем у S.

В работе [6] приведены асимптотические результаты для хвоста распределения задержки D в двухсерверной системе G/G/2 при условии, что незавершенное время обслуживания

имеет субэкспоненциальное распределение. Так, в случае «максимальной устойчивости», $\rho := ES/ET < 1$

$$C_1 \leqslant \liminf_{x\to\infty} \frac{P(D>x)}{(\overline{B}_I(x))^2} \leqslant \limsup_{x\to\infty} \frac{P(D>x)}{(\overline{B}_I(x))^2} \leqslant C_2, \tag{16}$$

где константы C_1, C_2 зависят от ET, ES. В частности, в случае правильно меняющейся функции $\overline{B}(x)$ имеет место асимптотика

$$P(D>x) \sim C(\overline{B}_I(x))^2, \quad x\to\infty, \tag{17}$$

где постоянная C зависит от ES, ET [6].

В случае «минимальной устойчивости», $1 < \rho < 2$, для правильно меняющейся $\overline{B}(x)$ имеет место асимптотика

$$P(D>x) \sim C\overline{B}_I(\frac{\rho}{\rho-1}x), \quad x\to\infty, \tag{18}$$

где постоянная C зависит от ES, ET.

2.2. Дисциплины распределения задач по серверам

В работе [9] рассматривается многосерверная система обслуживания, состоящая из s одинаковых вычислителей, каждый из которых имеет собственную очередь (практическим примером такой системы является Грид). Пуассоновский входной поток заявок обслуживается диспетчером, который, зная время обслуживания заявки S, назначает ей вычислитель. Предметом эмпирического исследования в [9] является оптимальная дисциплина работы диспетчера с точки зрения среднего времени ожидания EW и *замедления*, определяемого как EW/ES. При этом время обслуживание заявки моделируется при помощи усеченного распределения Парето

$$dB(x) = \frac{\alpha k^\alpha}{1-(k/p)^\alpha} x^{-\alpha-1}, \quad k \leqslant x \leqslant p < \infty. \tag{19}$$

Время обслуживания с таким распределением может принимать большие, но все же конечные значения. В работе рассмотрены три типа известных дисциплин обслуживания: назначение вычислителя случайным образом; циклическое назначение (Round-Robin); динамическое назначение (заявка отправляется на вычислитель с минимальной незавершенной работой). В работе [9] также предлагается новая дисциплина, SITA-E (Size Interval Task Assignment with Equal Load), которая позволяет сбалансировать нагрузку на все вычислители в среднем. Интервал возможных размеров задач $[k, p]$ делится на участки $k = x_0 < x_1 < \ldots < x_s = p$ так, чтобы в среднем загрузка на этих участках была одинакова:

$$\int_{x_0}^{x_1} x\, dB(x) = \cdots = \int_{x_{s-1}}^{x_s} x\, dB(x) = \frac{ES}{s}. \tag{20}$$

В результате численного моделирования в работе было показано, что наиболее эффективной дисциплиной работы диспетчера в случае конечной дисперсии для тяжелохвостого распределения времени обслуживания ($\alpha \approx 2$) динамическое распределение наиболее эффективно. Тогда замедление в большинстве случаев меньше 1, а среднее время ожидания на 1-2 порядка меньше, чем у других дисциплин. Однако, при $\alpha \approx 1$ динамическое распределение теряет свои преимущества из-за неограниченного роста дисперсии времен обслуживания. В то же время дисциплина SITA-E равномерно хорошо себя ведет с точки зрения обеих метрик (замедления и среднего времени ожидания) на всем интервале $\alpha \in (1, 2)$.

3. Модель вычислительного кластера

Вычислительный кластер — система, состоящая из нескольких узлов, объединенных быстрой коммуникационной сетью, предоставляющая пользователю вычислительные ресурсы. Для построения модели кластера будем рассматривать следующие упрощения:

1. поток заявок представляет собой процесс восстановления (нет пачек, времена между приходами независимы);
2. разделяемым ресурсом является количество процессоров и время вычисления; времена вычисления заявок и требуемое каждой заявке число процессоров независимы;
3. заявки поступают в бесконечную очередь и обслуживаются в порядке прихода (FIFO);
4. заявка поступает на обслуживание только в момент освобождения требуемого ей количества процессоров.

3.1. Теоретические результаты

В дополнение к обозначениям секции 2.1, пусть N_i — требуемое число процессоров для заявки i, на каждом из которых она будет выполняться в течение S_i единиц времени. При этом очевидно, $1 \leqslant N_i \leqslant s$. Тогда можно предложить следующую модификацию рекурсии Кифера-Вольфовица (6) для данной модели:

$$W_{i+1} = R\left(W_i(N_i) + S_i - T_i, \ldots, W_i(N_i) + S_i - T_i, W_i(N_{i+1}) - T_i, \ldots, W_i(s) - T_i\right)^+, \ i \geq 1. \tag{21}$$

Действительно, поскольку i-я заявка должна дождаться освобождения всех требуемых ей процессоров, то ей придется ждать освобождения наиболее загруженного процессора с номером N_i (т.е. обнуления соответствующей компоненты вектора W_i). Поскольку все сервера с меньшими номерами будут простаивать в ожидании наиболее загруженного, не принимая другие заявки, то их загрузку также можно считать равной $W_i(N_i)$. Отметим, что на практике такая неэффективная дисциплина используется редко. Примером более эффективного распоряжения ресурсами является алгоритм Backfill, позволяющий заполнять промежутки простоя отдельных процессоров заявками без существенного нарушения приоритетов и очередности.

Заметим, что при $N_i = s$ все компоненты вектора загрузки можно считать равными, и поэтому заявка с номером $i + 1$ начнет обслуживание в полностью пустой системе (все процессоры освобождаются с уходом заявки i). Такая ситуация при некоторых дополнительных предположениях может порождать *регенерации* в системе, что открывает возможности применения регенеративного моделирования для надежного оценивания стационарных характеристик системы (см., напр., [17]).

Суммарная незавершенная загрузка системы в данном случае в момент прихода заявки i составит

$$N_i \cdot (W_i(N_i) + S_i - T_i)^+ + \sum_{k=N_i+1}^{s} (W_i(k) - T_i)^+. \tag{22}$$

Поэтому условие стационарности должно включать в себя ограничения на распределение задач в системе. Проведенные предварительные эксперименты позволяют предполагать, что в случае, когда значения N_i распределены в интервале $[1, s]$, условие устойчивости имеет вид $\rho = ES/ET < 1$ (максимальная устойчивость), в отличие от классического условия $\rho < s$ для системы G/G/s. В частности, если каждая заявка будет требовать ровно s процессоров, система вырождается в классическую G/G/1, условием устойчивости которой является $\rho < 1$.

Отметим, что задержка в рассматриваемой системе $D_i := W_i(1)$ ограничена снизу значением задержки $D_i^{(low)} := W_i^{(low)}(1)$ в классической системе G/G/s с тем же входным

потоком с интервалами $\{T_i\}$ и временами обслуживания $\{S_i\}$. Действительно, из рекурсии (21) следует, что, в предположении индукции $D_i \geqslant D_i^{(low)}$

$$D_{i+1} = W_{i+1}(1) = (W_i(N_i) + S_i - T_i)^+ \geqslant (W_i(1) + S_i - T_i)^+ =$$
$$= (D_i + S_i - T_i)^+ \geqslant (D_i^{(low)} + S_i - T_i) \geqslant (D_i^{(low)} + P_i - T_i)^+ = D_{i+1}^{(low)}.$$

С другой стороны, величина D_i в рассматриваемой системе ограничена сверху задержкой $D_i^{(up)}$ для системы, в которой $N_i = s, i \geqslant 1$. Действительно,

$$D_{i+1} = W_{i+1}(1) = (W_i(N_i) + S_i - T_i)^+ \leqslant$$
$$\leqslant (W_i(s) + S_i - T_i)^+ = (D_i^{(up)} + S_i - T_i)^+ = D_{i+1}^{(up)}.$$

Последняя (доминирующая) система представляет систему G/G/1, с теми же интервалами между приходами T_i и временами обслуживания S_i, что и в исходной системе. Таким образом, для любых реализаций $\{T_i, S_i\}$ имеет место

$$D_i^{(low)} \leqslant D_i \leqslant D_i^{(up)}. \tag{23}$$

Неравенства (23) для моментов порядка α соответствующих стационарных величин (полученных в пределе при $i \to \infty$) принимают вид

$$E(D^{(low)})^\alpha \leqslant ED^\alpha \leqslant E(D^{(up)})^\alpha. \tag{24}$$

Однако в «нижней» и «верхней» системах обслуживания (при условии $ES/ET < 1$) имеют место одинаковые достаточные условия конечности моментов времени ожидания. Следовательно условие $ES^{\alpha+1} < \infty$ является достаточным для конечности момента ED^α в предложенной модели. Таким образом, доказана

Теорема. В предложенной модели кластера (в предположении $\rho = ES/ET < 1$) условие $ES^{\alpha+1} < \infty$ является достаточным для конечности момента порядка α времени ожидания в системе, т.е. $ED^\alpha < \infty$.

3.2. Численный эксперимент

Моделирование предложенного типа систем обслуживания проводилось на вычислительном кластере ЦКП КарНЦ РАН [2]. В качестве модельного примера рассматривался сам кластер.

За период времени с 01.03.2010 г. по 01.12.2010 г. на кластере было посчитано 3682 задачи, суммарное время вычисления составило 1072804.5 минут. Астрономическое время в минутах между указанными датами составляет 396000 минут. Таким образом, среднее время вычисления задачи составило $ES \approx 291$ минуту, а среднее время между приходами составило $ET \approx 107$ минут. Кластер имеет 80 вычислительных ядер, сгруппированных по 8 штук на каждый вычислительный узел. В процессе вычислений разрешено занимать все узлы, но не менее одного узла. Очевидно, достаточное условие устойчивости такой системы не выполнено, так как $ES/ET > 1$. Действительно, эксперименты показали, что в системе наблюдается неограниченный рост значений задержек. График зависимости величины задержки от номера задачи (для пуассоновского входного потока и экспоненциального времени обслуживания) приведен на рисунке 1 (одна реализация процесса).

Предположим, что заявкам разрешено занимать не более 4-х узлов одновременно. Очевидно, в такой системе задержки будут меньшими, чем в исходной, где разрешено занимать все узлы (см. рис. 2). Кроме того, имеет место сходимость суммы ряда автокорреляций к конечной величине, что говорит об отсутствии долговременной зависимости. График частичных сумм автокорреляций представлен на рисунке 3.

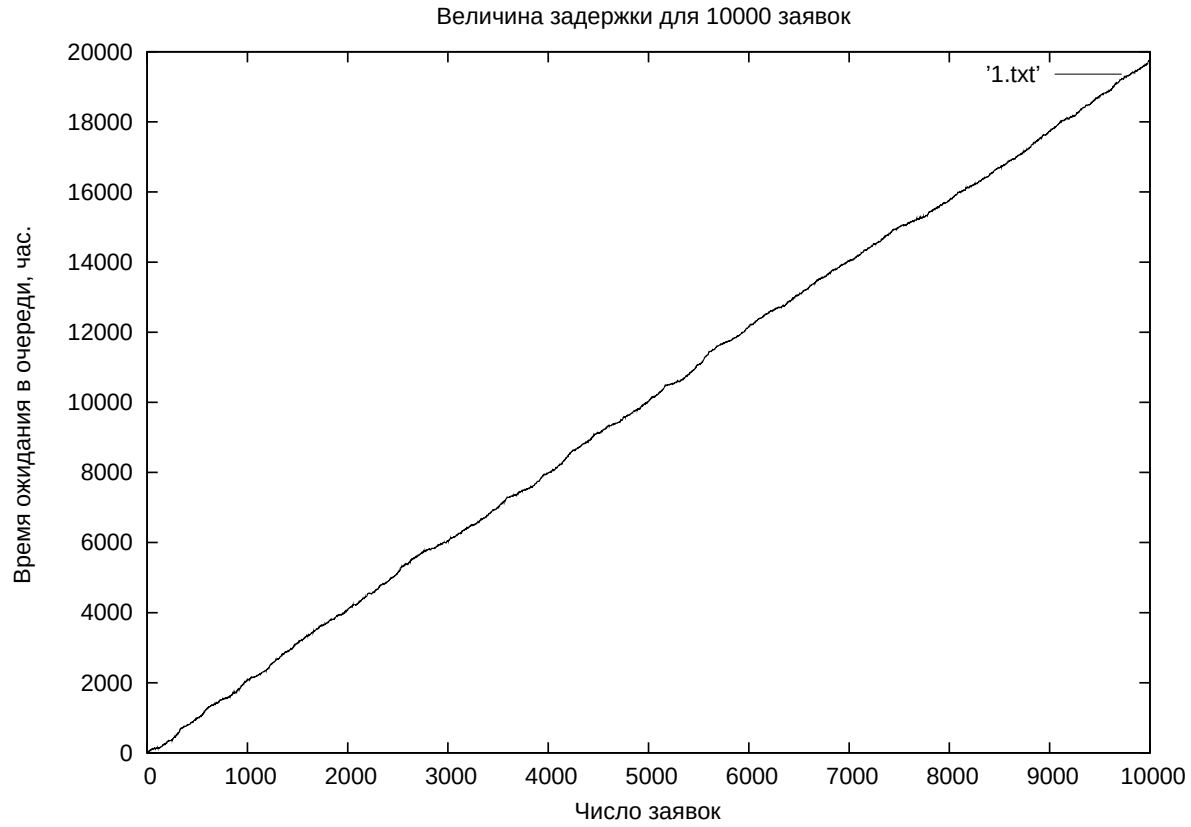


Рис. 1. Задержки в модели 10-узловой системы

Пусть теперь время обслуживания S имеет тяжелый хвост. Промоделируем этот случай при помощи распределения Парето с параметром $\alpha = 2.2$. Входной поток предполагается пуассоновским, с параметром $\lambda = 1$. В этом случае средняя загрузка равна [1] $\lambda/(\alpha-1) < 1$, и поэтому система должна обладать стационарным режимом. В то же время Рис. 4 показывает рост суммы автокорреляций, что говорит о возможном присутствии долговременной зависимости у процесса времени ожидания в очереди.

Однако, времена ожидания не уходят на бесконечность, хотя наблюдается тенденция длительных периодов нахождения времени ожидания выше или ниже своего среднего, что также косвенно указывает на долговременную зависимость (рис. 5).

Необходимо отметить, что модельные результаты отличаются от реального наблюдаемого поведения кластера, прежде всего потому, что на кластере ЦКП КарНЦ РАН предельный размер очереди существенно ограничен. Именно, каждому пользователю разрешается запускать не более 4 задач одновременно. Кроме того, используется алгоритм Backfill для выбора заданий из очереди, а также имеется ряд других ограничений, не отраженных в модели. Поэтому предложенная модель может рассматриваться лишь как предварительный вариант модели, описывающей функционирование кластера.

Для лучшей адаптации модели к реальным кластерам следует рассмотреть модели с конечным буфером для заявок, с возможностью появления пачек заявок, с более сложными дисциплинами выбора заявок на обслуживание (например, алгоритм Backfill и некоторые другие алгоритмы планировщиков). Аналитическое исследование в этих случаях существенно усложнится. В целом, точность предложенной модели и возможные направления ее развития должны быть установлены с помощью более обширных экспериментов.

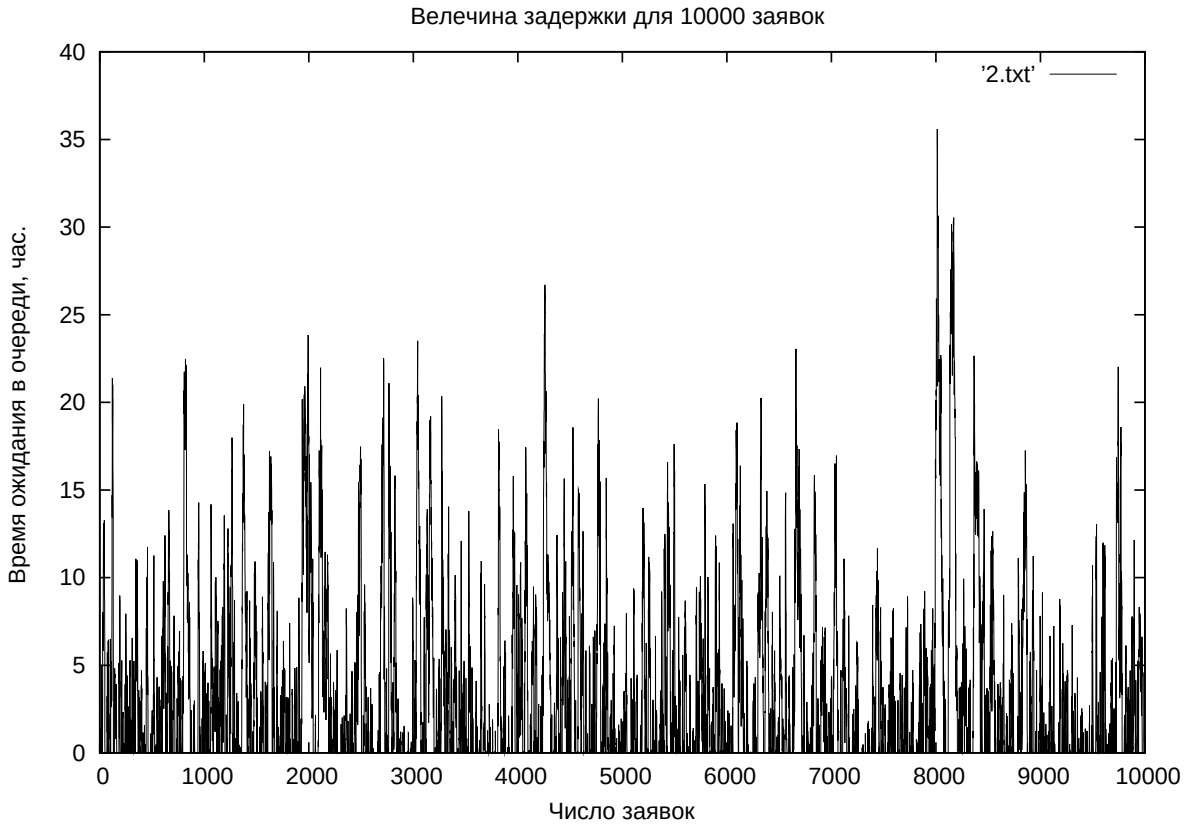


Рис. 2. Задержки в модели 10-узловой системы с правом занимать не более 4 узлов

4. Заключение

В работе рассмотрены некоторые важные результаты, касающиеся моментных свойств процесса обслуживания в многосерверных системах обслуживания, включая случай, когда время обслуживания имеет тяжелый хвост. Эти результаты могут применяться для оценивания качества обслуживания вычислительных кластеров и Грид. Предложена новая модель вычислительного кластера на основе модифицированной рекурсии Кифера-Вольфовица для многосерверной системы $GI/G/s$. Для этой модели получены условия конечности соответствующих моментов стационарного времени ожидания (задержки) в очереди. Приведены результаты вычислительных экспериментов по моделированию системы, в том числе для времени обслуживания с тяжелым хвостом.

Литература

1. Морозов Е.В., Румянцев А.С. Регенерация и корреляционные свойства стационарной задержки в одноканальной очереди. // Распределенные компьютерные и коммуникационные сети (DCCN-10). М., 2010. С. 58–67.

2. Центр высокопроизводительной обработки данных ЦКП КарНЦ РАН. URL: http://cluster.krc.karelia.ru (дата обращения: 13.12.2010 г.).

3. Borst S., Boxma O., Núñez-Queija R. Heavy Tails:The Effect of the Service Discipline. // Proceedings of Performance TOOLS 2002. 2002.

4. Crovella M., Bestavos A. Self-similarity in World Wide Web traffic: evidence and possible causes. // IEEE/ACM Transactions on Networking. 2002. Vol. 5. N. 6. P. 835–845.

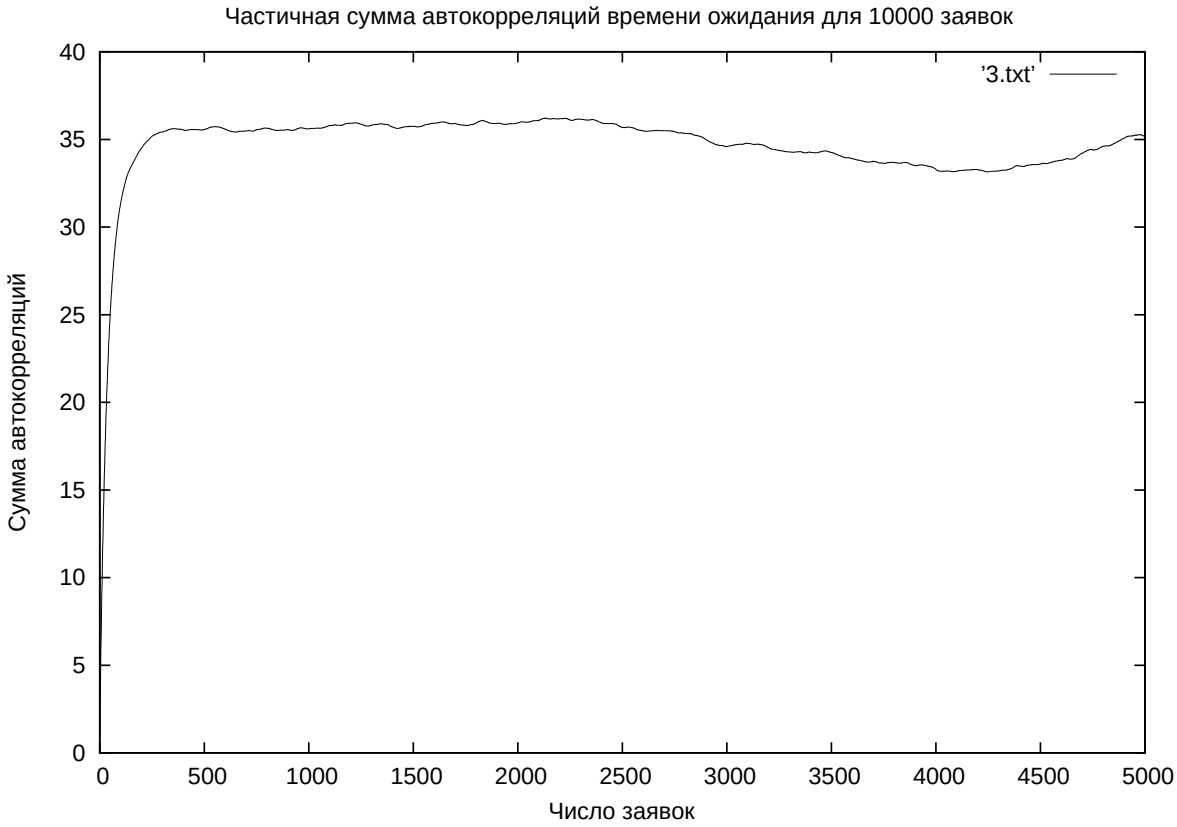


Рис. 3. Автокорреляции в 10-узловой системе с правом занимать не более 4 узлов

5. Daley D. The serial correlation coefficients of waiting times in a stationary single server queue. // Journal of Australian Mathematical Society. 1969. N. 8 P. 683–699.

6. Foss S., Korshunov D. Heavy Tails in Multi-Server Queue. // Queueing Systems. 2006. Vol. 52. P. 31-48.

7. Goldie C., Klüppelberg C. Subexponential Distributions // A practical guide to heavy tails. 1998. P. 435–459.

8. Greiner M., Jobmann M., Klüppelberg C. Telecommunication traffic, queueing models and subexponential distributions. // Queueing Systems. 1999. Vol. 33. P. 125–152.

9. Harchol-Balter M. The Effect of Heavy-Tailed Job Size Distributions on Computer System Design. // Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics, Washington, DC. June 1999.

10. Heath D., Resnick S., Samorodnitsky G. Heavy Tails and Long Range Dependence in ON/OFF Processes and Associated Fluid Models. // Mathematics of Operations Research 1998. Vol. 23. P. 145–165

11. Huang T., Sigman K. Steady-state Asymptotics for Tandem, Split-Match and Other Feedforward Queues with Heavy-Tailed Service. // Queueing Systems. 1999. Vol. 33. P. 233–259.

12. Jessen A., Mikosch T. Regularly Varying Functions. // Publications de L'Institut Mathematique. Nouvelle serie, tome 80 (94) 2006. P. 171–192

13. Keilson J., Seidmann A. M/G/∞ with Batch Arrivals. September 1987.

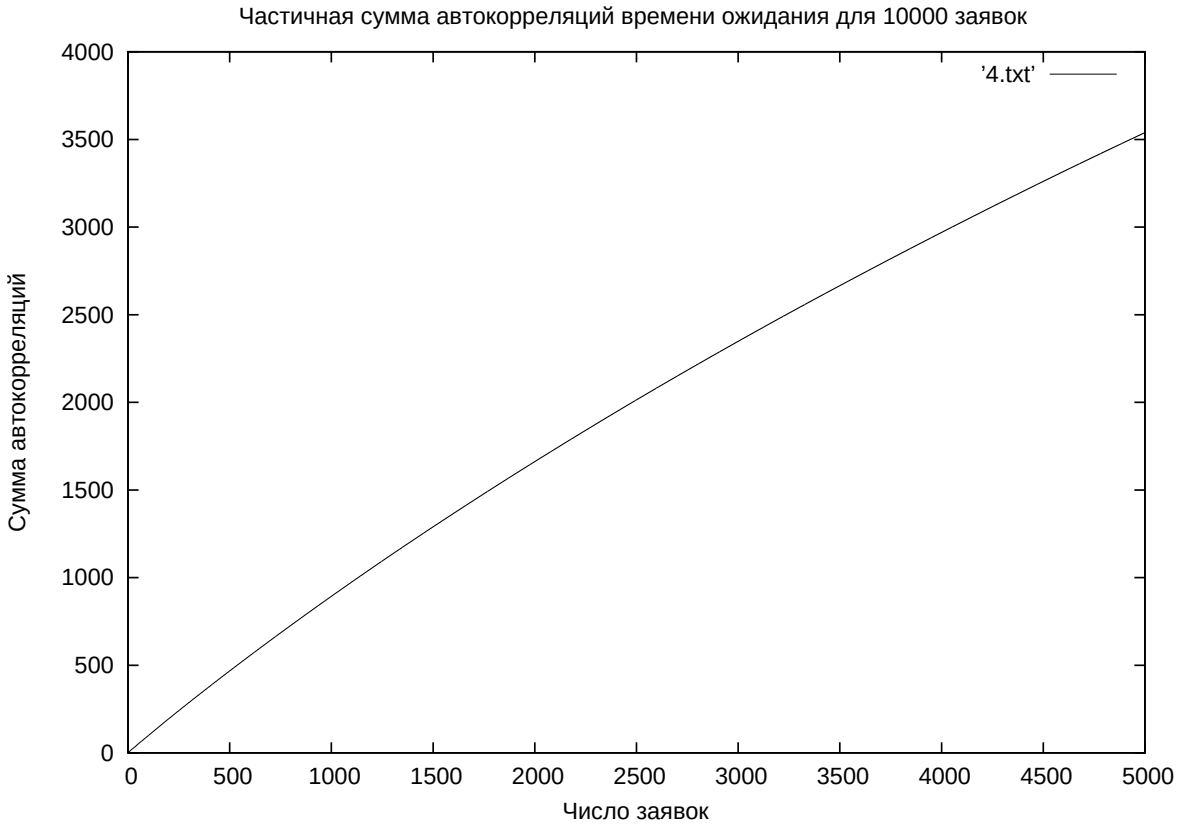


Рис. 4. Автокорреляции в 10-узловой системе, случай тяжелых хвостов

14. Leland W., Taqqu M., Willinger W., Wilson D. On the Self-Similar Nature of Ethernet Traffic // Journal IEEE/ACM Transactions on Networking (TON). February 1994. Vol. 2, N. 1.

15. Liu, L. Batch arrival infinite server queues with constant service times. // Proceedings of the First Symposium on Queueing Theory in China September 1993. P. 47–53

16. Morozov E., Pagano M., Rumyantsev A. Heavy-tailed Distributions with Applications to Broadband Communication Systems. // Proceedings of AMICT'2007. 2008. Vol. 9. P. 157–174.

17. Morozov E., Rumyantsev A. Moment properties of queueing systems and networks // Proceedings of ICUMT'2010 (Ультрасовременные телекоммуникации и системы управления). 2010.

18. Samorodnitsky G. Long Range Dependence. // Foundations and Trends in Stochastic Systems. 2006. Vol. 1. P. 163–257.

19. Scheller-Wolf A. Further delay moment results for FIFO multiserver queues. // Queueing Systems. 2000. Vol. 34 P. 387–400.

20. Scheller-Wolf A., Sigman K. Delay Moments for FIFO GI/GI/s queues. // Queueing Systems. 1997. Vol. 25. P. 77–95.

21. Scheller-Wolf A., Sigman K. New bounds for expected delay in FIFO GI/GI/c queues. // Queueing Systems. 1997. Vol. 26. P. 169–186.

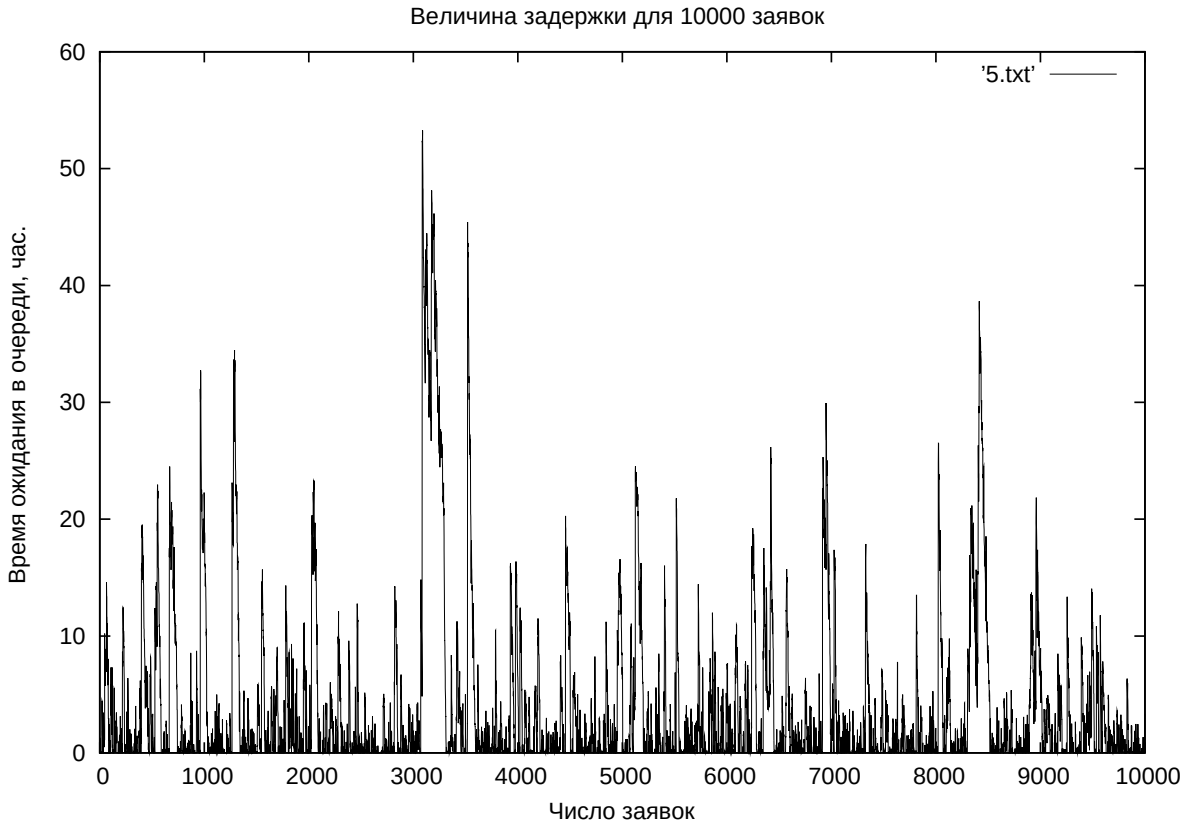


Рис. 5. Задержки в модели 10-узловой системы, случай тяжелых хвостов

22. Scheller-Wolf A., Vesilo R. Sink or Swim Together: Necessary and Sufficient Conditions for Finite Moments of Workload Components in FIFO Multiserver Queues. March 2008.

23. Scheller-Wolf A., Vesilo R. Structural interpretation and derivation of necessary and sufficient conditions for delay moments in FIFO multiserver queues. // Queueing Systems. 2006. Vol. 54. P. 221–232

24. Wolff R. On Finite Delay-Moment Conditions in Queues. // Operations Research. September-October 1991. Vol. 39. N. 5. P. 771–775.

Моделирование процессов в биомолекулярных системах методами квантовой и молекулярной механики (КМ/ММ)*

А.В. Немухин[1,2], Б.Л. Григоренко[1], Д.И. Морозов[1]

Химический факультет МГУ им. М.В. Ломоносова[1], Институт биохимической физики им. Н.М. Эмануэля РАН[2]

Обсуждаются алгоритмы и программные реализации комбинированного метода квантовой и молекулярной механики (КМ/ММ), активно используемого для моделирования процессов в биомолекулярных системах. Основное внимание уделено расчетам строения и оптических спектров перспективных флуоресцирующих белков, а также расчетам энергетических профилей реакций ферментативного катализа. Анализируется опыт расчетов КМ/ММ на суперкомпьютерах СКИФ-Полигон.

1. Введение

Метод моделирования механизмов реакций ферментативного катализа на основе комбинированного подхода квантовой и молекулярной механики (КМ/ММ), предложенного в 1976 г. [1], быстро завоевал популярность. Впоследствие подход КМ/ММ стал применяться для расчетов структуры и свойств других биомолекулярных процессов, включая прогнозирование спектров фоторецепторных и флуоресцирующих белков. Согласно подходу КМ/ММ та часть белковой системы, в которой происходят перераспределения химических связей или рассматриваются электронные возбуждения, включается в квантовую подсистему, и энергии и силы в ней рассчитываются различными приближениями квантовой химии. Большинство атомов, окружающих выделенную центральную часть, относится к молекулярно-механической подсистеме. Таким образом, в методе КМ/ММ учитывают конформационные изменения в белковой матрице, сопровождающие химическую реакцию или переходы между электронными состояниями. В приближениях КМ/ММ энергия каждой точки на поверхности потенциальной энергии (ППЭ) складывается из энергии квантовой части в поле ММ-подсистемы и молекулярно-механической энергии [2,3].

Анализ стационарных точек на ППЭ, т.е. геометрических конфигураций локальных минимумов, отвечающих реагентам, продуктам, возможным интермедиатам, а также седловых точек, через которые происходят переходы между минимумами составляет предмет исследования механизма реакции ферментативного катализа. Особое внимание уделяется расчетам энергетических барьеров на пути от реагентов к продуктам. Переход к экспериментально значимым величинам, таким как константы скорости химических реакций, осуществляется, как правило, в рамках теории переходных состояний.

При расчетах структуры и спектров фоторецепторных белков основную ценность представляет прогнозирование новых линеек биомаркеров в живых системах с улучшенными свойствами (высоким квантовым выходом флуоресценции, повышенной стабильностью, нетоксичностью и проч.) за счет рационального введения точечных мутаций, т.е. замен отдельных аминокислотных остатков в белковой матрице. Перебор огромного числа вариантов чисто экспериментальными средствами потребует слишком много времени и больших материальных затрат. Компьютерное моделирование с использованием современных квантовых методов расчетов структуры и спектров биологических хромофоров в белковых матрицах оказывает существенную поддержку подобным исследованиям. Вычислительные задачи связаны с решением уравнений КМ/ММ для большого фрагмента белковой частицы, содержащей хромофорную молекулу и ближайшие аминокислотные остатки с целью определить координаты всех атомов (общим числом до нескольких тысяч), соответствующие минимумам на поверхностях потенциальной энергии основного и электронно-возбужденных состояний, а также как можно точнее расчитать разности энергий между различными электронными состояниями модельной моле-

* Работа выполнена при поддержке РФФИ (проекты 10-03-00085 и 10-03-00139)

кулярной системы. Даже для одной композиции подобные расчеты крайне ресурсозатратны, но их необходимо выполнить для каждого потенциально перспективного варианта цветного белка [4].

2. Протокол вычислений для моделирования цветных белков

Опишем подробнее стратегию и протокол вычислений методом КМ/ММ на примере прогнозирования оптимального состава флуоресцирующих белков.

(1) Для выбранной композиции белковой макромолекулы найти равновесную геометрическую конфигурацию на поверхности потенциальной энергии основного состояния в рамках метода КМ/ММ с использованием алгоритмов теории функционала электронной плотности в квантовой подсистеме.

(2) Методами квантовой механики на основе многоконфигурационного метода самосогласованного поля (МК ССП) рассчитать вертикальные энергии возбуждений из найденной равновесной точки и оценить интенсивности соответствующих полос в оптическом спектре.

(3) Для выбранного интенсивного электронного перехода сканировать сечения поверхностей потенциальной энергии возбужденного электронного состояния, найти геометрические конфигурации точек минимумов и конических пересечений и соответствующие разности энергий, используя варианты метода МК ССП.

(4) Сформулировать заключение о перспективности данной композиции.

Шаги (1-4) повторяются для каждого варианта белковой макромолекулы с точечными мутациями.

Самыми ресурсозатратными процедурами являются расчеты методом МК ССП. Алгоритм МК ССП включает в себя следующие циклически повторяющиеся шаги:

(1) приведение интегралов из атомного базиса в текущий молекулярный;

(2) генерацию матрицы гамильтониана и оптимизацию коэффициентов многоконфигурационного разложения при частичной давидсоновской диагонализации;

(3) генерацию матрицы плотности первого и второго порядков;

(4) внесение улучшений в молекулярные орбитали, проверка критериев сходимости. При использовании небольших активных пространств молекулярных орбиталей, либо при использовании одного из алгоритмов МК ССП (ORMAS) лимитирующей расчёт стадией является стадия преобразования интегралов (1).

Данные по масштабируемости подобных расчетов на суперкомпьютере СКИФ МГУ «Чебышёв» приведены в Таблице 1.

Таблица 1. Масштабируемость преобразования интегралов при расчетах МК ССП

Количество процессоров	Время, мин
8	32.8
16	25.5
24	12.4
32	7.2
64	3.2

Лучшая масштабируемость достигается при больших размерах квантовой системы в силу большего количества молекулярных орбиталей. Кроме того, для больших систем в пакете квантовохимических программ GAMESS(US), активно используемом для расчетов, существует специальный алгоритм преобразования, использующий преимущества модели распределённой памяти DDI, таким образом улучшая масштабируемость.

Для использования в процессе МКССП в программном пакете GAMESS(US) присутствуют три вида прямого детерминантного варианта конфигурационных разложений: (1) полное конфигурационное взаимодействие (КВ) в активном пространстве (ALDET, Ames Laboratory Determinant Full CI); (2) полное КВ во множественных активных пространствах, с ограничениями по заселенности Occupation Restricted Multiple Active Spaces (ORMAS), используемое обычно для ограничения возбуждения и, тем самым, упрощения задачи полного КВ; (3) метод КВ с выбранным списком детерминантов (GENCI) [5,6].

ALDET: Полное КВ в выбранном активном пространстве. Основным ограничением для данного метода является количество памяти, доступной программе. В реальных условиях на суперкомпьютере СКИФ-«Чебышёв» при использовании 4 узлов (32 процессорных ядра) доступно порядка 128 Гб оперативной памяти, что ограничивает размеры доступных активных пространств до 18 электронов на 16 орбиталях. Данный код является наиболее масштабируемым кодом метода конфигурационного взаимодействия (КВ), поскольку активно использует модель распределённой памяти DDI для хранения векторов КВ [5].

ORMAS: Метод, используемый для ограничения размера задачи полного КВ. Используется, когда задача полного КВ становится слишком большой (>1000000 детерминантов), а также если необходимо ограничить или выделить только нужные переходы электронов между орбиталями. Разбиение большого активного пространства на несколько частей, с заданием возможных границ заполнения этих подпространств электронами, приводит к значительному уменьшению задачи полного КВ. Тем не менее, остаётся возможность сохранить многие эффекты полного КВ, поскольку в каждом из маленьких подпространств решается задача полного КВ. Являясь очень гибким вариантом метода КВ, ORMAS позволяет использовать достаточно большие пространства (например 20 на 16 орбиталях и более) при относительно небольших затратах памяти и практически без затрат дискового пространства (полностью прямой метод), при его использовании, обычно, лимитирующей стадией расчёта становится первый шаг процедуры МКССП – преобразование интегралов. ORMAS, благодаря использованию распределённой памяти, также как и ALDET обладает хорошей масштабируемостью [6].

При применении метода ORMAS для моделирования реакций с переносом электрона в зеленом флуоресцентном белке модельная система состояла из 81 атома (соответственно включала 805 молекулярных орбиталей). Полное пространство, необходимое для моделирования реакции переноса электрона с аниона глутаминовой кислоты Glu222 на хромофор зеленого флуоресцирующего белка составляет 18 электронов на 17 орбиталях, при этом количество детерминантов составляет порядка 25 миллионов. Задача становится технически невозможной на текущих системах при использовании метода ALDET. Поэтому использовался метод ORMAS со следующими параметрами: большое пространство разбивалось на 3 подпространства (занятые орбитали аниона глутаминовой кислоты, занятые орбитали хромофора, вакантные орбитали), переходы между подпространствами ограничивались до 1 электрона с занятых орбиталей глутамата на вакантные и полной заморозкой занятых орбиталей хромофора. Ограничение вводилось для того чтобы самое низшее возбуждённое соответствовало переходу с переносом заряда с аниона на хромофор.

3. Расчеты реакций ферментативного катализа

В настоящее время метод КМ/ММ на основе конформационно-подвижных эффективных фрагментов может быть применен для расчетов энергетических профилей реакционного пути (сечений поверхностей потенциальной энергий) основного электронного состояния с использованием наиболее востребованных квантово-химических подходов – приближений теории функционала электронной плотности, многоконфигурационных приближений самосогласованного поля, теории возмущений Мёллера-Плессе, в КМ-подсистеме. Число атомов, относимых к КМ-части, может превышать 100; более важным параметром является число базисных функций для представления молекулярных орбиталей – не более 1500. ММ-подсистема может содержать до 3000 атомов, сгруппированных не более чем в 700 эффективных фрагментов. Для расчетов вкладов в энергию и градиенты энергии по ядерным координатам в ММ-части могут быть использованы параметры популярных силовых полей AMBER, CHARMM, OPLSAA и других. Приведенные ограничения на параметры вычислительной схемы не являются принципиальными и могут быть изменены в следующих версиях программы [7].

Нами рассматривалась молекулярная система, моделирующая фермент-субстратный комплекса пенициллин-ацилазы (ПА) с пенициллином G. Исходная информация о координатах тяжелых атомов была получена из базы данных белковых структур (PDB) для кристаллической структуры фермента ПА с ингибирующим субстратом SOX (пенициллин G сульфоксид) – код PDB:1GM9. Далее методами компьютерного моделирования ингибирующий субстрат был заменен на пенициллин G, добавлены атомы водорода и проведены расчеты равновесных геомет-

рических конфигураций фермент-субстратного комплекса методом КМ/ММ с полной оптимизацией координат в субстрат-связывающей области. ММ-подсистема включала 2400 атомов, сгруппированных в 670 эффективных фрагментов. Расчеты энергий и сил в КМ-части проводились в приближении B3LYP/6-31G*, в ММ-части – с использованием параметров силового поля AMBER. Тщательное сопоставление равновесных геометрических параметров (расстояний между тяжелыми атомами, валентных углов), рассчитанных методом КМ/ММ, и измеренных в соотвествующей кристаллической структуре показывает хорошее соответветствие теоретических и экспериментальных величин.

Как для всех сериновых гидролаз, первая стадия реакции гидролиза пенициллина ферментом ПА (стадия ацилирования) проходит через образование первого тетраэдрического интермедиата. Особенности реакционного механиза для данного фермента и данного субстрата с приведением структур интермедиатов будут описаны в отдельной работе. Здесь мы обращаем внимание на сопоставление результатов двух методик КМ/ММ: приближении внедренного кластера и приближении конформационно-подвижных эффективных фрагментов. Следует отметить, рассчитанные в приближении B3LYP/6-31G*/AMBER геометрические параметры и для фермент-субстратного комплекса, и для тетраэдрического интермедиата в обоих подходах достаточно близки. Однако, разность энергий, вычисленная в подходе внедренного кластера, +5 ккал/моль, явно не выглядит правдоподобной. Пересчет разности энергий в варианте КМ/ММ с конформационно-подвижными фрагментами (-17 ккал/моль) позволяет получить ожидаемуй выигрыш энергии на первом сегменте реакции.

4. Резюме

Таким образом, известны алгоритмы, позволяющие проводить масштабные расчеты комбинированным методом КМ/ММ как для реакций ферментативного катализа, так и для моделирования структуры и спектров фоторецепторных белков.

Литература

1. A. Warshel, M. Levitt. Theoretical studies of enzymatic reactions: dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. // *J. Mol. Biol*. 1976. V. 103. P. 227-249.
2. F. Bernardi, M. Olivucci, M. A. Robb. Simulation of MC-SCF results on covalent organic multibond reactions: molecular mechanics with valence-bond (MM-VB). // J. *Am. Chem. Soc.* 1992. V. 114. P. 1606-1616.
3. J. Aqvist, A. Warshel. Simulation of enzyme reactions using valence bond force fields and other hybrid quantum-classical approaches. // *Chem. Rev*. 1993. V. 93. P. 2523-2544.
4. M. Matz, K. Lukyanov and S. Lukyanov. Family of the green fluorescent protein: journey to the end of the rainbow. Bioessays, 2002, V. 24(10). P. 953-962
5. J.Ivanic, K.Ruedenberg. Full CI (ALDET) and general CI (GENCI) //Theoret.Chem.Acc., 2001, V. 106, P. 339-351
6. J.Ivanic. Occupation restricted multiple active space (ORMAS). // J.Chem.Phys., 2003, V. 119, P. 9364-9385
7. A. Nemukhin, B. Grigorenko, A. Bochenkova. A QM/MM approach with effective fragment potentials applied to the dipeptide-water structures. // J.Molec.Struct.(Theochem), 2002, V. 581, p.167

Массивно-многопоточная реализация двумерных БИХ фильтров

А.В. Никоноров[1], В.А. Фурсов[1], П.Ю. Якимов[2]

Институ систем обработки изображений РАН[1], Самарский Государственный Аэрокосмический Университет имени академика С.П. Королева[2]

В работе рассматривается информационная технология реализации двумерного фильтра с бесконечной импульсной характеристикой (БИХ-фильтра). Для обеспечения физической реализуемости БИХ-фильтр строится в виде параллельного соединения 4-х фильтров одного квадранта. Параметры каждого фильтра одного квадранта определяются путем идентификации по тестовым фрагментам, формируемым из исходного искаженного изображения. После определения параметров фильтра осуществляется анализ его устойчивости. Для обработки крупноформатного изображения с использованием построенных указанным способом четырех фильтров одного квадранта предлагается схема эффективной массивно-многопоточной реализации БИХ фильтра в системах на основе графических процессоров (GPU).

1. Введение

В работе [1] рассматривалась информационная технология определения характеристик и обработки изображений с использованием двумерных фильтров с бесконечной импульсной характеристикой (БИХ-фильтров). В частности, для решения задачи идентификации параметров фильтра в указанной работе предложено использовать малые тестовые фрагменты, которые формируются из искаженного изображения с использованием априорной информации о геометрической форме регистрируемых объектов. При этом двумерный БИХ-фильтр строится в виде параллельного соединения физически реализуемых фильтров с опорной областью в виде квадранта, обеспечивающих компенсацию сильных искажений с использованием опорной области небольших размеров.

Вопросы повышения производительности обработки одномерными БИХ фильтрами за счет использования параллельных вычислительных ресурсов достаточно хорошо исследованы. В работе [2] рассматривалась реализация одномерных БИХ фильтров в многопоточных вычислительных средах. В работе [3] приведены результаты GPU-реализации БИХ фильтра. В ряде случаев возможна декомпозиция обработки изображений, в результате которой задачи сводится к последовательному применению одномерных фильтров [4]. Однако, в общем случае, такую декомпозицию провести нельзя.

В настоящей работе рассматривается достаточно универсальная массивно-многопоточная реализация двумерных БИХ фильтров, с использованием графических процессоров и технологии CUDA, обеспечивающая значительное увеличение производительности по сравнению не только с CPU-процессорами, но и с обычными CUDA-процедурами, использующими неоптимизированные коды.

2. Технология формирования и идентификации двумерного БИХ-фильтра

В общем случае выражение, определяющее способ вычисления выходного отсчета $y(n_1,n_2)$ двумерного БИХ-фильтра, имеет вид [5]:

$$y(n_1,n_2)=\sum_{r_1}\sum_{r_2}a(n_1-r_1,n_2-r_2)x(r_1,r_2)-\sum_{\substack{k_1\\(k_1k_2\neq\ n_1n_2)}}\sum_{k_2}b(n_1-k_1,n_2-k_2)y(k_1,k_2). \qquad (1)$$

Нетрудно заметить, что в данном случае требование рекурсивной вычислимости не выполняется (вычисляемый выходной отсчет на рисунке 1, а обозначен кружком в центре опорной области). Для физической реализуемости двумерного БИХ-фильтра он представляется в виде параллельного соединения БИХ-фильтров одного квадранта [6] (рисунок 1, б-д).

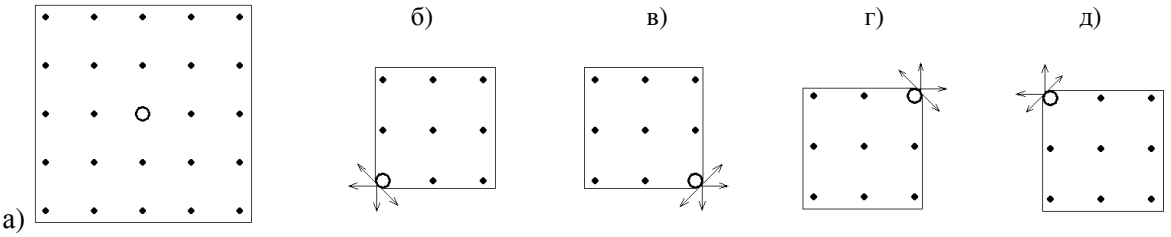


Рис. 1. Исходная маска 5×5 – а) и соответствующие ей маски и допустимые направления рекурсии для: 1-го квадранта – б); 2-го квадранта – в); 3-го квадранта – г); 4-го квадранта – д).

Для заданной опорной маски в виде квадранта и некоторой пары тестовых фрагментов, один из которых сформирован путем компьютерного ретуширования, записывается систему линейных уравнений [1]:

$$\mathbf{Y} = \mathbf{S}\varphi + \mathbf{o}, \qquad (2)$$

где, в соответствии с (1)

$$\mathbf{Y} = \begin{bmatrix} y_1(n_1,n_2) \\ y_2(n_1,n_2) \\ \vdots \\ y_N(n_1,n_2) \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} x_1(r_1,r_2) & \cdots & y_1(k_1,k_2) & \cdots \\ x_2(r_1,r_2) & \cdots & y_2(k_1,k_2) & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ x_N(r_1,r_2) & \cdots & y_N(k_1,k_2) & \cdots \end{bmatrix}, \mathbf{o} = \begin{bmatrix} \xi_1(n_1,n_2) \\ \xi_2(n_1,n_2) \\ \vdots \\ \xi_N(n_1,n_2) \end{bmatrix},$$

N – число отсчетов на тестовых фрагментах, по которым сформирована система (2), $\varphi = \left[a(n_1 - r_1, n_2 - r_2), \ \cdots \ b(n_1 - k_1, n_2 - k_2), \ \cdots \right]^T$ - искомый вектор параметров фильтра.

Для обеспечения возможности идентификации фильтра высокого порядка в работе [1] матрицу **S** в (5) предложено составлять из блочных матриц, каждая из которых формируется по простым фрагментам, на каждом из которых функция яркости изменяется в одном направлении, но эти направления для разных фрагментов различны. Примеры таких фрагментов приведены в работе [1].

Для вычисления параметров фильтра по данным системы (2) может использоваться простейшая оценка метода наименьших квадратов (МНК):

$$\hat{\varphi} = \left[\mathbf{S}^T\mathbf{S}\right]^{-1}\mathbf{S}^T\mathbf{Y}, \qquad (3)$$

либо более устойчивая к грубым ошибкам типа сбоев оценка метода наименьших модулей (МНМ-оценка).

3. Анализ устойчивости фильтра

Анализ устойчивости фильтра необходим вследствие того, что получающиеся в результате идентификации фильтры одного квадранта могут оказаться неустойчивыми, что может привести к искажениям обработанного изображения [6]. Поэтому в качестве промежуточного этапа технологии предлагается осуществлять анализ устойчивости полученного фильтра.

Передаточная функция БИХ-фильтра, который мы идентифицируем, выглядит следующим образом:

$$H_z(z_1, z_2) = \frac{\sum_{r_1=0}^{N_1}\sum_{r_2=0}^{N_2} a(r_1,r_2)\, z_1^{-r_1} z_2^{-r_2}}{\sum_{k_1=0}^{N_1}\sum_{k_2=0}^{N_2} b(k_1,k_2)\, z_1^{-k_1} z_2^{-k_2}}, \qquad (4)$$

где N_1, N_2 – это размеры входной маски (будем считать, что выходная маска имеет такие же размеры, что и входная).

Нам необходимо исследовать поведение функции, находящейся в знаменателе. Для удобства обозначим:

$$B(z_1, z_2) = \sum_{k_1=0}^{N_1} \sum_{k_2=0}^{N_2} b(k_1, k_2) z_1^{-k_1} z_2^{-k_2} .$$

В качестве основы для проведения анализа устойчивости целесообразно опираться на теорему Шэнкса [7]. В соответствии с этой теоремой для каждой точки $b = |z_2|$ единичной окружности $|z_2| = 1$ на комплексной плоскости z_2 для всех $|z_1| \geq 1$ должно выполняться условие:

$$B(z_1, b) \neq 0,$$

а для каждой точки $a = |z_1|$ единичной окружности $|z_1| = 1$ на комплексной плоскости z_1 для всех $|z_2| \geq 1$ должно выполняться условие:

$$B(a, z_2) \neq 0.$$

Для проверки выполнения первого из указанных условий строится годограф корней характеристического уравнения

$$B(z_1, b) = 0$$

на плоскости z_1 при условии, что параметр $b = |z_2|$ «пробегает» все точки на единичной окружности в плоскости z_2. Если этот годограф пересекает единичную окружность на плоскости z_1, это свидетельствует о неустойчивости фильтра. Устойчиваость возможна только в случае, если пересечения годографов с единичной окружностью отсутствуют, и все они оказываются внутри круга единичного радиуса плоскости z_1.

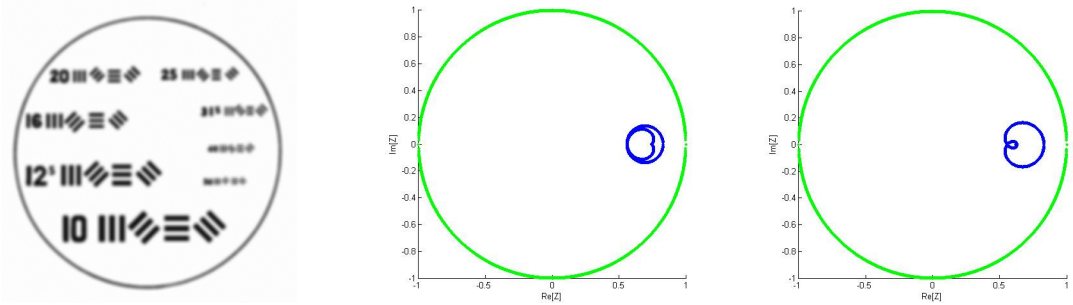
Аналогично годограф корней характеристического уравнения

$$B(a, z_2) = 0$$

на плоскости z_2 образованный при условии, что параметр $a = |z_1|$ «пробегает» все точки на единичной окружности в плоскости z_1, должен находиться внутри круга единичного радиуса на плоскости z_2.

При проведении анализа устойчивости следует иметь в виду, что числа *a* и *b* в общем случае являются комплексными, поэтому при вычислении корней следует приравнивать нулю как действительную, так и мнимую часть полинома в левой части характеристического уравнения.

На рисунках 2,б и 2,в показано поведение годографа устойчивого фильтра первого квадранта, идентифицированного с использованием миры, показанной на рисунке 2,а.



а) б) в)

Рис. 2. а) Мира, использованная для идентификации параметров БИХ-фильтра
б) Годограф корней характеристического уравнения $B(z_1, b) = 0$
в) Годограф корней характеристического уравнения $B(a, z_2) = 0$

Из рисунка 2 можно заключить, что исследуемый БИХ-фильтр, полученный идентификацией из тестовых фрагментов, является устойчивым. На рисунке хорошо видно, как годографы корней уравнения $B(z_1, z_2) = 0$ хорошо вписываются в единичную окружность.

4. Массивно-многопоточная реализация БИХ-фильтра в GPU системе

Поквадрантная реализация БИХ фильтра [1] позволяет естественным образом провести распределение вычислений на четыре потока. В настоящее работе предлагается массивно-многопоточных алгоритм реализации БИХ фильтра. Такая реализация в первую очередь ориентирована на использование в GPGPU системах, однако также может быть использована в современных многоядерных CPU системах.

В работе [9] была предложена процедура эффективной реализации рекуррентной обработки изображения локальным окном. С небольшими изменениями эта процедура может быть использована для решения задачи двумерной КИХ фильтрации.

В настоящей работе предлагается эффективная процедура реализации квадрантного БИХ фильтра. БИХ фильтр можно представить в виде последовательного соединения двух фильтров-компонент. Первый КИХ компонент зависит только от входного сигнала и соответствует числителю формулы (4). Второй, зависит только от выхода и соответствует знаменателю в формуле (4). Будем называть его рекурсивным компонентом фильтра.

Рассмотрим реализацию рекурсивного компонента. Будем рассматривать один квадрант, (x>0, y>0), для остальных трех квадрантов процедура выполняется аналогично. Передаточная функция для такого компонента имеет вид:

$$\frac{1}{\sum_{i=0,j=0}^{m-1} b_{i,j} z_1^{-i} z_1^{-j}}, \quad b_{0,0} = 1 \cdot$$

Схема вычислений описывается далее в терминах потоков (thread), блоков (block) и сетки (grid). Эти термины стандартны для архитектур CUDA и OpenCL [8, 10]. Предлагаемая схема содержит два уровня параллелизма – на уровне блоков и на уровне потоков.

Каждый блок выполняет расчет для одного столбца данных. Для единообразия, полагаем, что номер блока равен номеру столбца данных, т.е. блок с индексом *g* должен *N* раз выполнить следующее суммирование:

$$y_{k,g} = \sum_{i=0,j=0}^{m-1} b_{i,j} y_{k-i,g-j}, \quad k = 0,1,...N. \tag{5}$$

Так как вычисления выполняются рекурсивно, блоку для вычисления значения в некоторой точке (k,t) необходимо чтобы блоки с номерами меньше j выполнили вычисления для не менее чем i строк. Таким образом, если все блоки начинают вычисления одновременно, блок с индексом *b* должен дождаться завершения вычислений предыдущими блоками.

Потоки каждого блока выполняют суммирование (5) параллельно, по схеме каскадного суммирования или редукции[8, 10]. В таком суммировании должно быть задействовано $2^{2]\log_2 m[}$ потоков и тогда количество операций выполняемых потоками параллельно составит

$$O_{cs} = 2]\log_2 m[, \tag{6}$$

где под]m[понимается наименьшее целое, большее m.

Тогда количество вычислений для каждого блока составит:

$$O_{bs} = 2N]\log_2 m[. \tag{7}$$

Так как количество (6) для всех блоков одинаковое и блок с индексом *g* должен ожидать завершения вычисления для (*g*-1) блоков, то время ожидания для блока составит примерно

$$O_{bw} = 2(g-1)]\log_2 m[. \tag{8}$$

Если предположить, что N блоков начинают вычисления одновременно, то для последнего блока, с учетом (7) и (8), количество операций составит

$$O = (4N - 2)]\log_2 m[\,. \qquad (9)$$

Количество операций, требуемое для последовательной реализации алгоритма, составляет $O_S = N^2 m^2$. Таким образом, теоретическая оценка ускорения параллельной реализации позволяет предположить значительное ускорение. Однако модель многопоточности современных GPU накладывает некоторые дополнительные условия на реализацию предлагаемого алгоритма.

В общем случае, если количество блоков равно G, то алгоритм выполняется полосами]N/G[раз, а суммарная сложность должна составить:

$$O =]N / G[(4N - 2)]\log_2 m[\,.$$

Зависимость (7) имеет квадратичный характер по N, что должно сказываться на производительности при G << N.

Возможность увеличения количества блоков G зависит от возможностей GPU. В настоящей работе проведено исследование производительности при увеличении G для видеоадаптера GF 9600. Время обработки для одной полосы и для всего изображения для N = 512, m = 5 и числе потоков 16 для различных значений G приведено на рисунке 3.

Рис. 3. Зависимость времени обработки от количества блоков.

Увеличение количества блоков приводит к увеличению накладных расходов планировщика потоков. Однако планировщик GPU исключает тупиковые ситуации, и увеличение числа блоков позволяет увеличить производительность.

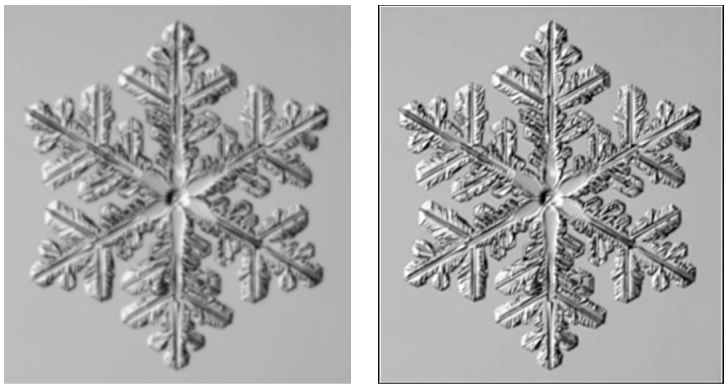
5. Примеры обработки

Экспериментальные исследования качества предложенной технологии проводились в следующей последовательности.

- Автоматизированный поиск тестовых фрагментов.
- Ретушь найденных тестовых фрагментов.
- Идентификация параметров БИХ-фильтров по тестовым фрагментам.
- Проверка на устойчивость.
- Обработка изображений с помощью полученных квадрантных БИХ-фильтров.

В качестве тестового изображения использованы снимки кристаллов льда. В результате все идентифицированные фильтры получились устойчивыми, что позволило получить значительное улучшение обработанных изображений.

На рисунке 4 приведен пример обработки изображений с использованием описанной технологии. а) –«размытое» изображение, б) – обработанное.



а) размытое изображение б) обработанное изображение

Рис. 4. Результат обработки изображения кристалла льда.

6. Заключение

Рассмотренная технология обработки изображений с использованием БИХ-фильтров является существеным развитием технологии описанной в работе [1]. Включение в качестве одного из этапов технологии проверки устойчивости полученного в результате идентификации БИХ-фильтра обеспечивает повышение качества и существенное повышение надежности обработки крупноформатных изображений.

Наиболее важным результатом работы является рекуррентная реализация БИХ-фильтра на GPU-процессорах. В данном случае путем оргинальной организации процедур обработки удалось преодолеть традиционно существовавшее мнение, что эффективная реализация рекурсивных процедур на GPU-процессорах невозможна. Также интересным для дальнейших исследований является анализ эффективности параллельной реализации БИХ-фильтров в RDMA системах.

Литература

1. Милюткин М.Г., Никоноров А.В., Фурсов В.А., Параллельная реализация двумерных БИХ-фильтров в распределенной системе обработки изображений, Труды международной конференции ПаВТ 2010, 2010 г., 268-275.
2. W. Sung and S. K. Mitra, Effiicient Multi-Proccessor Implementtation of Recursive Digital Filters, ICASSP, 1986.
3. Mccool M.D., Signal Processing and General-Purpose Computing and GPUs, Signal Processing Magazine, IEEE, 2008, pp. 109-114.
4. Мурызин С.А., Сергеев В.В., Фролова Л.Г. Исследование эффективности двумерных параллельно-рекурсивных КИХ-фильтров // Компьютерная оптика. - М.: МЦНТИ, 1992. - Вып.12. - С.65-71.
5. Методы компьютерной обработки изображений / Под ред. Сойфера В.А., Москва, Физматлит, 2001.
6. Зимин Д.И., Фурсов В.А. Построение устойчивых алгоритмов обработки изображений путем аппроксимации фильтров с бесконечной импульсной характеристикой // сб. Компьютерная оптика, № 28, 2005, с. 124-127.
7. D. E. Dudgeon, R. M. Mersereau, Multidimensional digital signal processing, Prentice-Hall, Inc., Englewood Cliffs, 1984.
8. А. В. Боресков, А. А. Харламов, Основы работы с технологией CUDA, ДМК Пресс, 2010, 230 с.
9. S. Bibikov, V. Fursov, A. Nikonorov, P. Yakimov, Memory Optimization for Recurrent CUDA Image Processing, PRIA 2010 Proceedings, 2010, pp. 176-179.
10. NVIDIA CUDA C Best Practices Guide, Santa Clara, 2010, 75p.

Архитектура и принципы реализации параллельной СУБД PargreSQL*

К.С. Пан, М.Л. Цымблер

Южно-Уральский государственный университет

В работе описана архитектура и принципы реализации параллельной СУБД PargreSQL для кластерных вычислительных систем, разрабатываемой на основе свободно распространяемой СУБД PostgreSQL.

1. Введение

СУБД PostgreSQL [1] представляет собой свободно распространяемую реляционную СУБД с открытым исходным кодом. Научный проект Омега [2] направлен на разработку прототипа параллельной СУБД для мультипроцессорных вычислительных систем с кластерной архитектурой.

Параллельная СУБД PargreSQL разрабатывается в рамках проекта Омега. Базовой идеей этой разработки является внедрение поддержки фрагментного параллелизма [3] в СУБД PostgreSQL. В данной работе описана архитектура и основные принципы разработки СУБД PargreSQL.

Работа имеет следующую структуру. В разделе 2 приводится краткий обзор работ по тематике исследования. Раздел 3 содержит описание архитектуры СУБД PostgreSQL. В разделах 4 и 5 описана архитектура и принципы реализации СУБД PargreSQL. Раздел 6 содержит заключение.

2. Работы по тематике исследования

В настоящее время СУБД PostgreSQL представляет собой надежную свободно распространяемую на уровне исходных кодов альтернативу коммерческим СУБД. Существует достаточно большое количество практических приложений баз данных на основе PostgreSQL и исследовательских проектов, посвящённых расширению и улучшению PostgreSQL.

В работе [4] рассматривается внедрение поддержки XML в PostgreSQL. Добавление новых типов данных для обеспечения поддержки стандарта обмена медицинской информацей HL7 в PostgreSQL описывается в [5]. Авторы работы [6] предлагают расширение PostgreSQL для обработки изображений. В работе [7] представлен подход к интеграции PostgreSQL с Semantic Web.

Исследования, посвященные использованию PostgreSQL для параллельной обработки запросов могут быть представлены следующими работами. В [8] предлагается расширение PostgreSQL для распределенной обработки запросов. Описаны необходимые изменения в исполнителе запросов PostgreSQL и предложены соответствующие способы увеличения производительности.

СУБД ParGRES [9] представляет собой промежуточное программное обеспечение (middleware) с открытым исходным кодом для высокопроизводительной обработи OLAP-запросов. ParGRES использует внутризапросный параллелизм на кластерных вычислительных системах и адаптивное виртуальное распределение базы данных. СУБД GParGRES [10] является продолжением продукта ParGRES для грид-сред. GParGRES использует репликацию базы данных и меж- и внутризапросный параллелизм для эффективной обработки OLAP-запросов в грид. Предложенный подход подразумевает разбиение данных на двух

*Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-07-00241-а).

уровнях: на уровне грид (реализовано в GParGRES) и на уровне узлов (реализовано в ParGRES).

Нами предлагается внедрение фрагментного параллелизма [11] в СУБД PostgreSQL на основе методов параллельной обработки запросов, разработанных в рамках проекта Омега [12, 13].

3. Архитектура PostgreSQL

В основе архитектуры PostgreSQL лежит модель «клиент-сервер». В сеансе работы с PostgreSQL участвуют три вида взаимодействующих процессов (см. Рис. 1): *приложение-клиент (frontend)*, *серверный процесс (backend)* и *демон (daemon)*. Демон осуществляет прием соединений, устанавливаемых клиентами, и создает отдельный серверный процесс для обработки запросов каждого отдельного клиента.

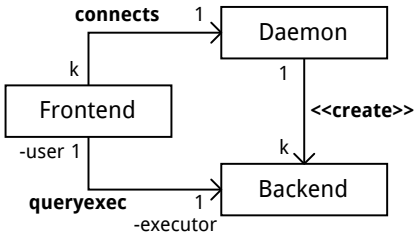


Рис. 1. Процессы СУБД PostgreSQL

Порядок взаимодействия клиента и СУБД представлен на Рис. 2. Сначала клиент устанавливает соединение с демоном. Демон принимает соединение от клиента и затем с помощью системного вызова `fork()` создает серверный процесс. После этого клиент отправляет запрос серверному процессу, который выполняет обработку этого запроса и отправку результатов обратно клиенту.

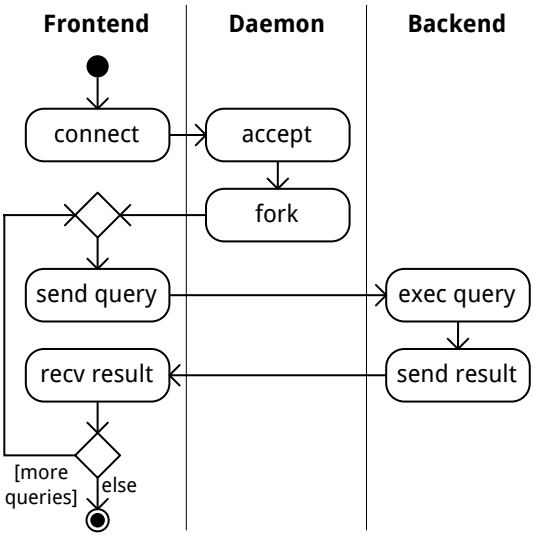


Рис. 2. Взаимодействие клиента и сервера PostgreSQL

Обработка запроса состоит из следующих этапов (см. Рис. 3):

- *parse* — разбор запроса на языке SQL;
- *rewrite* — преобразование запроса;
- *plan/optimize* — составление плана запроса и его оптимизация;

- *execute* — выполнение плана запроса.

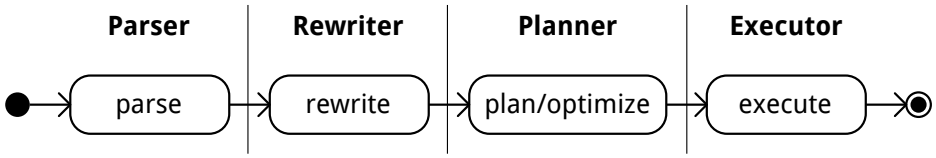


Рис. 3. Обработка запроса в СУБД PostgreSQL

СУБД PostgreSQL содержит следующие подсистемы, представленные на Рис. 4:

- *Parser* — подсистема, которая осуществляет разбор SQL-запросов;
- *Rewriter* — подсистема, выполняющая преобразование запроса в соответствии с правилами подстановки, которые хранятся в базе данных (например, для реализации представлений);
- *Storage* — подсистема хранения данных и метаданных;
- *Planner* — подсистема, которая выполняет составление плана запроса;
- *Executor* — подсистема, исполняющая план запроса;
- *libpq* — библиотека, реализующая протокол взаимодействия клиента (libpq-fe) и сервера (libpq-be).

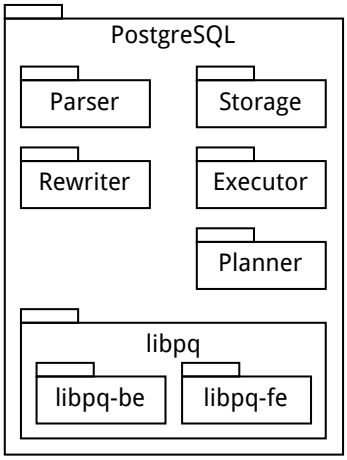


Рис. 4. Подсистемы СУБД PostgreSQL

Размещение компонентов СУБД PostgreSQL приведено на Рис. 5.

На клиенте размещается библиотека `libpq` и приложение пользователя. Все остальные компоненты размещаются на узлах сервера.

4. Архитектура PargreSQL

PargreSQL использует идею фрагментного параллелизма [3]. Общая схема параллельной обработки запроса представлена на Рис. 6. Каждое отношение (таблица) базы данных делится на горизонтальные *фрагменты*, распределяемые по процессорным узлам вычислительной системы. Способ фрагментации определяется *функцией фрагментации*, вычисляющей для каждого кортежа отношения номер процессорного узла, на котором должен быть

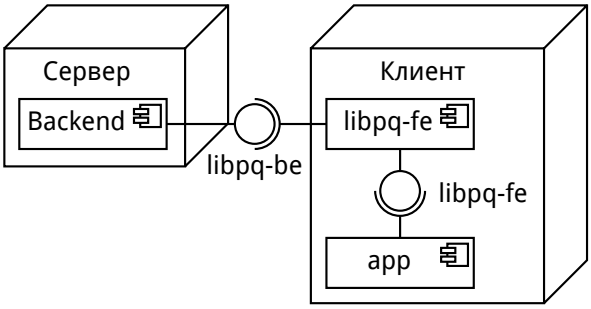


Рис. 5. Размещение компонентов PostgreSQL

размещен этот кортеж. Запрос выполняется в виде нескольких параллельных процессов (*агентов*), каждый из которых обрабатывает отдельный фрагмент отношения. Полученные фрагменты сливаются в результирующее отношение.

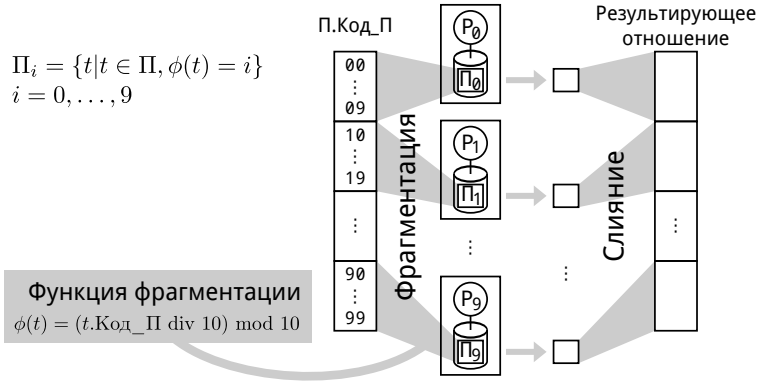


Рис. 6. Параллельная обработка запроса на основе фрагментного параллелизма

Архитектура клиент-серверного взаимодействия параллельной СУБД PargreSQL, в отличие от последовательной СУБД PostgreSQL, предполагает, что клиент взаимодействует с двумя или более серверами одновременно (см. Рис. 7).

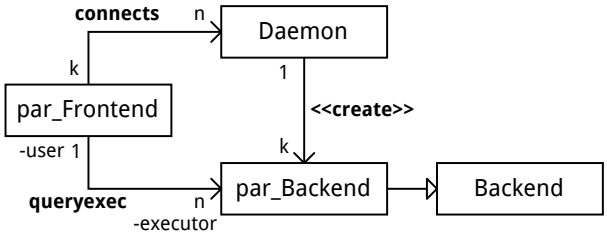


Рис. 7. Процессы СУБД PargreSQL

Порядок взаимодействия клиента и СУБД PargreSQL представлен на Рис. 8. Клиентское приложение подключается сразу ко всем экземплярам СУБД и отправляет им одинаковый запрос.

Параллельная обработка запроса состоит из следующих этапов (см. Рис. 9):

- *parse* — разбор запроса на языке SQL;
- *rewrite* — преобразование запроса;
- *plan/optimize* — составление последовательного плана запроса и его оптимизация;

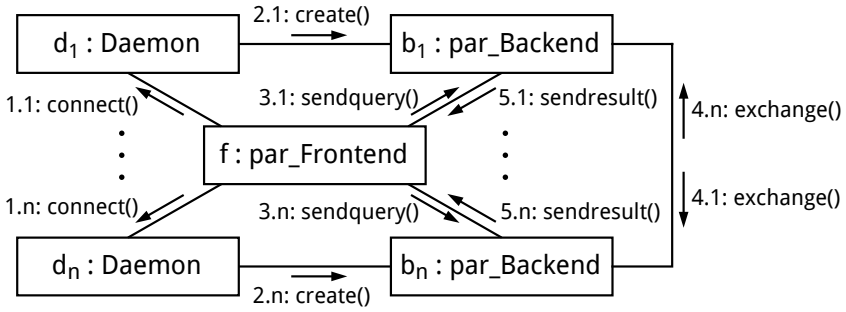


Рис. 8. Взаимодействие клиента и серверов PargreSQL

- *parallelize* — формирование параллельного плана запроса на основе последовательного путем вставки операторов *exchange*;
- *execute* — выполнение плана запроса;
- *balance* — балансировка загрузки серверных процессов.

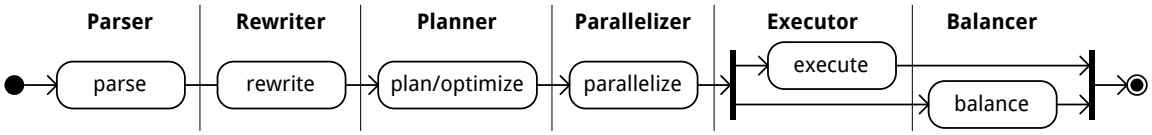


Рис. 9. Обработка запроса в СУБД PargreSQL

Архитектура PargreSQL представлена на Рис. 10. PostgreSQL является подсистемой в рамках системы PargreSQL. Для разработки PargreSQL необходимо внести изменения в исходные тексты следующих подсистем PostgreSQL: Storage, Executor и Planner.

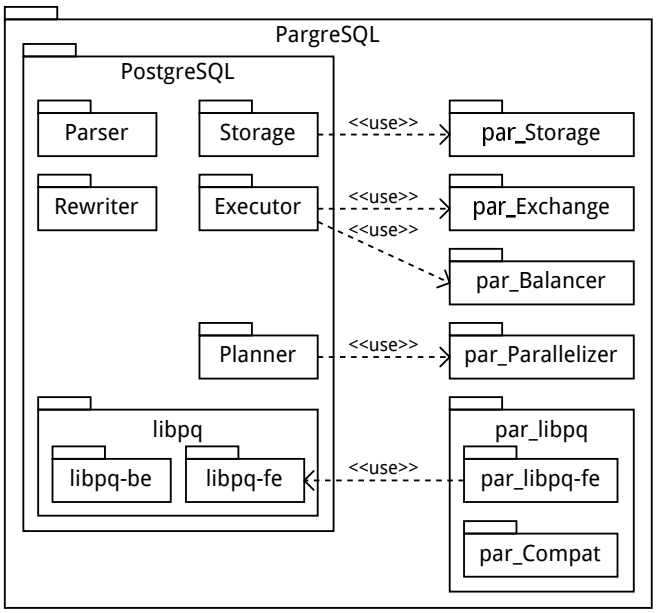


Рис. 10. Архитектура СУБД PargreSQL

Данные изменения обеспечивают внедрение следующих новых подсистем:

- *par_Storage* — подсистема хранения данных о фрагментации отношений;

- *par_ Exchange* — подсистема, реализующая оператор *exchange* [3], который выполняет обмен кортежами между экземплярами СУБД;
- *par_ Parallelizer* — подсистема, выполняющая добавление в нужные места последовательного плана запроса операторов *exhange*;
- *par_ Balancer* — подсистема, выполняющая динамическую балансировку загрузки серверных процессов.

Задача оператора *exchange* — передать все кортежи, которые должны быть обработаны ядрами PargreSQL на других вычислительных узлах, и получить все кортежи, предназначенные для обработки ядром PargreSQL на данном узле. Реализация оператора *exchange* инкапсулирует все аспекты, связанные с распараллеливанием запроса, так как он имеет стандартный итераторный интерфейс и может быть помещен в любое место дерева запроса.

В PargreSQL также входят следующие новые подсистемы, которые не требуют изменения оригинальных подсистем PostgreSQL:

- *par_ libpq-fe* — надстройка над `libpq-fe`, реализующая тиражирование запроса;
- *par_ Compat* — подсистема, реализующая прозрачное для приложения подключение `par_libpq-fe`.

Размещение компонентов СУБД PargreSQL приведено на Рис. 11.

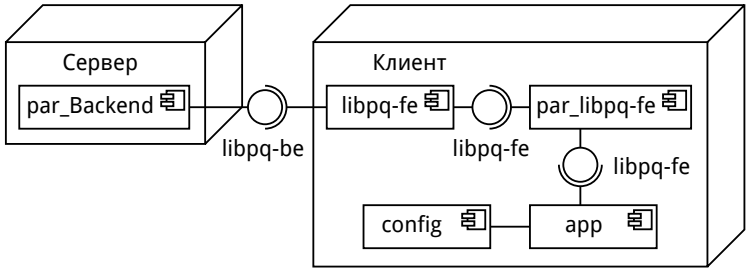


Рис. 11. Размещение компонентов PargreSQL

На клиенте размещаются библиотеки `par_libpq` и `libpq-fe` и приложение пользователя вместе с конфигурационным файлом в формате XML, в котором определяются параметры работы приложения (адреса узлов, фрагментация таблиц и др.). Остальные компоненты размещаются на узлах сервера.

5. Принципы реализации PargreSQL

PargreSQL разрабатывается в соответствии со следующими основными принципами: масштабируемость, минимальность и прозрачность.

Масштабируемость заключается в том, что параллельная СУБД PargreSQL, запущенная на одном вычислительном узле, работает так же, как последовательная СУБД PostgreSQL.

Масштабируемость реализуется путем использования оператора *exchange* [3]. Оператор *exchange* вычисляет значение функции пересылки для каждого поступающего кортежа и передает кортеж на вычислительный узел, номер которого совпадает со значением функции пересылки.

Таким образом, оператор *exchange* не изменяет кортеж (передает его в вышележащий узел плана), если значение функции пересылки совпадает с номером текущего узла. Когда

PargreSQL запускается на одном узле, функция пересылки тождественно равна номеру единственного узла — нулю.

В исходные тексты PostgreSQL вносятся *минимальные* изменения. Изменения в структурах данных и алгоритмах инкапсулируются в новых файлах исходных текстов, подключаемых к исходным текстам PostgreSQL.

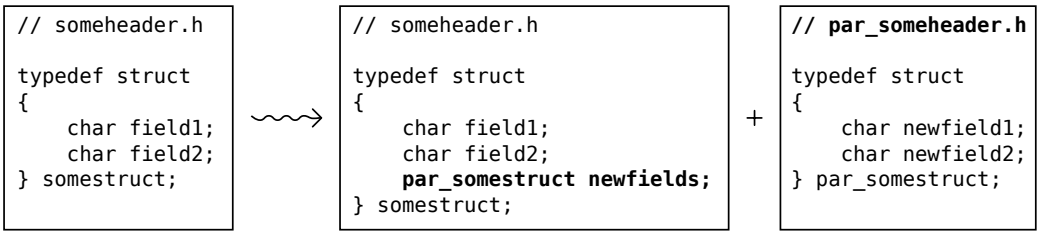


Рис. 12. Техника внесения изменений в определения структур данных

На Рис. 12 показан пример применения данного подхода для добавления новых полей в оригинальную структуру данных. В новом файле описывается тип `par_somestruct`, содержащий новые поля, а в оригинильную структуру добавляется новое поле типа `par_somestruct`.

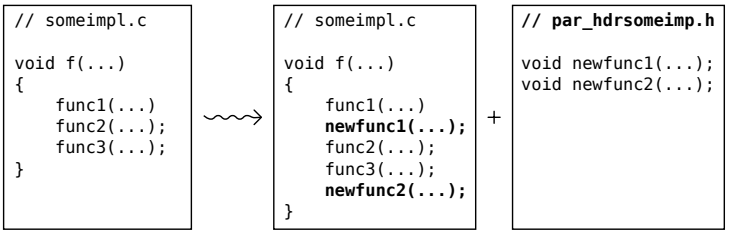


Рис. 13. Техника внесения изменений в исходные тексты функций

На Рис. 13 показан пример применения данного подхода для изменения оригинальных алгоритмов. В тело оригинальной функции добавляется вызов новой функции `newfunc()`, а сама функция `newfunc()` определяется в файле исходных текстов новой подсистемы.

Использование PargreSQL является *прозрачным* для пользовательских приложений. Подключение PargreSQL к прикладным программам, которые до этого использовали PostgreSQL, производится с минимальными изменениями в исходных кодах приложения.

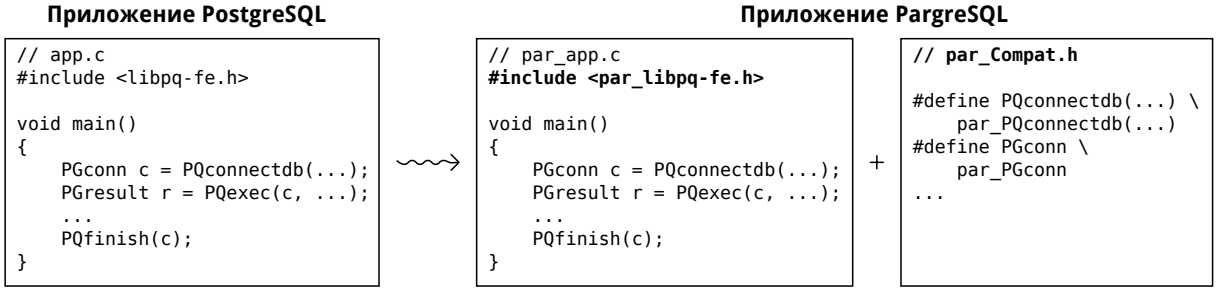


Рис. 14. Прозрачность использования `par_libpq`

Прозрачность реализуется следующим образом. Пользовательское приложение вместе с заголовочным файлом `par_libpq-fe.h` подключает заголовочный файл `par_Compat.h`. Этот файл содержит объявление макросов, заменяющих вызовы функций подсистемы `libpq` на вызовы функций подсистемы `par_libpq`. Таким образом, для адаптации PostgreSQL-приложения в исходном тексте приложения требуется изменение одной строки кода.

На Рис. 14 показан прозрачный способ подключения `par_libpq`.

6. Заключение

В данной работе описана архитектура и принципы реализации параллельной СУБД PargreSQL для многопроцессорных вычислительных систем с кластерной архитектурой. PargreSQL основана на свободной СУБД PostgreSQL и использует фрагментный параллелизм.

Направлением дальнейших исследований является завершение разработки PargreSQL и проведение экспериментов по исследованию ее ускорения и масштабируемости.

Литература

1. *Stonebraker M., Kemnitz G.* The POSTGRES next-generation database management system // Communications of the ACM. Oct. 1991. Vol. 34, No. 10. P. 78–92.
2. *Sokolinsky L., Axenov O., Gutova S.* Omega: The Highly Parallel Database System Project // Proceedings of the First East-European Symposium on Advances in Database and Information Systems (ADBIS'97), St.-Petersburg, September 2–5, 1997. St.-Petersburg: Nevsky Dialect. 1997. Vol. 2. P. 88–90.
3. *Соколинский Л.Б.* Организация параллельного выполнения запросов в многопроцессорной машине баз данных с иерархической архитектурой // Программирование. 2001. № 6. С. 13-29.
4. *Samokhvalov N.* XML Support in PostgreSQL // SYRCoDIS, volume 256 of *CEUR Workshop Proceedings*. 2007.
5. *Havinga Y., Dijkstra W., de Keijzer A.* Adding HL7 version 3 data types to PostgreSQL // CoRR, abs/1003.3370, 2010.
6. *Guliato D., de Melo E.V., Rangayyan R.M., Soares R.C.* POSTGRESQL-IE: An Image-handling Extension for PostgreSQL // Journal of Digital Imaging, 22(2):149–165. 2009.
7. *Levshin D.V., Markov A.S.* Algorithms for integrating PostgreSQL with the semantic web // Programming and Computer Software, 35(3):136–144. 2009.
8. *Lee R., Zhou M.* Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing // Database Systems for Advanced Applications, volume 4443 of *Lecture Notes in Computer Science*, P. 1086–1097. Springer. 2007.
9. *Paes M., Lima A.A.B., Valduriez P., Mattoso M.* High-Performance Query Processing of a Real-World OLAP Database with ParGRES // VECPAR, volume 5336 of *Lecture Notes in Computer Science*, P. 188–200. Springer. 2008.
10. *Kotowski N., Lima A.A.B, Pacitti E., Valduriez P., Mattoso M.* Parallel query processing for OLAP in grids // Concurrency and Computation: Practice and Experience, 20(17):2039–2048. 2008.
11. *DeWitt D.J., Gray J.* Parallel Database Systems: The Future of High Performance Database Systems // Communications of the ACM, 35(6):85–98. 1992.
12. *Sokolinsky L.B.* Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // Programming and Computer Software, 27(6):297–308. 2001.
13. *Lepikhov A.V., Sokolinsky L.B.* Query processing in a DBMS for cluster systems // Programming and Computer Software, 36(4):205–215. 2010.

Сравнение параллельных реализаций симплекс-метода для безошибочного решения задач линейного программирования *

А.В. Панюков, В.В. Горбик

Южно-Уральский государственный университет

В работе рассмотрены подходы к решению задачи линейного программирования с произвольно заданной точностью. Абсолютная точность вычислений достигается применением в алгоритмах симплекс-метода дробно-рациональных вычислений без округления. Если при этом m – минимальная из размерностей задачи, l – число бит, необходимых под один численный элемент исходных данных, то пространственная сложность алгоритма не превосходит $4lm^4 + o(m^3)$, при этом вычислительная сложность одной итерации симплекс-метода не превосходит $O(lm^4)$, а эффективность распараллеливания (т.е. отношение ускорения к числу процессоров) в предложенной реализации параллельного алгоритма составляет в асимптотике 100%. Известные полиномиальные алгоритмы в линейном программировании предполагают приближенные вычисления с наперед заданной точностью, которые могут быть реализованы с помощью представления данных в формате с плавающей точкой необходимой длины. Представлены результаты вычислительных экспериментов на основе реализаций параллельных версий алгоритмов решения задач линейного программирования.

1. Введение

В настоящее время широко распространены предрассудки, не основанные на доказательствах и порождающие ошибки в расчетах: (1) распространение свойства ассоциативности операций сложения и умножения в поле действительных чисел на конечное множество машинных “действительных” чисел; (2) распространение свойства непрерывной зависимости от параметров решения системы, полученной после «эквивалентных» преобразований, на исходную систему имеют популярные коммерческие пакеты MatLab, MathCad и т.п., а также свободно распространяемый пакет SciLab. Использование в вычислениях разного числа процессоров во многих случаях дает существенно различающиеся результаты, демонстрируя необходимость доказательных вычислений (см. работы [1], [2], [3], [4] [5]).

Потенциал имеющихся пакетов, поддерживающих символические вычисления, не позволяет решать реальные проблемы математического и имитационного моделирования. Возможность обеспечения вычислений с произвольной точностью в программах пользователя дает библиотека GMP [6]. Однако, ее использование требует от пользователя разработки собственного интерфейса для организации распределенных и параллельных вычислений [7]. Развитием библиотеки GMP является библиотека ExactComputational [8], которая предоставляет своим объектам возможность их использования в параллельных вычислениях.

Ориентация на применение многопроцессорных вычислительных систем в составе персональных компьютеров или рабочих станций (параллельные вычисления) и на применение сетевых технологий (распределенные вычисления) требует разработки новых параллельных методов их решения. Они должны быть лишены недостатков «традиционных» методов таких, например, как последовательный характер вычислений. Анализ способов распараллеливания показывает эффективность распараллеливания «по информации». Поэтому, весьма перспективной становится SPMD-технология программирования (Single Program - Multiple Data). При этой технологии вычислительный процесс строится на основе единственной программы, запускаемой на всех процессорах вычислительной системы или на

*Работа выполнена при поддержке РФФИ, проект № 10-07-96003-р_урал_а

многих станциях локальной сети. Копии программы могут выполняться по разным ветвям алгоритма, обрабатывая подмножества данных. Неизбежна синхронизация во времени и при обработке общих данных. Данная идеология используется в стандарте MPI (Message Passing Interface [9]). Такая технология параллельного программирования и обусловила разработку соответствующих методов. В то же время не отрицаются известные традиционные методы, сокращающие общее число операций и исключающие перебор. Иной методологический подход открывает дорогу к решению задач большой размерности и эффективной параллельной работе многих процессоров.

Ранее авторами разработаны алгоритмы и программное обеспечение для абсолютно точного решения систем линейных алгебраических уравнений [10] и вычисления обобщенной обратной матрицы Мура-Пенроуза [11] на многопроцессорных вычислительных системах с использованием классов overlong и rational из библиотеки ExactComputational [8]. Теоретическое и практическое исследование данного программного обеспечения демонстрирует высокую эффективность использования многопроцессорных вычислительных систем.

Целью работы является развитие программного обеспечения безошибочных дробно-рациональных и приближенных вычислений, обеспечивающих заданную гарантированную оценку погрешности, для параллельных и распределенных вычислительных систем и его применение для решения задач линейного программирования.

2. Техника реализации симплекс-метода

Применение симплекс-метода для практических задач линейного программирования остается вне конкуренции, несмотря на появившиеся полиномиальные алгоритмы. В настоящее время используются две техники реализации симплекс-метода:

- метод симплекс таблиц;
- метод обратной матрицы (модифицированный симплекс метод).

Для сохранения целостности изложения приведем их особенности на примере задачи линейного программирования

$$\max\left\{c^T x \ : Ax = b \geq 0,\ x \geq 0;\ c, x \in \mathbf{R}^n;\ b \in \mathbf{R}^m\right\}. \tag{1}$$

2.1. Метод симплекс-таблиц

Данный метод на итерации k пересчитывает симплекс-таблицу

$$S^{(k)} = \left|\begin{array}{c|c} Z^{(k)} = c_{B(k)}^T B(k)^{-1} b & z^{(k)} = -c^T + c_{B(k)}^T B(k)^{-1} A \\ \hline X_{B(k)} = B(k)^{-1} b & B(k)^{-1} A \end{array}\right|$$

где $B(k)$ – базисная матрица, содержащая все относящиеся к базисным переменным k-й итерации столбцы матрицы A (базисные столбцы); $c_{B(k)}$ – вектор коэффициентов целевой функции, относящихся к базисным переменным k-й итерации. При этом левый столбец симплекс-таблицы содержит вектор $X_{B(k)} = B(k)^{-1}b$ значений базисных переменных k-й итерации и значение $Z^{(k)}$ целевой функции на этом решении. Столбцы матрицы S, соответствующие базисным переменным, являются ортами (т.е. из них может быть составлена единичная матрица). Верхняя строка содержит вектор $z^{(k)} = -c^T + c_{B(k)}^T B(k)^{-1} A$ невязок двойственной задачи. Значения элементов строки $z^{(k)}$, относящиеся к базисным столбцам являются нулевыми.

Критерием оптимальности текущего базисного решения является неотрицательность вектора z. Если же существует небазисная переменная $x_i:\ z_i^{(k)} < 0$, то ее введение в число базисных приведет к увеличению целевой функции задачи. В этом случае если множество $L = \{l:\ S_{li}^{(k)} > 0\} = \emptyset$, то целевая функция задачи неограничена, иначе введение переменной x_i в число базисных приведет к увеличению целевой функции на величину

$$\Delta_i = -\frac{X_{B(k)_{l^*}} \cdot z_i^{(k)}}{S_{l^*i}^{(k)}}, \quad \text{где } l^* = \arg\min_{l \in L} \frac{X_{B(k)_{l^*}}}{S_{li}^{(k)}}. \tag{2}$$

Переход от таблицы k-й итерации к таблице $(k+1)$-й итерации осуществляется применением процедуры исключения Жордана-Гаусса для столбца i (ведущего столбца), используя в качестве ведущей строку l^*

$$(\forall l = 0,1,2,\ldots,m;\ j = 0,1,2,\ldots,n)\left(S_{lj}^{(k+1)} = \begin{cases} S_{lj}^{(k)} - \dfrac{S_{l^*j}^{(k)}}{S_{l^*i}^{(k)}} S_{li}^{(k)}, & \text{если } l \neq l^*, \\ \dfrac{S_{l^*j}^{(k)}}{S_{l^*i}^{(k)}}, & \text{если } l = l^* \end{cases}\right). \tag{3}$$

Легко заметить, что выполнение итерации, включая пересчет симплекс-таблицы, потребует не более $(m+n)$ операций деления и сравнения, а также не менее $m(n+1)$ операций сложения и умножения. Алгебраическая пространственная сложность табличного симплекс-метода в основном определяется числом операндов в симплекс-таблице, т.е. равна $mn + O(m)$.

2.2. Метод обратной матрицы

В данном методе, в отличие от табличного, на каждой итерации вместо пересчета симплекс-таблицы пересчитывается матрица $B(k)^{-1}$. Наличие обратной матрицы позволяет для текущего базиса легко находить $y(k)^T = c_{B(k)}^T B(k)^{-1}$ – соответствующее решение двойственной задачи. Текущий базис является оптимальным если соответствующее ему двойственное решение допустимо: $y(k)^T A \geq c^T$. Если же в матрице A найдется столбец $A_{i(k)}:\ y(k)^T A_{i(k)} < c_{i(k)}$, то введение его в число базисных приведет к увеличению целевой функции.

Образом столбца $A_{i(k)}$ в симплекс-таблице $S(k)$ будет вектор $g = B(k)^{-1} A_{i(k)}$. Поэтому ведущей строкой, определяющей выводимый из базиса столбец, будет

$$r = \arg \min_{l:\ g_l > 0} \frac{X_{B(k)_l}}{g_l}, \tag{4}$$

а новые значения базисных переменных равны

$$(\forall l = 1,2,\ldots,m)\left(X_{B(k+1)_l} = \begin{cases} X_{B(k)_l} - \dfrac{g_l}{g_r} X_{B(k)_r}, & \text{если } l \neq r, \\ \dfrac{X_{B(k)_l}}{g_r}, & \text{если } l = r; \end{cases}\right). \tag{5}$$

Базисные матрицы $B(k) = \left(b_{lj}^{(k)}\right)_{l,j=1}^{m}$ и $B(k+1) = \left(b_{lj}^{(k+1)}\right)_{l,j=1}^{m}$ отличаются только r-м столбцом, поэтому элементы обратной матрицы $B(k+1)^{-1} = \left(\beta_{lj}^{(k+1)}\right)_{l,j=1}^{m}$ следующим образом вычисляются через элементы матрицы $B(k)^{-1} = \left(\beta_{lj}^{(k)}\right)_{l,j=1}^{m}$

$$\beta_{lj}^{(k+1)} = \begin{cases} \beta_{lj}^{(k)} - \dfrac{g_l}{g_r} \beta_{rj}^{(k)}, & \text{если } l \neq r, \\ \dfrac{1}{g_r} \beta_{rj}^{(k)}, & \text{если } l = r. \end{cases} \tag{6}$$

Оценим алгебраическую вычислительную сложность рассматриваемого метода. Очевидно, что в общем случае на заключительной итерации потребуется проверка всех n ограничений двойственной задачи, для чего потребуется не более mn операций умножения и $m(n-1)$ операций сложения. На промежуточных итерациях для установления недопустимости двойственного решения эта величина как правило является существенно меньшей. Пересчет обратной матрицы потребует не более m операций деления, не более m^2 операций умножения и не более (m^2-1) операций сложения/вычитания. Поскольку $m < n$,то затраты вычислительных ресурсов на пересчет обратной матрицы могут быть существенно ниже затрат на пересчет симплекс-таблицы. Алгебраическая пространственная сложность метода обратной матрицы определяется числом элементов в исходных данных и в обратной матрице, т.е. равна $mn + m^2 + O(m)$, т.е. больше чем в табличном симплекс-методе.

2.3. Сопоставление методов

Подводя итог изложенному, делаем вывод, что применение метода обратной матрицы оправдано когда

- требуется найти решение двойственной задачи;
- число n существенно превосходит m;
- матрица A разрежена;
- возможна оптимизация проверки допустимости двойственного решения.

Отметим, что алгебраическая сложность дает адекватные оценки используемого вычислительного ресурса только когда все операнды имеют одинаковую длину, например, при использовании стандартных типов данных. При выполнении дробно-рациональных вычислений без округления память необходимая для операндов динамически изменяется (как правило возрастает), поэтому в этом случае более адекватными являются оценки фактически достаточного объема памяти и числа элементарных операций с битами. Эти величины принято называть битовой сложностью (пространственной и вычислительной соответственно).

2.4. Битовая сложность абсолютно точной реализации симплекс-метода

Практическая реализуемость вычислений без округления, в частности, безошибочного решения задачи линейного программирования, определяется требуемыми для вычислений ресурсами: числом бит оперативной памяти и количеством операций с битами. Далее через $S(\lambda)$ будем обозначать число бит, требуемое для представления объекта λ, а через $C(\lambda)$ – число битовых операций, выполненных при нахождении представления объекта λ. Например, число бит, требуемое для представления целых чисел будет равно $S(0) = 1$, $(\forall z \in \mathbf{Z} \setminus \{0\})S(z) = \lceil \log_2 |z| \rceil$. Число бит, требуемое для представления рационального числа $r = p/q$, $p, q \in \mathbf{Z}$ имеет верхнюю оценку $S(r) \le O(S(p) + S(q))$.

Легко проверить, что если $S(p)$, $S(q)$ – память, требуемая для представления рациональных чисел p, q, то память, требуемая для представления результата арифметической операции $\circ \in \{+, -, /, \times\}$ над данными числами будет $S(p \circ q) \le S(p) + S(q)$. Для *битовой* вычислительной сложности выполнения операции $\circ \in \{+, -, /, \times\}$ с помощью классических арифметических алгоритмов (умножение/деление столбиком) справедлива оценка $C(p \circ q) \le S(p)S(q)$. Использование алгоритмов быстрого умножения дает оценку $C(p \circ q) \le S(p) + S(q)$, которая будет использована в работе.

Оценим число бит оперативной памяти достаточное для решения задачи линейного программирования с применением вычислений без округления. Поскольку как элементы симплекс-таблицы, так и элементы обратной матрицы являются решениями систем линейных алгебраических уравнений, то сначала найдем число бит, требуемое для представления

определителя матрицы с элементами, имеющими заданную верхнюю оценку пространственной сложности.

Утверждение 1 *Пусть $B = (b_{ij})$ – целочисленная $m \times m$ матрица, $l = \max\limits_{i,j=1,2,\ldots,m} S(a_{ij})$. Тогда*

$$S(\det A) \le n(\log_2 m + l).$$

Доказательство. Рассмотрим верхнюю оценку абсолютной величины определителя

$$|\det B| = \left| \sum_{\sigma} \prod_{k=1}^{m} b_{k\sigma(k)} \right| \le \sum_{\sigma} \prod_{k=1}^{m} \left| b_{k\sigma(k)} \right| \le m! L^m \le (mL)^m,$$

где $L = \max\{|b_{ij}| : \ i, j = 1, 2, \ldots, m\}$.

Из данной оценки следует $S(\det B) = \log_2 |\det B| \le m(\log_2 m + l)$ ■

Утверждение 2 *Пусть $B = (b_{ij})$ – $m \times m$ рациональная матрица, $l = \max\limits_{i,j=1,2,\ldots,m} S(b_{ij})$. Тогда $S(\det B) \le m(\log_2 m + (2m+1)l)$.*

Доказательство. Пусть K_r равно наименьшему общему всех кратному всех знаменателей строки $r = 1, 2, \ldots, m$ матрицы B. Очевидно, что $S(K_r) \le lm$. Пусть K представляет диагональную матрицу: $\mathtt{diag}(K) = \{K_r : \ r = 1, 2, \ldots, m\}$. Рассмотрим матрицу $\tilde{B} = KB$. Данная матрица является целочисленной. Верхняя оценка числа бит, необходимых для одного элемента матрицы $\tilde{B}$, имеет вид

$$\tilde{l} = \max_{i,j=1,2,\ldots,m} S(\tilde{b}_{ij}) = \max_{i,j=1,2,\ldots,m} S(K_i b_{ij}) \le l(m+1).$$

Принимая во внимание утверждение 1, а также $\log_2 |\det K^{-1}| \le lm$, получим

$$S(\det B) = S\left(\det K^{-1} \cdot \det(\tilde{B})\right) \le m(\log_2 m + (2m+1)l) \ \blacksquare$$

Из формул Крамера для систем линейных алгебраических уравнений и доказанных утверждений вытекает

Утверждение 3 *Если один численный элемент исходных данных имеет пространственную сложность не более l, то*

- *элементы симплекс-таблицы и обратной матрицы имеют пространственную сложность не более $4lm^2 + lm + m\log_2 m + 1$;*
- *столбец симплекс-таблицы и обратной матрицы имеют пространственную сложность не более $4lm^3 + lm^2 + m^2 \log_2 m + m$;*
- *для представления симплекс-таблицы достаточно $4lm^3 n + o(lm^2 n)$ бит;*
- *для представления обратной матрицы достаточно $4lm^4 + o(lm^3)$ бит.*

Поскольку $m < n$, а наиболее критическим ресурсом при использовании точных дробно-рациональных вычислений является память, то целесообразным является использование метода обратной матрицы.

3. Параллельные версии симплекс метода

3.1. Метод симплекс-таблиц

Для реализации метода симплекс-таблиц наиболее подходящей является декомпозиция симплекс-таблицы по столбцам на число блоков равное числу процессоров N. Все столбцы

Процесс $K = 1, 2, \ldots, N$				
$S_{00} = Z$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 1}$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$z_{\lceil \frac{Kn}{N} \rceil}$
$S_{10} = X_{B1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{1\lceil \frac{Kn}{N} \rceil}$
$S_{20} = X_{B2}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{2\lceil \frac{Kn}{N} \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$S_{m0} = X_{Bm}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{m\lceil \frac{Kn}{N} \rceil}$

Рис. 1. Декомпозиция симплекс-таблицы по процессорам

симплекс-таблицы, за исключением левого столбца, делятся в равных пропорциях между N процессами, левый столбец т.е. вектор значений базисных переменных и значение целевой функции на нем, рассылаются всем процессам и обрабатываются ими независимо. Пример разбиения симплекс-таблицы S на блоки $S(K),\ K = 1, 2, \ldots, N$ представлен на рис. 1.

Принятые соглашения позволяют предложить следующую параллельную реализацию итерации метода симплекс-таблиц.

Алгоритм TabularSimplex

k-я итерация

- **Данные:** симплекс-таблица $S(K)$ для каждого процесса $K = 1, 2, \ldots, N$.
- **Шаг 1.** Каждому процессу $K = 1, 2, \ldots, N$ найти столбец $i_K :\ z_{i_K} < 0$. Если столбец i_K не найден, то вернуть "Пусто", иначе найти строку

$$l_K = \arg \min_{l:\ S_{l\, i_K} > 0} \frac{X_{Bl}}{S_{l\, i_K}}.$$

  Если строка l_K не найдена, то завершить выполнение всех процессов и вернуть "Не ограничена". В противном случае вычислить значение $\Delta_{i_K} = -X_{Bl_K} z_{i_K} / S_{l_K i_K}$.
  Комментарий. *При выполнении данного шага будет либо установлена неразрешимость задачи, либо найдены данные для изменения базиса: ведущий столбец i_K – кандидат для ввода в базис, ведущая строка l_K, определяющая столбец выводимый из базиса, и приращение целевой функции Δ_{i_K}, – либо установлено отсутствие таких кандидатов.*

- **Шаг 2.** Если $\{K :\ \exists i_K\} \neq \emptyset$, то определить ведущий процесс

$$K^* = \arg \max_{K:\ \exists i_K} \Delta_{i_K}$$

  и прийти на **Шаг 3**. Иначе каждому процессу $K = 1, 2, \ldots, N$ для

$$k = \lceil \frac{(K-1)n}{N} \rceil + 1,\ \lceil \frac{(K-1)n}{N} \rceil + 2, \ldots, \lceil \frac{Kn}{N} \rceil$$

  положить

$$x_k = \begin{cases} X_{Bl}, & \text{если } (S_{lk} = 1) \bigwedge ((\forall i = 0, 1, \ldots, l-1, l+1, \ldots, m)\, S_{ik} = 0), \\ 0, & \text{в противном случае.} \end{cases}$$

Таблица 1. Алгебраический вычислительный ресурс реализаций метода симплекс-таблиц

Оператор	Один процессор	N процессоров	
	Количество алгебраических операций	Количество пересылаемых операндов	Нагрузка на один процесс
Проверка условия оптимальности	$[1, n]$	–	$[1, n/N]$
Определение ведущей строки	$2m + n + 2$	–	$2m + n/N + 2$
Выбор ведущего процесса	–	N	$\lceil \log_2 N \rceil$
Пересылка ведущего столбца	–	$m + 2$	–
Пересчет симплекс-таблицы	$2(m + 1)(n + 1)$	–	$2m(1 + n/N)$
Итого:	$[1, n] + 2mn +$ $4(m + n + 1)$	$m + N + 2$	$[1, n/N] +$ $2(n/N)(m + 1) +$ $4m + 2 + \log_2 N$

Вернуть x – оптимальное решение задачи, Z – оптимальное значение целевой функции.
Комментарий. *При выполнении данного шага будет либо установлена оптимальность текущего базисного решения задачи и возвращены найденные оптимальные решение и значение целевой функции, либо найден ведущий процесс и продолжен процесс оптимизации.*

- **Шаг 3.** Ведущему процессу K^* передать остальным процессам ведущий столбец

$$S_{i_{K^*}} = (z_{i_{K^*}},\ S_{1i_{K^*}},\ S_{2i_{K^*}},\ \ldots,\ S_{mi_{K^*}})^T$$

и номер ведущей строки l_{K^*}.

- **Шаг 4.** Каждому процессу $K = 1, 2, \ldots, N$ пересчитать по формуле (3) симплекс-таблицу $S(K)$. Перейти к следующей итерации.

- Конец алгоритма

Из изложенного выше видно, что описание параллельной версии табличного симплекс-метода не намного сложнее чем для обычного алгоритма. При этом используется одна широковещательная коммуникация между процессами при нахождении ведущего процесса K^* для обмена значениями Δ_{i_K}, $K = 1, 2, \ldots, N$ и одна широковещательная коммуникация для передачи ведущего столбца $S_{i_{K^*}}$ и числа l_{K^*}. В таблице 3.1 приведена сводка требуемого алгебраического (т.е. количества используемых алгебраических операций) вычислительного ресурса для последовательной и параллельной реализаций метода симплекс-таблиц. Из таблицы и описания алгоритма видно, что процессоры загружены равномерно, а эффективность распараллеливания растет с ростом сложности задачи, достигая в пределе 100%.

3.2. Метод обратной матрицы

Из описания метода обратной матрицы следует, что максимальный эффект от распараллеливания при поиске вводимой в базис переменной достигается при декомпозиции

исходных данных: вектора c^T и матрицы A, – по столбцам в равных пропорциях между процессорами на блоки

$$c(K)^T = \left(\begin{array}{cccc} c_{\lceil \frac{(K-1)n}{N} \rceil+1} & c_{\lceil \frac{(K-1)n}{N} \rceil+2} & \cdots & c_{\lceil \frac{Kn}{N} \rceil} \end{array} \right), \quad K = 1, 2, \ldots, N; \tag{7}$$

$$A(K) = \left(\begin{array}{cccc} a_1\big(\lceil \frac{(K-1)n}{N} \rceil+1\big) & a_1\big(\lceil \frac{(K-1)n}{N} \rceil+2\big) & \cdots & a_1\big(\lceil \frac{Kn}{N} \rceil\big) \\ a_2\big(\lceil \frac{(K-1)n}{N} \rceil+1\big) & a_2\big(\lceil \frac{(K-1)n}{N} \rceil+2\big) & \cdots & a_2\big(\lceil \frac{Kn}{N} \rceil\big) \\ \vdots & \vdots & \ddots & \vdots \\ a_m\big(\lceil \frac{(K-1)n}{N} \rceil+1\big) & a_m\big(\lceil \frac{(K-1)n}{N} \rceil+2\big) & \cdots & a_m\big(\lceil \frac{Kn}{N} \rceil\big) \end{array} \right), \quad K = 1, 2, \ldots, N. \tag{8}$$

При этом предполагается. что вектор двойственных переменных y размещен в каждом процессе $K = 1, 2, \ldots, N$.

Эффект от распараллеливания при пересчете обратной матрицы B^{-1} будет максимальным при ее декомпозиции по строкам в равных пропорциях на блоки по числу процессов

$$B^{-1}(K) = \left(\begin{array}{cccc} b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)1} & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)2} & \cdots & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+1\right)m} \\ b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)1} & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)2} & \cdots & b_{\left(\lceil \frac{(K-1)m}{N} \rceil+2\right)m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\left(\lceil \frac{Km}{N} \rceil\right)1} & b_{\left(\lceil \frac{Km}{N} \rceil+2\right)2} & \cdots & b_{\left(\lceil \frac{Km}{N} \rceil+2\right)m} \end{array} \right), \quad K = 1, 2, \ldots, N. \tag{9}$$

Из описания метода обратной матрицы следует целесообразность декомпозиции вектора базисных переменных X_B и вектора g по таким же блокам строк

$$X_B(K) = \left(\begin{array}{c} x_{B\lceil \frac{(K-1)m}{N} \rceil+1} \\ x_{B\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ x_{B\lceil \frac{Km}{N} \rceil} \end{array} \right); \quad c_B(K) = \left(\begin{array}{c} c_{B\lceil \frac{(K-1)m}{N} \rceil+1} \\ c_{B\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ c_{B\lceil \frac{Km}{N} \rceil} \end{array} \right);$$

$$g(K) = \left(\begin{array}{c} g_{\lceil \frac{(K-1)m}{N} \rceil+1} \\ g_{\lceil \frac{(K-1)m}{N} \rceil+2} \\ \vdots \\ g_{\lceil \frac{Km}{N} \rceil} \end{array} \right); \qquad K = 1, 2, \ldots, N. \tag{10}$$

Способ (9) декомпозиции обратной матрицы по процессам позволяет каждому процессу вычислить субвектор двойственных переменных

$$y(K) = \left(B(K)^{-1}\right)^T c_{B(K)}. \tag{11}$$

Очевидно, что вектор действенных переменных

$$y = \sum_{K=1}^{N} y(k).$$

Изложенное выше позволяет предложить следующую параллельную реализацию итерации метода обратной матрицы.

Алгоритм InvBasMatrSimplex

k-я итерация

- **Данные:** в каждом процессе $K = 1, 2, \ldots, N$ размещены
  y – вектор двойственных переменных,
  $M(K)$ – вектор номеров базисных переменных процесса,
  $X_B(K)$ – вектор значений базисных переменных процесса,
  $c_B(K)$ – вектор значений коэффициентов целевой функции при базисных переменных процесса,
  $A(K)$ – блок матрицы ограничений процесса,
  $c(K)^T$ – блок коэффициентов целевой функции процесса,
  $B^{-1}(K)$ – блок обратной матрицы процесса.

- **Шаг 1.** Каждому процессу $K = 1, 2, \ldots, N$ найти столбец

$$i_K : \ z_{i_K} = A(K)^T_{i_K} y - c(K)_{i_K} < 0.$$

  Если столбец i_K не найден, то вернуть "Пусто".

- **Шаг 2.** Если $\{K : \ \exists i_K\} \neq \emptyset$, то определить ведущий процесс

$$K^c = \arg \min_{K:\ \exists i_K} z_{i_K}.$$

  и прийти на **Шаг 3**. Иначе сформировать решение задачи: положить $x[1 : n] = 0$, каждому процессу $K = 1, 2, \ldots, N$ для

$$r = \lceil \frac{(K-1)m}{N} \rceil + 1, \ \lceil \frac{(K-1)m}{N} \rceil + 2, \ldots, \lceil \frac{Km}{N} \rceil$$

  положить

$$x_{M(K)_r} = X_{B(K)_r}.$$

  Вернуть x – оптимальное решение задачи, y – оптимальное решение двойственной задачи.

- **Шаг 3.** Процессу K^c разослать всем процессам $K = 1, 2, \ldots, N$ столбец $A_{i_{K^c}}$ и $c_{i_{K^c}}$.

- **Шаг 4.** Каждому процессу $K = 1, 2, \ldots, N$ вычислить $g(K) = B(K)^{-1} A_{i_{K^c}}$ и

$$r(K) = \arg \min_{l:\ g(K)_l > 0} \left[h(K, l) = \frac{X_B(K)_l}{g(K)_l} \right].$$

  Если $r(K)$ не найдено, то вернуть "Не ограничена" и завершить выполнение алгоритма, в противном случае вернуть $r(K), \ h(K, r(K))$.

- **Шаг 5.** Определить ведущий процесс

$$K^r = \arg \min_{K:\ \exists r(K)} h(K, r(K)).$$

- **Шаг 6.** Ведущему процессу K^r разослать всем процессам $K = 1, 2, \ldots, N$ ведущую строку $r^* = r(K^r)$ обратной матрицы

$$\left(B(K)^{-1} \right)^{(r^*)} = \left(\ b_{r^*1}, \ \ b_{r^*2}, \ \ \cdots, \ \ b_{r^*m} \ \right)$$

  и значение $g(K^r)_{r^*}$. Положить $M(K)_{r^*} = i_{K^c}$, $c_B(K)_{r^*} = c_{i_{K^c}}$.

- **Шаг 7.** Каждому процессу $K = 1, 2, \ldots, N$ вычислить новые значения базисных переменных процесса $x_B(K)$ по формулам (5), новый блок обратной матрицы процесса $B^{-1}(K)$ по формулам (6), и двойственное субрешение блока

$$y(K) = \left(B(K)^{-1}\right)^T c_B(K).$$

- **Шаг 8.** Выполнить между процессами глобальный обмен полученными на шаге 8 субрешениями и вычислить значения двойственных переменных

$$y = \sum_{k=1}^{N} y(k).$$

  Перейти к следующей итерации.

- Конец алгоритма

Таким образом, описание параллельной версии метода обратной матрицы оказывается сложнее описания табличного симплекс-метода. В основном это объясняется большей степенью неоднородности используемых структур. При выполнении одной итерации используется пять широковещательных коммуникаций между процессами:

1) широковещательная коммуникация между процессами при нахождении ведущего процесса K^c для обмена значениями z_{i_K}, $K = 1, 2, \ldots, N$,

2) широковещательная коммуникация для передачи вводимого в базис (ведущего) столбца $A_{i_{K^*}}$ и его номера,

3) широковещательная коммуникация между процессами при нахождении ведущего процесса K^r для обмена значениями $x_B(K)_{r(K)}/g(K)_{r(K)}$, $K = 1, 2, \ldots, N$,

4) широковещательная коммуникация для передачи ведущей строки $\left(B(K)^{-1}\right)^{(r^*)}$ и ее номера,

5) широковещательная коммуникация для обмена между процессами двойственными субрешениями и нахождения двойственного решения.

В таблице 3.2 приведена сводка требуемого алгебраического вычислительного ресурса для последовательной и параллельной реализаций метода обратной матрицы. Из таблицы и описания алгоритма видно, что процессоры загружены равномерно, а эффективность распараллеливания растет с ростом сложности задачи, достигая в пределе 100%.

4. Заключение

Положительный эффект от распараллеливания, в случае использования дробно-рациональных вычислений без округления состоит не только в ускорении вычислений, но и возможности решать задачи большей размерности, т.к. достаточно легко достичь границ, когда матрица целиком не будет умещаться в оперативной памяти одного узла.

В основе вычислений всех рассмотренных нами коммерческих программ лежат типы данных с плавающей точкой, поэтому они не могут гарантировать высокую точность решения. Исключением из ряда продуктов, считающих приближенно, являются две открытых реализации симплекс метода, использующих в вычислениях библиотеку точных вычислений GNU MP, что неизбежно ведет к существенному увеличению времени счета. Однако они не используют преимущества параллельного программирования, методы которого, при

Таблица 2. Алгебраический вычислительный ресурс метода обратной матрицы

Оператор	Один процессор	N процессоров	
	Количество алгебраических операций	Количество пересылаемых операндов	Нагрузка на один процесс
Проверка условия оптимальности	$2m[1,n]$	–	$2m[1,n/N]$
Выбор c-ведущего процесса	–	N	$\log_2 N$
Передача ведущего столбца	–	$m+1$	–
Нахождение g	$m(m-1)$	–	$m(m-1)/N$
Нахождение ведущей строки	$2m$	N	$2m/N+\log_2 N$
Модификация X_B	$1+2m$	1	$2m/N$
Модификация B^{-1}	$3m^2$	m	$3(m^2/N)$
Вычисление $y(K)$	–	–	$(m/N)(2(m/N)-1)$
Вычисление y	$m(2m-1)$	N	$\log_2 N$
Итого:	$2m[1,n]+$ $6m^2+2m-1$	$2m+3N+2$	$2m[1,n/N]+$ $6m^2/N+2m/N+$ $3\log_2 N$

умелом использовании, позволяют сократить время расчетов и, следовательно, расширить круг решаемых задач (задачи большей размерности).

Основной задачей данной работы была разработка программы Plinpex решения задачи линейного программирования, основанной на параллельном алгоритме и использующей вычисления без округлений и интервальные вычисления на основе типов данных с плавающей точкой заданной точности. Plinpex основан на предложенных методах для адаптации используемых типов данных к MPI.

Проведенный эксперимент показал, что реализованный метод эффективен на задачах разной размерности и соотношением количества ограничений к количеству переменных. В результате проведенного эксперимента можно сделать выводы о высокой эффективности распараллеленного алгоритма симплекс метода. Согласно проведенным экспериментам, эффективность распараллеливания для типов данных с плавающей точности заданной точности прямо пропорционально зависит от длины мантиссы (количества бит точности) и составляет 70-80%, и показатель эффективности еще выше для точных дробно-рациональных вычислений. Однако, общее время вычислений может быть улучшено некоторыми оптимизациями реализации алгоритма, что является целью для дальнейшей работы.

Список литературы

1. *Ю.Г. Евтушенко, А.И. Голиков.* Параллельные алгоритмы решения задач линейного программирования. // Российская конференция "Дискретная оптимизация и исследование операций": Материалы конференции (Алтай, 27 июня – 3 июля 2010 г.). – Новосибирск: Изд-во Института математики, 2010. – С. 25–29.

2. *В.А. Гаранжа, А.И. Голиков, Ю.Г. Евтушенко.* Параллельная реализация метода Ньютона для решения больших задач линейного программирования. // Журнал вычислительной математики и математической физики. – Т. 49. – No 8. – С. 1369–1384.

3. *V. Y. Pan, J. H. Reif.* Fast and Efficient Parallel Linear Programming and Linear Least Squares Computations //Proceedings of the VLSI Algorithms and Architectures, Aegean Workshop on Computing. – Springer-Verlag. – 1986. P. 283–295.

4. *Ju. Hall.* Towards a practical parallelization of the simplex method // Journal Computational Management Science. – 2010. Vol. 7. – Number 2. – P. 139-170.

5. *G. G. Yarmish.* A Distributed Implementation of the Simplex Method, UMI Dissertations Publishing, 2001.

6. GNU Multiple Precision Arithmetic Library: http://gmplib.org/, 2010.

7. *A. V. Panyukov, V. V. Gorbik* Exact and Guaranteed Accuracy Solutions of Linear Programming Problems by Distributed Computer Systems with MPI // Tambov University REPORTS: A Theoretical and Applied Scientific Journal. Series: Natural and Technical Sciences. – Volume 15, Issue 4, 2010. – P. 1392-1404. http://vestnik.tsutmb.ru/old/index.php?module=subjects&func=viewpage&pageid=165

8. *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 No 3. – 2009. – С. 251.

9. MPI: A Message Passing Interface Standard: http://www.MPI-forum.org/docs/MPI-11-html/MPI-report.html, 1995.

10. *А.В. Панюков, М.И. Германенко, В.В. Горбик.* Параллельные алгоритмы решения систем линейных алгебраических уравнений с применением вычислений без округления // Параллельные вычислительные технологии (ПАВТ–2007): Труды международной научной конференции (Челябинск, 29 января – 2 февраля 2007 г.). – Челябинск: Изд-во ЮУрГУ, 2007, т. 2, с. 238–249.

11. *А.В. Панюков, М.И. Германенко* Приложение для безошибочного нахождения обобщенной обратной матрицы методом Мура-Пенроуза и безошибочное решение систем линейных алгебраических уравнений // Информационные технологии моделирования и управления. – 2009. – No 1 (53). – С. 78–87.

12. Netlib library collection: ftp://netlib2.cs.utk.edu/lp/data, 1996.

Параллельная реализация алгоритма обучения системы текстовой классификации

Т.А. Пескишева, Е.В. Котельников

Вятский государственный гуманитарный университет

В статье рассматривается алгоритм параллельного обучения системы текстовой классификации на основе метода опорных векторов (SVM) и стратегии многоклассовой классификации «один против всех». Предложено три метода распределения нагрузки между узлами вычислительного кластера. Для каждого из методов приводятся результаты экспериментов на текстовой коллекции Reuters-21578 и характеристики производительности.

1. Введение

Текстовая классификация (рубрикация) – отнесение текстовых документов к одной или нескольким заранее заданным категориям (рубрикам) в соответствии с определенными признаками. В условиях экспоненциального роста объема накапливаемой и используемой человечеством текстовой информации, хранящейся в электронном виде, задача тематической текстовой рубрикации становится все более актуальной, но вместе с тем сложной и трудоемкой.

На сегодняшний день для решения данной задачи используется множество различных программных продуктов. Существующие системы и модули текстовой классификации демонстрируют приемлемую скорость и точность обработки лишь на небольших и средних по объему данных. В случае значительного роста объема обрабатываемой информации, а также увеличения числа рубрик, по которым необходимо классифицировать документы, их производительность существенно снижается [1].

Решением данной проблемы может быть применение высокопроизводительных методов машинного обучения в реализации системы текстовой классификации. Одним из них является метод опорных векторов (Support Vector Machines, SVM), предложенный В. Н. Вапником [2].

Метод опорных векторов был разработан для решения задач распознавания образов двух классов. Применение SVM сводится к двум основным процессам: обучению и классификации (или распознаванию). Наиболее трудоемким является процесс обучения, который происходит на основе множества векторов. Каждый вектор представляет собой набор признаков, характеризующих распознаваемый объект. В задаче тематической текстовой классификации элементами векторов будут веса слов, входящих в определенный текстовый документ [3]. Вес дает числовую оценку значимости данного слова для определения тематики текста. Каждому вектору сопоставлено число, обозначающее класс, к которому относится вектор.

В процессе обучения в n-мерном пространстве признаков строится линейная или нелинейная гиперплоскость, разделяющая векторы разных классов. Обучение сводится к решению задачи квадратичной оптимизации с предельными ограничивающими условиями и одним ограничением линейного равенства. При этом SVM выделяет так называемые *опорные векторы* – такие векторы, которые находятся ближе всего к разделяющей гиперплоскости. Только опорные векторы несут всю информацию о разделении классов, так что остальные векторы могут в дальнейшем не учитываться.

Особенностями задачи текстовой классификации является, во-первых, очень высокая размерность пространства признаков – десятки, иногда сотни тысяч, во-вторых, большое количество классов (рубрик) – от нескольких десятков до сотен.

Проблема использования SVM на реальных коллекциях текстов связана как с указанными особенностями задачи текстовой классификации, так и с тем, что методы решения задачи квадратичной оптимизации известны, но вычислительно сложны.

В результате время обучения рубрикатора оказывается неприемлемо длительным. Повышение скорости обучения на основе SVM возможно за счет использования многопроцессорных вычислительных систем и комплексов.

Цель данной работы – описать параллельный алгоритм обучения рубрикатора системы многоклассовой текстовой классификации на основе метода опорных векторов и проанализировать эффективность методов распределения нагрузки между узлами вычислительного кластера.

Статья построена следующим образом. Раздел 2 посвящен общим подходам к решению задачи многоклассовой классификации в SVM. В разделе 3 предлагаются три метода распределения нагрузки между узлами кластерной системы при параллельном обучении SVM. В разделе 4 рассматриваются программные и аппаратные аспекты проведенных экспериментов, а также описывается коллекция обучающих текстов Reuters-21578. В разделе 5 приводятся и анализируются результаты экспериментов, а также вычисляются характеристики производительности параллельных методов. Раздел 6 является заключительным и содержит общие выводы по работе и рекомендации.

2. Стратегии многоклассовой классификации в SVM

Существуют два основных подхода к решению проблемы многоклассовой классификации в SVM. Первый называется *«решение за один шаг»* или *«все вместе» («all-together»)* [4]. В этом подходе задача квадратичной оптимизации усложняется за счет увеличения размерности дополнительных переменных.

В другом подходе решение задачи многоклассовой классификации сводится к решению последовательности задач с двумя классами. При этом может быть несколько стратегий генерации такой последовательности: «один против всех» («one-against-all»), «каждый против каждого» («one-against-one»), «турнир на выбывание» или «ориентированный ациклический граф SVM» (Directed Acyclic Graph SVM, DAGSVM).

В стратегии *«один против всех»* [5] для N классов обучается N классификаторов, каждый из которых отделяет «свой» класс от всех остальных классов. На этапе распознавания неизвестный вектор X подается на все N классификаторов. Принадлежность вектора X определяется тем классификатором, который выдал наибольшую оценку $f(X)$.

Стратегия *«каждый против каждого»* [6] выделяет $N(N–1)/2$ классификаторов, обучающихся отличать все возможные пары классов друг от друга. Для распознаваемого вектора каждый классификатор выдает оценку $f_{ij}(X)$, отражающую принадлежность к классам i и j. Результатом является класс с максимальной суммой

$$\sum_{i \neq j} g\left(f_{ij}(X)\right), \tag{1}$$

где g – монотонно неубывающая функция, например тождественная или логистическая.

Стратегия *«турнир на выбывание»* [7] также предполагает обучение $N(N–1)/2$ классификаторов, различающих все возможные пары классов. В отличие от предыдущей стратегии, на этапе классификации вектора X между классами устраивается турнир. При этом на каждом шаге распознавание вектора X осуществляет только один классификатор – «победивший» класс продолжает борьбу и определяет следующий используемый классификатор. Процесс осуществляется до тех пор, пока не останется один победивший класс, который и будет считаться результатом распознавания.

Методы сведения многоклассовой классификации к бинарной обучаются быстрее и дают меньшее число ошибок, в то время как в подходе «решение за один шаг» получается меньшее число опорных векторов.

В [8] подробно описаны параллельные реализации стратегий сведения многоклассовой классификации к бинарной, а также результаты экспериментов с ними. Однако, несмотря на то, что все предложенные способы распараллеливания продемонстрировали свою эффективность, для дальнейшего использования в системе текстовой классификации была выбрана стратегия «один против всех». Выбор именно этой стратегии связан с тем, что она позволяет относить каждый классифицируемый объект сразу к нескольким классам. В задачах текстовой класси-

фикации это существенно, так как часто возникают ситуации, когда документ может быть отнесен сразу к нескольким темам.

3. Подходы к распределению нагрузки

В данном разделе предлагается три метода распределения нагрузки на узлы кластера при параллельном обучении классификатора на основе метода опорных векторов. Во всех подходах используется стратегия сведения многоклассовой классификации к бинарной «один против всех».

Все предлагаемые подходы используют принцип «master-slave». Данный принцип состоит в следующем. Главный узел («master») передает подчиненным узлам («slave») задания, которые они должны выполнить. Подчиненные узлы независимо друг от друга выполняют свои задания, затем полученные результаты возвращают главному узлу.

Методы распределения нагрузки разделены на две группы в зависимости от того, в какой момент времени принимается решение о передаче обучающих векторов на узлы: заранее для примеров всех классов на основе имеющейся информации о векторах (статическое распределение) или в процессе обучения, для каждого класса отдельно (динамическое распределение).

3.1. Статическое распределение на основе примеров

В этом подходе главный узел распределяет примеры всех классов по узлам предварительно, до стадии обучения. Распределение осуществляется на основе известной информации относительно примеров.

Для равномерной загрузки вычислительных узлов желательно иметь информацию о времени обучения для каждого класса. Очевидно, до завершения процесса обучения эти данные недоступны, поэтому в первом методе (**статическое распределение на основе примеров**) для распределения нагрузки используется информация о количестве обучающих примеров в каждом классе и «жадный алгоритм». В общем случае работа «жадного» алгоритм («Greedy algorithm») заключается в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. На практике при решении задачи сбалансированного распределения нагрузки «жадный» алгоритм почти всегда дает решение в достаточной степени близкое к оптимальному.

В предлагаемом подходе жадный алгоритм применяется следующим образом. Сначала главный узел сортирует классы по убыванию количества примеров. Каждому подчиненному узлу выделяется по одному классу в порядке невозрастания количества примеров. После того, как были распределены первые N классов, остальные классы распределяются следующим образом. Очередной класс выделяется тому узлу, у которого общее количество всех выделенных ему примеров минимально (независимо от того, сколько классов уже выделено данному узлу).

При этом для каждого подчиненного узла формируется и отправляется собственный список рубрик, подлежащих обработке. На основе этого списка отбирается и пересылается набор обучающих векторов для каждого подчиненного узла. Узел, получив эти данные, строит правило классификации (модель) для каждой переданной ему рубрики. Завершив обучение, узел передает построенные модели на главный узел. Работа кластера завершается, когда последний узел закончит вычисления и передаст результаты главному узлу.

Данный подход показан на рис. 1, операции, выполняемые только главным узлом, обозначены буквой «M», а подчиненными узлами – буквой «S».

В рассматриваемом методе реальная загрузка узлов оказывается иногда далека от оптимальной по той причине, что время обучения SVM зависит не только от количества обучающих векторов, но и от их взаимного расположения в n-мерном пространстве. Для обеспечения большей сбалансированности требуется знать время обучения (или количество опорных векторов) для каждого класса.

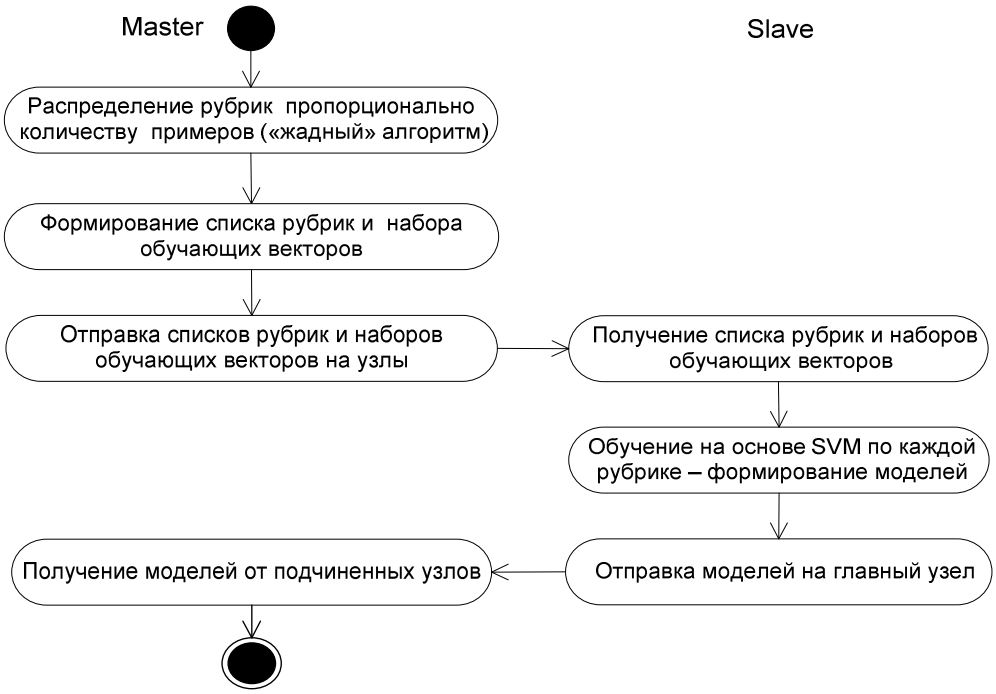


Рис. 1. Выполнение параллельного алгоритма при статическом распределении на основе примеров

3.2. Статическое распределение на основе опорных векторов

В некоторых ситуациях может требоваться многократное обучение классификатора на основе SVM на одинаковых данных, например, при поиске оптимальных параметров. В этом случае время обучения (и опорные векторы) становится известно после выполнения первого прохода и для остальных проходов эта информация может использоваться для распределения рубрик по вычислительным узлам. Предлагается учитывать не время обучения, поскольку оно зависит от аппаратных характеристик узлов, состояния сети и других случайных факторов, а количество опорных векторов, которое является объективной характеристикой набора обучающих векторов для данного алгоритма и параметров обучения. Время обучения классификатора находится в прямой зависимости от количества опорных векторов. Чем больше опорных векторов в классе, тем больше времени потребуется для построения классификатора.

Во втором методе (**статическое распределение на основе опорных векторов**) главный процесс также выполняет предварительное распределение классов по узлам на основе «жадного» алгоритма. Отличие от первого метода в том, что классы сортируются не по количеству примеров, а по количеству опорных векторов. В результате получается более сбалансированная нагрузка на узлы и уменьшается время простоя узлов.

3.3. Динамическое распределение

Основным недостатком первого метода является отсутствие в реальных задачах достаточной сбалансированности нагрузки на узлы вычислительной системы: некоторые узлы намного раньше завершают вычисления и простаивают, как результат – неоптимальное использование ресурсов кластера. Во втором методе этот недостаток отсутствует, но область его применения ограничена задачами поиска оптимальных параметров, в общем случае метод не применим.

Идея третьего метода (**динамическое распределение**) заключается в том, что узлы загружаются по мере их освобождения (динамически). Схема данного метода показана на рис. 2.

Прежде всего, главный узел отправляет все обучающие векторы на каждый подчиненный узел и формирует список свободных узлов. Подчиненные узлы сообщают о своей готовности и в ответ получают информацию о том, какую часть набора обучающих данных должен обработать каждый из них. По окончании вычислений подчиненный узел пересылает результат главному узлу и ждет сообщение с указаниями о необходимости обработки очередной части набора

данных, либо команду завершения. Главный узел выделяет подчиненному узлу следующую порцию обучающих векторов.

Данный подход не использует априорную информацию о количестве обучающих примеров или опорных векторов, т.е. является универсальным, однако при этом увеличивается количество обменов информацией между главным и подчиненными узлами кластера и время ожидания узлов по сравнению со статическими методами, что в некоторой степени уменьшает эффективность алгоритма.

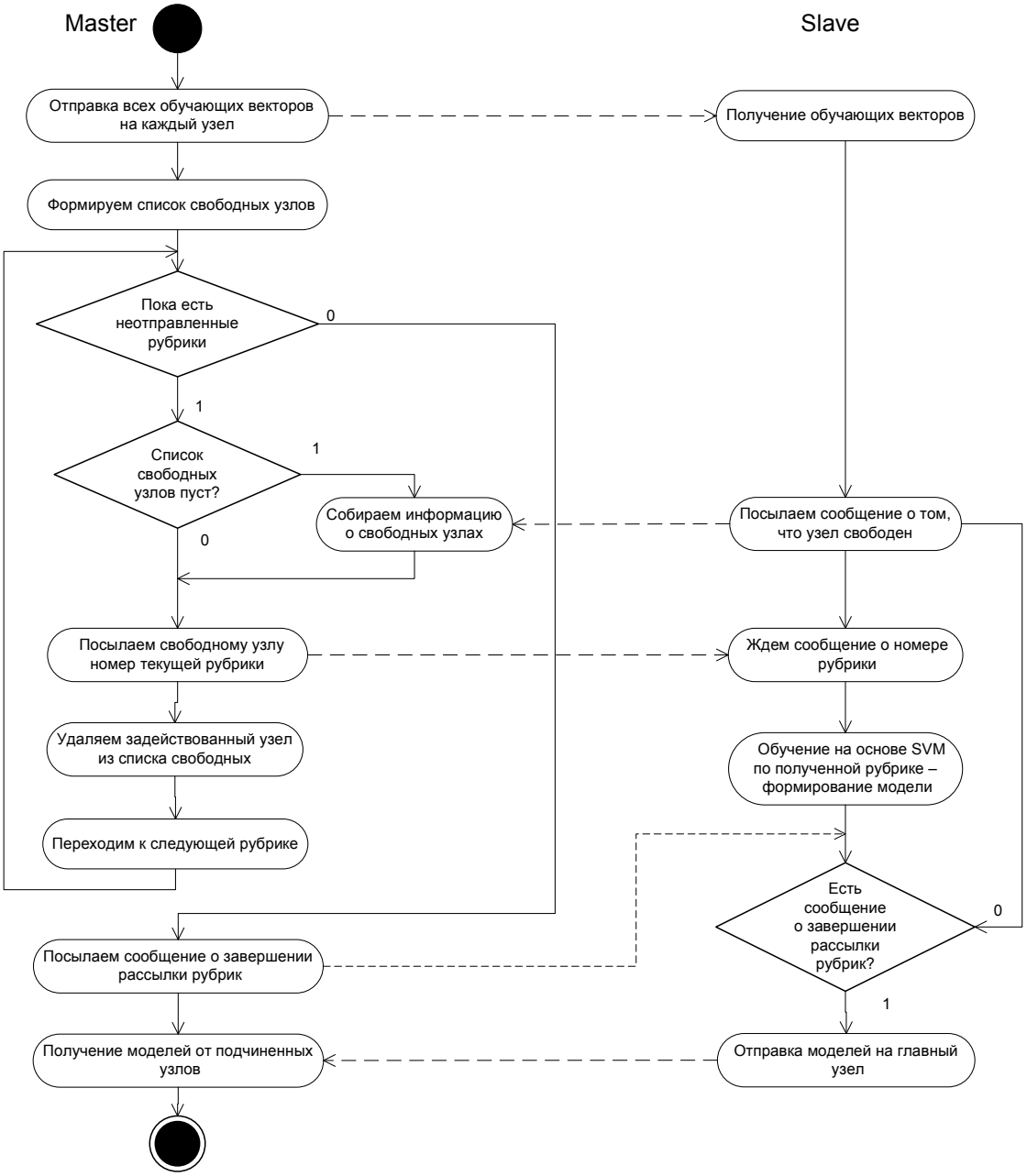


Рис. 2. Выполнение параллельного алгоритма при динамическом распределении нагрузки

4. Условия проведения экспериментов

Для тестирования и анализа предложенных методов распределения нагрузки в параллельном алгоритме обучения многоклассовой классификации было разработано программное при-

ложение для кластерной архитектуры. Приложение создавалось в среде Microsoft Visual Studio 2008 на языке C# на основе принципов объектно-ориентированного программирования.

Для кластерной архитектуры параллельная реализация алгоритмов обучения проводилась с использованием библиотеки MPI.NET версии 1.0 [9]. Библиотека MPI.NET представляет собой свободно доступную реализацию интерфейса передачи сообщений MPI для среды Microsoft.NET и позволяет разрабатывать приложения для MPI на C# и других языках .NET.

Расчеты проводились на вычислительном кластере Вятского государственного гуманитарного университета, состоящем из 30 вычислительных узлов. Каждый вычислительный узел представляет собой персональный компьютер с процессором Intel Core 2 Duo 2 ГГц и 2 Гб оперативной памяти. Узлы связаны сетью Gigabit Ethernet. Кластер функционирует на базе операционной системы Microsoft Windows HPC Server 2008, на каждом узле установлена среда выполнения MPI.NET Runtime версии 1.0.

Для экспериментального исследования использовалась коллекция финансовых новостей агентства Reuters (Reuters-21578, Distribution 1.0) [10]. Обучающий набор документов был выделен в соответствии с общепринятым подходом, названным «ModApte split» [10]. При этом исключались документы, не помеченные ни одной рубрикой. Таким образом, обучающий набор в наших экспериментах представлен 7775 документами, каждый из которых относится к одной или нескольким из 115 рубрик.

Для формирования векторов использовался морфологический анализатор Mystem 1.0 от компании Yandex [11], терминами считались все слова в нормальной форме, исключая стоп-слова. Для взвешивания терминов и получения числовых значений компонентов векторов применялся метод TF.IDF [12]. В результате получены 7775 обучающих векторов, размерность которых составляет 6103.

5. Результаты экспериментов

В ходе экспериментов тестировались три предложенных метода распределения нагрузки между узлами вычислительного кластера. Количество узлов для каждого метода менялось от 3 до 29 (главный узел здесь учитывается, хотя непосредственно для обучения SVM не используется, а играет роль диспетчера), проводилось 5 запусков одного эксперимента с фиксированными параметрами, затем результаты усреднялись.

Для вычисления характеристик производительности параллельных алгоритмов было измерено время выполнения последовательной версии решения данной задачи на одном узле кластера. Полученное значение составляет 68,7 с.

Результаты экспериментов приведены на рис. 3.

Вычислим основные характеристики производительности – ускорение и эффективность.

Ускорение вычисляется как отношение времени решения задачи на одном вычислительном узле T_s ко времени решения на p идентичных вычислительных узлах T_p:

$$S = \frac{T_s}{T_p}. \tag{2}$$

Эффективность отражает долю времени выполнения алгоритма, в течение которой вычислительные узлы были реально задействованы для решения задачи:

$$E = \frac{S}{p}. \tag{3}$$

Значения ускорения и эффективности приведены на рис. 4, 5.

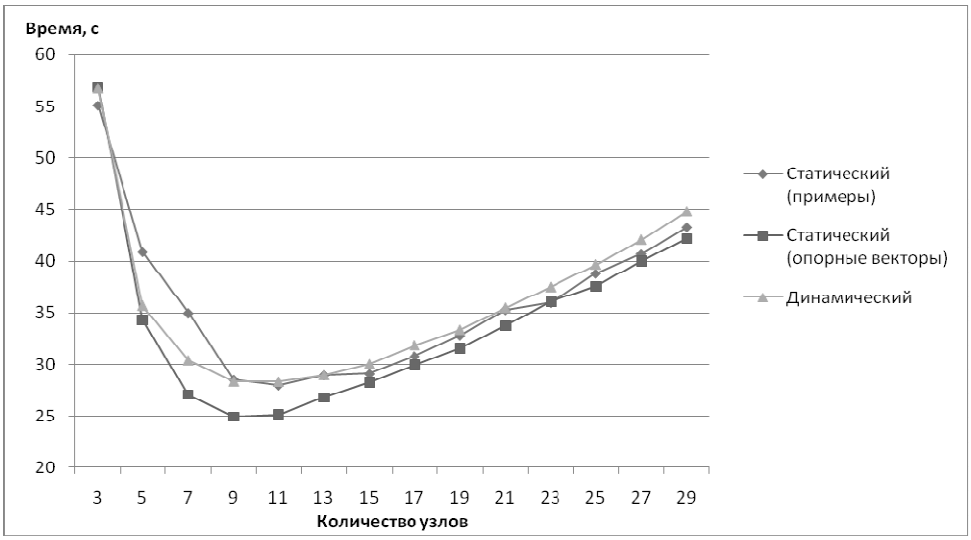


Рис. 3. Результаты экспериментов для трех методов

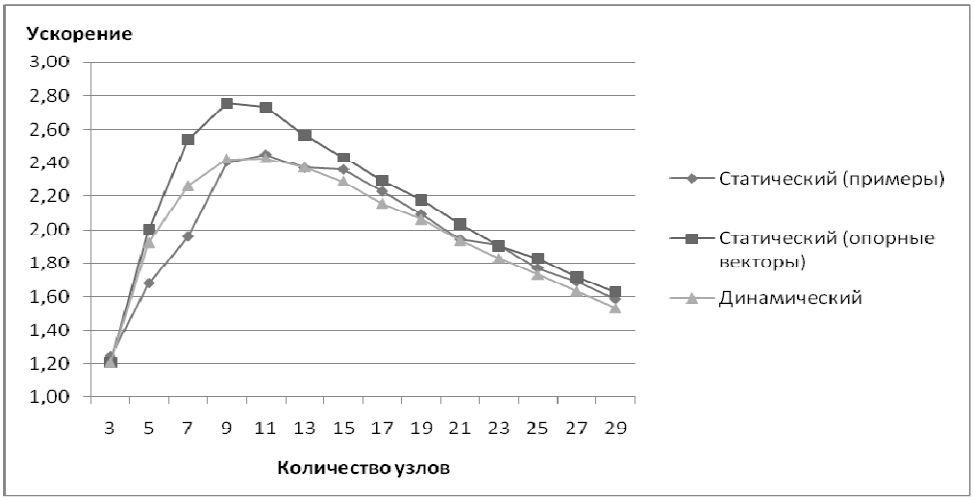


Рис. 4. Зависимость ускорения от количества узлов

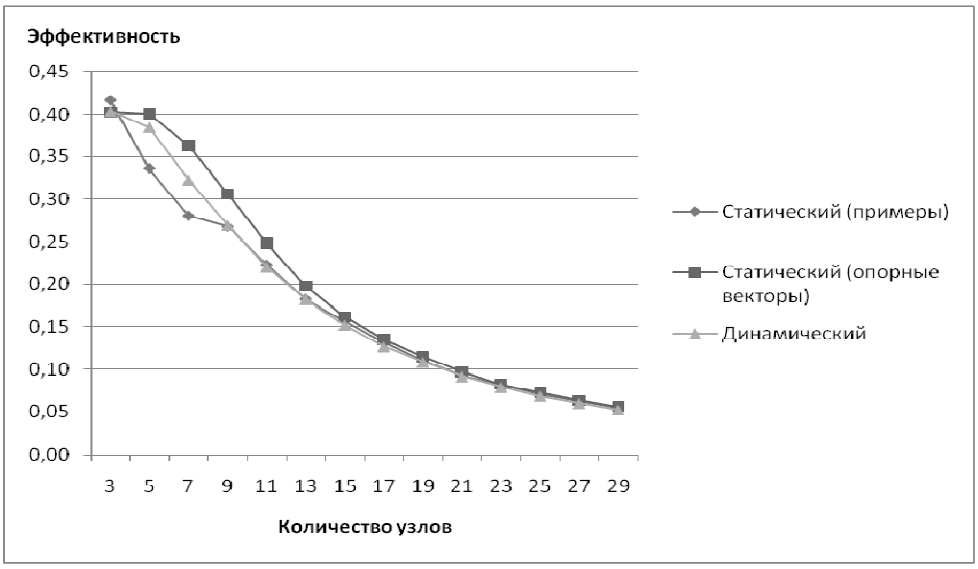


Рис. 5. Зависимость эффективности от количества узлов

По результатам экспериментов можно сделать следующие выводы.

1. Максимальное ускорение во всех методах достигается при количестве узлов 9–11, с увеличением числа узлов наблюдается ухудшение. Связано это с тем, что при количестве узлов больше 11 (для данной коллекции) время, затраченное на ожидание и пересылку информации о текущих рубриках, оказывается больше, чем эффект за счет распараллеливания. Для других обучающих коллекций большего размера точка минимума будет сдвигаться в сторону увеличения количества узлов.

2. При увеличении числа узлов эффективность падает в связи с увеличением времени ожидания каждым узлом списка рубрик (при высокой латентности коммуникационной сети).

3. Наибольшее значение ускорения (2,75 для 9 узлов), как и наибольшие средние значения ускорения и эффективности достигаются в методе статического распределения на основе опорных векторов. В то же время метод статического распределения на основе примеров почти всегда показывает худшие результаты. Такие результаты получаются вследствие того, что первый метод использует априорную информацию о количестве опорных векторов и, следовательно, о времени обучения для каждой рубрики. В то же время количество примеров для данной рубрики не всегда соответствует времени обучения.

Например, в табл. 1 приведены данные по нескольким рубрикам из рассматриваемой коллекции Reuters-21578.

Таблица 1. Данные по десяти рубрикам коллекции Reuters-21578 с наибольшим количеством примеров

Условные номера рубрик	Количество примеров в рубрике	Время обучении для рубрики, с	Количество опорных векторов
31	2877	8,53	766
1	1650	9,79	857
62	538	6,77	520
38	433	4,96	386
25	389	5,42	414
112	369	6,13	438
46	347	5,40	436
114	212	2,89	211
96	197	4,89	387
17	181	4,12	331

Из табл. 1 видно, что количество примеров в рубрике слабо связано с временем обучения, при этом время обучения сильно коррелирует с количеством опорных векторов.

Таким образом, в методе статического распределения на основе опорных векторов сбалансированность распределения нагрузки по узлам оказывается в среднем выше, чем у остальных двух методов, за счет априорной информации.

6. Заключение

Таким образом, можно сделать вывод, что предлагаемые методы распределения нагрузки между узлами вычислительного кластера можно применять для повышения эффективности решения задачи текстовой классификации. При этом для коллекции Reuters-21578 достигается максимальное ускорение 2,75, максимальная эффективность 0,42.

По результатам экспериментов можно рекомендовать использование для обучающих коллекций подобного уровня 9–11 узлов вычислительного кластера, дальнейшее увеличение ведет к падению ускорения.

В случае, когда априорная информация о количестве опорных векторов (времени обучения) для каждой рубрики неизвестна, могут применяться методы статического распределения нагрузки на основе примеров и динамического распределения, причем последнему следует отдавать предпочтение. При решении задачи поиска оптимальных параметров классификатора на первом проходе рекомендуется использовать метод динамического распределения, а при по-

следующих – метод статического распределения на основе опорных векторов, причем можно корректировать распределение рубрик на каждом шаге в зависимости от результатов (количества опорных векторов для рубрик) предыдущего прохода.

Литература

1. Пескишева Т.А. Современные системы и модули автоматической рубрикации текстовых документов: Вятский государственный гуманитарный университет. – Киров, 2010. – 37 с.: – Библиогр. 30 назв. – Рус. – Деп. в ВИНИТИ 01.07.2010, №410 – В 2010.
2. Vapnik V. Statistical learning theory. Wiley, New York, 1998.
3. Sebastiani F. Machine Learning in Automated Text Categorization. ACM Computing Surveys, Vol. 34, No. 1, March 2002, pp. 1–47.
4. Weston J., Watkins C. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.
5. Bottou L., Cortes C., Denker J., Drucker H., Guyon I., Jackel L., Le Cun Y., Muller U., Sackinger E., Simard P., Vapnik V. Comparison of classifier methods: a case study in handwriting digit recognition. In International Conference on Pattern Recognition, pp. 77–87. IEEE Computer Press, 1994.
6. Krebel B. Pairwise classification and support vector machines. In B. Schölkopf, C. J. C. Burges, A. J. Smola, editors. Advances in Kernel Methods – Support Vector Learning, pp. 255–268, Cambridge, MA, 1999. MIT Press.
7. Platt J., Cristianini N., Shawe-Taylor J. Large Margin DAGS for Multiclass Classification. In Advances in Neural Information Processing Systems, 12 ed. S. A. Solla, T. K. Leen and K.-R. Muller, MIT Press, 2000. Pp. 547–553.
8. Котельников Е. В., Стародубова Т. А. Параллельные алгоритмы многоклассовой классификации на основе метода опорных векторов // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы восьмой Международной конференции-семинара. – Казань: КГТУ, 2008, с. 228-233.
9. Project MPI.NET // The Open Systems Lab, Indiana University.

   URL: http://www.osl.iu.edu/research/mpi.net (дата обращения: 01.02.2011).
10. Reuters-21578, Distribution 1.0.

    URL: http://www.daviddlewis.com/resources/testcollections/reuters21578 (дата обращения: 01.02.2011).
11. Морфологический анализатор Mystem от компании Yandex.
    URL: http://company.yandex.ru/technology/mystem (дата обращения: 01.02.2011).
12. Salton G. Term-weighting approaches in automatic text retrieval // Information Processing & Management. – [s.l.] : Pergamon Press, 1988. – 5 : Vol. 24. – pp. 513-523.

Сервисно-ориентированный подход к использованию систем инженерного проектирования и анализа в распределенных вычислительных средах*

Г.И. Радченко

Южно-Уральский государственный университет

Рациональной альтернативой созданию собственного суперкомпьютерного центра для решения сложных задач инженерного моделирования является аренда вычислительных и программных ресурсов в режиме удаленного доступа у центров коллективного пользования, функционирующих при крупных университетах, академических институтах и других организациях. Однако при этом возникает целый комплекс проблем, связанных с организацией прозрачного и безопасного доступа к таким ресурсам. В статье предложено описание технологии CAEBeans, обеспечивающей автоматизированную генерацию проблемно-ориентированных грид-сервисов, позволяющих использовать программные системы для инженерного проектирования и анализа в распределенных вычислительных средах.

1. Введение

Системы компьютерного проектирования (CAE - Computer Aided Engineering), ориентированные на разработку сложных технологических процессов, конструкций, и материалов, являются сегодня одним из ключевых факторов обеспечения конкурентоспособности любого высокотехнологического производства. Использование таких систем дает возможность проводить *виртуальные* эксперименты, которые в реальности выполнить затруднительно или невозможно. Это позволяет значительно повысить точность анализа вариантов проектных решений и в десятки раз сократить путь от генерации идеи до ее воплощения в реальном промышленном производстве [18].

Точность результатов компьютерного моделирования во многом зависит от степени детализации сеток, используемых для проведения вычислительных экспериментов. На сегодняшний день, постоянно возрастает вычислительная сложность задач инженерного проектирования, и требуются значительные вычислительные ресурсы для выполнения инженерного моделирования [2]. Решение этой проблемы заключается в использовании многопроцессорных систем.

Процесс решения задач инженерного проектирования с использованием суперкомпьютерных ресурсов для рядового пользователя сопряжен с определенными трудностями. С одной стороны, от него требуется наличие специфических знаний, умений и навыков в области высокопроизводительных вычислений. С другой стороны, для решения задач инженерного проектирования пользователю требуется изучить интерфейс и особенности работы всех программных компонентов, входящих в технологический цикл решения задачи [12]. Все эти факторы затрудняют широкое внедрение систем компьютерного инженерного проектирования в практику НИОКР.

Рациональной альтернативой созданию собственного суперкомпьютерного центра является аренда вычислительных и программных ресурсов в режиме удаленного доступа у центров коллективного пользования, функционирующих при крупных университетах, академических институтах и других организациях. Однако при этом возникает целый комплекс проблем, связанных с обеспечением безопасности вычислительных систем и данных. Указанный комплекс проблем можно решить посредством применения концепции грид вычислений (Grid Computing) [14] и родственной ей концепции облачных вычислений (Cloud Computing) [15] в соответствии с которыми, пользователю предоставляется конечный проблемно-

* Проект выполнен при поддержке Совета по грантам Президента Российской Федерации (номер проекта МК-1987.2011.9) и Российского фонда фундаментальных исследований (проект № 11-07-00478-а).

ориентированный сервис, обеспечивающий решение задач на базе ресурсов распределенных вычислительных систем.

Подход к сервисно-ориентированному предоставлению прикладных программных пакетов в виде ресурсов распределенных вычислительных сред переживает бум в последние несколько лет [1, 6]. Но данные системы не ориентированы на создание виртуальных испытательных стендов, обеспечивающих проблемно-ориентированную постановку и решение задач инженерного моделирования профессиональными инженерами.

В работах [13, 19] раскрываются аспекты разработки системы совместного проектирования, ориентированной на решение задач инженерного проектирования в динамических распределенных группах разработчиков. В связи с потребностью постоянного взаимодействия между участниками команды инженеров в процессе разработки, в качестве базовой платформы данной системы была выбрана концепция P2P-вычислений. Но применение такого подхода ориентировано не на предоставление конечному пользователю готового продукта, предоставляющего ресурсы распределенной сети, а на поддержку процесса совместного проектирования командой разработчиков.

В работах [16, 17] представлена концепция грид-среды для совместного проектирования (Collaborative Manufactoring Grid) система совместной разработки, основанная на стандартах грид-систем. Реализована возможность решения задач инженерного моделирования посредством указания параметров моделирования и получения конечных результатов через веб-портал. В качестве недостатков этой системы можно отметить отсутствие стандартизированного подхода к созданию виртуальных испытательных стендов; отсутствие возможности конкретизации проблемно-ориентированных оболочек, обеспечивающих «подгонку» задач к конкретным требованиям пользователя; отсутствие реализации стандарта WSRF. Также, отсутствует брокер ресурсов, обеспечивающий прозрачное для пользователя предоставление ресурсов распределенной вычислительной среды, наиболее соответствующих требованиям текущей задачи.

2. Технология CAEBeans

2.1 Основные понятия

Технологии CAEBeans представляет собой комплекс моделей, методов и алгоритмов, направленных на автоматизированное создание иерархий распределенных проблемно-ориентированных оболочек над инженерными (CAE) пакетами на основе сервисно-ориентированного подхода и концепции облачных вычислений.

Задача инженерного моделирования включает в себя:

- геометрическую модель объекта исследования и (или) вычислительную сетку, разбивающую моделируемую область на дискретные подобласти;
- граничные условия, физические характеристики и параметры взаимодействия отдельных компонентов исследуемой области;
- описание неизвестных величин, значения которых требуется получить в результате решения задачи;
- требования к аппаратным и программным ресурсам, обеспечивающим процесс моделирования.

Параметром задачи инженерного моделирования назовем величину, значение которой влияет на конечный результат моделирования и может варьироваться в определенном диапазоне значений. Каждый параметр задачи инженерного моделирования имеет определенную семантику, отражая свойство проблемной области, либо некоторую особенность процесса решения задачи [9].

Полным дескриптором задачи назовем множество параметров, однозначно описывающих задачу инженерного моделирования.

Логическим планом решения задачи инженерного моделирования назовем ориентированный граф, в вершинах которого могут находиться узлы двух типов [11]:

- *действия*, выполняемые отдельными инженерными пакетами;
- *узлы управления* потоком решения задачи.

Определим *класс задач инженерного моделирования* как множество задач, у которых совпадает структура полного дескриптора задачи и для которых можно описать единый логический план, приводящий к решению.

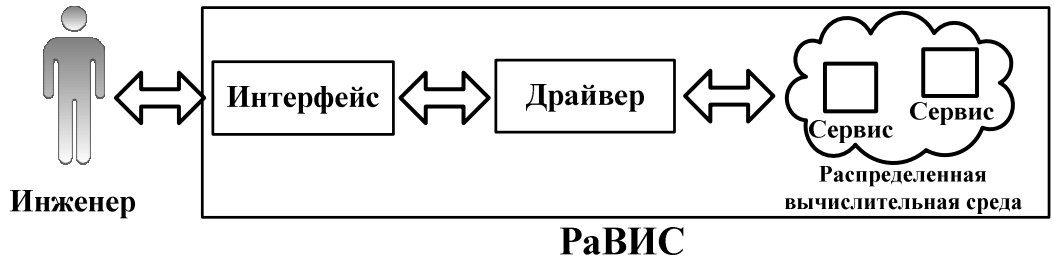


Рис. 1. Обобщенная схема распределенного виртуального испытательного стенда.

Внедрение ресурсов CAE-пакетов в распределенные вычислительные среды основывается на концепции распределенного виртуального испытательного стенда. *Распределенный виртуальный испытательный стенд (РаВИС)* – это программно-аппаратный комплекс, обеспечивающий проведение работ инженерного моделирования в распределенной вычислительной среде в рамках определенного класса задач.

РаВИС включает в себя (см. рис. 1):

- *интерфейс,* обеспечивающий постановку определенного класса задач инженерного моделирования;
- *драйвер:* набор программных средств, обеспечивающих использование сервисов распределенной вычислительной среды для проведения виртуального эксперимента;
- *сервисы распределенной вычислительной среды:* множество вычислительных систем, входящих в распределенную вычислительную среду, в совокупности с установленными на них программными компонентами, обеспечивающими решение задач инженерного моделирования и поддерживающими безопасные стандартизованные методы удаленного взаимодействия.

Процессы разработки и функционирования РаВИС определяются технологией CAEBeans. *Технология CAEBeans* – это совокупность теории и практической техники, на которые опирается процесс создания и использования распределенных виртуальных испытательных стендов. Технология CAEBeans включает в себя:

1) концептуальные средства, которые определяют методы разработки и структуру РаВИС;
2) организационные средства, которые определяют форму труда и распределение обязанностей в команде разработчиков и пользователей РаВИС;
3) программные средства разработки и среду исполнения РаВИС.

2.2 Архитектура CAEBeans

Оболочка CAEBean – это основная структурная единица, формирующая РаВИС. В соответствии с технологией CAEBeans выделяются четыре слоя структуры РаВИС, каждый из которых представляется своим типом оболочек CAEBeans (см. рис. 2):

1) концептуальный слой (проблемный CAEBean);
2) логический слой (потоковый CAEBean);
3) физический слой (компонентный CAEBean);
4) системный слой (системный CAEBean).

Концептуальный слой РаВИС формируется на основе оболочек CAEBeans, которые мы будем называть *проблемными*. Пользовательский интерфейс, предоставляемый проблемным CAEBean, является основным средством взаимодействия пользователя с системой CAEBeans. Посредством проблемного CAEBean, ориентированного на решение конкретного класса задач инженерного моделирования, пользователь может произвести постановку задачи; проследить за ходом решения поставленной задачи; получить результаты решения. Технология CAEBeans предусматривает возможность формирования *дерева проблемных CAEBeans*. Она позволяет адаптировать проблемно-ориентированный интерфейс конкретного класса задач инженерного моделирования под нужды определенной категории пользователей. Дочерние проблемные оболочки могут конкретизировать класс задач, решаемых родительской оболочкой, путем выделе-

ния и инкапсуляции групп проблемных параметров, значения которых определяются на данном уровне абстракции.

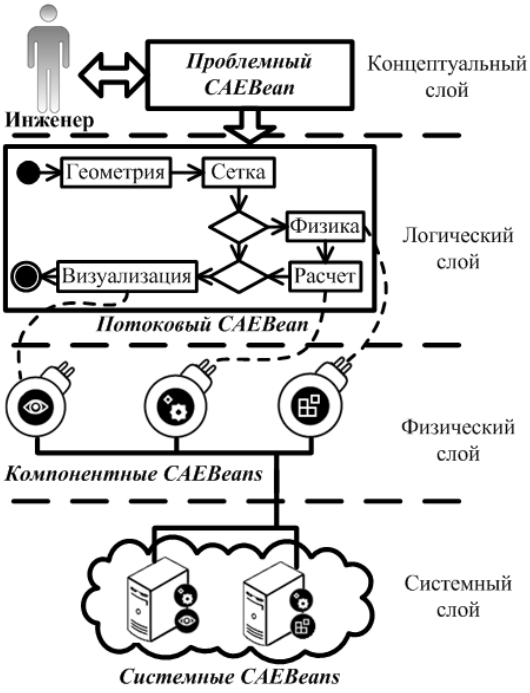


Рис. 2. Обобщенная схема слоев РаВИС.

Логический слой РаВИС представлен *потоковым CAEBean*, реализующим логический план решения определенного класса задач компьютерного моделирования (см рис. 3). Для формирования логического плана используются элементы нотации *диаграммы деятельности* стандарта UML 2.0. *Поток управления* – это ребро, задающее протекание управления, но не данных. Информация, необходимая для исполнения любого узла логического плана содержится в полном дескрипторе задачи. В ходе решения задачи, любой узел может обратиться к дескриптору для получения значений входных параметров и сохранения результатов в соответствующих выходных параметрах.

Процесс выполнения деятельности можно представить себе как обход экземпляра графа логического плана, содержащего маркеры управления. Маркер управления указывает на наличие управления на некотором этапе процесса вычисления, представленным узлом или потоком. Маркеры управления не имеют внутренней структуры. Выполнение действия разрешается, если во всех его входных потоках присутствуют маркеры. Некоторые виды узлов управления могут начинать работу в момент, когда на заданном подмножестве входных ребер появятся маркеры управления. Когда работа узла завершается, в каждом из его исходящих потоков управления появляется маркер управления.

Узел действия логического плана реализует определенное действие инженерного моделирования. При исполнении узла действия, потоковый CAEBean обеспечивает взаимодействие с дескриптором задачи: получение значений входных параметров для инициации работы действия и запись значений выходных параметров, вычисленных в результате исполнения действия. Действие реализуется соответствующим компонентным CAEBean, находящимся на физическом слое РаВИС.

Компонентные CAEBeans, представляющие физический слой РаВИС, отвечают за процесс постановки и решения отдельных действий инженерного моделирования средствами конкретных инженерных пакетов. Основная функция компонентного CAEBean – преобразование проблемно-ориентированного описания действия инженерного моделирования в *компонентно-ориентированную форму*. По окончании процесса решения, компонентный CAEBean обеспечивает преобразование компонентно-ориентированных результатов решения задачи в проблемно-ориентированные. Компонентный CAEBean задает абстрактное действие, которое не может быть реализовано само по себе, так как не обеспечено реальными программными, аппаратными

и лицензионными ресурсами. В процессе исполнения, каждому компонентному CAEBean должен быть найден и предоставлен физический ресурс, отвечающий всем требованиям, необходимым для реализации специфицированной в нем деятельности. В рамках системы CAEBeans, физические ресурсы реализуются посредством системных CAEBeans.

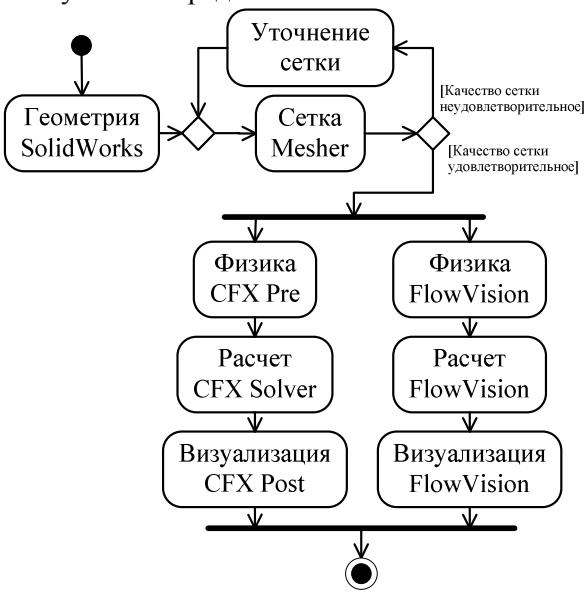


Рис. 3. Пример логического плана решения задачи инженерного моделирования.

Системным CAEBean называется оболочка системного слоя РаВИС, предоставляющая функциональные возможности физического ресурса в распределенной вычислительной среде и обеспечивающая сервисно ориентированный подход к постановке задач и получению результатов. Системный CAEBean обеспечивает изолированное рабочее пространство для исполнения каждого действия инженерного моделирования; предоставляет программный интерфейс для загрузки исходных данных, удаленного запуска действия инженерного моделирования и передачи результатов решения.

2.3 Разработка и использование РаВИС

Программные средства CAEBeans можно разбить на два типа: средства исполнения и средства разработки РаВИС. Разработка проблемно-ориентированной части распределенного виртуального испытательного стенда ведется в системе *Конструктор*. Посредством Конструктора прикладной программист формирует оболочки концептуального, логического и физического слоя РаВИС, отвечающие за решение конкретного класса задач инженерного моделирования.

В соответствии с технологией CAEBeans работа РаВИС осуществляется с помощью следующих четырех программных систем:

1) клиент;
2) сервер;
3) брокер ресурсов;
4) набор целевых систем.

Клиент предоставляет унифицированный интерфейс конкретного РаВИС. *Сервер* отвечает за хранение и исполнение РаВИС. *Брокер ресурсов* – это автоматизированная система регистрации, анализа и предоставления ресурсов распределенной вычислительной среды. *Целевая система* – это совокупность грид-сервисов, которые имеют доступ к пространству программных, аппаратных и лицензионных ресурсов некоторого узла грид, и поддерживает аутентификацию и авторизацию пользователей.

Разработка РаВИС – это сложный процесс, который невозможен без участия специалистов в различных областях информационных технологий и инженерного проектирования и анализа. Можно выделить три основные роли в разработке и использовании РаВИС:

1) *инженер* - это пользователь системы CAEBeans, решающий задачу инженерного моделирования на основе созданного для этой цели распределенного виртуального испытательного стенда;
2) *прикладной программист* - это специалист в области разработки РаВИС посредством технологии CAEBeans;
3) *системный программист* - это специалист в области информационных технологий, отвечающий за формирование распределенной вычислительной среды для исполнения РаВИС.

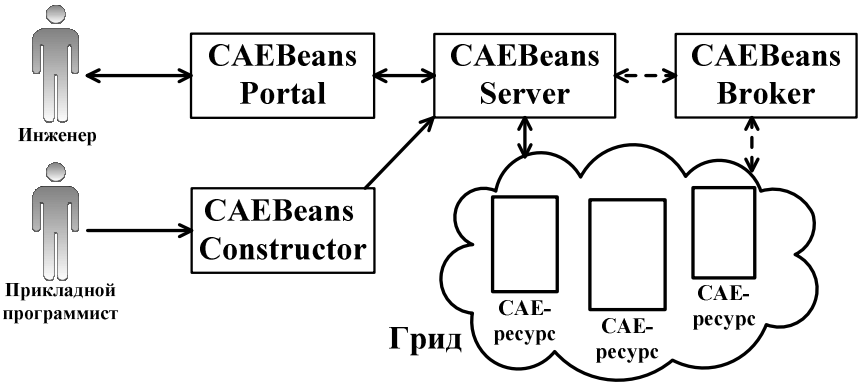


Рис. 4. Общая схема взаимодействия компонентов системы CAEBeans.

3. Система CAEBeans

Для поддержки технологии CAEBeans был разработан программный комплекс *система CAEBeans*, обеспечивающий разработку и исполнение РаВИС В систему CAEBeans [10] входят следующие компоненты (см. рис. 4):

1. *CAEBeans Constructor* – интегрированная среда разработки проблемно-ориентированных оболочек для грид;
2. *CAEBeans Portal* – это веб-приложение, обеспечивающее выбор, загрузку, запуск и получение результатов моделирования CAE-задач;
3. *CAEBeans Server* – хранилище и интерпретатор CAE-проектов;
4. *CAEBeans Broker* – автоматизированная система регистрации, анализа и предоставления CAE-ресурсов;
5. *CAE-ресурсы* – грид-сервисы, обеспечивающие удаленную постановку и решение задач средствами некоторого инженерного пакета на базе конкретной целевой системы.

CAEBeans Constructor – это интегрированная среда разработки РаВИС на основе CAE-проектов. CAEBeans Constructor предоставляет прикладному программисту пользовательский интерфейс для разработки оболочек CAEBeans концептуального, логического и физического слоев. В соответствии с этим, пользовательский интерфейс, обеспечивающий разработку CAE-проектов в среде CAEBeans Constructor, разделен на 3 секции, обеспечивающих разработку проблемных, логических и физических оболочек CAEBean соответственно.

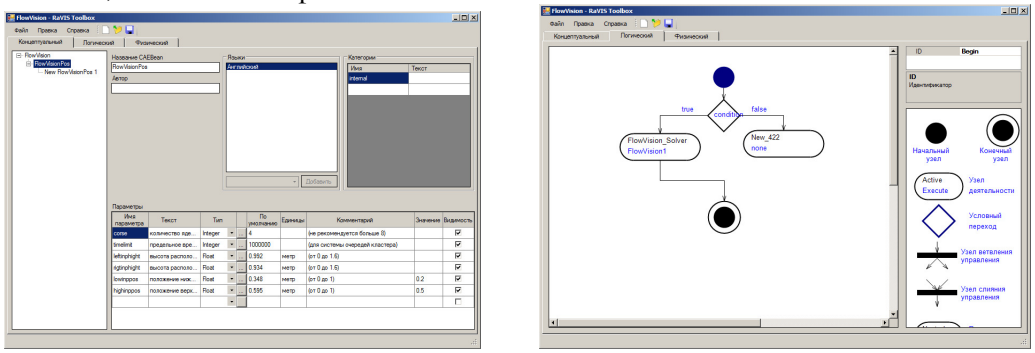


Рис. 5. Пользовательский интерфейс концептуального и логического слоя CAE-проекта.

CAEBeans Portal – это веб-приложение, доступное через интернет, обеспечивающее пользовательский интерфейс для постановки и решения задач инженерного моделирования средствами системы CAEBeans. Встроенный генератор веб-форм обеспечивает автоматическую генерацию пользовательского интерфейса для постановки задач инженерного моделирования, на основе описания параметров соответствующего проблемного CAEBean.

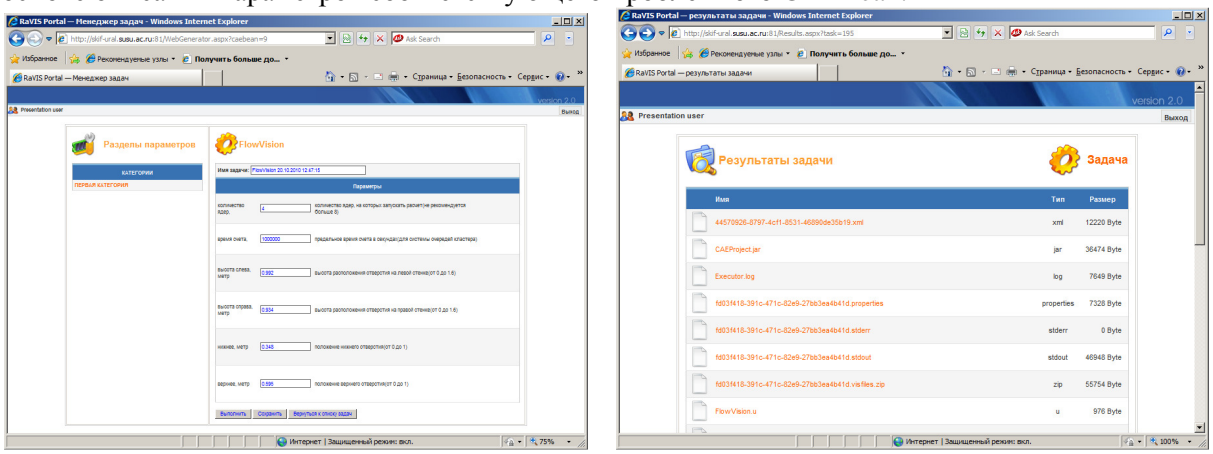


Рис. 6. Пользовательский интерфейс постановки задачи и получения результатов моделирования в CAEBeans Portal.

В качестве платформы для реализации грид-сервисов, был выбрана технология грид-вычислений на базе системы UNICORE 6. Основным достоинством использования системы UNICORE 6 для разработки распределенных вычислительных систем можно считать наличие богатого арсенала различных клиентов, обеспечивающих взаимодействие пользователя с ресурсами вычислительной сети, а также развитых средств обеспечения безопасности при разработке грид-приложений.

CAEBeans Server – это грид-сервис, обеспечивающий хранение и интерпретацию CAE-проектов. Процесс исполнения логического плана поддерживается посредством *монитора логического плана*. Он постоянно производит проверку состояния всех узлов логического плана. Готовность узла к исполнению зависит от наличия маркеров управления в потоках управления, входящих в узел. В зависимости от типа узлов, может требоваться наличие одного или нескольких маркеров управления на входных потоках управления. Алгоритм работы монитора логического плана приведен на рисунке 7.

```
// пока происходит исполнение логического плана
while (running) {
       // цикл по всем узлам логического плана
       foreach (AbstractWorkflowNode node in workflow) {
              if node.ready(){    //Если узел готов к исполнению
                     node.reset();//Сброс       информации       о       поступивших
                                  //маркерах управления
                     new Thread(node).start(); //Исполнение узла
              } //if                          //в отдельном потоке
       }//foreach
}//while
```

Рис. 7. Алгоритм работы монитора потокового CAEBean**.**

CAE-ресурсом является экземпляр системного CAEBean, предоставляющий ресурсы некоторого инженерного пакета на базе конкретной целевой системы. CAE-ресурс обеспечивает:

– получение данных для решения задачи средствами базового инженерного пакета из CAEBeans Server или внешнего источника данных;
– запуск и автоматизированное решение задачи инженерного моделирования;
– передачу результатов решения CAEBeans Server или во внешнее хранилище данных.

CAEBeans Broker обеспечивает автоматизированную регистрацию, поиск и выделение CAE-ресурсов для реализации действий инженерного проектирования. CAEBeans Broker обеспечивает распределение действий инженерного моделирования по виртуальным CAE-ресурсам. Система CAEBeans Broker обеспечивает распределение действий инженерного моделирования

по виртуальным CAE-ресурсам. В структуре системы CAEBeans Broker можно выделить следующие основные блоки (см. рис. 8.):

- *входная очередь заданий (inputQueue)* обеспечивает получение от сервера запросов на предоставление CAE-ресурсов;
- *распределитель заданий* обеспечивает анализ поступающих запросов и их распределение по очередям заданий;
- *каталог ресурсов* хранит информацию о характеристиках всех физических вычислительных узлов, входящих в грид, и виртуальных CAE-ресурсах находящихся на данных узлах;
- *блок перераспределения ресурсов* обеспечивает возможность удаления старых и формирования новых CAE-ресурсов на основе доступных физических ресурсов;
- *очередь ожидания (waitQueue),* в которую попадают задания, которые не могут быть исполнены на существующем наборе CAE-ресурсов, но можно использовать перераспределение для получения CAE-ресурсов, удовлетворяющих требованиям.

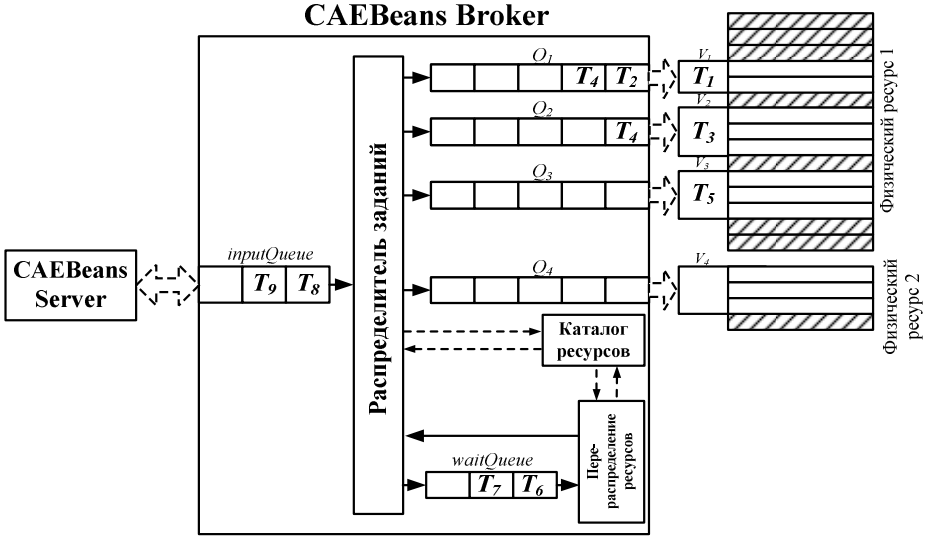


Рис. 8. Структура очередей CAEBeans Broker.

4. Испытания системы CAEBeans

Для проверки возможностей, предоставляемых системой CAEBeans в области создания и исполнения виртуальных испытательных стендов, был разработан набор РаВИС, обеспечивающих решение различных задач инженерного моделирования средствами различных CAE-пакетов. Были произведены исследования методов взаимодействия с современными инженерными пакетами, и проанализированы API, предоставляющие методы автоматизированного решения задач инженерного моделирования. Для испытания системы CAEBeans были разработаны оболочки для решения задач средствами наиболее распространенных пакетов инженерного моделирования: ANSYS CFX [8], ANSYS Mechanical [5], ABAQUS [7], DEFORM [4].

В качестве основной испытательной задачи выбрана задача моделирования процесса закалки и охлаждения труб и анализа влияния различных аспектов процесса закалки на качество производимой продукции («Распределенный виртуальный испытательный стенд «Термообработка»), решаемая по заказу ОАО «Челябинский трубопрокатный завод» [3].

Перед созданием компьютерной модели процесса термической обработки были произведены тепловизионные исследования процесса закалки труб непосредственно на производстве. Был произведен сбор информации о геометрии заготовок и температурных полях, величине разностенности с разверткой по длине и окружности заготовки, величине начальной овальности с разверткой по длине и кривизне оси заготовки.

При разработке проблемного CAEBean была предусмотрена возможность изменения следующих технических параметров индукционной установки: количество индукторов, частота и

сила тока, длина индукторов, количество и конфигурация водяных струй, давление и расход воды, скорость движения трубы через индукционную установку, частота вращения труб (см. рис. 9). Также, была предусмотрена возможность моделирования термообработки труб из различных марок сталей (путем указания физических характеристик материала).

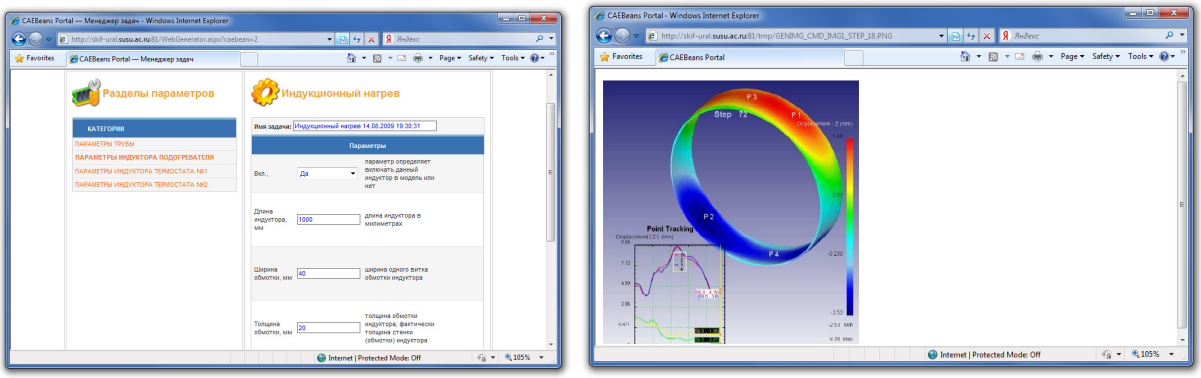


Рис. 9. Интерфейс постановки задачи в распределенном виртуальном испытательном стенде «Термообработка» и пример визуализации результата моделирования.

Система CAEBeans была развернута на базе суперкомпьютерных ресурсов СКЦ ЮУрГУ. Системы CAEBeans Portal, CAEBeans Server и CAEBeans Broker были установлены на вычислительных узлах суперкомпьютера СКИФ-Урал. В системе CAEBeans Constructor был разработан CAE-проект РаВИС. В CAEBeans Portal был загружен проблемный CAEBean созданного виртуального испытательного стенда. На CAEBeans Portal была создана тестовая учетная запись и для нее предоставлены права на разработанный виртуальный испытательный стенд.

На узлах суперкомпьютера СКИФ Урал была установлена и настроена грид-система Unicore. В Unicore была создана целевая система, реализующая системный CAEBean для взаимодействия с пакетом DEFORM. На вычислительных узлах суперкомпьютера СКИФ Урал был установлен пакет DEFORM.

Доступ инженера к CAEBeans Portal осуществляется посредством интернет-обозревателя. После авторизации менеджер задач отображает список доступных испытательных стендов и список запущенных расчетов. В менеджере задач можно создать и запустить новую задачу моделирования индукционного нагрева. По окончанию расчетов изображения с результатами моделирования доступны в разделе результатов задачи.

5. Заключение

В работе были рассмотрены вопросы, связанные с внедрением систем инженерного проектирования и анализа в распределенные вычислительные среды. Было произведено исследование современных подходов по организации распределенных вычислительных систем. Были исследованы основные аспекты внедрения систем инженерного проектирования и анализа в распределенную вычислительную среду. На основе проведенных исследований была предложена концепция распределенного виртуального испытательного стенда, обеспечивающая проблемно-ориентированный подход к решению конкретных классов задач инженерного проектирования посредством ресурсов, предоставляемых вычислительными грид-средами. Была предложена технология CAEBeans, в соответствии с которой, выделяются четыре слоя архитектуры РаВИС, каждый из которых представляется своей оболочкой CAEBeans: Концептуальный слой (проблемный CAEBean), Логический слой (потоковый CAEBean), Физический слой (компонентный CAEBean), Системный слой (системный CAEBean). Разработан прототип комплекса программных средств «Система CAEBeans», обеспечивающий поддержку разработки и исполнения РаВИС. В состав системы входят компоненты: CAEBeans Constructor, CAEBeans Portal, CAEBeans Server, CAEBeans Broker, CAE-ресурсы. На основе разработанного прототипа произведено испытание системы посредством разработки виртуальных испытательных стендов, ориентированных на решение задач инженерного моделирования на базе ряда наиболее распространенных пакетов инженерного моделирования.

Литература

1. Астафьев А.С. и др. Научная сервис-ориентированная среда на основе технологий Web и распределенных вычислений. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). – М.: Изд-во МГУ, 2009. с. 463-467.
2. Бегунов А.А. Применение результатов моделирования для оптимизации и управления технологическими процессами // Параллельные вычислительные технологии: тр. Междунар. науч. конф. (28 янв. – 1 февр. 2008 г., г. Санкт-Петербург). Челябинск: Изд. ЮУрГУ. 2008. С. 31-38.
3. Дорохов В.А., Маковецкий А.Н., Соколинский Л.Б. Разработка проблемно-ориентированной GRID-оболочки для решения задачи овализации труб при закалке // Параллельные вычислительные технологии: Труды международной научной конференции (28 января - 1 февраля 2008 г., г. Санкт-Петербург). -Челябинск: Изд-во ЮУрГУ. -2008. С. 520.
4. Дорохов В.А. Разработка виртуального испытательного грид-стенда для исследования эффекта овализации труб при термической обработке // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта - 3 апреля 2009 г.). Челябинск: Изд-во ЮУрГУ, 2009. С. 457-462.
5. Лёушкин В.В., Соколинский Л.Б., Чайко К.А., Юрков В.В. Разработка проблемно-ориентированной Grid-оболочки для моделирования резьбовых соединений труб для нефтяных скважин в распределенных вычислительных средах // Параллельные вычислительные технологии: Труды международной научной конференции (28 января - 1 февраля 2008 г., г. Санкт-Петербург). Челябинск: Изд-во ЮУрГУ. 2008. С. 534.
6. Марьин С.В., Ковальчук С.В. Сервисно-ориентированная распределенная среда управления прикладными вычислительными пакетами // Сборник статей участников Всероссийского конкурса научных работ студентов и аспирантов «Телематика’2010: телекоммуникации, веб-технологии, суперкомпьютинг» / СПбГУ ИТМО. СПб, 2010. С. 205-206.
7. Насибулина Р.С. и др. Методы организации программных интерфейсов к инженерным пакетам в среде GPE // Параллельные вычислительные технологии: Труды международной научной конференции (28 января - 1 февраля 2008 г., г. Санкт-Петербург). Челябинск: Изд-во ЮУрГУ, 2008. С. 537.
8. Радченко Г.И., Соколинский Л.Б., Шамакина А.В. Разработка компонентно-ориентированных CAEBean-оболочек для пакета ANSYS CFX // Параллельные вычислительные технологии (ПаВТ'2008): Труды международной научной конференции (28 января - 1 февраля 2008 г., г. Санкт-Петербург). Челябинск: Изд-во ЮУрГУ, 2008. С. 438-443.
9. 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. С. 134-139.
10. Радченко Г.И. Грид-система CAEBeans: интеграция ресурсов инженерных пакетов в распределенные вычислительные среды // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта - 3 апреля 2009 г.). Челябинск: Изд-во ЮУрГУ, 2009. С. 281-292.
11. Радченко Г.И. Технология построения проблемно-ориентированных иерархических оболочек над инженерными пакетами в грид-средах // Системы управления и информационные технологии. № 4(34). 2008. С. 57-61.
12. Buriola T.M., Scheer S. CAD and CAE Integration Through Scientific Visualization Techniques for Illumination Design // Tsinghua Science & Technology. Vol. 13, No. 1. 2008. P. 26-33.

13. Fan L.Q. et al. Development of a distributed collaborative design framework within peer-to-peer environment. // Computer-Aided Design. Vol. 40. 2008. P. 891–904.
14. Foster I., Kesselman C. The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure. San Francisco: Morgan Kaufman: 2003, 748 p.
15. Hayes B. Cloud computing // Communication of the ACM. Vol. 51. Issue 7. 2008. P. 9-11.
16. Li Z. et al. Architecture of collaborative design grid and its application based on LAN // Advances in Engineering Software. Vol. 38, № 2. 2007. P. 121-132.
17. Li Z. et al. Conception and implementation of a collaborative manufacturing grid // The International Journal of Advanced Manufacturing Technology. Vol. 34, № 11-12. 2007. P. 1224-1235.
18. Raphael B., Smith I.F.C. Fundamentals of computer aided engineering. John Wiley, 2003. 306 p.
19. Yin J.W., Zhang W.Y., Li Y., Chen H.W. A peer-to-peer-based multi-agent framework for decentralized grid workflow management in collaborative design. // The International Journal of Advanced Manufacturing Technology. Vol. 41, № 3-4. 2009. P. 407-420.

Исследование производительности алгоритмов множественного выравнивания нуклеотидных и белковых последовательностей на вычислительном кластере

К.В. Романенков

В статье содержится описание проведенного тестирования различных параллельных алгоритмов множественного выравнивания (модификация последовательного алгоритма Muscle с помощью системы Parus, ClustalW-MPI, Dialign P) на многопроцессорных кластерных системах. Для исследования было сформировано 3 файла в FASTA формате с семействами белков и семействами нуклеотидных повторов в геноме человека. В статье также даны краткие характеристики систем, на которых производился запуск программ, и анализ полученных результатов.

1. Введение

Алгоритмы множественного выравнивания представляют собой инструмент для установления функциональных, структурных или эволюционных взаимосвязей между биологическими последовательностями.

Несмотря на то, что задача множественного выравнивания сформулирована более 15 лет назад, она не утрачивает своей актуальности в свете развития новых технологий и методов. Появление новых, более высокоуровневых средств (к примеру, кроссплатформернного биоинформатического проекта UGENE[1]) не отменяет наличия качественной и быстродействующей базы, каковой является набор алгоритмов множественного выравнивания. Однако сложность задачи обусловливается экспоненциальным ростом времени счета не параллельной программы при увеличении либо числа биологических последовательностей, либо их длины. Поэтому существующие алгоритмы жертвуют качеством выдаваемого ответа, увеличивая эффективность собственного выполнения.

Активное внедрение многопроцессорных и многоядерных архитектур дает возможность значительно сократить время решения реальных задач, особенно тех из них, в которых исследуемая область плохо сужается в процессе итеративного подхода (то есть каждый шаг в независимости от предыдущего высчитывается по новой), позволяя реализовывать ту или иную вариацию перебора. В качестве примера реальной задачи, помимо целей эволюционной биологии, для которой исторически и создавались алгоритмы, можно привести проблему синтеза лекарственных препаратов (особенно антибиотиков), когда скорость мутаций в популяциях бактерий практически сравнялась, а местами и опередила скорость синтеза новых лекарственных препаратов традиционным способом [2 – 4]. Еще одним примером может служить раздел моделирования различных биологических комплексов (например, клеточная мембрана).

2. Виды множественного выравнивания

Практически все алгоритмы множественного выравнивания базируются на парном выравнивании последовательностей, которые, в свою очередь, зачастую основаны на алгоритмах динамического программирования (Алгоритм Нидельмана-Вунша, алгоритм Смита-Ватермана). Оба этих метода используют матрицу размера (N+1)*(M+1), где N и M длины соответствующих последовательностей, затем, исходя из значений, занесенных в эту таблицу согласно ходу алгоритма, строится оптимальное парное выравнивание.

Процедура множественного выравнивания аналогична парному, с той лишь разницей, что уже выровненные последовательности становятся так называемым профилем выравнивания, внутри которого запрещено производить перемещение последовательностей друг относительно друга, и которые можно считать одной последовательностью для дальнейшей работы

алгоритма. Изначально производится кластеризация входных данных, с той целью, чтобы выравнивания происходили, по возможности, в наиболее схожих последовательностях.

Существует множество реализаций алгоритмов множественного выравнивания, основанных на различных моделях:

1. Прогрессивные.
2. Итеративные.
3. Скрытые Марковские модели.
4. Генетические алгоритмы.

Наибольшее распространение получили алгоритмы, представляющие первые две группы.

Прогрессивное выравнивание характеризуется меньшим временем выполнения, так как процесс кластеризации идет только в самом начале метода, таким образом, гарантируется, что каждая последовательность будет выровнена не более одного раза. Этот подход позволяет ощутимо сократить время счета, однако получаемые результаты не всегда могут похвастаться отменной биологической корректностью, так как зависят от того, как будет произведена начальная кластеризация.

Алгоритм итеративного выравнивания подразумевает возможность проведения повторной кластеризации, если в ходе работы метода обнаружится, что точность выравнивания не удовлетворяет заданным критериям. Такой подход позволяет достичь наперед заданной точности, но при больших объемах входных данных это требует серьёзных временных затрат.

3. Рассмотренные алгоритмы

Для исследования были выбраны параллельные версии (основанные на библиотеке MPI) следующих алгоритмов:

1. ClustalW-MPI [5].
2. Dialign P [6].
3. Параллельная реализация Muscle с помощью Parus [7].

Первый алгоритм является прогрессивным методом, второй алгоритм представляет собой итеративный подход, а третий объединяет в себе итеративную и прогрессивную схемы. Существует довольно много последовательных алгоритмов множественного выравнивания, в том числе и opensourse проекты, но для параллельных версий оказались доступны исходные коды только этих программ.

4. Обзор использованных вычислительных кластеров

При исследовании были задействованы два многопроцессорных комплекса, к которым имеется доступ из сети МГУ:

1. СКИФ МГУ "ЧЕБЫШЕВ".
2. BlueGene P.

Ниже приведены краткие характеристики обеих систем:

Таблица 1. СКИФ МГУ "ЧЕБЫШЕВ"

Пиковая производительность	60 TFlop/s
Производительность на Linpack	47.04 TFlop/s (78.4% от пиковой)
Модель процессора	Intel Xeon E5472 3.0 GHz

Таблица 2. BlueGene P

Пиковая производительность	27.2 TFlop/s
Производительность на Linpack	23.2 TFlop/s (85% от пиковой)
Модель процессора	4-х ядерный процессор, каждое из которых представляет собой PowerPC450 850 MHz

Dialign P запускался на BlueGene P, а остальные алгоритмы на СКИФЕ.

5. Обзор входных данных

Для проведения исследования было сформировано 3 файла в FASTA [8] формате, два из них содержат белковые последовательности, а в остальных находятся нуклеотидные цепочки.

5.1 Файл Pfam.fasta

Файл содержит последовательности из 13 семейств белков, информация взята из протеиновой базы данных Pfam, характеристики:

1. Число последовательностей - 1011
2. Средняя длина — 526.

5.2 Файл Seq.fasta

Файл содержит нуклеотидные последовательности LTR (Long Terminal Repeat) класса 5 в геноме человека, характеристики:

1. Число последовательностей - 1500
2. Средняя длина – 1200.

5.3 Файл Short.fasta

Его содержимое не несет явно выраженной смысловой нагрузки, а файл является сокращенной версией Pfam.fasta, сформированным с целью исследования производительности алгоритма Dialign P, так как он весьма требователен к объему памяти на узле, характеристики:

1. Число последовательностей - 100
2. Средняя длина – 390.

6. Анализ результатов

На рисунках 1, 2, 3 представлены результаты проведенного тестирования, можно выделить следующие наиболее интересные особенности представленных диаграмм:

1. Алгоритм Dialign P не присутствует на первых двух графиках, так как он выполняется только для сравнительно небольших объемов данных, но и на них он проигрывает по времени выполнения параллельной программы остальным рассматриваемым алгоритмам. Одинаковое время для одного, двух и четырех процессоров означает, что Dialign P не уложился в отведенные ему 15 минут. Надо заметить, что программа не предоставляет отчет о времени, затраченном на работу, поэтому самостоятельно были выбраны точки для замера времени, и в измеряемом отрезке не были учтены временные затраты на считывание входных данных и распределение их по процессорам, в отличие от остальных алгоритмов.

2. Алгоритм Muscle показывает хорошую масштабируемость на больших объемах данных, но на меньшем числе последовательностей не демонстрирует достаточное сокращение времени выполнения, а то и его прирост.

3. Малое изменение времени работы ClustalW-MPI на одном и двух процессорах обусловливается тем, что один из процессов является управляющим. Программа показывает лучшую масштабируемость, но временные показатели у неё хуже чем у Muscle.

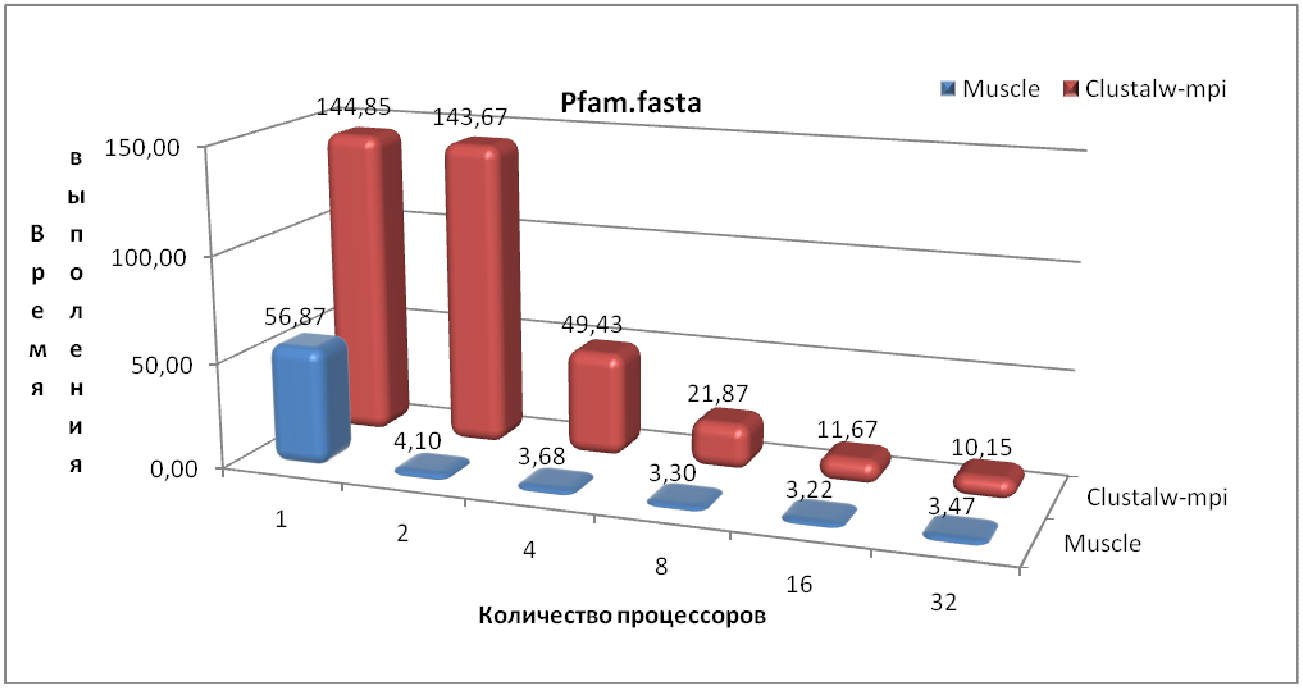


Рис. 1. Время выравнивания (в минутах) последовательностей из файла Pfam.fasta

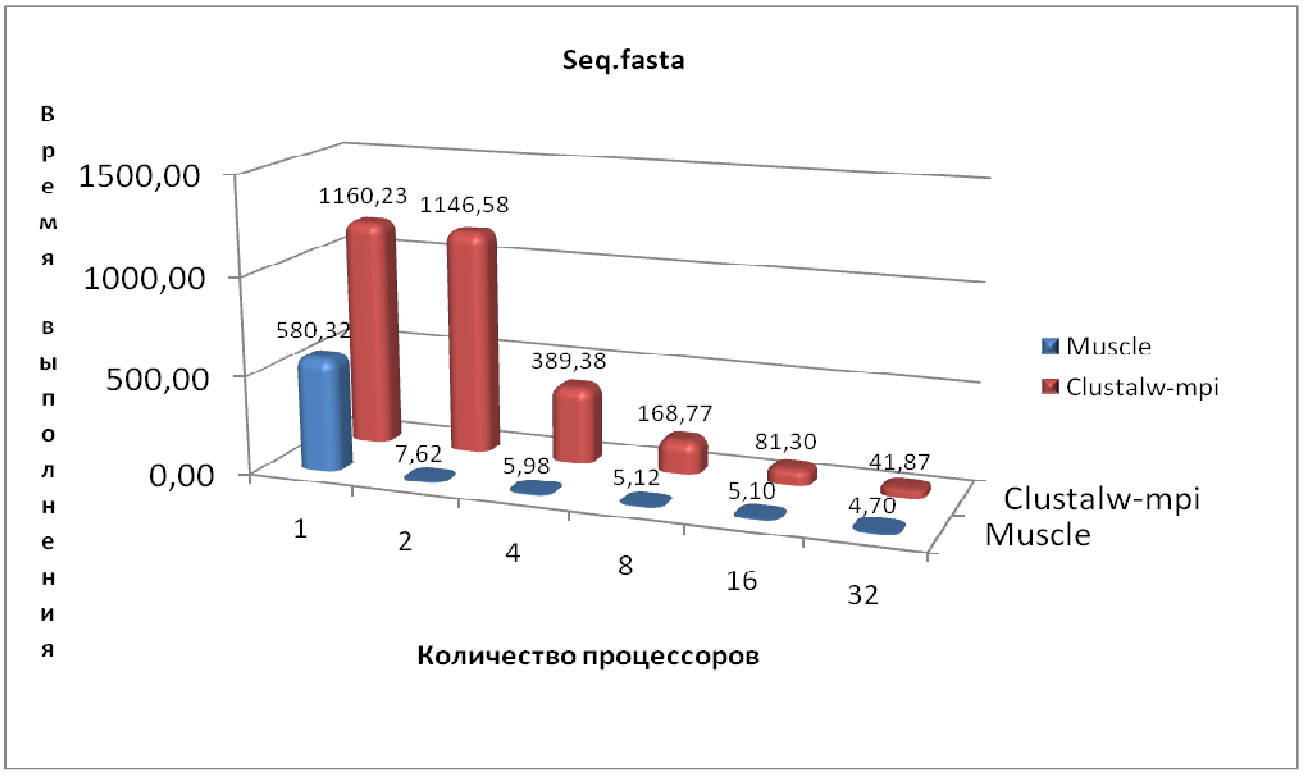


Рис. 2. Время выравнивания (в минутах) последовательностей из файла Seq.fasta

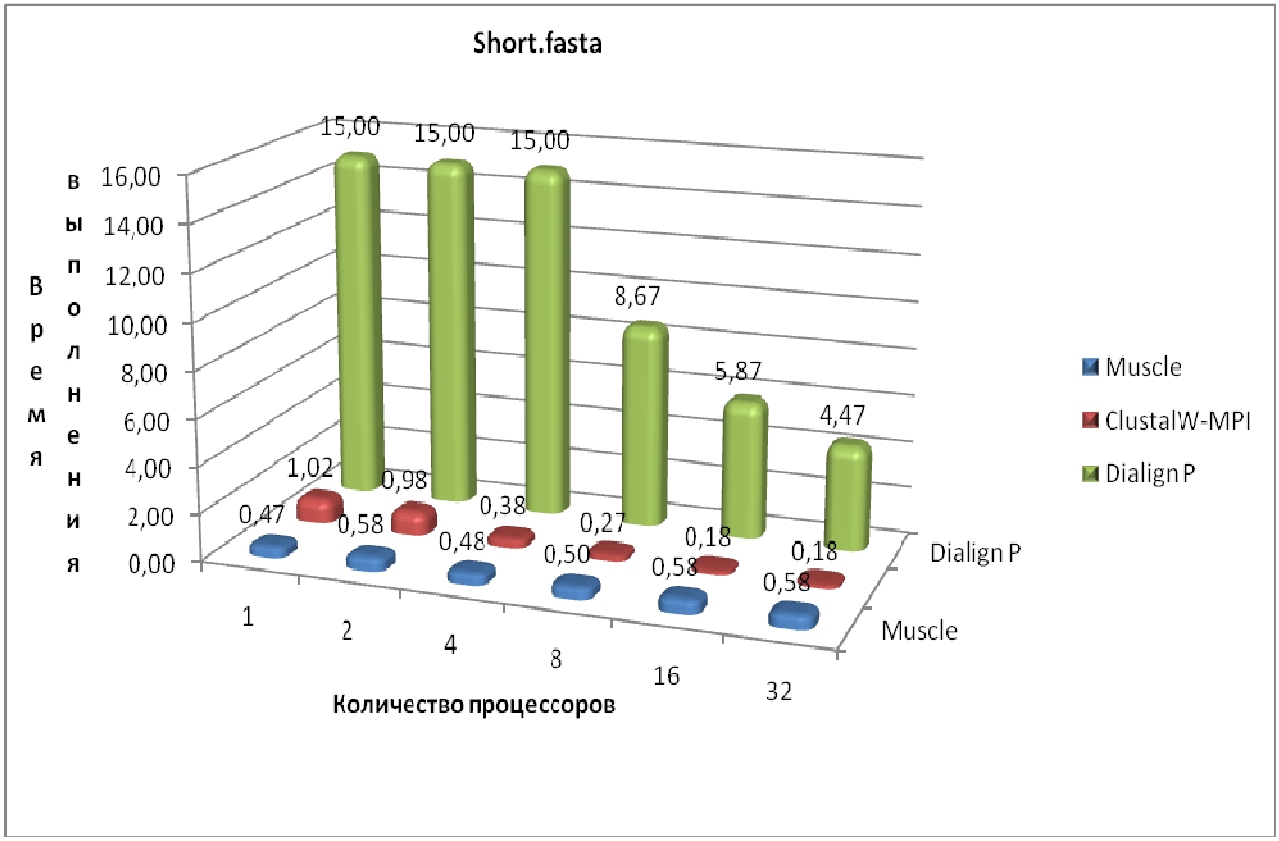


Рис. 3. Время выравнивания (в минутах) последовательностей из файла Short.fasta

7. Заключение

Полученные результаты позволяют говорить о том, что на сегодняшний день не существует доступных универсальных алгоритмов множественного выравнивания, одинаково быстро работающих и для малого числа коротких последовательностей, и для большого числа длинных. Почти все программы при построении парного выравнивания используют матрицу, размерности которой равняются длине выравниваемых участков, поэтому для последовательностей достаточно больших размеров требуется колоссальное количество памяти. Решение этой проблемы в рамках использования концепции динамического программирования не представляется очевидным. Таким образом, задача создания эффективного и универсального алгоритма множественного выравнивания является достаточно важной и нужной, особенно с учетом развития многопроцессорных систем (в частности, ввода в эксплуатацию многопроцессорного комплекса «Ломоносов» в МГУ).

Литература

1. Fursov, M. Y.; Oshchepkov, D. Y; Novikova, O. S. (2009). "UGENE: interactive computational schemes for genome analysis". *Proceedings of the Fifth Moscow International Congress on Biotechnology* 3: 14–15. ISBN 5-7237-0372-2.
2. Howard Hughes Medical Institute (2005, September 22). Gaining Ground In The Race Against Antibiotic Resistance. *ScienceDaily*. Retrieved January 13, 2011, URL: http://www.sciencedaily.com/releases/2005/09/050922021043.htm (дата обращения: 10.01.2011).
3. Robert F. Resistance is futile. URL: http://news.sciencemag.org/sciencenow/2007/03/28-03.html (дата обращения: 10.01.2011).
4. Mark Joffe, The Canadian Journal of Infectious Diseases & Medical Microbiology, 2006 Sep–Oct; 17(5): 285.
5. Kuo-Bin Li ClustalW-MPI: ClustalW analysis using distributed and parallel computing //Bioinformatics Vol. 19, No. 12, 2003, pp. 1585-1586. ISSN: 1460-2059 (Electronic),ISSN: 1367-4803 (Print).
6. Martin Schmollinger, Kay Nieselt, Michael Kaufmann and Burkhard Morgenstern DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors // BMC Bioinformatics, 2004, 5:128, ISSN: 1471-2105 (Электронный).
7. Alexey N. Salnikov The modification of MUSCLE multiple sequence alignment algorithm for multiprocessors Proceedings of the 3-rd Moscow conference on computational molecular biology, Moscow, Russia, July 27-31 2007, pp. 270-271.
8. Pearson and Lipman. "Improved tools for biological sequence comparison", 1988. PNAS 85(8): 24444 – 2448.

Механизмы ускорения взаимодействия устройств в вычислительных структурах с сетевой организацией связей

Г.Г. Стецюра

Институт проблем управления РАН

Рассмотрены решения, ускоряющие реакцию систем управления и обработки данных на внутренние и внешние события за счет совмещения обмена данными с распределенной обработкой данных и применения однотактной распределенной беспроводной коммутации. Показана целесообразность использования для этих целей оптических технических средств.

1. Введение

Так как современные вычислительные и управляющие системы содержат множество взаимосвязанных компонентов, то для эффективного решения прикладных задач естественно требуется быстрое и гибкое взаимодействие компонентов системы и быстрая реакция на внутренние и внешние события. Такие же и даже более жесткие требования выдвигают внутрисистемные задачи управления – задачи поддержки "жизнедеятельности" систем.

Ярким примером повышения требований к управлению состоянием системы служит выдвинутая IBM концепция "Autonomic Computing" (AC), возникшая как реакция на возрастающую сложность вычислительных и управляющих комплексов [1]. Концепция AC ориентирована на обеспечение без участия человека четырех свойств системы: самоконфигурирования при изменяющихся условиях функционирования, самовосстановления при любых нарушениях в работе системы, непрерывной самооптимизации и самозащиты от произвольного вида враждебных воздействий (4 «само»). Выступая с AC, фирма IBM предлагает создавать совершенно новый вид систем и фактически *инициирует междисциплинарные исследования в области управления поведением сложных полностью автоматических комплексов, действующих в реальном масштабе времени.* Эта инициатива была широко воспринята.

Для достижения объявленных IBM целей, видимо, целесообразно параллельно со средствами обработки данных иметь дополнительную систему средств, непрерывно наблюдающую за состоянием всех цифровых средств и обеспечивающую достижение указанных четырех "само". Такая система должна, возможно, в большей степени, чем основная система, обеспечивать высокую скорость реакции на возникающие события, автоматически прогнозировать появление нестандартных нежелательных ситуаций, их обнаруживать и устранять. Таким образом, задачи самоуправления функционированием систем выдвигают дополнительные требования к управлению взаимодействием устройств систем, и их также надо учитывать.

Для всех указанных применений в настоящем докладе предложена совокупность механизмов низкого уровня, ускоряющих и упрощающих реализацию трех видов процессов в цифровых системах – совмещения обмена данными с распределенными вычислениями, децентрализованного устранения конфликтов доступа к ресурсам, децентрализованной коммутации.

Рассматривается организация функционирования этих механизмов, и приводятся ссылки на публикации других авторов по конкретной физической реализации необходимых устройств с использованием, прежде всего, оптических средств. Оптические средства позволяют получить простые, быстродействующие и мало энергоемкие решения, и количество работ по созданию оптических устройств с требуемыми в этой работе свойствами непрерывно возрастает.

Так результаты, полученные в NRL [2–4], непосредственно применимы при реализации устройств из разделов 2.2, 2.5 – 2.8. Для разделов 2.1 и 2.3 важны исследования в IBM по оптическим переключателям [5, 6]. Электронные варианты устройств для этих разделов были созданы в ИПУ [7]. В разделе 2.4 устройства, переключающиеся медленнее микросекунды, реализуемы многими способами [8]; для более высоких скоростей имеются публикации только на уровне описания результатов физических экспериментов [9–12].

2. Механизмы ускорения взаимодействия узлов цифровых систем

В восьми подразделах данного раздела рассмотрены существенно различающиеся между собой механизмы, обеспечивающие ускорение взаимодействия устройств систем управления и обработки данных (далее "узлов"). Все изложенные результаты получены в Институте проблем управления РАН (ИПУ).

2.1 Групповые операции (совмещение обмена данными с распределенными вычислениями)

Обычно логические элементы вычислительных средств действуют следующим образом. Поступающий на вход элемента сигнал перестраивает структуру элемента так, что на его выходе появляется сигнал, создаваемый за счет энергии источника питания элемента. На перестройку элемента требуется время, и возникающая временная задержка сигнала определяет его быстродействие. При последовательном соединении элементов сигнал с выхода одного элемента поступает на вход другого элемента, перестраивает его и т.д.: задержка накапливается.

Чтобы избежать накопления задержки и сделать время вычисления такого вида практически не зависящим от количества участвующих в нем узлов, был разработан метод вычислений, совмещающий процесс передачи данных с вычислением [7, 13–15]. Один из вариантов реализации метода иллюстрирует рис. 1.

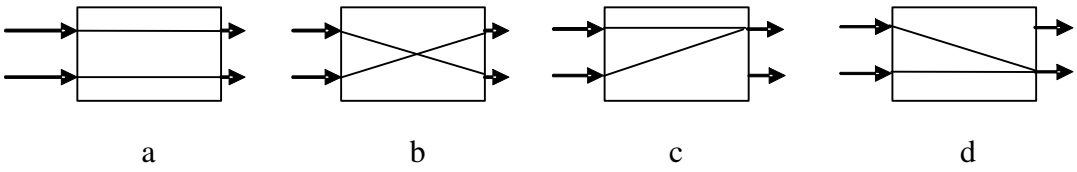


Рис. 1. Логический элемент

Здесь a, b, c, d – четыре состояния логического элемента, представленного прямоугольником. В элемент входят две линии и из него выходят также две линии. Элемент выполняет указанные на рис.1 соединения между входными и выходными линиями. Сигнал, подаваемый на элемент, поступает только на одну из входных линий. Будем считать, что наличие сигнала на верхней входной линии интерпретируется как двоичная единица (1), сигнал на нижней линии интерпретируется как логический ноль (0). Элемент в состоянии “a” передает сигнал на выход без изменения, в состоянии “b” инвертирует значение входного сигнала, в состоянии “c” превращает любое значение входа в "1" на выходе, в состоянии "d" превращает любое значение входа в "0" на выходе. Переключение состояний выполняет внешнее по отношению к элементу устройство.

Указанных состояний элемента достаточно для выполнения логических операций, нахождения *max* и *min,* выполнения арифметических операций сложения, вычитания и умножения.

Рассмотренный логический элемент размещается в системе совмещающей вычисления с передачей данных способом, показанным на рис.2.

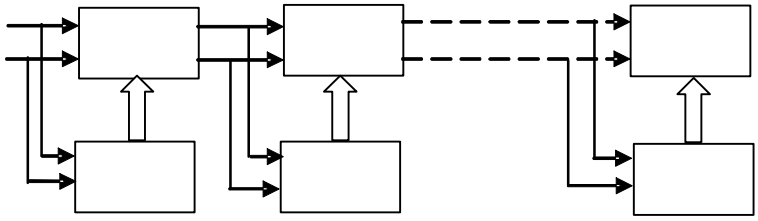


Рис.2. Система с совмещением вычислений и передачи данных.

Здесь каждый прямоугольник из верхнего ряда обозначает рассмотренный выше логический элемент, каждый прямоугольник из нижнего ряда – устройство, управляющее состоянием соответствующего элемента из верхнего ряда. Фигурная стрелка обозначает канал управления

состоянием элемента. Показанные на рис.2 ответвления входов в устройства предназначены для приема приходящих данных. Совокупность элемента с управляющим им устройством составляет узел системы. По каналу, соединяющему элементы, данные передаются последовательно поразрядно.

Распределенное вычисление выполняется над операндами, каждый из которых размещен в соответствующем узле рис. 2. Так можно, например, выполнить распределенное суммирование чисел. При этом состояние элемента из рис. 1 определяет соответствующий разряд операнда в узле, и *перестройка связей в элементе выполняется до прихода по каналу разряда, на значение которого эта перестройка должна повлиять.*

Отсюда следует: *все узлы одновременно устанавливают входящие в них логические элементы в требуемое состояние, после этого сигнал проходит через все элементы без задержки на установку элементов.* Операция выполняется за время переноса сигнала через элементы (один такт) и практически не зависит от числа последовательно соединенных элементов.

Отметим, что все элементы, включая первый, не создают новый сигнал, они лишь транслируют поступивший сигнал. Сигнал создает внешнее устройство – генератор сигналов, посылающий сигналы на верхний вход первого элемента в цепочке. Отсюда следует второй важный результат: *в системе не создаются мощные сигналы, требуемые для взаимодействия между узлами системы.* Это уменьшает энергопотребление и рассеивание энергии в системе, и позволяет создавать распределенные системы, потребляющие малую мощность, например, системы интеллектуальных датчиков.

Выше рассмотрен способ совмещения, использующий изменение пути прохождения сигнала, но можно также изменять свойства сигналов. В публикации [7] показано использование для этого поляризации оптического сигнала, в [13] использована замена без задержки передаваемого по линии двоичного сигнала на противоположное значение. Было бы полезно выявить и другие способы изменения сигналов, передаваемых без их задержки.

Операции, выполняющие рассмотренные распределенные вычисления названы групповыми операциями.

2.2 Децентрализованное приоритетное управление доступом к ресурсу (ДПУ)

Способ доступа ДПУ был разработан для решения следующей задачи. Имеется группа узлов, которым требуется доступ к некоторому ресурсу, в частности к каналу связи. Эта потребность возникает в произвольные не коррелированные для разных узлов моменты времени. Для решения задачи в ИПУ был разработан ряд способов, объединенных под общим названием "ДПУ". Наиболее полно варианты ДПУ изложены в [14]. Здесь приведем один из вариантов ДПУ – децентрализованное кодовое управление (ДКУ).

При ДКУ узлам присваиваются различающиеся коды приоритетов, определяющие право источников на занятие канала.

Источник ожидает наступления момента времени, когда можно начинать борьбу. Пусть, например, этот момент определяется наступлением паузы – отсутствием сигнала в связывающем источники канале в течение времени $T_{пз}$.

В интервале времени, когда можно проводить борьбу, источники передают последовательно разряды своих кодов приоритетов, начиная со старшего разряда. Сигналы, представляющие двоичные 1 и 0 кода приоритета, должны быть такими, чтобы можно было обнаружить наличие сигнала 1 в канале при наличии в нем сигнала 0. Источник передает сигналы с интервалом времени $T_р$. Сигнал передается течение времени $T_рQ$ ($0<Q<1$). В течение времени $T_р(1 - Q)$ источник проверяет состояние линии. Источник, передавший 0 и обнаруживший 1 в канале, прекращает борьбу за канал. Источник, передавший все разряды кода приоритета, занимает канал. Изложенные действия – это операция, определяющая в канале максимальное значение в группе чисел. Для корректности работы алгоритма достаточно, чтобы выполнялось соотношение $T_р \geq 2\tau / (1 – Q)$. Здесь τ – время прохождения сигнала по каналу [14].

ДКУ с помощью регулирования значений приоритетов позволяет достигнуть очень гибкого управления работой распределенных систем, и обеспечивает гарантированный доступ узлов к ресурсу. Приоритеты узлов могут изменяться в динамике, учитывая текущее состояние узла.

ДКУ эффективно работает при достаточно коротких расстояниях между узлами системы, так как интервал времени между передачей разрядов кода приоритета учитывает время прохождения сигнала по каналу. ДКУ асинхронно.

Позднее фирма Philips опубликовала интерфейс I^2C, использующий близкий способ разрешения конфликтов, но работающий синхронно и более ограниченно использующий приоритеты доступа [16]. Интерфейс I^2C применяется во многих микроконтроллерах.

2.3 Групповые команды, групповые программы

Групповые операции, приведенные в разделе 2.1, используются в групповых командах и групповых программах. Групповая команда представляет собой сообщение, передаваемое между узлами, предварительно объединенными каналом так, как это сделано в разделе 2.1.

Групповая команда содержит операнд и перемещается через цепочку узлов (см. раздел 2.1). Вторые операнды находятся в каждом из узлов. При прохождении сообщения через узел *без задержки сообщения* выполняется требуемая операция над двумя указанными операндами. *Ключевое свойство* совмещенных операций – результат вычисления помещается на место, которое в сообщении занимает операнд. После прохождения сообщения через цепочку узлов в нем вместо первого операнда будет находиться результат действия всех узлов.

Чтобы указать узлу, как он должен обрабатывать операнд, в сообщении, перед операндом помещается код групповой операции, понимаемой всей группой узлов. Последовательность из кода групповой операции и операнда составляет простейшую групповую команду. В общем случае групповая команда имеет переменную длину и содержит группу операндов. Код групповой команды может содержать сложные указания по обработке содержимого сообщения, определяющие различные способы обработки разных операндов в групповой команде. Так код групповой операции информирует узлы о структуре сообщения.

Действия групповой команды могут распространяться не только на сообщение, содержащее эту команду, но и на определенную в команде группу последующих сообщений. Групповая команда может также изменять состояние конкретных узлов системы, в которые она поступает, тем самым в динамике реконфигурируя систему, включая изменение прав отдельных узлов. Например, так можно задавать ветвление вычислений, передачу от одного узла другому прав лидера – узла, имеющего право посылать групповые команды.

Следующий шаг в построении групповых взаимодействий – определение групповой программы. Групповая программа – это сообщение, в разных частях тела которого размещаются определенные выше групповые команды. Чтобы структуру групповой программы можно было формировать произвольным образом, будем код каждой групповой операции отмечать специальной меткой. На рис. 3 показан пример сообщения – групповой программы.

Рис. 3. Сообщение – групповая программа

Здесь серым цветом обозначены коды групповых команд, без окраски –данные групповых команд.

Основное назначение групповых команд и программ – формирование данных, необходимых для управления сложной системой. Так, например, в работах [17, 18] даны способы получения взаимно согласованных решений при дублировании вычислений и регулировании доступа к ресурсам. Однако групповые команды можно использовать для проведения распределенных вычислений в прикладных задачах при соответствующем изменении вычислительных алгоритмов.

В таблице 1 показано полученное ускорение вычислений для ряда задач по сравнению с другими известными методами [19]. Здесь n – количество компонентов в задаче (размерность матрицы, количество членов уравнения, полинома и т.д.). В строках таблицы для заданного количества процессоров показано время, требуемое для выполнения вычисления. Время выражено в условных единицах T. Сравниваются наиболее быстрый из известных методов с методом совмещения передачи данных и распределенного вычисления.

В системах, использующих параллельные многоразрядные высокоскоростные каналы связи между узлами, синхронная передача разрядов данных усложняется, и во многих случаях последовательная поразрядная передача становится конкурентоспособной. Способ обработки данных, примененный в групповых командах, естественно сочетается с таким последовательным способом передачи.

Изложенная организация групповых программ ориентирована на самые быстрые взаимодействия узлов. Теперь рассмотрим возможность использования групповых программ в смешанных системах, использующих помимо локальных также удаленные связи, например, соединяющие кластеры через интернет при проведении облачных вычислений или в grid computing.

Таблица 1. Ускорение распределенных вычислений

Вид операции	время вычисления		число процессоров
	метод совмещения	обычно: ≥	
Распределенное умножение матриц	Tn T	$Tn\log_2 n$ $T(\lceil \log_2 n \rceil+1)$	n^2 n^3
Алгебраические уравнения, метод Гаусса-Жордана, нахождение главных элементов	Tn	$Tn\log_2 n$	$n(n+1)$
Вычисление значения полинома	T	$T\log_2 n$	n
Вычисление значения произведения полиномов	$2T$	$2T\log_2 n$	n
Свертка	T	$T\log_2 n$	n
Дискретное преобразование Фурье	Tn	$Tn\log_2 n$	n^2
Сортировка	$T(\lceil n/2 \rceil+1)$	$Tn(\log_2 n)$	n

Удаленную передачу необходимо проводить с помощью сообщений, удовлетворяющих международным стандартам, например, протоколу TCP/IP. В это сообщение поместим групповую команду или групповую программу, адресованную удаленным узлам. Формирование сообщения в формате TCP/IP выполняет узел-передатчик, обеспечивающий удаленную связь. Передатчик сохраняет у себя полученную групповую программу, создает сообщение TCP/IP, упаковывает в него в качестве данных групповую программу, адресованную удаленным узлам, и направляет это сообщение удаленному приемнику сообщений через сеть общего пользования. Приемник извлекает из принятого сообщения групповую программу и направляет ее узлам удаленной системы.

Отметим, что в последние годы проводится много работ по внедрению активных сетей, сообщения которых содержат программы, влияющие на конфигурацию сети и внутреннее поведение узлов сети [20, 21]. В этом отношении программы активных сетей подобны групповым программам, но ориентированы на значительно более медленные процессы.

Поэтому там, где требуется быстрое время реакции на события, например, в управляющих системах, целесообразно сообщения оформлять в виде групповых команд или программ.

Кроме того, команды в активных системах в отличие от групповых программ не позволяют организовать совмещенные с обменом данными распределенные вычисления.

2.4 Распределенная оптическая коммутация: парные соединения

Цель раздела – показать возможность уменьшения сложности распределенных коммутаторов и ускорения процесса коммутации за счет использования оптических средств. Показано, что для коммутации n узлов возможно построение однотактных коммутаторов, содержащих $n\log_2 n$ элементов коммутации, каждый из которых принимает одно из двух возможных состояний. Ускорение и уменьшение сложности коммутации достигается за счет установления непосредственных связей между источниками и приемниками сигналов.

Рассматривается только коммутация макроустройств, например, процессоров или компьютеров в сосредоточенном многомашинном комплексе или электронных модулей на печатной плате.

Уменьшение сложности достигается за счет того, что элемент коммутации, имеющий только два состояния, может одновременно коммутировать много каналов и его сложность не увеличивается при увеличении количества каналов.

При этом каждый узел содержит все необходимые ему оптические средства коммутации.

Чтобы указать такому локальному средству коммутации на конкретный из n узлов обработки данных, с которым требуется установить соединение, узлу требуется задать не менее $\log_2 n$ битов информации, что и определяет минимальное количество коммутационного оборудования. В докладе достигнута именно эта величина. В практически используемых средствах коммутации оно больше, и может достигать значения ~ n^2.

Рассмотрим распределенный коммутатор, соединяющий n узлов и содержащий в каждом из таких узлов локальный демультиплексор, соединяющий этот узел с остальными узлами. В разделе приведены примеры возможной реализации такого коммутатора, в которых для построения демультиплексора требуется $\log_2 n$ элементов коммутации, принимающих два состояния. Использованы известные решения, в которые внесен ряд дополнений.

Вначале рассмотрим коммутатор, выполненный как микро-электромеханическая система (MEMS) [8], содержащая электромеханически перемещаемые зеркала. Пусть информация между узлами передается оптическими сигналами, а локальный демультиплексор имеет структуру, пример которой показан на рис. 4.

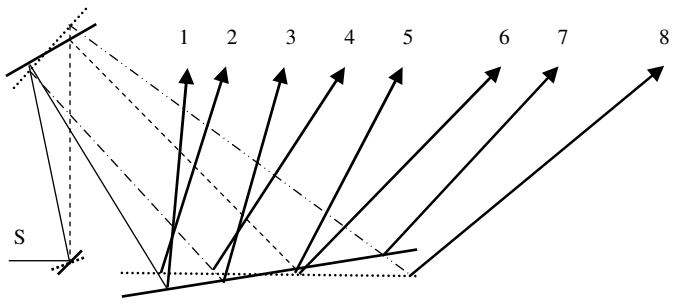


Рис. 4. Коммутатор с перемещаемыми зеркалами

Здесь показаны три поворачивающихся зеркала, на первое из которых направляет луч света лазерный источник сигналов S. Отразившись от всех зеркал, луч займет одно из восьми указанных положений, определяемых положением зеркал.

Таким образом, достаточно иметь $\log_2 n$ управляемых зеркал, чтобы одновременно, за один такт, выбрать одно из n положений луча. Чтобы направить лучи 1 – 8 произвольно расположенным приемникам сигналов, возможно на пути лучей придется поместить дополнительную систему неподвижных зеркал, направляющую лучи в заданном направлении. Требуется n^2 таких неподвижных зеркал для обслуживания связи со всеми n узлами. Неподвижные зеркала всех демультиплексоров располагаются вне узлов. Эти зеркала составляют часть среды распространения лучей, могут быть выполнены в виде монолитной пластины с зеркалами и их сложность существенно меньше сложности остальных компонентов рассматриваемой системы коммутации. Зеркала ориентируются так, чтобы связать места, где могут быть размещены коммутируемые узлы, а система наращивается размещением узлов в отведенные для них места без какой-либо настройки демультиплексоров.

Последнее из зеркал рис. 4 должно иметь размер, достаточный для отражения от него всех n лучей. Поэтому при больших n желательно использовать всю площадь зеркала: $n^{1/2}$ лучей размешается в плоскости рисунка и $n^{1/2}$ – в перпендикулярной плоскости. При этом для перемещения луча между источником S и рассматриваемым демультиплексором требуется расположить дополнительно демультиплексор на $n^{1/2}$ выходов. Он отличается от основного в следующем. Любой выходящий из него луч должен поступать в основной демультиплексор парал-

лельно лучу от источника S, но со сдвигом. Это достигается установкой на выходе демультиплексора $n^{1/2}$ неподвижных зеркал, создающих требуемое изменение направления лучей.

Выделим особенности описанного узла.

1. Количество перемещаемых зеркал и их индивидуальных средств управления положением равно количеству двоичных разрядов числа, представляющего коммутируемые узлы. Положения, в которое перемещаются зеркала, не зависят от уровня сигнала, управляющего положением зеркала.

2. Каждый из двоичных элементов коммутации, переходя в одно из двух состояний, позволяет переключать одновременно группу виртуальных каналов (лучей); размеры группы варьируются от 2 до n для разных элементов демультиплексора. Для управления любой группой требуется однобитовый сигнал. В каждый момент времени реализуется только один из виртуальных каналов.

3. Каждое зеркало способно коммутировать передаваемые параллельно многоразрядные слова.

Отмеченные в п.п. 2 и 3 особенности и обеспечивают уменьшение сложности коммутатора.

4. Источники света S могут быть вынесены не только за пределы коммутаторов, но и за пределы всей системы. В элементах коммутации не создаются новые сигналы. Все это уменьшает энергопотребление в системе.

На переключение положения зеркал требуется более микросекунды, и электромеханические переключатели можно использовать длительно используемой системы соединений. Однако использование физических сред с изменяемыми оптическими свойствами позволяет создавать быстродействующие коммутаторы [9–12, 22]. Так, например, диоксид ванадия (VO_2) за время порядка 12 фмтсек переходит из прозрачного состояния в зеркально отражающее свет [9–11]. Рис. 5 показывает, как может выглядеть демультиплексор, использующий это свойство.

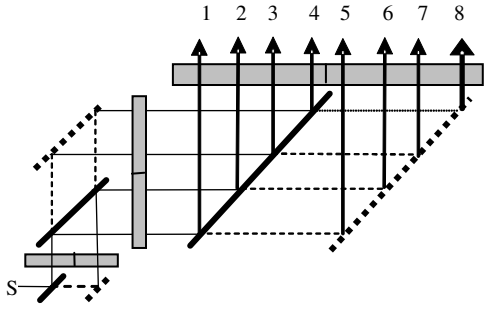


Рис. 5. Коммутатор, использующий VO_2

Здесь сплошными наклонными линиями показаны три управляемых зеркала из VO_2, пунктирными линиями – обычные зеркала или призмы. Источник S направляет луч света на управляемое зеркало, которое или отразит свет на вторую группу зеркал или пропустит его на неуправляемое зеркало. Так действуют все пары зеркал. В результате луч света на выходе примет одно из восьми положений. Все 8 направлений взаимно параллельны, и, чтобы направить их на произвольно расположенные приемники, требуется вне узлов установить указанные выше зеркала.

Пропускающее луч зеркало в незначительной степени и отражает его, что создает помеху. Для избавления от помех введем указанные на рис. 5 прямоугольниками управляемые пары фильтров, которые блокируют лучи от одного зеркала из пары зеркал, нежелательные при данной комбинации управляющих сигналов. Это могут быть фильтры, изменяющие пропускную способность при подаче управляющего сигнала, либо управляемые зеркала из VO_2, отводящие нежелательный сигнал в сторону. Для управления фильтрами не требуются дополнительные сигналы, и сложность по управлению сохраняется.

Для построения демультиплексора можно сочетать пластины с двойным лучепреломлением с электрооптическими поляризаторами Поккельса [12].

Известны и многие другие способы реализации оптического коммутатора, но в значительной их части величина отклонения луча зависит от величины управляющего сигнала, что создает дополнительные трудности. В приведенных примерах этот недостаток отсутствует. Более

подробно материалы раздела 2.4 изложены в [22].

Приведенные парные взаимодействия можно использовать также для соединения модулей на уровне электронной платы, расширяя возможности известного решения [23]

Особенности 1 – 4, приведенные выше для MEMS-демультилексора, необходимо обеспечивать и при других способах построения демультиплексоров, выполняющих рассматриваемые парные соединения.

2.5 Распределенная оптическая коммутация: связь узла с группой узлов

Задача следующая: узел должен одновременно передать одинаковые данные группе связанных с ним узлов. Рассылка выполняется по инициативе передающего узла.

Для таких рассылок данных, помимо сложного передающего оборудования, требуются большие затраты энергии. Ниже приведено решение, устраняющее указанные проблемы. В нем использованы некоторые результаты работ [2–4].

Узлы группы посылают узлу, имеющему ретрорефлектор, немодулированные лучи света, которые этот узел должен модулировать двоичными сигналами и возвратить узлам группы в виде общих для всех узлов данных.

Рассмотрим структуру, показанную на рис. 6. Здесь два луча от двух источников группы (сплошная и пунктирная линии) падают на ретрорефлектор (на рис. 6 это уголковый отражатель – катафот).

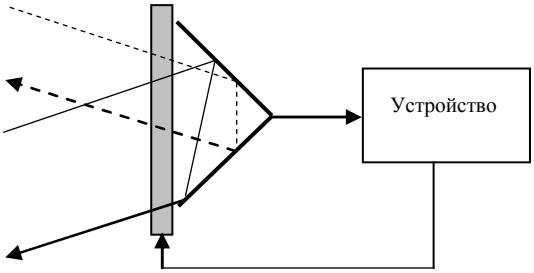


Рис. 6. Применение ретрорефлектора

Отражаясь от катафота, каждый луч возвращается к пославшему его источнику. Перед катафотом расположен управляемый сигналами приемника фильтр, который модулирует значение возвращаемых источникам сигналов. Модулировать сигнал можно, изменяя в фильтре любые свойства отраженного сигнала или изменяя прозрачность граней уголкового отражателя. Можно также расположить перед катафотом управляемое зеркало, которое будет прерывать возврат сигнала адресату.

В отличие от [2] ретрорефлектор здесь используется для связи не между парой устройств, а с группой устройств. В [3, 4] каждый приемник имеет индивидуальные средства для связи с каждым источником.

2.6 Распределенная оптическая коммутация: устранение конфликтов доступа

Пусть имеется узел (центр), с которым требуется установить связь группе узлов, но это должно быть разрешено одновременно только одному из них. Центр имеет устройство связи, показанное на рис. 6. Частично свет, падающий на катафот центра, проходит в приемное устройство, которое обнаруживает поступление сигнала, соответствующего логической единице в поступившем в центр сообщении.

Возникающий конфликт будем устранять с помощью алгоритма ДКУ.

Конфликтующим узлам группы приписаны различающиеся коды приоритетов доступа. Узлы группы посылают центру лучи, модулированные двоичными сигналами, представляющими коды приоритета узлов, и центр при поступлении к нему двоичной единицы хотя бы от одного из узлов группы сообщает об этом всем узлам группы.

Такое оповещение можно выполнить многими способами, например, прерывая возврат све-

тового сигнала в его источники.

Будем считать, что поступившие в центр двоичные сигналы центр интерпретирует так: наложение сигналов "1" и "0" воспринимается как сигнал "1", множество одновременно поступающих сигналов "1" ("0") воспринимается как "1" ("0").

В соответствии с алгоритмом ДКУ узлы, пославшие в центр сигнал "0" и получившие в ответ сообщение от центра о приходе к нему сигнала "1", перестают участвовать в устранении конфликта. После передачи всех разрядов приведенная процедура нахождения максимума выделит единственный источник с наибольшим значением кода приоритета.

Возможны различные модификации приведенной процедуры с изменением как логики процесса [9], так и его физической реализации, например при использовании сигналов разной длины волны для передачи данных и управления.

2.7 Распределенная оптическая коммутация: вызов узлом группы узлов

Пусть узлу системы требуется установить соединение с группой узлов и затем передать этим узлам данные. Пусть все узлы имеют ретрорефлектор и модулятор, показанные на рис. 6.

Выделим в системе фиксированный узел вызова. Этот узел имеет фиксированное положение, и для посылки на его катафот луча узлам не требуется привлекать коммутатор. Все узлы следят за узлом вызова, и с его помощью, действуя в соответствии с разделом 2.6, любые узлы могут провести борьбу за право доступа к узлу вызова. Узел, получивший это право, модулируя лучи устройств с помощью узла вызова, сообщит требуемым ему узлам о необходимости взаимодействия.

После этого выбранные узлы направят лучи на катафот запрашивающего узла, и начнется связь этого узла с группой узлов.

Узел вызова используется только для управления обменом данными, который далее выполняется по индивидуальным связям.

2.8 Распределенная оптическая коммутация: связь с подвижными узлами

В системе могут быть подвижные узлы, ими могут быть как периферийные устройства, так и устройства обработки данных. Необходимо рассмотреть взаимодействие и с такими узлами. Ниже рассмотрим оптические связи, позволяющие получить новые полезные свойства, но вначале укажем на использование для таких связей ДПУ, использующее радиосвязь. Установление связи источника с приемником состоит из двух шагов. На первом шаге при наличии нескольких источников – претендентов на передачу данных алгоритм ДКУ выделяет одного из них. На втором шаге происходит собственно обмен данными. На обоих шагах достаточно иметь единственную частоту радиосигналов, общую для всех узлов. Однако при этом узлы должны иметь источники радиосигналов, что не всегда желательно. Применение оптических средств позволяет разместить источник сигналов в единственном узле.

Рассмотрим следующую систему. В поле видимости узлов находится ретранслятор сигналов, который принимает оптические сигналы на одной частоте f0 и ретранслирует их на другой частоте f1. Кроме этого ретранслятор имеет источник непрерывного оптического излучения на частоте f0, который также видят все узлы системы. Каждый узел имеет два приемника оптических сигналов – для сигналов частоты f0 и частоты f1.

Пусть выделен узел-источник данных, который должен разослать эти данные заданным узлам-приемникам данных. Взаимодействие узлов осуществляется следующим образом. Все узлы получают непрерывный сигнал частоты f0 от ретранслятора, и только источник данных возвращает сигналы, несущие свои данные в ретранслятор на частоте f0. Для этого источник имеет ретрорефлектор сигналов и расположенный перед ним модулятор оптических сигналов, аналогичные показанным на рис. 6.

При передаче данных источник модулирует с помощью модулятора поступающий на ретрорефлектор свет и возвращает его в ретранслятор. Источник модулирует сигнал (разрешает возврат сигнала в ретранслятор), если ему требуется передать единицу. Предполагается синхронность посылок, позволяющая определить" передачу" нуля. Усложнив схемотехнику, можно перейти к несинхронной передаче. Ретранслятор полученные модулированные сигналы

ретранслирует всем узлам, используя оптические сигналы на частоте f1. Сообщение, содержащее данные, имеет адресную часть, идентифицирующую приемники данных.

Наличие ретранслятора позволяет узлам передавать данные, не имея собственных источников оптических сигналов – используется только источник в ретрансляторе. Изложенное решение позволяет узлам перемещаться относительно ретранслятора. Сложность в узлах устройств, управляющих описанным взаимодействием узлов, не зависит от количества последних.

Теперь рассмотрим вопрос выделения единственного источника данных из числа многих узлов, желающих стать источниками данных. Для этого используем ДКУ в следующей редакции. Все узлы получают непрерывный сигнал f0 от ретранслятора, и этот сигнал модулируют только те узлы, старший разряд кода приоритета которых равен единице. Модулированный сигнал возвращается в ретранслятор, который ретранслирует его всем узлам на частоте f1. Если узлы в старшем разряде кода приоритета имели "0" и получили сигнал f1, то они прекращают борьбу за право быть источником. Оставшиеся узлы выполняют указанные действия со следующим разрядом кода приоритета и т.д. В результате будет выделен единственный узел, который и становится источником.

Ретранслятор может быть размещен среди узлов. Для этого должна быть использована отражающая свет среда, на которую посылает свет ретранслятор, и отраженный от среды свет поступает к узлам и от узлов к ретранслятору. При необходимости такое расположение позволяет разместить средства ретрансляции в каждом узле.

3. Заключение

Полученные решения направлены на ускорение важных процессов в системах обработки данных и управления – устранения конфликтов при доступе к ресурсам, распределенной обработки данных, распределенной коммутации. С этой целью разработаны механизмы со следующими возможностями:

– децентрализованное приоритетное управление (ДПУ) позволяет узлам системы разрешать без выделенного центра конфликты доступа к ресурсам;

– групповые команды и программы, совмещающие передачу данных с распределенными вычислениями, позволяют быстро оценивать состояние системы и воздействовать на ее структуру и функционирование;

– распределенная коммутация непосредственно, без привлечения промежуточных активных устройств, связывает бесконфликтно источники данных с приемниками. Конфликт возможен только на входе в приемник, но он разрешается с привлечением ДПУ.

Значительная часть приведенных в докладе результатов возможна только с привлечением оптических средств, интенсивно разрабатываемых в настоящее время во многих исследовательских центрах. Следует отметить, что с России вопросам применения оптики в вычислительных системах значительное внимание уделял В.С. Бурцев [24].

Полученные решения также полезны при реализации четырех "само" инициативы "Autonomic Computing" фирмы IBM [1], требующих для управления функционированием систем обработки данных децентрализованного управления взаимодействием узлов и особо быстрой реакции на события.

Литература

1. An architectural blueprint for autonomic computing. Autonomic Computing White Paper, Third Edition. IBM. 2005.
2. Gilbreath G.C. et al. Large-aperture multiple quantum well modulating retroreflector for free-space optical data transfer on unmanned aerial vehicles// Opt. Eng. 2001. no.7. P. 1348-1356.
3. Rabinovich W.S. et al. Free space optical data links using cat's eye modulating retroreflectors. http://esto.nasa.gov/conferences/estc2003/papers/A6P3(Rabinovich).pdf
4. Rabinovich W.S. et al. 45-Mbit/s cat's-eye modulating retroreflectors// Optical Engineering. 2007. Vol. 46. no. 10. P. 1-8.

5. Green W.M., Rooks M.J., Sekaric L., Vlasov Y.A. Ultra-compact, low RF power, 10 Gb/s silicon Mach-Zehnder modulator // Optics Express. 2007. Vol. 15. Issue 25. P. 17106-17113.

6. Made in IBM Labs: Breakthrough Chip Technology Lights the Path to Exascale Computing http://www-03.ibm.com/press/us/en/pressrelease/33115.wss

7. Стецюра Г.Г. Методы совмещения вычислений и передачи данных в мультипроцессорных системах и локальных сетях. М.: Ин-т пробл. упр., 2005. 86 С. http://www.ipu.ru/labs/lab31kom/ggs.zip

8. Kaajakari V. Practical MEMS. Small Gear Publ. 2009. 496 P.

9. Suh J. Y. et al. Modulated optical transmission of subwavelength hole arrays in metal-VO_2 films// Applied Physics Letters. 2006. Vol. 88. P. 133115-1 – 133115-3.

10. Huber R. et al. THz Slow Motion of an Ultrafast Insulator-Metal Transition in VO_2: Coherent Structural Dynamics and Electronic Correlations. Ultrafast Phenomena XVI. Springer Series in Chemical Physics. 2009. Vol. 92. Part 3. P. 179-181. http://www.springerlink.com/content/t5737480x7393g27/

11. Виноградова О.П. и др. Синтез и свойства нанокристаллов диоксида ванадия в силикатных пористых стеклах// Физика твердого тела. 2008. том 50. вып. 4. С. 734-740. http://journals.ioffe.ru/ftt/2008/04/p734-740.pdf

12. Kulcke W. et al. Fast, Digital-Indexed Light Deflector // IBM Journal. 1964. Jan. P. 64-67.

13. Прангишвили И.В., Стецюра Г.Г. Микропроцессорные системы. М.: Наука. 1980. 237 С.

14. Прангишвили И.В., Подлазов В.С., Стецюра Г.Г. Локальные микропроцессорные вычислительные сети. М.: Наука. 1984. 176 С.

15. Подлазов В.С., Стецюра Г.Г. О потребности в новых логических элементах//Журнал радиотехники и электроники РАН (электронный журнал). 2009. №6. http://jre.cplire.ru/jre/journal.html

16. The I^2C-Bus Specification Version 2.1. Philips. 2000. 46 P.

17. Стецюра Г.Г. Быстрые распределенные алгоритмы получения взаимно согласованных решений в системах жесткого реального времени// Автоматика и телемеханика. 2008. №10. С. 162-167.

18. Стецюра Г.Г. Быстрые децентрализованные алгоритмы устранения конфликтов и тупиков при доступе к ресурсам в системах обработки данных и управления//Автоматика и телемеханика. 2010. №4. С. 181-190.

19. Стецюра Г.Г. Совмещение вычислений и передачи данных в системах с коммутаторами. Автоматика и телемеханика. 2008. № 5. С. 170-179.

20. Sixto Ortiz Jr. Active Networks: The Programmable Pipeline//Computer. 1998. Vol. 31. no. 8. P. 19-21.

21. Hussain S. A. Active and programmable networks for adaptive architectures and services. Auerbach Publ. 2007. P. 329.

22. Стецюра Г.Г. Уменьшение сложности распределенного коммутатора для параллельных систем обработки данных // Автоматика и телемеханика. 2010. №5. С. 147-154.

23. Method and apparatus for free-space optical interconnects between arbitrary locations in a field using lenses, steering elements and a curved reflecting surface//Patent Application Publication Jun. 21, 2007 US 2007/0139749 A1

24. Бурцев В.С. Параллелизм вычислительных процессов и развитие архитектуры суперэвм (сборник статей) М.: Торус пресс. 2006. 414 С.

Перенос излучения в природных и искусственных средах и супервычисления*

Т.А. Сушкевич, С.А. Стрелков, С.В. Максакова, В.В. Козодеров, Б.А. Фомин, А.Н. Волкович, А.Б. Гаврилович, Е.В. Дмитриев, Л.Д. Краснокутская, С.Д. Устюгов, В.П. Шари, В.А. Фалалеева, П.П. Григорьева, А.К. Куликов

Институт прикладной математики им. М.В. Келдыша РАН

Настоящая работа ориентирована на приложения в аэрокосмических системах наблюдений теории переноса излучения в диапазоне спектров солнечного и собственного излучения от жесткого ультрафиолета до миллиметровых волн. Особое внимание уделяется современным проблемам информационно-математического обеспечения нанодиагностики природных и техногенных сред, опасных явлений и объектов на основе аэрокосмического гиперспектрального дистанционного зондирования.

1. Введение

С 1992 года, когда на "Саммите Земли" в Рио-де-Жанейро была торжественно принята Рамочная конвенция ООН об изменении климата, РКИК (Framework Convention on Climate Change, UN FCCC) — Соглашение, подписанное более чем 180 странами мира, включая Россию, все страны бывшего СССР и все промышленно развитые страны, об общих принципах действия стран по проблеме изменения климата, практически все страны, входящие в ООН, приобщились к климатическим проблемам. РКИК вступила в силу 21 марта 1994 года (Россия ратифицировала РКИК в 1994).

С 7 по 18 декабря 2009 года в Копенгагене проходил климатический саммит ООН. Войдет ли данное событие в новейшую глобальную историю? И если войдет, то, с какими глобальными процессами будет связан прошедший климатический саммит? Задолго до начала саммита планируемая международная конференция преподносилась как всемирно-историческое событие, способное предопределить вектор развития человечества на десятилетия вперед. Итог известен. По мнению широкого круга аналитиков и экспертов крупнейший в истории международный климатический форум, в котором приняли участие около 20 000 участников, а также более 100 глав государств из 193 стран-членов ООН, завершился провалом. "Лучше, чем ничего" — так оценили итоги своей почти двухнедельной работы участники климатической конференции.

Непозволительная роскошь в кризисное время — заседать в широком составе довольно длительный срок и не принимать важных документов. Тем не менее, кое-каких результатов достичь удалось. Например, понимания, что ***климатические проблемы в силу глобальности характера, "не имеют ответственных и виноватых"***, в связи с чем, весьма сложно договориться о том, кто и в каком размере будет компенсировать ущерб окружающей среде. "Тень мирового кризиса витала над Копенгагеном". Для широкого круга лидеров суверенных государств, прибывших на климатический саммит в Копенгаген, тема парниковых выбросов и всемирного потепления была не столь важна по сравнению с перспективой долгового кризиса 2008 года.

Ученых — специалистов по глобальным изменениям окружающей среды и климата пригласили, но к их мнению не прислушивались... В настоящее время мировое научное сообщество располагает практически достаточными фундаментальными знаниями и научным потенциалом, чтобы, объединив совместные усилия, провести достоверные комплексные и системные исследования на основе "сценариев", реализуемых на суперкомпьютерах

*Работа выполнена при поддержке РФФИ (проекты 09-01-00071, 11-01-00021).

с привлечением данных длительных временных рядов космических наблюдений двойного назначения. Однако такого объединения ученых не происходит...

И сейчас, когда в России объявлены приоритеты "модернизации" и прорывные направления, среди которых видим "Информационно-телекоммуникационные системы", в том числе "супервычисления" и "грид-системы", а также "Рациональное природопользование" (в частности, влияние на экологическую и климатическую систему последствий естественно-природных и техногенных катастроф), НЕОБХОДИМО консолидироваться (поскольку задачи комплексные) и вновь занять ведущие позиции в компьютерном моделировании радиационных задач дистанционного зондирования Земли и других планет, радиационного баланса Земли, радиационного форсинга и радиационных блоков в моделях климата и прогноза (где до сих пор используются упрощенные плоские приближения для расчета переноса солнечного излучения) [1, 2].

2. Постановка задачи математического моделирования

Радиационное поле системы "атмосфера – земная поверхность (суша, океан)" — это ***неотъемлемая компонента*** биосферы, экосистемы и климата Земли, с одной стороны. С другой стороны, радиационные характеристики системы являются ***носителями информации*** о состоянии окружающей среды, атмосферы, облачности, океана, гидрометеоров и всевозможных выбросов с загрязняющими примесями (последствия техногенных аварий, военных действий, лесных и степных пожаров, извержений вулканов и т.д.), а также промышленной и транспортной инфраструктуры. ***Электромагнитное излучение***, регистрируемое разными средствами, является ***основным источником информации*** о строении и физических свойствах планетных атмосфер и поверхностей при дистанционном зондировании. Для пассивных систем наблюдений ***источниками излучения*** являются ***внешний солнечный поток*** коротковолнового диапазона спектра (ультрафиолетовый, видимый, ближний инфракрасный) и ***собственное излучение*** планеты длинноволнового диапазона спектра (инфракрасный, миллиметровый), когда применимо квазиоптическое приближение теории переноса излучения.

В рамках развития вычислительных средств рассматриваются следующие модели переноса излучения.

I. Спектральная, пространственная и угловая структуры поля яркости — интенсивности светового поля (солнечного излучения) при известных условиях освещения рассчитываются как решения общей краевой задачи для уравнения переноса.

II. Спектральные и пространственные структуры интегральных характеристик поля излучения рассчитываются как решения задач, отвечающих (математически) точным или разной степени приближенности линейным и нелинейным моделям, которые получаются из интегро-дифференциального уравнения переноса с помощью аппарата разложений решения по сферическим функциям, при контролируемых условиях и ограничениях.

Нас интересует проблема расчета радиационного поля Земли в масштабах всей планеты. Радиационные проблемы требуют учета эффектов, связанных с поверхностью Земли. Проблемы энергетики и радиационного баланса Земли, когда Солнце играет роль источника излучения, обычно решаются в приближении плоского слоя без разделения вкладов атмосферной радиации и излучения земной поверхности, которое учитывают с некоторым усредненным альбедо. В нашем подходе атмосфера рассматривается как элемент "оптической" системы переноса излучения и суммарное излучение САЗ рассчитывается с использованием оптического передаточного оператора (ОПО), который формулируется на базе математического аппарата линейно-системного подхода и интеграла "суперпозиции".

Рассматривается общая краевая задача (ОКЗ) для кинетического уравнения переноса излучения в сферической системе атмосфера-Земля, освещаемой внешним параллельным солнечным потоком. На основе теории передаточного оператора и метода функций влия-

ния САЗ факторизуется на подобласти с различными оптическими свойствами и разными радиационными режимами. На основе линейно-системного подхода построено обобщенное решение задачи с оптическим передаточным оператором (ОПО), позволяющим учитывать пространственно неоднородную (мозаичную) подстилающую поверхность, а также гетерогенную структуру атмосферы (приземный слой, многоярусная облачность, стратосфера, мезосфера). Ядрами ОПО являются функции влияния. Функция влияния каждой подобласти определяется как решение первой краевой задачи (ПКЗ) для кинетического уравнения и является универсальной характеристикой системы переноса излучения, инвариантной относительно конкретных структур неоднородностей на границах, отражающих и пропускающих излучение.

Теоретическое построение и алгоритмы расчета оптического передаточного оператора основаны на теории обобщенных решений, теории интегральных преобразований обобщенных функций и рядов общей теории регулярных возмущений (асимптотический метод, когда решение выражается в виде рядов по малому параметру). Подход, разработанный на этих строгих математических основах, называем методом функций влияния [2]. Общность разработанной методики состоит в том, что она распространяется на разные диапазоны и условия дистанционного зондирования. Важно, чтобы "сценарий" и атмосферный канал рассматривались в рамках теории переноса излучения. Поэтому предпочтительнее избегать частого употребления термина "оптический", который сужает область применимости.

3. Нанодиагностика и супервычисления

Важнейшей современной тенденцией развития подходов и технологий дистанционного аэрокосмического зондирования, наряду с улучшением пространственного и спектрального разрешения, является переход от измерений относительных градаций яркостей, представляемых с помощью многоспектральных изображений (обычно не более десятка спектральных каналов), к получению абсолютно калиброванных данных в нескольких десятках, сотнях и даже тысячах каналов (гиперспектральные измерения). Информационная продукция, получаемая в результате обработки многоспектральных и гиперспектральных данных, открывает широкие возможности реализации инновационных технологий, главная особенность которых — получение количественных оценок параметров состояния наблюдаемых объектов природно-техногенной сферы в сравнении с их качественными оценками в традиционных подходах, а также установление элементного состава объектов. В изображениях от условных "псевдо-цветов" можно перейти к "природным" цветам. Цель исследований — теоретически обосновать возможности новых перспективных гиперспектральных методик аэрокосмического и наземного дистанционного зондирования системы атмосфера-Земля по спектрам солнечного и собственного излучения.

В настоящее время возросла актуальность таких исследований в связи с подписанием Международного Соглашения (более 40 стран) по организации международного глобального проекта по изучению Земли, международной космической системы аэрокосмического и наземного глобального мониторинга, международных Центров оперативной космической информации по катастрофическим и экологически опасным явлениям, а также в связи с новым этапом развития нанотехнологий для космических исследований, в частности, гиперспектрального (высокого спектрального разрешения) дистанционного зондирования объектов разной природы. Возможно, уже и не такая далекая перспектива - использование данных аэрокосмического дистанционного зондирования для решения проблем оперативного и системного управления территориями для обеспечения их устойчивого развития.

Новые перспективные возможности математического моделирования атмосферной радиации Земли связаны с разработкой информационно-математической системы для широкой области приложений на суперкомпьютерах и многопроцессорных вычислительных кластерах с распараллеливанием вычислений и распределением ресурсов, а также внедрением

GRID-технологий. В США, Японии, Германии, Англии, Франции, России появились суперкомпьютеры нового поколения, ориентированные на массовый параллелизм. Необходимо отметить, что сложные большие трудоемкие многомерные задачи пока что использовались для выполнения стратегических космических проектов.

Для диагностики и прогнозирования требуется серьезное исследование глобальных систем наблюдений и численное моделирование возможности дистанционного зондирования и мониторинга на основе "сценариев" развития в атмосфере и на земной поверхности техногенных чрезвычайных аварий и естественно-природных катастроф.

4. О супервычислениях и параллельных алгоритмах

Даже на современных высокопроизводительных вычислительных системах стоят проблемы скорости вычислений и оптимальной организации распараллеливания расчета при больших размерностях разностной сетки, а также передачи больших массивов результатов расчета по сетям от суперкомпьютера к рабочей станции оператора для последующей обработки.

Библиотека программ численного решения краевых задач теории переноса излучения в рассеивающих, поглощающих и излучающих средах (атмосфера, океан, облачность, дымы, гидрометеоры, водные бассейны) составляется из набора программ на Fortran, каждая из которых позволяет рассчитывать радиационные характеристики при заданных модели и методике (краевая задача теории переноса, геометрия, численный метод и т.д.) в определенном диапазоне длин волн.

Предложенная архитектура программного обеспечения с функциональным наполнением, ориентированным на решение задач мониторинга развития и оценки последствий воздействия техногенных аварий и природных катастроф, а также природно-ресурсных, экологических, геоэкоинформационных и т.п. задач, позволяет осуществлять модификацию и адаптацию вычислительно-информационной системы применительно к конкретным проблемам математического моделирования радиационных процессов в системе Земля-атмосфера или восстановления набора параметров зондируемой среды.

Используем следующие приемы распараллеливания вычислений:
(1) распределенные вычисления по физическим моделям:
— многоспектральные (по длине волны);
— оптико-геофизическая погода (по коэффициентам общей краевой задачи);
— по источникам;
(2) распределенные вычисления на основе методического распараллеливания — декомпозиции краевых задач:
— по подобластям;
— по параметрам вектора функций влияния;
— по параметрам вектора пространственно-частотных характеристик;
— по компонентам векторных функционалов;
(3) алгоритмическое распараллеливание для многомерных моделей:
— однократное рассеяние по характеристикам;
— многократное рассеяние по интегралам столкновений и по подобластям с разными сеточно-характеристическими схемами.

БОЛЬШИЕ многомерные вычислительные задачи в теории переноса излучения не адаптировались к алгоритмам решения задач алгебры с использованием теории графов, которые разрабатывал В.В. Воеводин, с одной стороны. С другой стороны, архитектура и ресурсы памяти первых отечественных многопроцессорных вычислительных систем не позволяли реализовать эффективно параллельные алгоритмы. В задачах космических исследований используется Фортран (с прямым доступом к файлам) для обеспечения преемственности и наследования огромного накопленного потенциала. Не было подходящих

компиляторов.

В течении 20 лет мы искали свои подходы к организации супервычислений с распараллеливанием алгоритмов и проводили опытные испытания программной системы на всех доступных суперкомпьютерах. С 2002 года приспособились к суперЭВМ в МСЦ, но при переходе на многоядерные процессоры и 64-разрядную память обнаружили потерю точности. Тестовыми являются результаты, полученные на БЭСМ-6 и на основе которых шла апробация переносимости программного обеспечения на разные ЭВМ. Скорее всего, проблема в компиляторах Фортрана, которые почему-то оказываются некачественными (подавали несколько рекламаций разным разработчикам). Но исправленных компиляторов не могли дождаться, поскольку ЭВМ уже выводили из строя...

Последние надежды на гибридные суперкомпьютеры с приличными компиляторами фирмы Portland Group (США). Данное программное обеспечение не имеет аналогов, а многолетняя репутация фирмы–разработчика позволяет надеяться на высокое качество упомянутого продукта. Надеемся, что использование этих компиляторов позволит как значительно упростить и ускорить разработку, переносимость и адаптацию прикладных программ для создаваемого суперкомпьютера, так и активировать и направить в нужную сторону отечественные исследования и разработки в данной области.

В последние годы, наконец-то, создали суперкомпьютеры соответствующего уровня производительности, в том числе и в России, а не только в США, Японии, Китае, ЕС, Индии и т.д.! В космосе сейчас присутствуют более 40 стран, а проблемами климата, прогноза погоды и экологической безопасности озабочены все страны ООН. У военных интерес к космосу поддерживался с начала космической эры. Если в Японии первый суперкомпьютер установили в "Центре по изучению климата" , то в США самые мощные суперкомпьютеры устанавливают в Центрах NASA...

Стоит ЗАДАЧА оценивания информационного содержания гиперспектральных аэрокосмических измерений и заполнения существующего пробела в понимании реальных особенностей формирования спектральных образов наблюдаемых объектов и процессов для НАНОДИАГНОСТИКИ их состояния. ***Необходимо обосновать оптимальное число измерительных каналов гиперспектрометров, их ширины и расположения по спектру длин волн и миниспутников на основе введенных информационных мер распознавания объектов и опасных явлений.***

НАУЧНАЯ ИДЕЯ основана на использовании существенных различий в спектральном ходе поглощения, рассеяния, излучения и пропускания основных компонент системы атмосфера-Земля и спектральных характеристик отражения объектами природно-техногенной сферы для выделения интервалов длин волн спектра многократно рассеянного солнечного и собственного излучения, информативных в отношении конкретных компонент, и на этой основе ИДЕНТИФИЦИРОВАТЬ КОМПОНЕНТЫ ПО ИХ СПЕКТРАЛЬНЫМ ХАРАКТЕРИСТИКАМ.

5. Заключение

СОВРЕМЕННАЯ ЗАДАЧА ДИСТАНЦИОННОГО ЗОНДИРОВАНИЯ И МОНИТОРИНГА состоит уже не столько в том, чтобы сделать космический снимок с использованием оптико-электронных аппаратов и цифровых каналов передачи космических данных на землю, определить альбедо или яркость земной поверхности (объектов) или провести радиационную коррекцию, ВАЖНЕЕ ВОССТАНОВИТЬ параметры состояния и количественные данные природных и других объектов по их спектральным образам.

В масштабах планеты стоит актуальная проблема создания международного глобального мониторинга Земли с целью исследования её эволюции и прогнозирования естественно-природных стихийных бедствий и антропогенно-техногенных катастрофических процессов.

Это ГРАНДИОЗНЫЕ ЗАДАЧИ, которые охватывают ряд важных направлений фун-

даментальных исследований в разных областях знаний (математики, физики, химии, биологии, геофизики, метеорологии, инженерно-конструкторских разработок), имеющих междисциплинарный характер и тематически объединяемых задачами комплексного изучения окружающей природной, космической и техногенной среды с использованием кинетической теории переноса излучения, спектральных методов молекулярной физики, методов и средств космических исследований и космического землеведения с использованием перспективных гиперспектральных технологий дистанционного зондирования и НАНОДИАГНОСТИКИ, МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ и эффективных ЧИСЛЕННЫХ МЕТОДОВ с распараллеливанием СУПЕРВЫЧИСЛЕНИЙ на современных и перспективных суперкомпьютерах.

В масштабах планеты стоит актуальная проблема создания международного глобального мониторинга Земли с целью исследования её эволюции и прогнозирования естественно-природных стихийных бедствий и антропогенно-техногенных катастрофических процессов.

Для диагностики и прогнозирования требуется серьезное исследование глобальных систем наблюдений и численное моделирование возможности дистанционного зондирования и мониторинга на основе "сценариев" развития в атмосфере и на земной поверхности техногенных чрезвычайных аварий и естественно-природных катастроф.

Всемирная система мониторинга и иерархия моделей — главные инструменты для прогнозирования изменений в природных процессах и разделения естественных и антропогенных воздействий.

В последние годы специалисты в области климата и глобальных изменений окружающей среды признали важность радиационного форсинга, который может быть не менее 40%, что привело к необходимости совершенствования радиационных блоков в моделях климата, циркуляции и т.д.

Одно из важных направлений Федеральной программы по космосу связано с использованием космических данных для оптимального управления регионами с целью обеспечения их устойчивого развития.

Естественно стоит вопрос: а в каких вузах готовят молодых специалистов для работы в этих перспективных высокотехнологичных направлениях? И достаточно ли для их подготовки овладение только персональными компьютерами или графическими системами и ГИС? Мой ответ — без супервычислений и суперкомпьютеров прорывных достижений в глобальном мониторинге Земли не получится!

Литература

1. Сушкевич Т.А. О пионерских работах по математическому моделированию радиационного поля Земли при освоении космоса / Методы и алгоритмы обработки спутниковых данных // Современные проблемы дистанционного зондирования Земли из космоса. Физические основы, методы и технологии мониторинга окружающей среды, потенциально опасных явлений и объектов. Институт космических исследований РАН. Сборник научных статей. Выпуск 5. Том 1. М.: ООО "Азбука-2000" , 2008. С. 165–180. ISSN 2070-7401.

2. Сушкевич Т.А. Математические модели переноса излучения. М.: БИНОМ. Лаборатория знаний, 2005. 661 с.

Применение индексного метода глобальной оптимизации при решении обратных задач химической кинетики*

М.В. Тихонова[1], В.В. Рябов[2]

БашГУ[1], ННГУ им. Лобачевского[2]

Построение математических моделей и решение обратных задач физической химии связаны с минимизацией отклонения между расчетными и экспериментальными данными. Для этого требуется многократное решение вычислительно трудоёмких прямых задач. Для изучения механизмов сложных химических реакций металлокомплексного катализа, исследуемых в ИНК РАН, и определения их кинетических параметров в рамках данной работы предложено использовать параллельный индексный метод глобальной оптимизации, разрабатываемый в ННГУ им. Лобачевского. Метод использует редукцию размерности на основе кривых Пеано и информационно-статистический подход, дополненный схемой построения множественных отображений (вращаемые развёртки), позволяющих эффективно использовать сотни процессоров. Также для ускорения поиска используется смешанная локально-глобальная стратегия и другие модификации индексного метода.

1. Введение

Построение математических моделей сложных химических реакций предполагает наличие в них неизвестных кинетических параметров (константы скоростей, энергии активации и частота столкновений реагирующих молекул элементарных стадий), которые можно найти, решая задачу минимизации отклонения между расчётными данными (прямая задача) и данными натурных экспериментов. Таким образом, возникает задача идентификации математической модели (обратная задача химической кинетики), которая в общем случае является задачей глобальной оптимизации.

Многие физико-химические задачи предполагают значительный объем вычислений, обеспечивающих, тем не менее, достаточно низкую точность. Именно к таким задачам относятся обратные задачи изучения механизмов сложных химических реакций, представляющие собой оптимизационные задачи циклического решения множества прямых задач – систем дифференциальных и алгебраических уравнений. Для кинетики сложных химических реакций характерно наличие быстро и медленно меняющихся переменных. Так как стадии реакций протекают с различными скоростями, то решение прямых кинетических задач осложняются жесткостью систем дифференциальных уравнений, описывающих механизмы этих реакций. В связи с этим актуальным является решение обратных задач с использованием эффективных численных методов глобальной оптимизации на высокопроизводительных многопроцессорных вычислительных системах (суперкомпьютерах).

2. Постановка задачи

Прямая кинетическая задача для изотермической нестационарной модели в закрытой системе представляет собой задачу Коши для системы обыкновенных дифференциальных уравнений:

$$\frac{dx_i}{dt} = F_i,\ i = 1..M; \quad F_i = \sum_{j=1}^{N} S_{ij} w_j; \tag{1}$$

$$w_j = k_j \prod_{i=1}^{M} (x_i)^{|\alpha_{ij}|} - k_{-j} \prod_{i=1}^{M} (x_i)^{|\beta_{ij}|}; \tag{2}$$

* Работа выполнена при поддержке совета по грантам Президента Российской Федерации (грант № НШ-64729.2010.9).

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}.

Поскольку часть констант k_j, k_{-j}, как правило, неизвестны, возникает задача идентификации математической модели или обратная кинетическая задача, которая представляет собой задачу минимизации функционала отклонения между расчетными и экспериментальными данными:

$$F = \sum_{i=1}^{n} \sum_{j=1}^{M} \left| x_{ij}^{p} - x_{ij}^{э} \right| \to \min, \tag{3}$$

где
x_{ij}^{p} – расчетные значения концентраций наблюдаемых веществ, (*мольные доли*);
$x_{ij}^{э}$ – экспериментально полученные значения концентраций наблюдаемых веществ, (*мольные доли*);
n – количество точек эксперимента.

Так как при поиске констант скоростей элементарных стадий они могут попасть в область, где система дифференциальных уравнений, описывающая реакцию, может оказаться жесткой, для решения прямой задачи используется метод Мишельсена с автоматическим выбором шага [1].

Поскольку выходные данные модели (x_{ij}^{p}) зависят нелинейно от значений констант k_j, k_{-j}, задача (3) является в общем случае задачей многоэкстремальной оптимизации.

При наличии заданных отрезков варьирования констант k_j, k_{-j} (входных параметров), при условии липшицевости функционала F и при заданной точности ε требуемой оценки глобального оптимума такая задача является NP-трудной (с ростом числа входных параметров вычислительные затраты растут экспоненциально).

Для решения задачи (3), помимо тривиального равномерного перебора и методов Монте-Карло, существует целый ряд эффективных в том или ином смысле методов, использующих априорную информацию о целевой функции (липшицевость) и поисковую информацию (значения функции в точках испытаний, выбранных на предыдущих шагах). Для большинства подобных методов характерно построение неравномерного покрытия области поиска, более плотного в окрестности глобального оптимума.

Для нахождения энергии активации и частоты столкновений реагирующих в элементарной стадии молекул используется уравнение Аррениуса:

$$k = Ae^{-\frac{Ea}{RT}}, \tag{4}$$

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*.

2.1 Реакция карбоалюминирования олефинов и ацетиленов

Для применения новой методики идентификации была выбрана математическая модель реакции каталитического карбоалюминирования олефинов и ацетиленов с помощью триалкилаланов в присутствии комплексов переходных металлов. Эта реакция получила применение в лабораторной практике ИНК РАН как эффективный способ построения новых Me-C (метил-углерод), Et-C (этил-углерод) и C-C (углерод-углерод) связей [2].

В ИНК РАН г. Уфы предложен ряд схем, описывающих реакции каталитического карбоалюминирования олефинов и ацетиленов при различных катализаторах.

2.1.1 Et-C связи

Для исследования Et-C связи предложено две кинетические схемы:
1. 10 – стадийная схема с обратимой 1-й элементарной стадией (таблица 1);
2. 12 – стадийная схема с обратимой 1-й элементарной стадией (таблица 3).

Приоритетная задача – выбрать наиболее вероятную схему.

Таблица 1. 10-стадийная схема реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2.

1.	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} A_3$	$w_1 = k_1x_1x_2 - k_{-1}x_3$
2.	$A_2 + A_3 \xrightarrow{k_2} A_4 + A_5$	$w_2 = k_2x_2x_3$
3.	$A_4 + A_6 \xrightarrow{k_3} A_7$	$w_3 = k_3x_4x_6$
4.	$A_2 + A_7 \xrightarrow{k_4} A_4 + A_8$	$w_4 = k_4x_2x_7$
5.	$A_4 \xrightarrow{k_5} A_9 + A_{10}$	$w_5 = k_5x_4$
6.	$A_6 + A_{10} \xrightarrow{k_6} A_{11}$	$w_6 = k_6x_6x_{10}$
7.	$A_2 + A_{11} \xrightarrow{k_7} A_4 + A_{12}$	$w_7 = k_7x_2x_{11}$
8.	$A_7 \xrightarrow{k_8} A_{13} + A_{14}$	$w_8 = k_8x_7$
9.	$A_6 + A_{14} \xrightarrow{k_9} A_{15}$	$w_9 = k_9x_6x_{14}$
10.	$A_2 + A_{15} \xrightarrow{k_{10}} A_4 + A_{16}$	$w_{10} = k_{10}x_2x_{15}$

где w_j – скорость j-й элементарной стадии, *1/ч*;
x_i – концентрация i-го вещества (соответствует A_i), *мольная доля*;
k_j – приведенная константа скорости j-й элементарной стадии, *1/ч*.

В качестве A_i выступают вещества, приведённые в таблице 2.

Таблица 2. Список веществ 10-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов

$A_1 = L_2ZrCl_2$, L_2=Cp	$A_9 = C_2H_6$
$A_2 = AlEt_3$	$A_{10} = L_2Zr(Cl)CH_2CH_2AlEt_2$
$A_3 = L_2ZrClEtClAlEt_2$	$A_{11} = L_2Zr(Cl)CH_2CH_2(R)CH_2CHAlEt_2$
$A_4 = L_2ZrEtClAlEt_3$	$A_{12} = EtAlCH_2CH_2CHRCH_2$
$A_5 = ClAlEt_2$	$A_{13} = H_2CCEtR$
$A_6 = CH_2CHR$	$A_{14} = L_2ZrHClAlEt_3$
$A_7 = CH_2CHEtRL_2ZrClAlEt_3$	$A_{15} = L_2ZrCH_2CH_2(R)ClAlEt_3$
$A_8 = CH_2CHEtRAlEt_2$	$A_{16} = Et_2AlCH_2CH_2R$

Таблица 3. 12-стадийная схема реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2.

1.	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} A_3$	$w_1 = k_1x_1x_2 - k_{-1}x_3$
2.	$A_2 + A_3 \xrightarrow{k_2} A_4 + A_5$	$w_2 = k_2x_2x_3$
3.	$A_4 + A_6 \xrightarrow{k_3} A_7$	$w_3 = k_3x_4x_6$
4.	$A_2 + A_7 \xrightarrow{k_4} A_4 + A_8$	$w_4 = k_4x_2x_7$
5.	$A_4 \xrightarrow{k_5} A_9 + A_{10}$	$w_5 = k_5x_4$
6.	$A_6 + A_{10} \xrightarrow{k_6} A_{11}$	$w_6 = k_6x_6x_{10}$
7.	$A_2 + A_{11} \xrightarrow{k_7} A_4 + A_{12}$	$w_7 = k_7x_2x_{11}$
8.	$A_7 \xrightarrow{k_8} A_{13} + A_{14}$	$w_8 = k_8x_7$
9.	$A_6 + A_{14} \xrightarrow{k_9} A_{15}$	$w_9 = k_9x_6x_{14}$
10.	$A_2 + A_{15} \xrightarrow{k_{10}} A_4 + A_{16}$	$w_{10} = k_{10}x_2x_{15}$
11.	$A_6 + A_{15} \xrightarrow{k_{11}} A_{17}$	$w_{11} = k_{11}x_6x_{15}$
12.	$A_{17} \xrightarrow{k_{12}} A_{14} + A_{18}$	$w_{12} = k_{12}x_{17}$

В качестве A_i выступают вещества, приведённые в таблице 4.

Таблица 4. Список веществ 12-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов

$A_1 = L_2ZrCl_2$, L_2=Cp	$A_{10} = L_2Zr(Cl)CH_2CH_2AlEt_2$
$A_2 = AlEt_3$	$A_{11} = L_2Zr(Cl)CH_2CH_2(R)CH_2CHAlEt_2$
$A_3 = L_2ZrClEtClAlEt_2$	$A_{12} = EtAlCH_2CH_2CHRCH_2$
$A_4 = L_2ZrEtClAlEt_3$	$A_{13} = H_2CCEtR$
$A_5 = ClAlEt_2$	$A_{14} = L_2ZrHClAlEt_3$
$A_6 = CH_2CHR$	$A_{15} = L_2ZrCH_2CH_2(R)ClAlEt_3$
$A_7 = CH_2CHEtRL_2ZrClAlEt_3$	$A_{16} = Et_2AlCH_2CH_2R$
$A_8 = CH_2CHEtRAlEt_2$	$A_{17} = L_2ZrCH_2CHRCH_2CH_2RClAlEt_3$
$A_9 = C_2H_6$	$A_{18} = H_2CCRCH_2CH_2R$

Натурный эксперимент по проведению реакции каталитического карбоалюминирования олефинов и ацетиленов проводился при температуре 22°C и начальных концентрациях (мольные доли):

$$x_1=0.009,\ x_2=0.541,\ x_6=0.450.$$

В ходе эксперимента наблюдались концентрации пяти веществ и выдавались их процентные соотношения, приведённые в таблице 5.

Таблица 5. Соотношения концентраций наблюдаемых веществ

Время (с.) \ концентрация (%)	A6	A8	A12	A14	A16
0	100	0	0	0	0
0.25	36	15	15	8	26
1.25	30	27	20	3	20
2.75	31	33	16	1	19
5	23	34	15	16	12
8	33	28	19	5	15
24	6	33	29	26	6

2.1.2 Me-C связи

Для исследования Me-C связи предложено две кинетические схемы:
1. 8 – стадийная схема с катализатором Cp_2ZrCl_2 (таблица 6);
2. 12 – стадийная схема с катализатором $(CpMe_5)_2ZrCl_2$ (таблица 8).

Приоритетная задача – найти энергии активации элементарных стадий предложенных схем при различных катализаторах.

Таблица 6. 8-стадийная схема реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2

1.	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} A_3$	$w_1 = k_1x_1x_2 - k_{-1}x_3$
2.	$A_2 + A_3 \underset{k_{-2}}{\overset{k_2}{\rightleftharpoons}} A_4 + A_5$	$w_2 = k_2x_2x_3 - k_{-2}x_4x_5$
3.	$A_4 + A_6 \xrightarrow{k_3} A_7$	$w_3 = k_3x_4x_6$
4.	$A_2 + A_7 \xrightarrow{k_4} A_4 + A_8$	$w_4 = k_4x_2x_7$
5.	$A_7 \xrightarrow{k_5} A_{13} + A_{14}$	$w_5 = k_5x_7$
6.	$A_6 + A_{14} \xrightarrow{k_6} A_{15}$	$w_6 = k_6x_6x_{14}$
7.	$A_6 + A_{15} \xrightarrow{k_7} A_{14} + A_{18}$	$w_7 = k_7x_6x_{15}$
8.	$A_2 + A_{15} \xrightarrow{K_8} A_4 + A_{16}$	$w_8 = k_8x_2x_{15}$

В качестве A_i выступают вещества, приведённые в таблице 7.

Таблица 7. Список веществ 8-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2

$A_1 = L_2ZrCl_2$, L_2=Cp	$A_{10} = L_2Zr(Cl)CH_2CH_2AlMe_2$
$A_2 = AlMe_3$	$A_{11} = L_2Zr(Cl)CH_2CH_2(R)CH_2CHAlMe_2$
$A_3 = L_2ZrClMeClAlMe_2$	$A_{12} = MeAlCH_2CH_2CHRCH_2$
$A_4 = L_2ZrMeClAlMe_3$	$A_{13} = H_2CCMeR$
$A_5 = ClAlMe_2$	$A_{14} = L_2ZrHClAlMe_3$
$A_6 = CH_2CHR$	$A_{15} = L_2ZrCH_2CH_2(R)ClAlMe_3$
$A_7 = CH_2CHMeRL_2ZrClAlMe_3$	$A_{16} = Me_2AlCH_2CH_2R$
$A_8 = CH_2CHMeRAlMe_2$	$A_{17} = L_2ZrCH_2CHRCH_2CH_2RClAlMe_3$
$A_9 = C_2H_6$	$A_{18} = H_2CCRCH_2CH_2R$

Таблица 8. 9-стадийная схема реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора $(CpMe_5)_2ZrCl_2$

1.	$A_1 + A_2 \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} A_3$	$w_1 = k_1x_1x_2 - k_{-1}x_3$
2.	$A_3 + A_6 \xrightarrow{k_2} A_{19}$	$w_2 = k_2x_3x_6$
3.	$A_2 + A_{19} \xrightarrow{k_3} A_3 + A_8$	$w_3 = k_3x_2x_{19}$
4.	$A_{19} \xrightarrow{k_4} A_1 + A_8$	$w_4 = k_4x_{19}$
5.	$A_{19} \xrightarrow{k_5} A_{13} + A_{20}$	$w_5 = k_5x_{19}$
6.	$A_6 + A_{20} \xrightarrow{k_6} A_{21}$	$w_6 = k_6x_6x_{20}$
7.	$A_6 + A_{21} \xrightarrow{k_7} A_{18} + A_{20}$	$w_7 = k_7x_6x_{21}$
8.	$A_{21} \xrightarrow{K_8} A_1 + A_{16}$	$w_8 = k_8 x_{21}$
9.	$A_2 + A_{21} \xrightarrow{K_9} A_3 + A_{16}$	$w_9 = k_8 x_2x_{21}$

В качестве A_i выступают вещества, приведённые в таблице 9.

Таблица 9. Список веществ 9-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора $(CpMe_5)_2ZrCl_2$

$A_1 = L_2ZrCl_2, \quad L_2 = CpMe_5$	$A_{12} = MeAlCH_2CH_2CHRCH_2$
$A_2 = AlMe_3$	$A_{13} = H_2CCMeR$
$A_3 = L_2ZrClMeClAlMe_2$	$A_{14} = L_2ZrHClAlMe_3$
$A_4 = L_2ZrMeClAlMe_3$	$A_{15} = L_2ZrCH_2CH_2\,(R)ClAlMe_3$
$A_5 = ClAlMe_2$	$A_{16} = Me_2AlCH_2CH_2R$
$A_6 = CH_2CHR$	$A_{17} = L_2ZrCH_2CHRCH_2CH_2RClAlMe_3$
$A_7 = CH_2CHMeRL_2ZrClAlMe_3$	$A_{18} = H_2CCRCH_2CH_2R$
$A_8 = CH_2CHMeRAlMe_2$	$A_{19} = L_2ZrC\,H_2CHMeRClClAlMe_2$
$A_9 = C_2H_6$	$A_{20} = L_2ZrHClClAlMe_2$
$A_{10} = L_2Zr(Cl)CH_2CH_2AlMe_2$	$A_{21} = L_2ZrC\,H_2C\,H_2RClClAlMe_2$
$A_{11} = L_2Zr(Cl)CH_2CH_2(R)CH_2CHAlMe_2$	

Натурные эксперименты по проведению реакций каталитического карбоалюминирования олефинов и ацетиленов в присутствии катализаторов Cp_2ZrCl_2 и $(CpMe_5)_2ZrCl_2$ проводились при трех температурах (15°C, 22°C и 30°C) и начальных концентрациях (мольные доли):

$$x_1=0.009,\ x_2=0.541,\ x_6=0.450.$$

В ходе эксперимента наблюдались концентрации пяти веществ (A_6, A_8, A_{13}, A_{16}, A_{18}) и выдавались их процентные соотношения.

2.2 Постановка задачи многоэкстремальной оптимизации

Алгоритмы, развиваемые Нижегородской научной школой многоэкстремальной оптимизации, предполагают следующую постановку задачи[1]:

$$\varphi^* = \varphi(y^*) = \min\{\varphi(y): \ y \in D\},$$
$$D = \{y \in R^N: \ a_i \le y_i \le b_i, \ 1 \le i \le N\}, \tag{5}$$

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. [4]), 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].

3. Индексный метод глобальной оптимизации

3.1 Редукция размерности и множественные отображения

Редукция многомерных задач к одномерным с помощью разверток имеет такие важные свойства, как непрерывность и сохранение равномерной ограниченности разностей функций при ограниченности вариации аргумента. Однако при этом происходит потеря части информации о близости точек в многомерном пространстве, так как точка $x \in [0,1]$ имеет лишь левых и правых соседей, а соответствующая ей точка $y(x) \in R^N$ имеет соседей по 2^N направлениям. А при использовании отображений типа кривой Пеано близким в N-мерном пространстве образам y', y'' могут соответствовать достаточно далекие прообразы x', x'' на отрезке [0,1]. Как результат, единственной точке глобального минимума в многомерной задаче соответствует несколько (не более 2^N) локальных экстремумов в одномерной задаче, что, естественно, ухудшает свойства одномерной задачи.

Сохранить часть информации о близости точек позволяет использование множества отображений

$$Y_L(x) = \{y^1(x), \ldots, \ y^L(x)\} \tag{6}$$

вместо применения единственной кривой Пеано $y(x)$ (см. [4], [7]). Каждая кривая Пеано $y^i(x)$ из $Y_L(x)$ может быть получена в результате поворота развертки вокруг начала координат. При этом найдется отображение $y^i(x)$, которое точкам многомерного пространства y', y'', которым при исходном отображении соответствовали достаточно далекие прообразы на отрезке [0,1], будет сопоставлять более близкие прообразы x', x''.

Максимальное число различных поворотов развертки, отображающей N-мерный гиперкуб на одномерный отрезок, составляет 2^N. Использование всех из них является избыточным. В

[1] Здесь применяются общепринятые обозначения для задач многоэкстремальной оптимизации. Следует помнить, что в приложении к задачам химической кинетики роль вектора x играет вектор кинетических параметров k_j, k_{-j}.

используемой схеме (см. [7]) преобразование развертки осуществляется в виде поворота на угол $\pm\pi/2$ в каждой из координатных плоскостей. Число подобных пар поворотов определяется числом координатных плоскостей пространства, которое равно $C_N^2 = \frac{N(N-1)}{2}$, а общее число преобразований будет равно $N(N-1)$. Учитывая исходное отображение, приходим к заключению, что данный способ позволяет строить до $N(N-1)+1$ развертки для отображения N-мерной области на соответствующие одномерные отрезки.

3.2 Параллельный индексный метод и локально-глобальная стратегия

Использование множества отображений $Y_L(x) = \{y^1(x), \ldots, y^L(x)\}$ приводит к формированию соответствующего множества одномерных многоэкстремальных задач

$$\min\{\varphi(y^l(x)) : x \in [0,1],\ 1 \le l \le L. \quad (7)$$

Каждая задача из данного набора может решаться независимо, при этом любое вычисленное значение $z = \varphi(y')$, $y' = y^i(x'')$ функции $\varphi(y)$ в i-й задаче может интерпретироваться как вычисление значения $z = \varphi(y')$, $y' = y^s(x''')$ для любой другой s-й задачи без повторных трудоемких вычислений функции $\varphi(y)$. Подобное информационное единство позволяет решать исходную задачу (5) путем параллельного решения индексным методом L задач вида (7) на наборе отрезков [0,1]. Каждая одномерная задача решается на отдельном процессоре. Для организации взаимодействия на каждом процессоре создается L очередей, в которые процессоры помещают информацию о выполненных итерациях.

Подробное описание решающих правил параллельного индексного алгоритма глобальной оптимизации приведено в работе [6].

Локально-адаптивный алгоритм является модификацией индексного метода глобального поиска, состоящей в том, что, начиная с некоторого шага, при выборе точек итераций используется дополнительная информация – текущие оценки плотности вероятности для расположения точки искомого оптимума. Оценки плотности определяются по значениям функционалов задачи, вычисленных в точках выполненных итераций. Таким образом, плотность переоценивается после каждой итерации, причем максимумы плотности соответствуют окрестностям точек текущих оптимальных значений. Подробно решающие правила локально-адаптивного метода приведены, например, в [6]. Существенным параметром этого метода является целое число $0 \le \alpha \le 30$, влияющее на характер сходимости. При $\alpha=0$ поиск носит глобальный характер, при $\alpha=30$ – локальный.

Смешанный алгоритм является модификацией индексного метода глобального поиска, состоящей в том, что, начиная с некоторого шага итерации, определяемые правилами индексного метода, чередуются с итерациями, определяемыми правилами локально-адаптивного алгоритма. Частота чередования является параметром метода.

4. Результаты вычислений

По экспериментальным данным поставлены 8 численных экспериментов, в результате которых были найдены константы скоростей стадий для 4-х схем реакции и для различных температур, в случае Me-C связей.

Поскольку данные натурных экспериментов содержат погрешности, достичь значений функционала F, достаточно близких к нулю, часто не удаётся. Однако численные оценки глобального оптимума, полученные модифицированным индексным методом, позволяют решить значительную часть поставленных задач.

Характерное время получения численных оценок глобального оптимума, приведённых в работе, – 2-3 недели (для одной реакции) последовательным индексным методом, тогда как параллельный метод при запуске на 90-150 вычислительных ядрах (при текущей схеме распараллеливания) позволяет получать аналогичные оценки за считанные часы.

4.1 Et-C связи

Оценки констант скоростей элементарных стадий для 10-ти и 12-ти стадийных схем реакции карбоалюминирования в присутствии катализатора Cp_2ZrCl_2 приведены в таблице 10 и 11, соответственно.

Сопоставления расчета эксперименту представлены на рисунках 1 и 2, соответственно.

Таблица 10. Константы скоростей элементарных стадий для 10-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2, *1/ч*

k_1 =	6215.977	k_4 =	9517.975	k_7 =	5023.656	k_{10} =	197.410
k_2 =	3582.618	k_5 =	311.589	k_8 =	3387.914	k_{11} =	6261.753
k_3 =	5165.258	k_6 =	8.304	k_9 =	2778.785		

Значение функционала в этой точке равно F = 3.040.

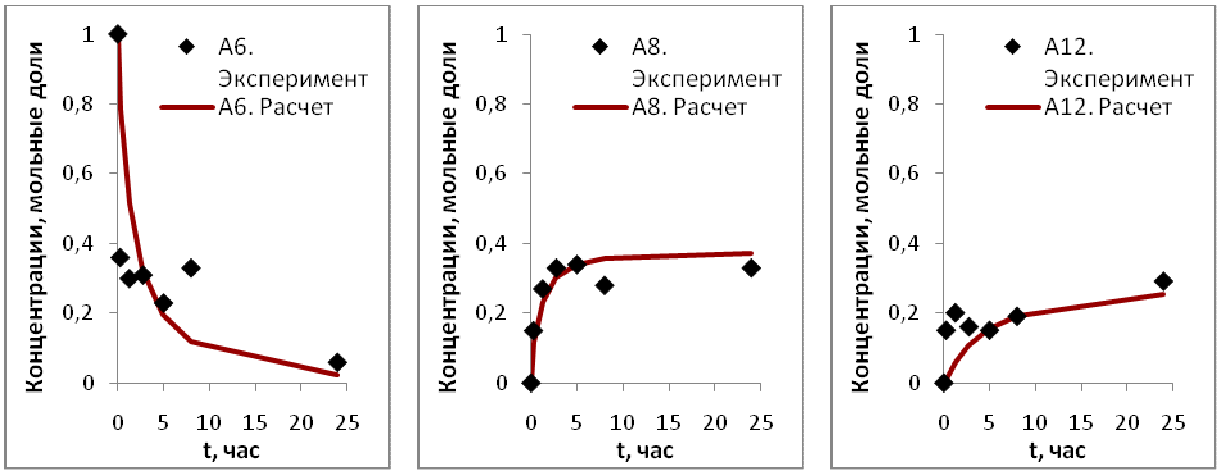


Рис. 1. Сопоставление расчетных и экспериментальных данных для 10-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2

Таблица 11. Константы скоростей элементарных стадий для 12-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2, *1/ч*

k_1 =	6215.977	k_4 =	9517.975	k_7 =	5023.656	k_{10} =	197.410	k_{13} =	6261.753
k_2 =	3582.618	k_5 =	311.589	k_8 =	3387.914	k_{11} =	8608.858		
k_3 =	5165.258	k_6 =	8.304	k_9 =	2778.785	k_{12} =	8280.793		

Значение функционала в этой точке равно F = 2.336.

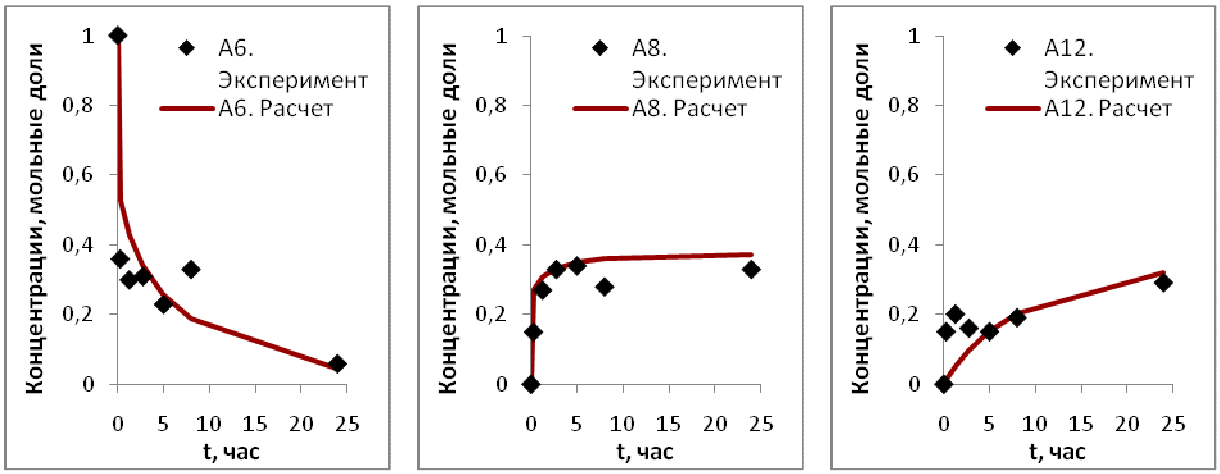


Рис. 2. Сопоставление расчетных и экспериментальных данных для 12-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2

Таким образом, 12-ти стадийная схема реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2 с константами скоростей стадий из таблицы 11 описывает экспериментальные данные лучше, чем 10-ти стадийная схема на том же наборе констант для соответствующих стадий.

4.2 Me-C связи

Из уранения Аррениуса (5) методом наименьших квадратов была построена прямая зависимость *lnk* от *1/T:*

$$\ln k = \ln A - \frac{E_a}{R} \cdot \frac{1}{T}, \tag{8}$$

из которой рассчитывались энергии активации и фактор частоты.

Константы скоростей k_i, энергии активации E_A, и частоты A столкновений реагирующих молекул элементарных стадий для 8-ми и 9-ти стадийных схем реакции карбоалюминирования в присутствии катализатора Cp_2ZrCl_2 и $(CpMe_5)_2ZrCl_2$ приведены в таблице 12 и 13, соответственно.

Таблица 12. Константы скоростей k_i, энергии активации E_A, и частоты A столкновений реагирующих молекул элементарных стадий для 8-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора Cp_2ZrCl_2

№ стадии	k_i, 1/ч			E_A, ккал	A, 1/ч
	15°C	22°C	30°C		
1 прямая	3651.282	963.296	6213.230	6.642	2.348E+08
2 прямая	5301.671	1223.001	5755.712	1.428	3.823E+04
5 прямая	2641.762	9064.790	6318.821	9.808	9.960E+10
6 прямая	3.825	3.520	5304.723	84.650	2.375E+64
7 прямая	5812.840	5284.581	8269.808	4.154	7.607E+06
8 прямая	452.127	198.221	2740.944	21.341	4.153E+18
1 обратная	2581.947	7485.201	4862.523	7.066	7.869E+08
2 обратная	7697.298	1150.064	8596.040	1.896	1.079E+05

Таблица 13. Константы скоростей k_i, энергии активации E_A, и частоты A столкновений реагирующих молекул элементарных стадий для 9-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора $(CpMe_5)_2ZrCl_2$

№ стадии	k_i, 1/ч			E_A, ккал	A, 1/ч
	15°C	22°C	30°C		
2 прямая	24.882	1094.217	1096.353	43.075	2.671E+34
3 прямая	5012.975	2963.111	8005.526	5.643	7.498E+07
4 прямая	6382.100	9440.219	9000.231	3.896	6.309E+06
5 прямая	554.666	1018.228	1972.817	14.648	7.506E+13
7 прямая	4311.682	7464.450	7793.124	6.749	6.357E+08
8 прямая	4818.578	1245.889	6315.464	3.595	1.553E+06
9 прямая	6772.312	6884.311	9357.148	3.776	4.782E+06
1 обратная	5377.660	7635.653	9823.151	6.936	1.024E+09

Зависимость (8) для для 1-й обратной стадии 9-ти стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора $(CpMe_5)_2ZrCl_2$ приведена на рисунке 3.

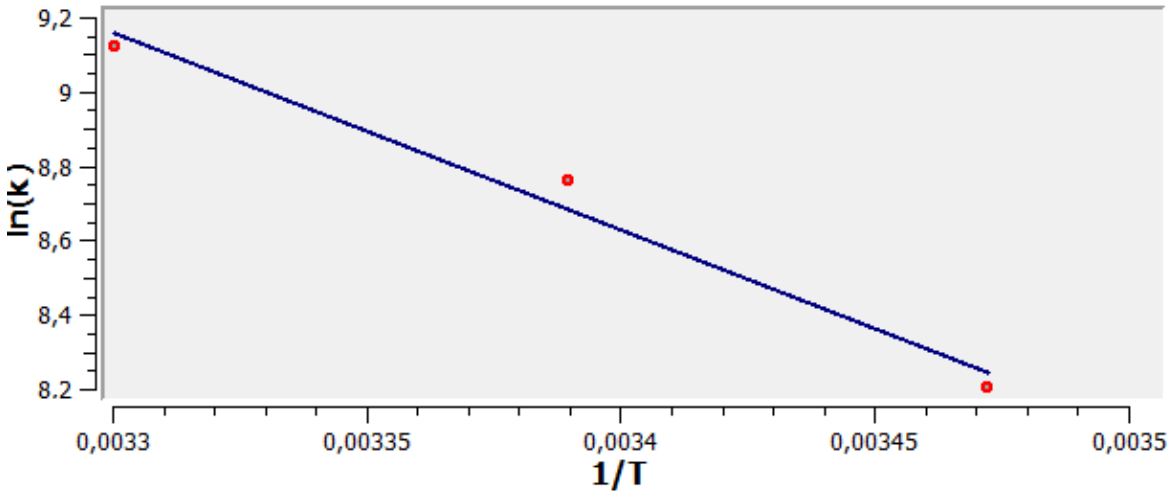


Рис. 3. Зависимость *lnk* от *1/T* для 1-й прямой элементарной стадии 9-стадийной схемы реакции карбоалюминирования олефинов и ацетиленов в присутствии катализатора $(CpMe_5)_2ZrCl_2$

Схемы реакции карбоалюминирования олефинов и ацетиленов, представленные в таблицах 6 и 8, имеют структурно-одинаковые 1-ю и 2-ю элементарные стадии, отличающиеся только используемым катализатором. Сравнивая значения энергий активации из таблиц 12 и 13 для этих стадий, можно сделать вывод: энергия активация первой обратной стадии для обеих схем имеет сравнительно одинаковое количественное значение, но при использовании катализатора Cp_2ZrCl_2 для осуществления 2-й элементарной стадии молекулы должны преодолеть меньший энергетический барьер, чем при использовании катализатора $(CpMe_5)_2ZrCl_2$.

5. Заключение

Применение параллельного индексного метода глобальной оптимизации позволяет получать хорошие численные оценки констант скоростей элементарных стадий реакции (и сопутствующих величин), что продемонстрировано на примере реакции карбоалюминирования олефинов и ацетиленов, за приемлемое время (применяемые ранее методики требовали нескольких месяцев вычислений и частичного ручного контроля с применением эвристик). Однако и в рамках данного исследования пока остаются открытыми такие вопросы, как выбор диапазонов изменений искомых кинетических констант (область поиска глобального оптимума) и учёт

структурных связей между задачами (например, для одной и той же реакции при разных температурах кинетические константы растут с ростом температуры). Один из возможных способов учёта таких связей – построение аггрегированной задачи, что, однако, может привести к появлению функциональных ограничений и увеличению размерности в разы.

Литература

1. Тихонова М.В., Губайдуллин И.М., Спивак С.И. Численное решение прямой кинетической задачи методами Розенброка и Мишельсена для жестких систем дифференциальных уравнений // Ж. Средневолжского математического общества. Т.12. №2. 2010. С. 26–33.
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. Ускорение сходимости в задачах условной глобальной оптимизации. Нижний Новгород: изд-во Нижегородского гос. ун-та, 2005.
7. Баркалов К.А., Рябов В.В., Сидоров С.В. Использование кривых Пеано в параллельной глобальной оптимизации // Материалы Девятой международной конференции-семинара "Высокопроизводительные параллельные вычисления на кластерных системах", Владимир, 2009. С. 44–47.
8. Баркалов К.А., Рябов В.В., Сидоров С.В. О некоторых способах балансировки локального и глобального поиска в параллельных алгоритмах глобальной оптимизации // Ж. Вычислительные методы и программирование. Т.11. 2010. С. 382–387.

Гибридная суперкомпьютерная система*

Р.Т. Файзуллин, А.А. Свенч, В.А. Соловьев, В.Ф. Фефелов, И.Г. Хныкин

Омский государственный технический университет

В статье рассматривается гибридная суперкомпьютерная система на базе кластера центральных процессоров (ЦПУ), кластера графических процессоров (ГПУ) и системы хранения данных (СХД). Описывается программно-аппаратная база, используемая для построения. Формируется класс задач, которые могут оптимально выполняться с использованием гибридной системы. Рассматриваются прикладные задачи криптографического анализа, моделирования транспортных потоков в сложных системах, исследования адсорбции сложных органических молекул.

1. Введение

В настоящее время есть множество общедоступных вычислительных кластеров (суперкомпьютеров). Одни основаны на стандартной архитектуре CPU, другие на менее известной архитектуре GPU. Проведенный обзор показал, что вычислительных систем, объединяющих данные архитектуры в единый кластер, нет, либо они являются коммерческим секретом.

Архитектура CPU позволяет использовать практически любые шаблоны параллельных вычислений, в то время как GPU является менее гибким, но гораздо более производительным вариантом для решения многих вычислительных задач. В частности, архитектура современных GPU NVidia Tesla включает в себя множество масштабируемых блоков, не обладающих мощной управляющей логикой и большим объемом кэш-памяти. Эта архитектура может эффективно применяться при вычислениях с большим параллелизмом и интенсивной арифметикой. Все функции, выполнимые на GPU, не поддерживают рекурсии и имеют некоторые другие ограничения, которых нет в архитектуре CPU. С помощью созданной авторами модели программирования стало возможным разрабатывать сложные проекты с использованием параллельных вычислений, которые одновременно могут использовать «гибкость» CPU и более скоростные вычисления GPU.

Другим недостатком существующих суперкомпьютерных систем является непрозрачный доступ пользователя к своим проектам на кластере. Сегодня пользователю предлагается обучиться работать с арсеналом программного обеспечения (программой удаленного доступа, командной строкой, компилятором, планировщиком и т.д.), установленным на целевом кластере. Поэтому авторами разработана программная система, позволяющая осуществлять удаленную работу с проектами без необходимости погружаться в устройство операционной системы кластера.

2. Описание технологии и аппаратных средств

Гибридная суперкомпьютерная система включает в себя:

– Вычислительную систему, объединяющую кластер CPU(центральных процессоров) x86/64 – на базе процессоров Intel Xeon, и кластер GPU(графических процессоров) – на базе процессоров NVidia Tesla.

– Модуль системы хранения данных на базе оборудования Sun Microsystems.

– Инновационную модель программирования, позволяющую объединить вычисления, использующие CPU и GPU.

– Инновационную программную систему, которая управляет выполнением проекта, использующего гибридные вычисления, и предоставляет прозрачный интерфейс для работы пользователей с системой на базе web-технологий.

* Работа поддержана грантом конкурса «У.М.Н.И.К», конференция «ОМСКОЕ ВРЕМЯ – ВЗГЛЯД В БУДУЩЕЕ», 2010 г.

На сегодняшний день в распоряжении ОмГТУ находится пять вычислительных узлов с архитектурой CPU (mgr, cn01, cn02, cn03, cn04). Каждый вычислительный узел включает два 4-ядерных процессора HP X5472 DL 160G5. Пиковая производительность достигает 1 Tflop, а объем оперативной памяти составляет 40GB. Структура масштабируема и расширяема. Вычислительная мощность может быть увеличена за счет добавления дополнительных узлов.

В рамках гибридизации к суперкомпьютеру добавлен вычислительный узел cn05, осуществляющий управление кластером GPU. Для его построения выбрана платформа NVidia Tesla 10 с архитектурой CUDA GPU. Узел представляет собой персональный компьютер с подключенными к нему ячейками NVidia Tesla S1070, каждая из которых включает в себя 4 GPU с суммарной пиковой вычислительной мощностью 4 Tflops в операциях с одинарной точностью. То есть производительность всей системы, при относительно малых затратах увеличивается до 5 Tflops (при задействовании одной ячейки Tesla S1070). Для выполнения параллельных вычислений используется NVidia CUDA API для языков программирования C, C++, Fortran [2,3].

Другим дополнением суперкомпьютерной системы является модуль хранения данных (ssd) на базе системы Sun StorageTek 9900V. Данная система отличается высокой отказоустойчивостью.

Все узлы объединены в локальную высокоскоростную сеть (1Gb/сек). Параллельное выполнение программ осуществляется с помощью технологий MPI и CUDA. Поддерживаемые реализации MPI: OpenMPI, HPMPI, MPICH [1].

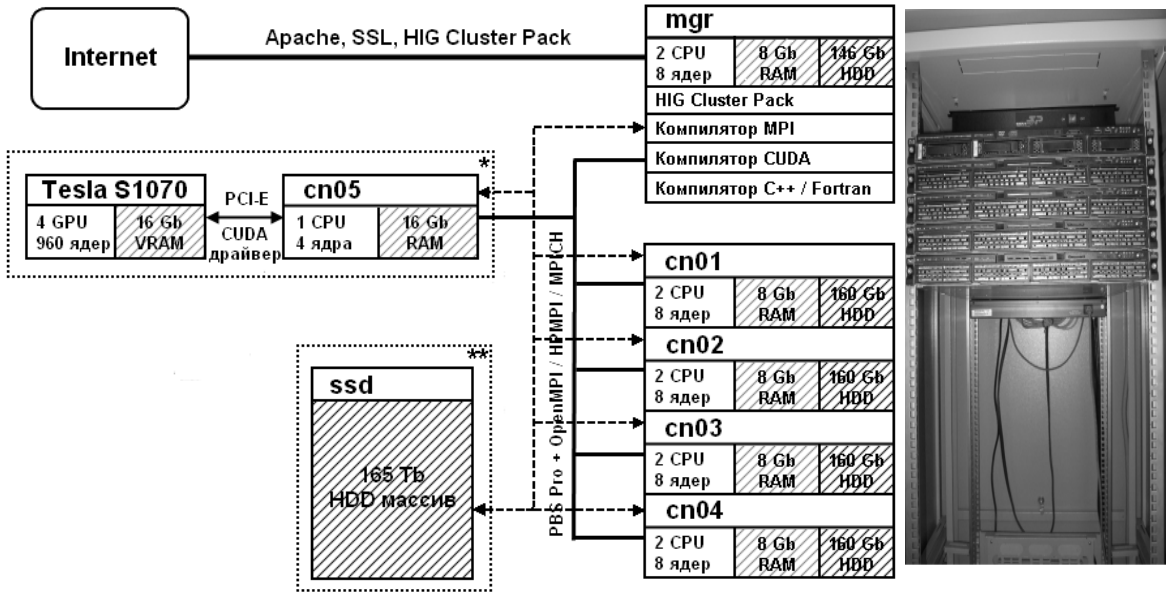


Рис.1. *Схема суперкомпьютерной системы ОмГТУ.*[1][2]

Для управления кластером один из его узлов выделен как управляющий (mgr), при необходимости он может быть использован для вычислений. На управляющем узле установлена авторская программная система для управления проектами на кластере (HIG_Cluster_Pack). С её помощью узел принимает и выполняет команды авторизованных пользователей через Интернет. Все соединения осуществляются по защищенному HTTPS протоколу (см. Рис.1).

[1] На момент написания статьи узел cn05 смоделирован с помощью персонального компьютера с 4 установленными графическими адаптерами NVidia GeForce GTX260.

[2] На момент написания статьи дисковый массив ssd имеет общий объем 12 Тб, планируется расширение до 165 Тб.

3. Программная система прозрачного доступа к кластеру

Программная система доступа к кластеру разработана авторами проекта и основана на базе web-сервера Apache. Данная система является инновационной полностью переносимой и расширяемой и может работать на вычислительных кластерах с любой аппаратной конфигурацией, операционной системой и программным обеспечением. На сегодняшний день данная система (HIG_Cluster_Pack) не имеет аналогов и успешно используется на кластере ОмГТУ.

Программа предоставляет следующие возможности:

– Трехфакторная система авторизации на уровне web-сервера, самой программы и операционной системы.

– Для каждого пользователя создается, так называемая, «песочница». То есть, все исполняемые файлы проекта могут обращаться только к ресурсам своего проекта и таким образом исчезает возможность злонамеренного повреждения кластера или проектов других пользователей.

– Пользователю предоставляется удобный web-интерфейс, который позволяет создать/удалить проект, загрузить проект, скомпилировать проект, запустить проект, посмотреть файлы проекта и т.п. При этом поддерживаются все необходимые настройки работы с проектом. Пользователю нет необходимости разбираться в опциях компилятора и планировщика заданий.

– Существует система полуавтоматической обработки заданий, когда пользователь с помощью определенных HTTPS запросов управляет работой проекта на кластере. Таким образом, появляется возможность создания программ, работающих на персональном компьютере и использующих вычислительные мощности кластера для просчета ресурсоемких блоков.

– Благодаря дружественному интерфейсу регистрация нового пользователя производится удобным как самому пользователю, так и администратору кластера способом. Пользователю достаточно сгенерировать запрос на сертификат доступа, где указываются необходимые данные. Администратору достаточно разрешить работу на кластере с данным сертификатом. Все остальные действия выполняются автоматически.

4. Прикладные задачи

Суперкомпьютерная система, смоделированная авторами в виде рабочего прототипа, ориентирована на широкий круг задач. В сочетании с инновационными авторскими разработками по созданию унифицированных средств доступа и управления, данная система позволяет использовать сразу все ведущие решения в области высокопроизводительных вычислений.

4.1. Математическое моделирование транспортных потоков на основе микроскопической схемы предиктор-корректор

Представляется возможным использование параллельных вычислений в задаче моделирования транспортных потоков, которая отличается большой сложностью в случае моделирования и оптимизации работы транспортной сети крупного города. Существующие модели, макроскопические и микроскопические в настоящее время ограничены однопроцессорными реализациями.

В качестве топографической основы модели транспортной системы города рассмотрим неориентированный граф, ребра это дороги или магистрали, узлы это перекрестки. Будем считать, что движение везде двухполосное, разделенное сплошной линией, т.е. обгоны запрещены и на всех перекрестках стоят светофоры. Транспортные средства – это точки, расположенные на ребрах. В начальном состоянии системы все транспортные средства имеют нулевую скорость. Для каждого транспортного средства случайно выбираются пункты назначения – точки на каком-либо ребре, для которых вычисляется оптимальный маршрут. В

результате поиска пути, каждому транспортному средству с номером i_s, поставлен в соответствие массив номеров ребер $ir_s(l)$, которые он должен пройти. Здесь $l = 1,..,L$, где L это число ребер в графе. Время считаем изменяющимся дискретно и вводим два характерных величины для времени Δt и $\Delta\tau$, где первая величина намного больше второй. Мы предполагаем, что Δt, это общий интервал времени, на которое независимо прогнозируется движение каждым водителем, а $\Delta\tau$, это шаг по времени после которого водитель вынужден корректировать свой прогноз.

Решение частных задач моделирования и оптимизации естественным образом приводит нас к требованию максимального повышения производительности вычислений. Как показывают расчеты, движение в режиме реального времени транспортных средств в количестве 100 000 единиц моделируется одним процессором, но задачи моделирования большей размерности и особенно оптимизационные задачи требуют повышения скорости вычислений на порядки.

Например, если речь идет об управлении движением транспорта, то число расчетов по необходимости должно быть велико, и они должны происходить существенно быстрее.

Как показывает опыт вычислений, число контролируемых транспортных средств, при подобном моделировании может достигать чисел порядка 10^6 и время расчета вариантов движения существенно меньше, чем время реализаций этих вариантов в действительности. Также, представляется перспективным перенести часть массовых вычислений на графические процессоры, выбирая оптимальные размеры региона или специальным образом распараллеливать вычисления на обычные и графические процессоры [4].

4.2. Исследование многоцентровой адсорбции молекул с возможностью различной ориентации в адсорбционном слое

В последнее время наблюдается значительный рост исследований, направленных на изучение поведения молекул, адсорбированных на поверхности твердых тел. С одной стороны это во многом это обуславливается значительным прогрессом в техническом обеспечении экспериментов (сканирующая туннельная микроскопия, атомно-силовая микроскопия и т.д.). С другой стороны необходимостью в переходе на качественно другой уровень в создании микро(нано)электронных приборов (полевых транзисторов, органических светодиодов, нелинейной оптики, газовых сенсоров, хранителей информации, микропроцессоров…) – на уровень атомов, молекул и их ансамблей. Данные экспериментов показывают, что при адсорбции молекулы занимают несколько активных центров поверхности и вследствие своей сложной структуры могут по-разному ориентироваться в адсорбционном монослое [5]. Несмотря на это, теоретических работ по исследованию подобных систем в свете многоцентровой адсорбции с возможностью различной ориентации молекул на поверхности очень мало.

Для построения модели адсорбционной системы была использована МРГ (модель решеточного газа). В рамках МРГ поверхность представляла собой двумерную решетку с квадратной симметрией. В качестве модельной молекулы-адсорбата был выбран димер – молекула, состоящая из двух одинаковых сегментов, расстояние между которыми равно постоянной решетки. Димер не обладает сферической симметрией и при адсорбции может занимать один или два активных центра поверхности (перпендикулярно или параллельно поверхности). Моделирование проводилось в большом каноническом ансамбле. Параметрами модели являлись: химический потенциал, температура, разница между теплотами адсорбции димера на два и на один активный центр, размер решетки. Модель исследовалась при помощи методов Монте-Карло и трансфер-матрицы.

Авторами модели были построены графики функции степени покрытия поверхности и изотермы адсорбции, на которых видны два ярко выраженных плато. Этим плато соответствуют упорядоченные фазы, возникающие в системе. В области низкого значения химического потенциала на поверхности наблюдается упорядоченная структура, состоящая только из димеров, адсорбированных на два активных центра. В области высокого значения химического потенциала – только из димеров, адсорбированных на один активный центр.

Интересно, что структур, состоящих одновременно из параллельно и перпендикулярно ориентированных димеров, на поверхности не образовывалось. Также были построены графики дифференциальной теплоты адсорбции и внутренней энергии системы от плотности монослоя, из которых видно, что в системе наблюдаются два режим адсорбции.

Для каждой упорядоченной структуры был введен свой уникальный параметр порядка, который равнялся единице, если в системе имела место соответствующая упорядоченная фаза, и стремился к нулю, если ее не было. Исходя из значений параметров порядка, была построена фазовая диаграмма.

Расчет модели, в силу большой вычислительной емкости и хорошей распараллеливаемости лежащих в ее основе численных методов (в частности, метода Монте-Карло) производился с помощью кластера ЦПУ, являющегося подсистемой гибридной суперкомпьютерной системы.

Литература

1. Стандарт MPI-2.0 // http://www.mpi-forum.org/docs/docs.html
2. CUDA Reference manual // http://developer.download.nvidia.com
3. NVidia Developer zone // http://developer.nvidia.com
4. Файзуллин Р.Т., Свенч А.А., Хныкин И.Г. Применение гибридной суперкомпьютерной системы в задачах криптоанализа // Доклады ТУСУР / Томск: ТУСУР, №1 (21), часть 1. 2010. С. 61-63.
5. Su G. J., Zhang H. M., Wan L. J., Bai C. L., Wandlowski T. Potential-induced phase transition of trimesic acid adlayer on Au(111) // J. Phys. Chem. B. – 2004. – V.108. – P.1931 – 1937.

Математическое моделирование транспортных потоков на основе микроскопической схемы предиктор-корректор

Р.Т. Файзуллин, В.А. Соловьев

Омский Государственный Технический Университет

В работе рассматриваются микроскопические модели транспортных потоков в крупном городе, основанные на модели следования за лидером и схеме предиктор-корректор. Учитываются различные изменения в потоке: скоростные характеристики транспортных средств, сужение магистралей, смены сигналов светофоров, случайный старт транспортных средств, с заданным пунктом назначения, транзитный поток транспортных средств через город и т.п. Для реализации модели предложен способ распараллеливания на несколько процессоров, ассоциированных с районами города. Результаты тестирования показывают эффективность параллельной версии для моделирования в режиме реального времени, и возможность управления транспортными потоками с числом транспортных средств порядка 10^6.

1. Введение

Суперкомпьютеры используются для решения задач с интенсивными вычислениями. Такие задачи часто возникают в различных областях физики и механики, в прогнозировании погоды, исследовании климата (включая исследование относительно глобального потепления), молекулярном моделировании (вычисляющем структуры и свойства химических составов, биологических макромолекул, полимеров, и кристаллов), физическом моделировании (моделирование самолетов, моделирование взрыва ядерного оружия, и исследование относительно ядерного сплава), в криптографическом анализе, и т.п.

Представляется возможным использование параллельных вычислений и в задаче моделирования транспортных потоков, которая отличается большой сложностью в случае моделирования и оптимизации работы транспортной сети крупного города. Существующие модели, макроскопические и микроскопические в настоящее время ограничены однопроцессорными реализациями [1-2].

2. Описание модели

Название предложенной модели сложилось на основе использованных подходов в ее реализации. Микроскопической модель является в силу детальной обработки транспортного потока, т.е. учет характеристик каждого транспортного средства в отдельности. Прямого отношения к схеме предиктор-корректор, существующей в теории дифференциальных уравнений [3], модель не имеет, но принцип работы аналогичен, что позволило использовать такое название как отражающее принцип действия модели.

В качестве топографической основы модели транспортной системы города рассмотрим неориентированный граф, ребра это дороги или магистрали, узлы это перекрестки. Будем считать, что движение везде двухполосное, разделенное сплошной линией, т.е. обгоны запрещены и на всех перекрестках стоят светофоры. Транспортные средства – это точки, расположенные на ребрах. В начальном состоянии системы все транспортные средства имеют нулевую скорость. Для каждого транспортного средства случайно выбираются пункты назначения – точки на каком-либо ребре, для которых вычисляется оптимальный маршрут. В результате поиска пути, каждому транспортному средству с номером i_s, поставлен в соответствие массив номеров ребер $ir_s(l)$, которые он должен пройти. Здесь $l = 1,..,L$, где L это число ребер в графе. Время считаем изменяющимся дискретно. За приращение времени примем $\Delta\tau$ – шаг по времени, после которого водитель вынужден корректировать свой прогноз. Также введем вторую величи-

ну, отвечающую за время, Δt – это общий интервал времени, на которое независимо прогнозируется движение каждым водителем. Логично утверждать, что Δt намного больше $\Delta \tau$.

2.1 Схема движения транспортных средств

Как упоминалось выше, для каждого транспортного средства заранее вычисляется путь до пункта назначения. Исходя из этого, каждому движущемуся транспортному средству с номером i_s поставлен в соответствие массив номеров ребер $ir_s(l)$, которые он должен пройти, скаляр v_{i_s} – скорость транспортного средства, номер ребра $l_{i_s}(t)$, на котором находится в данный момент транспортное средство, и $x_{i_s}(t)$ – текущая координата транспортного средства на этом ребре.

Для перемещения транспортных средств по графу дорог, на каждом такте движения по магистрали (по ребру графа) необходимо вычислить приращение координат всех транспортных средств на магистрали, с учетом скоростей и ситуации на магистралях.

Рассмотрим ситуацию, когда движение определяется лидером движения, т.е. если светофор открыт, то лидер идет с максимально допустимой скоростью, если же лидер стоит (или, то же самое, светофор закрыт), то скорость его тоже известна и итоговые расстояния между транспортными средствами прогнозируются однозначно разрешимой системой уравнений

$$x_i + v_i \Delta t - x_{i+1} - v_{i+1} \Delta t = Su, \quad i = 1,..,M \qquad (1)$$

где M – это количество машин на магистрали, u – это минимально допустимое расстояние между транспортными средствами, а константа S выбирается согласно условию: скорость меньше или равна максимально допустимой на магистрали.

Следует учесть, что транспортное средство, начинающее движение в момент времени t, вклинивается в поток транспортных средств, если ему не мешают другие транспортные средства, т.е. выезд открыт и скорость выезда достаточна, чтобы не совершилось столкновение. Также, транспортное средство, заканчивающее движение на магистрали, т.е. имеющее конечный пункт назначения на ребре графа, влияет на движение остальных транспортных средств, участвующих в движении. Поэтому желательно рассматривать более дискретизированные моменты времени, считая решение (1) «предиктором», решением, к которому в момент времени t стремятся транспортные средства.

Смещения и скорости в момент времени $t + \Delta \tau$ можно рассчитать по формулам

$$x_i(t + \Delta \tau) = x_i(t) + v_i(t)\Delta \tau + \alpha(x_i)\left(\frac{v_i(t + \Delta t) - v_i(t)}{\Delta t}\right)\Delta \tau^2 \qquad (2)$$

$$v_i(t + \Delta \tau) = v_i(t) + \alpha(x_i)\frac{v_i(t + \Delta t) - v_i(t)}{\Delta t}\Delta \tau \qquad (3)$$

где $v_i(t + \Delta t)$ найдено из (1), а $\alpha(x)$ это коэффициент имитирующий сужение дороги, или ограничение ускорения на повороте. Далее, принимая за t уже момент $t + \Delta \tau$, продолжим расчет на следующем шаге.

Отметим, что формула (1) описывает только одну из возможных стратегий, при которой транспортное средство ориентировано только на лидера, впереди идущее транспортное средство, или светофор. Назовем эту стратегию «эгоистичной». Можно предложить и другие стратегии, когда транспортное средство ориентируется не только на лидера, например, когда предиктор работает согласно выражению:

$$-x_{i-1} - v_{i-1}\Delta t + 2(x_i + v_i \Delta t) - x_{i+1} - v_{i+1}\Delta t = 0, \quad i = 1,..,M \qquad (4)$$

Например, (4) моделирует движение военной техники в составе колонны. Зная скорость лидера, мы можем рассматривать (4) как трехдиагональную систему уравнений. Такая «дружественная» стратегия позволяет передавать информацию о пробке вверх по потоку. Отметим, что, считая правую часть (4) за отрицательное число, например, кратное u, мы получаем целый спектр задач, лежащих между «эгоистичной» и «дружественной» моделями.

Очевидно, что результатами расчетов будут являться скорости и положения всех транспортных средств.

2.2 Моделируемые ситуации и постановки оптимизационных задач

Для проверки соответствия предложенной модели реальным ситуациям, возникающим при движении автотранспорта, естественным образом возникает необходимость построения типичных тестовых ситуаций. Естественной ситуацией, вытекающей из самых первых строк формулировки модели, является ситуация, представленная на рисунке 1, в которой транспортные средства движутся по дороге (ребру) в сторону вершины, пересекают ее, и продолжают движение уже по следующей дороге. Система состоит, как видно на рисунке 1, из двух ребер. На данном этапе светофоры не помещены в вершины, что позволяет транспортным средствам перемещаться с одной дороги на другую без остановок. Транспортные средства генерируются случайным образом на первом ребре. В качестве пункта назначения выбирается точка на втором ребре. Данная ситуация позволяет проверить корректность схемы в части вычисления прогнозируемых положений транспортных средств через дискретные интервалы времени.

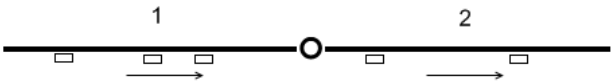


Рис. 1. Переход транспортных средств с одной дороги на другую

Добавляя светофоры в вершины графа, переходим к моделированию ситуации, которая получила название «работа Т-образного перекрестка». Отличие этого случая от предыдущего кроме дополнительного ребра, заключается в наличии светофора на перекрестке (в вершине). В данной ситуации транспортные средства «генерируются» на первой и второй дороге, а их целью является достижение конца третьей. Красный и зеленый сигналы светофора горят одинаковое количество времени.

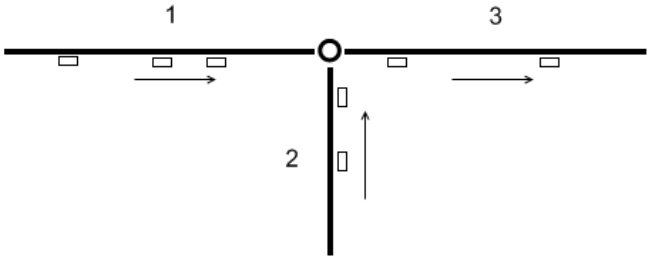


Рис. 2. «Т-образный перекресток»

Данная ситуация позволяет моделировать образование пробок и вычислять критическую плотность транспортных средств, при которых возникают пробки. Отметим, что в этой ситуации статистически показано преимущество «дружественной» модели перед «эгоистичной».

Более сложная ситуация проиллюстрирована на рисунке 3. В данном примере рассматривается решетка из 49 перекрестков (вершин), некоторые из которых соединены дорогами (ребрами) с другими. Транспортные средства случайным образом генерируются на случайных дорогах. Каждому из них задается случайный пункт назначения, и просчитывается оптимальный маршрут для его достижения. Затем транспортное средство включается в движение. Варьируя количество транспортных средств, генерируемых на каждом такте системы, можно загружать сеть различным количеством транспортных средств, выявляя места, в первую очередь подверженные образованию пробок. Будем называть подобные решетки регионом.

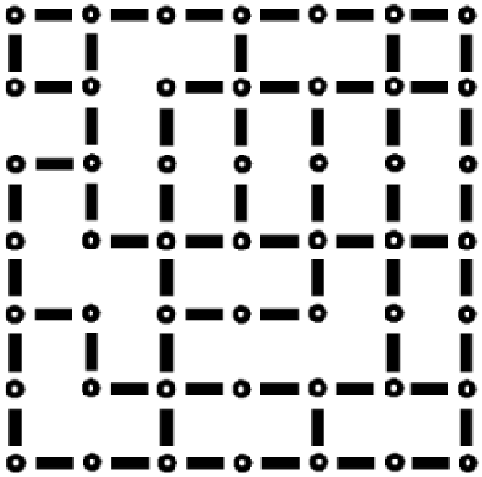


Рис. 3. Регион

Данная модель позволяет поставить задачу нахождения оптимальной работы светофоров для максимальной «прокачки» транспортных средств через регион. Без потери общности зададим время работы светофоров функцией сдвига фаз относительно основной частоты колебаний ω, $\beta_1,..,\beta_P$, где P, это число светофоров в регионе. По определенным путям или направлениям через регион следует транзитный транспорт (рисунок 4) средняя скорость которого при пересечении региона является целевой функцией, которую нам необходимо максимизировать с помощью выбора значений фаз. Данная задача является в некотором смысле развитием задачи поиска так называемой «зеленой волны».

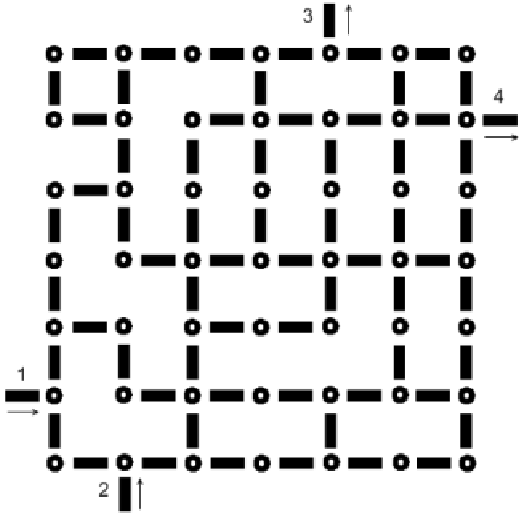


Рис. 4. «Прокачка» через регион

Вопрос о классе данной оптимизационной задачи в настоящее время открыт.

3. Распараллеливание при моделировании и оптимизации движения транспорта в нескольких регионах

Решение рассмотренных частных задач моделирования и оптимизации естественным образом приводит нас к требованию максимального повышения производительности вычислений. Как показывают расчеты, движение в режиме реального времени транспортных средств в количестве 100 000 единиц моделируется одним процессором, но задачи моделирования большей

размерности и особенно оптимизационные задачи требуют повышения скорости вычислений на порядки.

Например, если речь идет об управлении движением транспорта, то число расчетов по необходимости должно быть велико, и они должны происходить существенно быстрее.

Первая и очевидно наиболее естественная схема распараллеливания, это распараллеливание по регионам, где расчеты происходят независимо, а обмен ограничивается передачей контроля над транспортными средствами, пересекающими границу региона. На рисунке 5 приведена схема распараллеливания – индекс ребра или вершины отвечает номеру процессора (индекс вершины, это номер процессора отвечающего за светофоры). Очевидно, что регионы надо организовывать таким образом, чтобы контакты между регионами были минимальны, т.е. число контактирующих вершин было наименьшим.

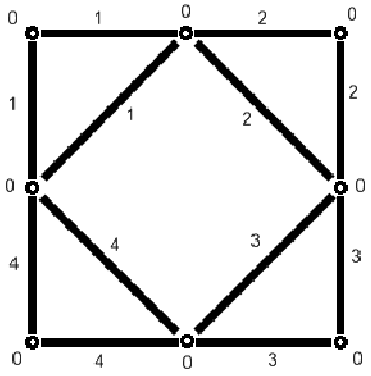


Рис. 5. Разбивка по регионам

В качестве примера распараллеливания по регионам можно привести схему районов г. Омска. На рисунке 6а приведены три крупных района г. Омска: Советский (I), часть Первомайского (II) и часть Кировского (III).

Рис. 6а. г. Омск

Соединения между ними осуществляются по единственному мосту и одной магистрали Красный путь.

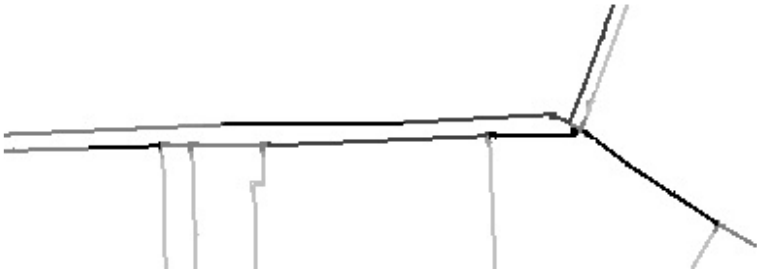


Рис. 6б. г. Омск

Интенсивность «серого» на рисунке 6б соответствует насыщенности транспортного потока. По результатам расчетов можно сделать вывод о соответствии между модельными «пробками» и «пробками», образующимися на практике.

4. Заключение

Как показывает опыт вычислений, число контролируемых транспортных средств, при подобном моделировании может достигать чисел порядка 10^6 и время расчета вариантов движения существенно меньше, чем время реализаций этих вариантов в действительности. Также, представляется перспективным перенести часть массовых вычислений на графические процессоры, выбирая оптимальные размеры региона или специальным образом распараллеливать вычисления на обычные и графические процессоры [4].

Литература

1. Швецов В.И. Математическое моделирование транспортных потоков в крупном городе с применением к Московской агломерации // Автоматика и Телематика, №11, 2005. С. 113-125.
2. P. Chakroborty, S. Agrawal, K. Vasishtha. Microscopic Modeling of Driver Behavior in Uninterrupted Traffic Flow //Journal of transportation engineering. ASCE. July/August 2004. P. 486-451.
3. Марчук Г.И. Методы вычислительной математики / Г.И. Марчук. — М.: Наука. Главная редакция физико-математической литературы, 1980. – 536 с.
4. Файзуллин Р.Т., Свенч А.А., Хныкин И.Г. Применение гибридной суперкомпьютерной системы в задачах криптоанализа // Доклады ТУСУР / Томск: ТУСУР, №1 (21), часть 1. 2010. С. 61-63.

Параллельная реализация алгоритма вершинной минимизации недетерминированных конечных автоматов

А.В. Цыганов

Ульяновский государственный педагогический университет им. И.Н. Ульянова

Задача вершинной минимизации недетерминированных конечных автоматов в общем случае является PSPACE-полной задачей и может потребовать значительных вычислительных затрат даже для автоматов с небольшим числом состояний. В настоящей работе рассматривается параллельная реализация одного из точных методов вершинной минимизации НКА — алгоритма Камеды-Вейнера. Программная реализация алгоритма выполнена на языке C++ с использованием двух технологий параллельного программирования: OpenMP и MPI. Приводятся результаты численных экспериментов, демонстрирующие некоторые статистические свойства рассматриваемого алгоритма и подтверждающие эффективность предложенной реализации.

Введение

Конечные автоматы (КА) находят самое широкое применение в различных областях науки и техники, например, в системах обработки текста, машинного перевода, распознавания речи и др. На практике исследователю очень часто приходится выбирать между двумя типами автоматов: детерминированными конечными автоматами (ДКА) и недетерминированными конечными автоматами (НКА). Кроме того, в некоторых приложениях важную роль может играть способ представления КА в памяти компьютера. Например, может потребоваться минимизация КА по числу состояний (вершинная минимизация), по числу переходов (дуговая минимизация) или по другим критериям, в частности, смешанным. Первая из этих задач является, пожалуй, самой известной. В то время как для ДКА задача вершинной минимизации имеет сложность $O(n \log n)$, аналогичная задача для НКА является PSPACE-полной [1]. Это означает, что вершинная минимизация даже небольших автоматов может потребовать больших вычислительных затрат.

К настоящему времени разработано значительное количество точных и приближенных алгоритмов вершинной минимизации НКА. Одним из самых первых точных методов является алгоритм Камеды-Вейнера [2]. Несмотря на почтенный возраст алгоритма, в многочисленных программах для работы с конечными автоматами (по крайней мере, бесплатных) его реализация отсутствует.

В настоящей работе будет рассмотрена параллельная реализация алгоритма Камеды-Вейнера, выполненная автором на языке C++ с использованием двух технологий параллельного программирования: OpenMP и MPI.

1. Постановка задачи и описание алгоритма Камеды-Вейнера

Приведем основные определения из теории КА, которые нам понадобятся для формулировки алгоритма Камеды-Вейнера (более подробно с соответствующей теорией можно ознакомиться в многочисленных монографиях, например, [3]).

Детерминированным конечным автоматом называется пятерка $A = (Q, \Sigma, \delta, s, F)$, где Q — некоторое конечное множество состояний (вершин) автомата, Σ — рассматриваемый алфавит, δ — функция переходов вида $\delta : Q \times \Sigma \to Q$, $s \subset Q$ — стартовое состояние (вход) автомата, $F \subset Q$ — множество финальных состояний (выходов).

Недетерминированным конечным автоматом называется пятерка $A = (Q, \Sigma, \delta, S, F)$, где

Q — некоторое конечное множество состояний автомата, Σ — рассматриваемый алфавит, δ — функция переходов вида $\delta : Q \times \Sigma \to 2^Q$ (2^Q — множество всех подмножеств множества Q), $S \subset Q$ — множество стартовых состояний, $F \subset Q$ — множество финальных состояний.

Задача вершинной минимизации НКА состоит в построении такого НКА, который был бы эквивалентен исходному, то есть задавал бы тот же самый регулярный язык, что и исходный автомат, но имел бы при этом как можно меньшее число состояний.

Введем следующие обозначения (аналогичные обозначениям [2]). Пусть A — некоторый КА, тогда через $\bar{A}$ обозначим соответствующий ему зеркальный автомат (то есть автомат с противоположным направлением дуг, входы которого являются выходами автомата A, а выходы — входами). Если A — НКА, то через $D(A)$ будем обозначать ДКА, полученный из A процедурой детерминизации (subset construction). Для ДКА B, через $\hat{B}$ будем обозначать соответствующий минимальный автомат, то есть автомат, полученный из B объединением эквивалентных состояний.

Пусть A — некоторый НКА, $B = D(A)$, $C = D(\bar{A})$, а $\hat{B}$ и $\hat{C}$ — соответствующие минимальные автоматы. Состояниями автоматов B, C, $\hat{B}$ и $\hat{C}$ являются подмножества множества состояний исходного автомата A (заметим, что после выполнения процедуры детерминизации число состояний в автомате может экспоненциально возрасти). Матрицей RAM (Reduced Automaton Matrix) назовем матрицу, состоящую из нулей и единиц, число строк в которой совпадает с числом состояний автомата $\hat{B}$, а число столбцов — с числом состояний автомата $\hat{C}$. Пусть $p_1, p_2, \ldots, p_m$ — состояния автомата $\hat{B}$, а $q_1, q_2, \ldots, q_n$ — состояния автомата $\hat{C}$, тогда по определению элемент r_{ij} матрицы RAM равен 0, если $p_i \cap q_j = \emptyset$ и 1 в противном случае.

Некоторую пару подмножеств строк и столбцов матрицы RAM назовем гридом (блоком), если, во-первых, на всех их пересечениях стоят 1 и, во-вторых, это множество нельзя пополнить ни строкой, ни столбцом без нарушения первого свойства. Покрытием Z назовем такое множество гридов, для которого каждая единица матрицы RAM принадлежит, по крайней мере, одному из них. Размером покрытия будем называть число гридов в нем. Минимальным покрытием назовем покрытие, содержащее наименьшее возможное количество гридов.

В работе [2] описана процедура, называемая правилом пересечений (intersection rule), позволяющая по заданному покрытию Z и автомату $\hat{B}$ построить НКА $I(Z, \hat{B})$, который может оказаться эквивалентным исходному. При этом число вершин автомата I равно размеру покрытия. Доказано, что минимальные автоматы следует искать только среди таких автоматов.

Сформулируем теперь алгоритм Камеды-Вейнера вершинной минимизации НКА A.

Алгоритм Камеды-Вейнера.

1. Построить автоматы $B = D(A)$ и $C = D(\bar{A})$.
2. Для автоматов B и C построить минимальные автоматы $\hat{B}$, $\hat{C}$.
3. По автоматам $\hat{B}$ и $\hat{C}$ построить матрицу RAM.
4. Найти все гриды матрицы RAM.
5. Найти минимальное покрытие матрицы RAM. Пусть $i_{\min}$ — размер минимального покрытия. Положим $i = i_{\min}$.

   А. Для каждого покрытия Z размера i построить НКА $I(Z, \hat{B})$ и проверить его эквивалентность исходному автомату A.

   Б. Если эквивалентного НКА не найдено, то положить $i = i + 1$ и перейти к п. А.

Описанный алгоритм останавливается либо при нахождении минимального автомата, либо, если i становится равно числу вершин автомата A. В отличие от ДКА, минимальный автомат для заданного НКА может оказаться не единственным. Очевидно, что с помощью алгоритма Камеды-Вейнера могут быть найдены все минимальные автоматы.

2. Программная реализация

Программная реализация алгоритма вершинной минимизации НКА Камеды-Вейнера была выполнена в виде проекта на языке C++ с использованием технологий параллельного программирования OpenMP и MPI и кросс-платформенной библиотеки HeO. С подробным описанием библиотеки можно ознакомиться в работе [5] (официальная страница проекта расположена по адресу: http://code.google.com/p/heo/, в настоящее время для скачивания доступна версия 1.1 библиотеки). Для реализации алгоритма минимизации были написаны все необходимые классы, методы и вспомогательные функции, а для проведения численных экспериментов — процедуры случайной генерации НКА с заданными свойствами.

Самыми трудоемкими операциями алгоритма Камеды-Вейнера являются шаги 4 и 5: поиск гридов матрицы RAM и ее покрытий. Последняя задача является разновидностью задачи о покрытии множества (SCP — Set Covering Problem), которая в общем случае является NP-трудной, и в худшем случае алгоритмы ее решения по сложности не отличаются от переборных, при этом нахождение минимального покрытия матрицы RAM каким-либо из известных алгоритмов решения SCP не гарантирует построения автомата, эквивалентного исходному. Поэтому для выполнения шагов 4 и 5 алгоритма было решено использовать метод «грубой силы» — параллельный перебор. Для нахождения гридов и покрытий матрицы RAM в программе используется генератор сочетаний из n элементов по k элементов, основанный на алгоритме Twiddle [6].

Сформулируем алгоритм нахождения гридов, использующий приведенное выше определение грида.

Алгоритм нахождения гридов.

Для каждого $k = 1, 2, \ldots, n$, где n — число столбцов матрицы RAM:

1) сгенерировать все возможные подмножества столбцов матрицы RAM, состоящие из k элементов;
2) для каждого подмножества столбцов Y выбрать такое подмножество X строк матрицы RAM, чтобы на пересечении всех строк и столбцов были только единицы;
3) проверить, можно ли к данному множеству столбцов Y добавить еще столбцы без нарушения условия 2 и, если это невозможно, добавить грид $X \times Y$ в список найденных гридов.

Для уменьшения перебора описанный выше алгоритм поиска гридов «по столбцам» применяется, если число столбцов в матрице RAM меньше числа строк, в противном случае поиск гридов осуществляется аналогичным образом «по строкам». Генерация подмножеств (сочетаний) осуществляется каждым потоком программы независимо, начиная со своего номера, с шагом N, где N — общее число потоков. Найденные гриды помещаются в специальный вектор grids (нумеруются). В OpenMP-версии программы вектор grids является общим для всех потоков, а в MPI-версии он у каждого потока свой. По окончании поиска в MPI-версии потоки обмениваются найденными гридами.

После нахождения гридов производится поиск и анализ всех покрытий матрицы RAM размера $i = i_{\min}, i_{\min}+1, \ldots, i_{\max}$, где $i_{\min}$ и $i_{\max}$ вычисляются автоматически или задаются пользователем. Для каждого i всеми потоками генерируются подмножества вектора grids, состоящие из i элементов и для каждого такого подмножества Z проверяется, является ли оно покрытием матрицы RAM. Если Z является покрытием, то согласно правилу пересечений находится НКА $I(Z, \hat{B})$ и проверяется его эквивалентность исходному автомату, для чего сначала строится ДКА $D = D(I)$, а затем минимальный автомат $\hat{D}$, после чего проверяется изоморфизм автоматов $\hat{D}$ и $\hat{B}$. Если автоматы $\hat{D}$ и $\hat{B}$ изоморфны, то автомат $I(Z, \hat{B})$ является искомым минимальным автоматом и помещается в специальный вектор, который в OpenMP-версии программы является общим для всех потоков, а в MPI-версии у каждого потока свой. Поиск минимальных автоматов продолжается либо до первого найденного,

либо до нахождения всех минимальных автоматов. По окончании поиска в MPI-версии все потоки передают найденные минимальные автоматы нулевому потоку.

В алгоритмах работы с КА активно используются структуры данных, основанные на множествах. Для работы с множествами в реализованном проекте используется библиотека BitMagic [7]. Данная библиотека позволяет эффективно работать с большими и разреженными множествами, а также позволяет выполнять их сериализацию и десериализацию, что необходимо в MPI-версии программы. Кроме того, для MPI-версии программы были написаны процедуры сериализации и десериализации автоматов (для обмена найденными решениями).

3. Численные эксперименты

Статистические свойства алгоритма Камеды-Вейнера (точнее говоря, свойства объектов, возникающих в процессе его работы) для разных типов НКА изучены крайне мало и параллельная реализация данного алгоритма позволяет изучать их с большей эффективностью.

В качестве примера рассмотрим результаты минимизации 1000 НКА с одним входом и двумя выходами без недостижимых и бесполезных состояний (trim-автоматов), сгенерированных по алгоритму, аналогичному алгоритму Лэсли (см., например, [8]). Входными данными для этого алгоритма генерации автоматов являются: число состояний автомата, число начальных состояний, число конечных состояний, размер алфавита и плотность переходов $D = \frac{T}{|Q|^2|\Sigma|}$, где T — это число переходов в автомате. Ниже приведены результаты работы алгоритма для $|Q| = 5$, $|\Sigma| = 2$, $D = 0{,}2$. Эксперименты проводились на суперкомпьютерном кластере «Скиф-политех» («Т-Платформы» HPC–0011102–001) Томского политехнического университета. Результаты расчетов проверялись в программе GAP [9].

На рис. 1 представлено распределение размеров матрицы RAM для рассматриваемой выборки.

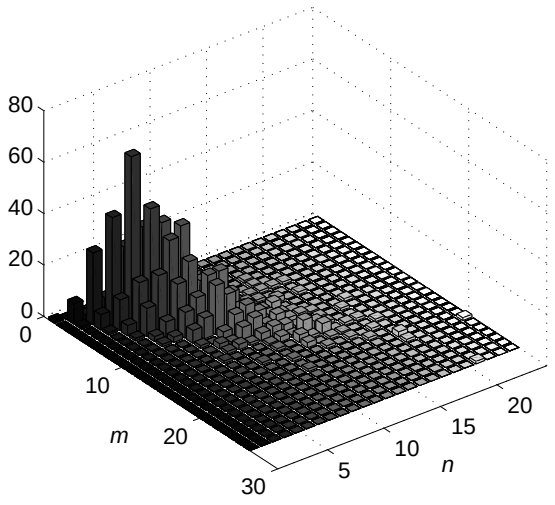


Рис. 1. Распределение размеров матрицы RAM

В таблице 1 приведены остальные результаты эксперимента. Обозначения: m, n — соответственно число строк и столбцов матрицы RAM; d — плотность единиц в матрице RAM; N_G — число гридов в матрице RAM; N_I — число найденных минимальных автоматов; $|Q_I|$ — число вершин в минимальном автомате; t — время вычислений в секундах для 64 потоков;

min — минимальное значение; max — максимальное значение; μ — среднее значение; σ — среднеквадратическое отклонение.

Таблица 1. Результаты эксперимента

	m	n	d	N_G	N_I	$\lvert Q_I \rvert$	t
min	1	1	0,3333	1	0	0	0,0017
max	26	24	1,0000	1329	6	4	36,9774
μ	7,2100	7,8330	0,6140	23,8160	0,7540	1,7020	0,1848
σ	3,0502	3,1405	0,0878	61,1850	1,0671	1,8699	1,5989

На рис. 2 приведена зависимость среднего времени вычислений в секундах (t) от числа потоков (N).

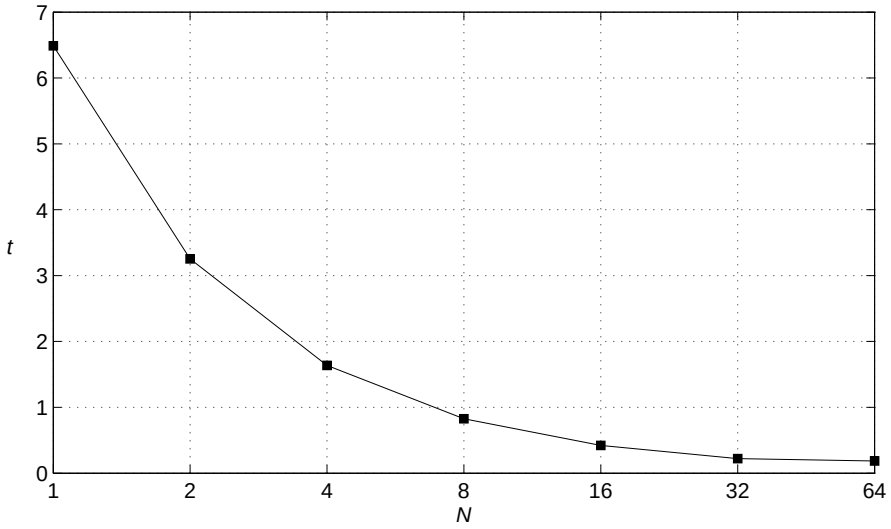


Рис. 2. Среднее время вычислений

Заключение

В работе описана параллельная реализация алгоритма вершинной минимизации НКА Камеды-Вейнера. Точное решение данной задачи даже для НКА с небольшим числом состояний может потребовать больших вычислительных затрат, поэтому актуальным остается вопрос разработки эффективных приближенных методов ее решения. В частности для нахождения минимального покрытия матрицы RAM могут использоваться метаэвристические алгоритмы, такие, например, как метод имитации отжига [10].

В настоящее время автором ведутся работы по реализации для подзадачи поиска минимального покрытия параллельной версии мультиэвристического алгоритма [11], основанного на использовании метода ветвей и границ в сочетании с комплексом эвристик для выбора разделяющего элемента (заметим, что данный алгоритм может быть применен для достаточно широкого круга задач дискретной оптимизации).

Автор выражает благодарность руководству и сотрудникам центра коллективного пользования «Суперкомпьютерный кластер» Томского политехнического университета и лично Дубакову Анатолию Алексеевичу и Белоцерковскому Александру Владимировичу.

Литература

1. Jiang T., Ravikumar B. Minimal NFA problems are hard // SIAM Journal on Computing, 22(6):1117–1141, December 1993.

2. Kameda T., Weiner P. On the state minimization of nondeterministic finite automata // IEEE Transactions on Computers. 1970. Vol. C-19, no. 7. P. 617–627.

3. Мельников Б. Ф. Недетерминированные конечные автоматы. Тольятти: Изд-во ТГУ, 2009. 160 с.

4. Gansner E. R., North S. C. An open graph visualization system and its applications to software engineering // Softw. Pract. Exper. 2000. V. 30, no. 11. P. 1203–1233.

5. Цыганов А. В., Булычов О. И. HeO: библиотека метаэвристик для задач дискретной оптимизации // Программные продукты и системы. 2009. №4. С. 148–151.

6. Chase P. J. Algorithm 382: Combinations of M out of N Objects [G6] // Communications of the Association for Computing Machinery 13:6:368, 1970.

7. Kuznetsov A., Shemanarev M., Tolstoy I., Lewis E., Khovayko O. BitMagic Library [Электронный ресурс]. URL: http://bmagic.sourceforge.net (дата обращения: 12.12.2010).

8. Almeida M., Moreira N., Reis R. On the performance of automata minimization algorithms: Tech. rep. / DCC-FC & LIACC, Universidade do Porto, 2007.

9. The GAP Group. GAP — Groups, Algorithms, and Programming, Version 4.4.12; 2008 [Электронный ресурс]. URL: http://www.gap-system.org (дата обращения: 5.01.2011).

10. Цыганов А. В. Имитационная нормализация в задаче минимизации недетерминированных конечных автоматов // Всероссийская конференция с элементами научной школы для молодежи «Проведение научных исследований в области обработки, хранения, передачи и защиты информации», 1–5 декабря 2009 г. Россия, г. Ульяновск : сборник научных трудов. В 4 т. Т. 2. Ульяновск: УлГТУ, 2009. С. 270–275.

11. Melnikov B. F., Tsyganov A. V., Bulychov O. I. A Multi-Heuristic Algorithmic Skeleton for Hard Combinatorial Optimization Problems // CSO'09: Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization. Washington, DC, USA: IEEE Computer Society, 2009. P. 33–36.

Параллельные компьютерные технологии в системах виртуального окружения. Цели и задачи

Н.Н. Шабров, С.Г. Орлов, А.К. Кузин, А.Е. Суетов,
С. Петербургский государственный политехнический университет

Мировое сообщество разработчиков программного обеспечения предприняло инициативу по кардинальному пересмотру стратегии развития и разработки программного обеспечения для высокопроизводительных вычислительных систем, которая воплощена в документе IESP Roadmap на период 2010 - 2019 годы. Этот документ отмечает, что в процессе моделирования с производительностью вычислительных систем на уровне петафлоп создаются объемы данных уровня петабайт, а в процессе моделирования с производительностью на уровне экзофлоп создаются объемы данных уровня экзобайт. При этом утверждается, что для визуализации объемов данных уровня Petabyte и Exabyte требуются как новые технологии анализа и визуализации результатов, так и новые программные и аппаратные средства визуализации. К новым технологиям анализа и визуализации больших объемов данных относятся технологии систем виртуального окружения типа CAVE 3D, которые интенсивно развиваются в мире в последнее время. Моделирование на суперкомпьютерах порождает сверхбольшие объемы данных, анализ и интерактивная визуализация которых в системах виртуального окружения в режиме real-time в свою очередь также требует применения суперкомпьютерных вычислений. В работе обсуждаются перспективные подходы решения этой проблемы.

Мировое сообщество разработчиков программного обеспечения предприняло инициативу по кардинальному пересмотру стратегии развития и разработки программного обеспечения для высокопроизводительных вычислительных систем, которая воплощена в документе IESP Roadmap на период 2010 – 2019 годы. Этот документ отмечает, что в процессе моделирования с производительностью вычислительных систем на уровне петафлоп создаются объемы данных уровня петабайт, а в процессе моделирования с производительностью на уровне экзофлоп создаются объемы данных уровня экзобайт. При этом утверждается, что для визуализации объемов данных уровня Petabyte и Exabyte требуются как новые технологии анализа и визуализации результатов, так и новые программные и аппаратные средства визуализации.

Созданию виртуальных сред распределенного совместного моделирования, научного анализа и интерактивной визуализации на основе систем виртуальной реальности типа CAVE 3D (**Computer Aided Virtual Environment**) в мире уделяется растущее внимание. Актуальность создания таких сред в настоящее время осознана всеми ведущими промышленными компаниями в мире, выпускающими конкурентоспособную продукцию. Эти среды наиболее востребованы в высокотехнологичных отраслях промышленности, таких как аэрокосмическая, автомобильная и авиационная. В последнее время системы типа CAVE 3D позиционируются инженерной общественность как средство и место принятия коллективного решения о судьбе проектируемого изделия. При этом технологии принятия решения в значительной мере базируется на технологиях распределенной работы (**Collaborative work**) по моделированию, анализу и визуализации результатов в среде виртуального окружения.

В настоящее время в мире ведутся **интенсивные** работы по совершенствованию специализированного программного обеспечения в направлении повышения качества масштабирования, улучшения функциональности виртуальных сред для использования последних группами инженеров - участников совместной распределенной работы. Это подразумевает получение разделенного доступа как к приложениям и инструментальным средствам разработки программного обеспечения, так и к средствам интерактивной визуализации и анализа результатов моделирования сложных физических процессов. Глобальная задача состоит в **создании, развитии и интеграции** распределенного совместного моделирования и интерактивных сред виртуальной реальности для анализа результатов в режиме реального времени. Решение этой задачи позволит пользователю получить доступ к основным функциям, необходимым для распределенной совместной работы в рамках

отведенной сессии. Разработка и создание виртуальных сред распределенного совместного моделирования, научного анализа и интерактивной визуализации неразрывно связаны с развитием высокопроизводительных вычислительных систем и в полной мере относится к вычислительному обеспечению развития прорывных технологий.

Задачи визуализации и понимания результатов вычислений на больших сетках с объемом данных уровня петабайт и экзобайт порождают проблемы связанные как с обработкой видеоизображений, так и проблемы связанные с ограничениями по скорости передачи результатов вычислений в систему визуализации. Особенно это относится к проблеме визуализации результатов моделирования динамически развивающегося процесса. Специализированные вычислительные системы в настоящее время способны обрабатывать и порождать огромные объемы данных, причем на практике решение задач моделирования, как правило, не выполняется в режиме реального времени. Системы визуализации, напротив, должны обрабатывать данные с достаточно высокой скоростью необходимой для работы пользователя в режиме реального времени. Актуальной задачей является решение проблем визуализации больших и сверхбольших объемов данных на основе параллельных компьютерных технологий.

Таким образом, моделирование на суперкомпьютерах порождает сверхбольшие объемы данных, анализ и интерактивная визуализация которых в системах виртуального окружения в режиме реального времени в свою очередь также требует применения суперкомпьютерных вычислений.

Наиболее перспективными подходами к решению этой проблемы представляются следующие взаимно дополняющие подходы:

1. Предварительная обработка результатов моделирования с целью редукции объемов данных. Следует разработать алгоритмы, определяющие наборы данных, которые бы содержали только ту информацию, которая является визуально значимой. В этой связи следует указать на исследования, выполненные в ИММ РАН и используемые при визуализации данных на сверхбольших сетках в системе CAVE 3D (1).
2. Компрессия геометрических данных, передаваемых в систему визуализации, с последующей их быстрой декомпрессией с использованием процессоров, установленных на графических ускорителях. Подход позволит решить проблему “бутылочного горлышка” – шины передачи данных на графический ускоритель.
3. Выполнение части вычислений непосредственно с помощью процессоров, входящих в систему визуализации. Такой подход оправдан в тех случаях, когда визуальное представление объекта может быть частично реконструировано по некоторым параметрам результатов моделирования, объем которых существенно меньше, чем объем восстанавливаемых данных. Алгоритм восстановления частично реализуется на графическом процессоре.
4. Интерактивная работа в системе виртуального окружения требует, чтобы время реакции системы на действия пользователя, находящегося в киберпространстве, составляло не более 2 секунд. В связи с этим для обработки видеоизображения требуются программные и аппаратные средства параллельных компьютерных технологий.

Программные и аппаратные средства НОЦ «Параллельные компьютерные технологии и моделирование в системах виртуального окружения» востребованы как промышленностью, так и организациями РАН. Результаты исследований в области моделирования и визуализации больших объемов данных с использованием систем виртуального окружения были продемонстрированы на выставке «Российский промышленник 2010» в С. Петербурге.

Совместно с ИММ РАН выполняются цикл работ по созданию специализированного программного обеспечения на основе суперкомпьютерных вычислений по интерактивной визуализации изоповерхностей на сетках с числом узлов до 10^9 (1). Ставиться задача о построении изоповерхности на нерегулярной сетке тетраэдров, в узлах которой задано скалярное поле. Сетка разбивается на домены, в каждом из которых содержится порядка 10^6 узлов (таким образом, имеется около 1000 доменов). Реализации параллельных алгоритмов

построения изоповерхностей выполнялись на основе многоядерной вычислительной архитектуре с использованием видеокластера производительностью 1,4 TF на базе процессора Intel Quad Core Xeon.

В работе представлен достаточный иллюстративный материал.

1. Akayev A.A, Kuzin A.K., Orlov S.G., Chetverushkin B.N., Shabrov N.N., Iakobovski M.V. Genereation of isosurface on Large Mesh. Proceeding of the IASTED International Conference ACIT 2010 in cooperation with

Авторы доклада благодарят РФФИ за поддержку в части проведенных исследований в рамках гранта № 09-07-12020 офи_м.

Средства визуальной поддержки процесса распараллеливания последовательных программ

В.Л. Авербух[1], Р.О. Сударикov[2]

ИММ УрО РАН[1], Уральский Государственный Университетет[2]

Одной из важных задач поддержки и организации супервычислений является задача распараллеливания огромных объемов прокладных программ, созданных в предшествующую эпоху для последовательных ЭВМ. Эти программы успешно решали задачи математической физики, моделирования химических процессов, небесной механики и др. После появления современных параллельных вычислителей с 1000 и 10 000 процессоров встает проблема превращения надежных и проверенных кодов в эффективные и мобильные параллельные программы.

Частично эта проблема решается за счет средств автоматического распараллеливания, присутствующих в ряде компиляторов и систем программирования, однако, для значительной части гигантского корпуса программ такие средства, по крайней мере, неэффективны. Для создания эффективных параллельных кодов необходимо вмешательство человека.

Цель нашей работы – создание визуального инструментария, помогающего в процессе распараллеливания. Рассматривается код, написанный на языке программирования С. Потенциальный пользователь – прикладной программист, понимающий суть работающей программы и свободно ориентирующийся в параллельном программировании. Он не разрабатывает новую программу, а переводит последовательный код на систему параллельных вычислений. Для начала мы выявляем набор действий, который пользователь производит во время работы.

Для создания инструментария используется язык C# и компонент WPF. На данном этапе рассматривается парадигма параллелизма на основе библиотеки MPI. В начальном варианте прототипа используется схема «Ферма». В качестве средств поддержки были реализованы подсветка синтаксиса и визуальная подсказка того, как строится программа,

Разработан работающий прототип инструментария, поддерживающий часть функционала. Наш прототип отличается от большинства прочих исследований в этой области, поскольку основной упор разработки инструментов поддержки создания параллельных программ сделан на развитие инструментов отладки параллельных программ.

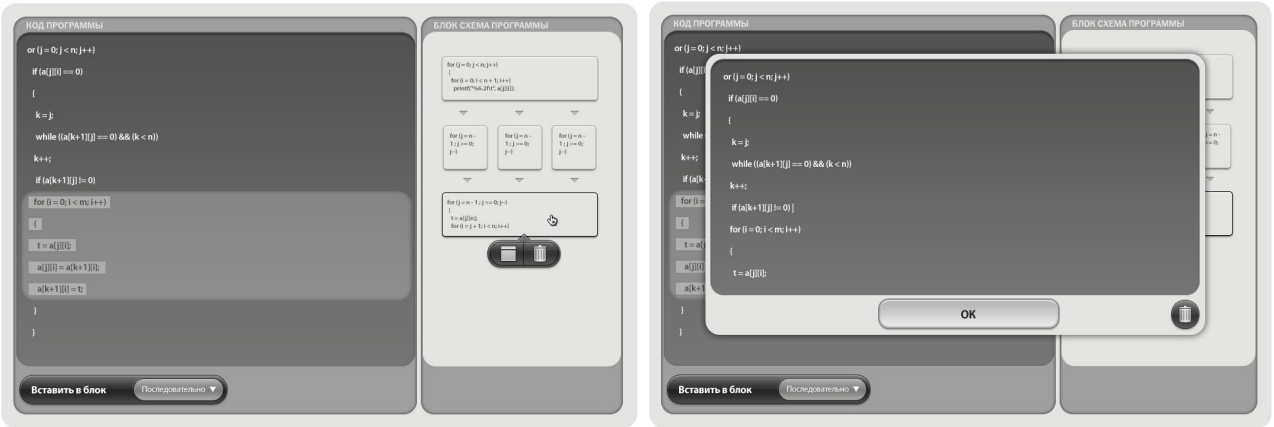


Рис. 1. Пример, демонстрирующий работу инструментария

На следующем этапе мы собираемся предоставить нашу систему специалистам для тестирования и высказывания рекомендаций по поводу необходимого функционала и общей реализации интерфейса. Также мы планируем расширять инструментарий, добавляя функционал в MPI направлении и создание подобного инструментария для библиотеки OpenMP. Дополнительно представляются важными вопросы визуальной поддержки создания гетерогенных систем, включающих в себя графические процессоры, и перевода программ на данные системы.

Исследование эффективности различных вариантов реализации коллективной операции Allgather на гибридном вычислительном комплексе МВС-Экспресс

Д.Л. Аверичева

ФГУП «НИИ «Квант»

При решении многих вычислительных задач на многопроцессорных системах часто используется несколько стандартных коллективных операций. Одной из базовых операций является операция Allgather. К примеру, Allgather используется при умножении матрицы на вектор, а, значит, во многих вычислительных задачах. Целью данного исследования является разработка эффективной реализации коллективной операции Allgather на гибридном вычислительном комплексе МВС-Экспресс. В МВС-Экспресс реализована программно-аппаратная поддержка общей памяти, основным средством разработки параллельных программ является библиотека односторонних коммуникаций shmem-express. Мотивация работы состояла в том, что на машинах с аппаратной поддержкой общей памяти реализация коллективных операций с использованием односторонних коммуникаций оказывалась эффективней реализации библиотеки MPI.

Гибридный вычислительный комплекс МВС-Экспресс состоит из 8-ми узлов, объединенных полносвязным коммутатором PCI-Express. В состав каждого вычислительного узла входят 2 четырехядерных процессора Opteron с частотой 2,6 ГГц и оперативной памятью 16ГБ.

Библиотека shmem-express обеспечивает взаимодействие независимых процессов, каждый – со своей локальной памятью, занумерованных по порядку от нуля, и в этом она похожа на MPI. Отличие от MPI состоит в том, что обмены данными между процессами являются односторонними. Для чтения/записи данных в память удаленного узла требуется знать локальный адрес на удаленном узле и номер узла.

Разработка эффективных алгоритмов на MPI подразумевает использование «рукопожатий», что при небольшой длине и большом количестве отправляемых сообщений значительно снижает производительность. При использовании библиотеки shmem необходимости применения двусторонних коммуникаций нет, записать или прочитать данные из памяти можно без ведома удаленного процесса. Однако, для разработки эффективных программ на shmem необходимо как можно реже прибегать к общей синхронизации процессов, например, при помощи введения неблокирующих операций можно значительно повысить производительность. Кроме того, прирост производительности дает также использование дополнительных функций, обеспечивающих прямой доступ к небольшим областям памяти удаленных процессов.

Литература

1. R.Thakur, R.Rabenseifner, W.Gropp. Optimization of Collective Communication Operations in MPICH
2. Лацис А.О. Вычислительная система МВС-Экспресс, http://www.kiam.ru/MVS/research/mvs_express.html.
3. Горбунов В.С., Лацис А.О., Иванов А.Н. О построении суперкомпьютеров на основе интерфейса PCI-EXPRESS //Материалы международной научно-технической конференции СКТ-2010, 27 сентября — 2 октября 2010, Дивноморское, Россия. — Таганрог.
4. Соколов А.А. Особенности модели параллельного программирования для архитектур с сетью МВС-Экспресс. XXXIX Международная научно-практическая конференция «НЕДЕЛЯ НАУКИ СПбГПУ», декабрь 2010.

Эффективный мелкозернистый параллельный алгоритм сортировки методом «слияния» на параллельной потоковой вычислительной системе

М.Ж. Акжолов

Институт проблем проектирования в микроэлектронике РАН

Предлагается эффективный мелкозернистый параллельный алгоритм сортировки, в основе которого лежит существующий классический алгоритм сортировки «слиянием». Алгоритм написан на входном языке модели параллельной потоковой вычислительной системы (ППВС) [1,2] и предназначен для демонстрации возможностей этой системы.

Сортировка происходит поэтапно. На каждом этапе имеется несколько пар порций, которые попарно сливаются в одну порцию, то есть количество порций уменьшается в два раза, а в каждой порции количество данных удваивается. Этот процесс продолжается, пока не останется одна порция. Слияние каждой пары порций происходит последовательно на одном вычислительном ядре (ВЯ), поэтому на заключительных этапах число используемых ВЯ резко сокращается.

Для того чтобы эффективно использовать вычислительные ресурсы, работа вышеуказанного алгоритма разбивается на два режима. Первый режим работает, пока количество пар порций способно загрузить работой все имеющиеся ядра. Начиная с некоторого этапа, во втором режиме, каждая порция делится на две части (младшую и старшую) относительно общего среднего значения, и попарное слияние частей производится в разных ядрах. Поэтому количество обрабатываемых порций сохраняется до конца работы программы, и это дает возможность эффективно использовать имеющиеся вычислительные ресурсы. Алгоритм хорошо масштабируется: удвоение количества ВЯ дает почти двойное ускорение.

Эксперименты показали, что при использовании хорошо подобранной функции распределения общее время параллельного выполнения программы значительно сокращается за счет улучшения равномерности загрузки ВЯ и минимизации объема передаваемой информации на всех уровнях иерархии вычислительной системы.

Надо отметить, что при выполнении данного алгоритма на модели ППВС, благодаря отсутствию глобальных (барьерных) синхронизаций, процессы слияния порций идут на двух или трех этапах одновременно, даже при переходе на второй режим.

Было проведено сравнение эффективности выполнения данного алгоритма с выполнением программы Integer Sort ($N=2^{23}$, класс A) из пакета NPB на вычислительном комплексе MVS100K МСЦ РАН с использованием технологии MPI. Оказалось, что она масштабируется лишь до 32 процессоров. Работа алгоритма для ППВС с соответствующим количеством данных демонстрирует устойчивое масштабирование до 2^{16} ВЯ, обгоняя более чем в 100 раз наилучший результат для MVS100K. При этом на последних этапах на одну порцию приходилось менее 100 значений.

Заметим, однако, что данный результат был достигнут на хорошо перемешанном исходном массиве. На плохо перемешанном массиве алгоритм работает даже хуже исходного варианта с одним первым режимом. Поэтому в общем случае придется решать задачу дважды: сначала для хорошего перемешивания, а затем для сортировки.

Литература

1. Стемпковский А.Л., Левченко Н.Н., Окунев А.С, Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // журнал "Информационные технологии" №10, 2008, с. 2-7.

2. Стемпковский А.Л., Климов А.В., Левченко Н.Н., Окунев А.С. Методы адаптации параллельной потоковой вычислительной системы под задачи отдельных классов // журнал «Информационные технологии и вычислительные системы», 2009. №3, с. 12-21.

Использование гибридных вычислительных систем на основе графических процессоров при решении задач геостатистического моделирования

М.В. Андреев, Р.К. Газизов, А.Л. Штангеев, А.В. Юлдашев, А.А. Яковлев

Уфимский государственный авиационный технический университет

Гибридные вычислительные системы, построенные на основе графических процессоров, обладают высокой производительностью при решении самого широкого круга класса научных и прикладных вычислительных задач. Кроме того, использование гибридных систем является энергетически и экономически эффективным [1]. Актуальными задачами являются адаптация существующих и разработка новых алгоритмов и программ численного моделирования для их эффективного выполнения на архитектурах гибридных вычислительных систем.

В последние годы геостатистическое моделирование резервуаров становится все более востребованным компонентом исследований для оптимизации стратегии разработки нефтегазовых месторождений. Геостатистика позволяет строить детализированные количественные модели неоднородностей строения резервуаров в тех их частях, где геофизические параметры ненаблюдаемы и их значения неизвестны. Однако, методы геостатистического анализа и моделирования, заложенные в современных коммерческих продуктах, не позволяют в полной мере учесть анизотропность и нестационарность пластовых свойств.

В докладе представлены работы по созданию высокопроизводительного симулятора для решения задач геостатистического моделирования [2], ориентированного на выполнение, как на традиционных вычислительных системах с многоядерными процессорами, так и на гибридных вычислительных системах с графическими процессорами. В основу симулятора положен новый метод построения обусловленных геостохастических геологических моделей по скважинным данным с использованием спектрального представления стационарных случайных полей, учитывающий анизотропность и нестационарность пластовых свойств, а также обладающий существенным ресурсом параллелизма.

На данный момент средствами технологии PGI Accelerator выполнена адаптация наиболее трудоемких участков кода симулятора под гибридные вычислительные системы с графическими процессорами NVIDIA с поддержкой CUDA. Применение высокоуровневой технологии PGI Accelerator позволило получить рабочую версию симулятора за достаточно короткое время.

Проведено тестирование производительности полученной версии симулятора при расчете реальных моделей на двухпроцессорной рабочей станции FSC CELSIUS V840 на базе процессоров AMD Opteron 2214 с различными графическими ускорителями (GeForce GTX 295, 460, 470, 480 и Tesla C1060). Тестирование показало ускорение суммарного времени выполнения адаптированных участков кода более 8 раз при расчете на CPU+GPU относительно суммарного времени их выполнения в многопоточной OpenMP-версии при расчете на 4 ядрах CPU.

В настоящее время проводится тестирование полученной версии симулятора на гибридных вычислительных системах, оборудованных вычислителями Tesla серии 2000. Исследуются возможности оптимизации данной версии симулятора, кроме того ведется разработка новой версии симулятора с использованием технологии CUDA.

Литература

1. Джораев А.Р. Гибридные вычислительные системы на основе GPU для задач биоинформатики // Компьютерные исследования и моделирование. 2010. Т. 2, № 2. С.163-167.
2. Андреев М.В., Галеев Э.Р., Штангеев А.Л., Юлдашев А.В., Яковлев А.А. Опыт портирования геостатистического симулятора на архитектуру GPU средствами технологии PGI Accelerator // Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2010): Материалы X международной конференции (Пермь, 1 – 3 ноября 2010 г.), в двух томах – Пермь: Издательство ПГТУ, 2010. – Том 1. С. 37-41.

Экспериментальная система распределенных вычислений в компьютерном базисе исчисления древовидных структур для сетей с недетерминированными ресурсами

С.Е. Артамонов[2], Ю.С. Затуливетер[1], В.А. Козлов[2], В.С. Подлазов[1], В.В. Сергеев[2], А.В. Топорищев[1], Е.А. Фищенко[1]

Учреждение Российской академии наук Институт проблем управления им. В. А. Трапезникова РАН[1], Общество с ограниченной ответственностью «ИДМ»[2]

Для распространения свойств универсальной программируемости системы ПАРСЕК, основанной на компьютерном базисе исчисления древовидных структур [1], с внутренних ресурсов компьютеров на распределенные вычислительные ресурсы локальных и глобальных сетей с использованием библиотеки функций управления протоколом TCP/IP реализован базис управления распределенными вычислениями в сетевой архитектуре Peer-to-Peer [2]. Особенность построенного решения – организация распределенных структурно-сложных вычислений в едином адресном пространстве, охватывающем оперативную память компьютеров, предоставляющих ресурсы через сети, см. таблицу.

Таблица. Формат единого адресного пространства распределенной оперативной памяти

Формат сквозного адреса (80bit)		
IP адрес (32bit)	IP порт (16bit)	Адрес в ОЗУ (32bit)

Распределенные вычисления осуществляются в предположении недетерминированности компьютерных ресурсов в сетях. При этом целостность распределенного процесса сохраняется при отказах или при непредсказуемом выключении/выключении компьютеров. Для присоединения компьютеров к распределенной системе требуется установка простейшей программы сетевой связи, которая передается при регистрации IP-адресов компьютеров.

Экспериментальная система испытана на задаче [3], которая допускает разбиение на многие слабосвязанные фрагменты. Обеспечивается сокращение времени счета пропорциональное числу вовлеченных компьютеров, связанных через Интернет.

В ходе испытаний системы экспериментально подтверждена осуществимость "бесшовного" программирования распределенных вычислений в едином адресном пространстве компьютерного исчисления древовидных структур. При этом показано:

- распределенные вычисления осуществляются в ресурсах общедоступных сетей без добавления каких-либо новых системных программных слоев;
- трудоемкость программирования практически не зависит от количества задействованных компьютеров, связанных сетями;
- эффективность системы выражается относительно малой долей расхода времени на управление распределенными процессами по сравнению со временем счета фрагментов прикладных задач высокой вычислительной сложности.

Литература

1. Затуливетер Ю.С., Халатян Т.Г. ПАРСЕК - язык компьютерного исчисления древовидных структур с открытой интерпретацией. Стендовый вариант системы программирования. - М., 1997 (Препр. ИПУ РАН).
2. Затуливетер Ю.С., Топорищев А.В. Язык Парсек: программирование глобально распределенных вычислений в модели исчисления древовидных структур // Проблемы управления. 2005. №4. С.12-20.
3. Каравай М.Ф., Пархоменко П.П., Подлазов В.С. Комбинаторные методы построения двудольных однородных минимальных квазиполных графов (симметричных блок-схем) // АиТ. 2009. №. 2. С. 153-170.

Применение GPU для решения задачи автоматической фильтрации облачных масс на графических процессорах

В.Х. Багманов, Р.К. Газизов, А.Х. Султанов, И.Р. Фатхулисламов

Уфимский государственный авиационный технический университет

В настоящее время растет интерес к параллельным системам обработки спутниковых изображений с использованием графических процессоров. Это связано с тем, что появилась насущная потребность в обработке крупноформатных изображений с высокой скоростью. Как показывает практика последних лет, GPU все чаще стали использовать для расчетов общего назначения, учитывая его высокую пропускную способность памяти и высокую вычислительную производительность.

На первой стадии процесса получения продуктов обработки спутниковых данных выполняется первичная обработка цифровых изображений. Обязательным этапом обработки полученных данных является выявление облачных масс. От точности решения данной задачи зависят множества других задач, например, точная географическая привязка спутниковых изображений, выявление движения облачных масс и т.д.

В настоящей работе был рассмотрен алгоритм по обнаружению облачных масс со спутника NOAA-14/AVHRR. Работа основана на методе, описанном в [1] и [2] и состоит из 7 шагов. Отличие от существующих методов состоит в выборе точек, которые были использованы для Расчета порогов.

Итак, для решения задачи предлагается следующий алгоритм:

1. Выбор точек, которые в дальнейшем будут использованы для расчета порогов.
2. Расчет порогов.
3. Поиск точек, температура которых ниже рассчитанного на предыдущем шаге порога.
4. Поиск точек, у которых высокий коэффициент отражения и высокая температура.
5. Поиск границ облачных масс.
6. Поиск полупрозрачных облаков по методу, который описан в [1].
7. Поиск точек, у которых коэффициент отражения выше, а температура ниже, чем на предыдущем снимке.

В данной работе была выполнена параллельная реализация алгоритма на языке программирования Matlab с использованием GPU. Для решения задачи был использован коммерческий продукт Jacket [3]. Для хранения исходного изображения использовали текстурную память графической карты. Исходное изображение было разделено на колонки, равные количеству потоков. Каждый поток производил поиск облачных пикселей в своей колонке. Шаги 1,2 и 5 были рассчитаны последовательно, шаги 3,4,6 и 7 параллельно.

В данной работе была рассмотрена GPU-реализация алгоритма по выявление облачных масс на спутниковых изображениях. Проведенные эксперименты показали, что использование графических процессоров в качестве вычислительного устройства себя оправдывает. В настоящий момент производится тестирование предложенного метода на других графических процессорах и производится оптимизация исходного кода.

Литература

1. Saunders, R.W., and K.T. Kriebel, 1988, An improved method for detecting clear sky and cloudy radiances from AVHRR data. Int. J. Remote Sensing, 9, 123-150.
2. Derrien M., B. Farki, L. Harang, H. LeGleau, A. Noyalet, D. Pochic and A. Sairouni, 1993, Automatic cloud detection applied to NOAA-11/AVHRR imagery. 46, 246-267.Mehta M., DeWitt D.J. Data Placement in Shared-Nothing Parallel Database Systems // The VLDB Journal. January 1997. Vol. 6, No. 1. P. 53-72.
3. http://www.accelereyes.com/products

Решение прямой и обратной задачи диагностики кровеносных сосудов на ЭВМ с параллельной архитектурой

Л.П. Басс[1], А.В. Быков[2], В.С. Кузнецов[3], Е.М. Лоскутов[4], О.В. Николаева[1], А.В. Приезжев[2], Ю.В. Яхно[4]

Институт Прикладной Математики им.М.В. Келдыша РАН[1],
Московский Государственный Университет и Университет г. Оулу (Финляндия)[2],
Российский Научный Центр «Курчатовский Институт»[3],
Институт Прикладной Физики РАН, Н. Новгород[4]

Рассматривается задача обучения нейронной сети для определения диаметра и глубины залегания кровеносного сосуда в биоткани по отраженному от фантома излучению постоянного лазера. Обучение осуществляется на основе результатов моделирования распространения лазерного излучения в сильно рассеивающей среде методом Монте-Карло на параллельных компьютерах.

Рассматривается модельная 3D-задача определения размера и расположения кровеносного сосуда в биоткани. В качестве модели объекта рассматривается параллелепипед размером 25 мм*25 мм*20 мм с включенной цилиндрической неоднородностью (сосудом) диаметра d, залегающего на глубине h параллельно верхней грани параллелепипеда. Зондирование объекта осуществляется тонким лазерным пучком, помещенным в центре верхней грани. Диагностика выполняется по диффузно рассеянному от 3D-области лазерному излучению, детектируемому на верхней поверхности параллелепипеда матрицей приемников размером $\Delta x=\Delta y=0.2$ мкм.

Моделирование процесса распространения лазерного излучения в сильнорассеивающей среде осуществляется методом Монте-Карло с использованием траекторий 10^9 фотонов в среде и занимает время порядка 200 минут на 100 процессорах суперкомпьютера МВС-100к. Распараллеливание вычислений осуществляется с помощью команд языка MPI. Для проверки точности решения задачи методом Монте-Карло используется решение уравнения переноса излучения сеточным методом дискретных ординат.

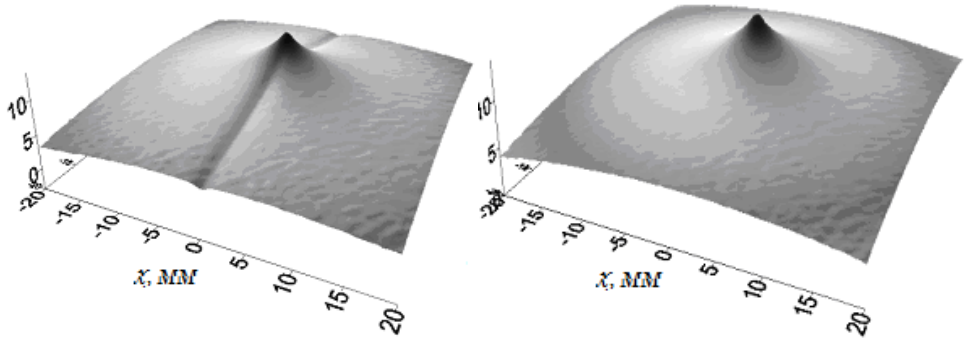


Рис. 1. Поверхностное распределение интенсивности $I(x,y)$ излучения, диффузно отраженного от сильнорассеивающей среды с цилиндрической неоднородностью (сосудом) диаметра d=1 мм, находящейся на глубине h=1 мм перпендикулярно оси x (а) и без сосуда (б).

Полученные распределения интенсивности излучения I (см. Рис 1) позволяют решить обратную задачу восстановления параметров сосуда h и d с помощью нейронной сети.

Отметим, что решение прямых задач настолько времязатратно, что возможно лишь при использовании параллельных вычислений.

Программный модуль для создания параметрических наборов данных, характеризующих нефтяные месторождения

Р.К. Газизов[1], А.В. Гладков[2], И.А. Исламгулов[1], Д.Е. Кондаков[2],
А.Л. Штангеев[1], А.В. Юлдашев[1]

Уфимский государственный авиационный технический университет[1],
ЗАО «Центр технологий моделирования»[2]

Гидродинамическое моделирование является сложным, но вместе с тем и очень полезным инструментом для анализа разработки нефтяных и газовых месторождений. Сложность уравнений, лежащих в основе современных гидродинамических симуляторов, приводит к тому, что с ростом достоверности модели существенно возрастает и время расчета. В тоже время, для оперативного анализа текущего состояния разработки зачастую требуется иметь результаты расчета различных вариантов модели здесь и сейчас. В связи с этим, в некоторых случаях может быть полезен параметрический набор заранее рассчитанных типовых гидродинамических моделей. К примеру, путем предварительного расчета с помощью гидродинамического моделирования кривой эффективности заводнения для различных значений соотношения подвижностей и коэффициента неоднородности Дикстра-Парсона может быть получена библиотека кривых заводнения, которую можно использовать для анализа истории и текущего состояния заводнения нефтяных месторождений.

В целях автоматизации формирования и дальнейшего наполнения такой библиотеки нами был создан кроссплатформенный (Linux/Windows) программный модуль, ориентированный на работу как на персональном компьютере, так и на многопроцессорной вычислительной системе, решающий задачи подготовки, расчета и обработки параметрического набора гидродинамических моделей в соответствии со следующими стадиями:

- предгенерация (формирование множества параметризованных входных файлов);
- генерация (формирование параметрического набора гидродинамических моделей на основе шаблона модели, а также полученных входных файлов);
- моделирование (расчет гидродинамических моделей, а также обработка результатов моделирования в целях экстракции необходимой информации);
- формирование библиотеки (заполнение результирующей таблицы информацией, полученной на предыдущей стадии).

Так как модели могут быть рассчитаны и обработаны независимо, использование многопроцессорных систем позволяет существенно ускорить наиболее трудоемкую стадию моделирования.

Кроссплатформенность программного модуля обеспечена различными средствами. Во-первых, программная реализация выполнена на языке Python. Во-вторых, на стадии моделирования использована консольная версия симулятора websim [1], рассчитанная на выполнение в Windows, однако также успешно запущенная в Linux средствами Wine. В-третьих, работа модуля на многопроцессорных системах обеспечена посредством поддержки менеджера ресурсов TORQUE, который с недавних пор может работать не только в Linux, но и в Windows [2].

Проведено экспериментальное исследование эффективности расчета параметрического набора из 1155 гидродинамических моделей при помощи разработанного программного модуля на кластерной системе УГАТУ. Основным результатом явилось практически линейное снижение времени выполнения с увеличением количества задействованных узлов кластера.

Литература

1. WebSim – Reservoir Simulation Online.
   URL: http://www.websim.ru (дата обращения: 06.02.2011).
2. Ильенко И.И., Лапа В.А., Чернявский Е.В. Портирование системы пакетной обработки Torque и планировщика Maui на операционную систему Windows //III межд. науч. конф. SSA’2010: докл. конф.: в 2 томах. Минск: ОИПИ НАН Беларуси. 2010. – Т.1. С. 60-64.

Задача о расстановке визуальных дорожных знаков

А.А. Горбенко, М.Л. Морнев, В.Ю. Попов

Уральский государственный университет им. А.М. Горького

Визуальная навигация широко используется в современной робототехнике (см., например, [1]). В большинстве случаев методы визуальной навигации основываются на том или ином способе выделения визуальных дорожных знаков. Даже при использовании каких-либо других подходов, например, реактивного движения или топологической навигации, применение дорожных знаков в качестве вспомогательного метода ориентирования существенно повышает качество работы навигационной системы. В качестве маяков могут использоваться как искусственные дорожные знаки, так и различные объекты из окружения.

Навигация на основе дорожных знаков представляет собой наиболее простой и эффективный метод ориентирования. Использование искусственных дорожных знаков обеспечивает наиболее надежный способ для поддержания автономного функционирования роботов. Однако применение искусственных дорожных знаков требует либо первоначального оборудования зоны действия робота, либо наличия у робота функции самостоятельной установки дорожных знаков. И в том, и в другом случае уменьшение количества используемых дорожных знаков существенно влияет на повышение рентабельности применения робота.

Использование естественных дорожных знаков позволяет избежать затрат, связанных с установкой маяков. Однако естественные дорожные знаки требуют более трудоемких способов обработки визуальной информации, чем искусственные (см., например, [2]). При этом поиск даже простейших закономерностей требует решения **NP**-трудных проблем (см., например, [3]). Кроме того, часто используются самообучающиеся навигационные системы (см., например, [4]), что приводит к быстрому разрастанию базы дорожных знаком и, в конечном счете, резкому падению производительности бортовых вычислительных систем. Таким образом, использование как искусственных, так и естественных дорожных знаков требует минимизации множества применяемых маяков. В первом случае мы уменьшаем затраты, связанные с установкой знаков. Во втором случае — минимизируем расход вычислительных ресурсов на поиск новых дорожных знаков и идентификацию имеющихся.

Нами дана алгоритмическая формализация задачи о минимизации количества используемых маяков. Доказана **NP**-трудность соответствующей алгоритмической проблемы. Получено сведение этой проблемы к задаче выполнимости. Найдены эффективные подходы к решению задачи о минимизации количества используемых маяков на суперкомпьютере при помощи генетических алгоритмов для задачи выполнимости.

Список литературы

1. Yang G., Hou Z.-G., Liang Z. Distributed visual navigation based on neural Q-learning for a mobile robot // International Journal of Vehicle Autonomous Systems. 2006. Vol. 4, N. 2-4. P. 225–238.

2. Basri R., Rivlin E. Localization and Homing Using Combinations of Model Views // AI. 1995. Vol. 78, N. 1-2. P. 327–354.

3. Popov V. The Approximate Period Problem // IAENG International Journal of Computer Science. 2009. Vol. 36, N. 4. P. 268–274.

4. Hagras H., Colley M., Callaghan V. Online Learning and Adaptation of Autonomous Mobile Robots for Sustainable Agriculture // Autonomous Robots. 2002. Vol. 13. P. 37–52.

Разработка и исследование параллельного алгоритма оптимизации развития динамической транспортной сети

Н.Л. Григоренко, А.В. Жарков, Д.Г. Пивовартчук,Н. Н. Попова

Московский Государственный Университет им. М.В.Ломоносова

Под динамической транспортной сетью понимается математическая модель сети поставок, объединяющая некоторое количество поставщиков, потребителей и дистрибьютеров, параметры которой стохастически увеличиваются со временем. Под развитием сети подразумевается увелчение пропускных способностей рёбер графа транспортной сети с течением времени. Под оптимальным развитием понимается такая последовательность увеличения пропускной способности опередeлёных рёбер графа, которая соотвествует критерию качеству. Критерий качества рассчитывается по всевозможным сценариям, порожденным марковским процессом развития транспортной сети. Для решение данной оптимизационной задачи необходимо решить подзадачу нахождения максимального потока в графе для каждого возможного набора параметров транспортной сети и затем методом динамического программирования найти оптимальную последовательность развития пропускных способностей рёбер.

Была выполнена последовательная реализация описанного алгоритма на основе библиотеки GLPK для решения подзадач линейного программирования симплекс-методом. Проведённый численный эксперимент и аналитические оценки сложности алгоритма показывают, что время решения задачи оптимизации растет экспоненциально. Предлагается параллельный алгоритм, который основан на распределении подзадач нахождения максимального потока по процессам. Обмен между данных процессами реализован с помощью интерфейса MPI. Распределение подазадач производит мастер-процесс методом динамической балансировки нагрузки. Рассмотрена возможность использования гибридной архитектуры BlueGene/P для ускорения вычислений симлекс-метода на основе автораспараллеливания компилятора. Проведённый числинный эксперимент показал, что механизм автораспараллеливания компилятора позволяет сократить время решения задачи до 1.5 раз при использовании четырёх ядер.

Исследование эффективности параллельного алгоритма на суперкомпьютерах BlueGene/P и СКИФ-МГУ показывает, что наиболее высокая эффективность использования процессоров достигается при минимальном времени выполнения последовательной части алгоритма, которая отвечает за нахождение оптимальной последовательности. Это происходит в случае, если размер графа транспортной сети достаточно большой по сравнению с числом моментов времени. Поэтому масштабируемость алгоритма оказывается достаточно высокой для транспортных сетей с числом рёбер более 10^4, развитие которой необходимо оптимизировать в течении 10 и более моментов времени.

Литература

1. А.В. Жарков, Д.Г. Пивовартчук. Разработка и исследование параллельного алгоритма решения задачи оптимизации развития инфраструктуры типа поставщик-потребитель // Труды Пятой Международной конференции «Параллельные вычисления и задачи управления» PACO’2010. Институт проблем управления им. В.А. Трапезникова РАН,--2010,--С. 433-446

Ю. К. Демьянович [1]

АЛГОРИТМЫ РАСПАРАЛЛЕЛИВАНИЯ С ИСПОЛЬЗОВАНИЕМ БИОРТОГОНАЛЬНЫХ СИСТЕМ

Yuri.Demjanovich@JD16531.spb.edu

1. Введение

Известны многочисленные трудности, возникающие при распараллеливании алгоритмов решения вычислительных задач (см. [1-5]). Многие из этих трудностей устраняются (хотя бы частично) при использовании различных методов локализации. В данной работе рассматривается одно из мощных средств локализации: использование биортогональных систем. Такие системы приводят к существенным упрощениям вычислений в весьма общей ситуации; в данном сообщении эти упрощения рассмотрены в нескольких (весьма важных на практике) частных ситуациях, а именно для интерполяционных задач (с использованием многочленов и сплайнов) и для одного варианта вэйвлетного разложения числового потока (см. [6-7]).

2. Распараллеливание в задаче интерполяции

Каа известно, общая задача интерполяции состоит в следующем: имеется набор функционалов $\{g_i\}_{i\in J}$, заданных на некотором классе функций $\mathbb{F}$, и множество чисел $\{y_i\}_{i\in J}$; требуется найти функцию $f \in \mathbb{F}$, удовлетворяющую условию

$$g_i(f) = y_i \qquad \forall i \in J. \tag{2.1}$$

В простейшем случае функции из множества $\mathbb{F}$ непрерывны, функционалы g_i представляют собой значения функции в точках некоторой сетки $X = \{x_i\}_{i\in J}$, лежащей на вещественной оси, так что

$$g_i(f) \stackrel{\text{def}}{=} f(x_i) \qquad \forall i \in J. \tag{2.2}$$

В этом случае задача (1.1) принимает вид

$$f(x_i) = y_i \qquad \forall i \in J \qquad f \in \mathbb{F}. \tag{2.3}$$

Расширенная задача интерполяции включает вычисления найденной функции $f(t)$ в точках $t = t_s$ некоторого (конечного) множества точек $T \stackrel{\text{def}}{=} \{t_s\}$ (вообще говоря, не совпадающего с сеткой X).

[1]Работа частично поддержана грантами РФФИ 10-01-00245 и 10-01-00297.

Автоматизация выявления причин потери производительности при взаимодействии процессов в MPI программах

А.В. Дергунов

Нижегородский государственный университет им. Н.И. Лобачевского

Наиболее часто для анализа производительности MPI программ используются системы, которые осуществляют сбор и визуализацию трассы выполнения программы (например, Intel Trace Collector and Analyzer). При этом анализ узких мест производительности осуществляется пользователем вручную с помощью детального анализа трассы. Разработанная автором система позволяет автоматизировать анализ трасс MPI программ с целью выявления причин потерь производительности при взаимодействии процессов.

Схема работы системы представлена на **Рис. 1**. Исходными данными является трасса выполнения MPI приложения (полученная, например, с помощью Intel Trace Collector). Входные данные анализируются модулем автоматического анализа трассы. Для этого используется описание шаблонов неэффективного взаимодействия MPI процессов на специальном языке, разработанном в рамках системы. Результат анализа – список выявленных причин недостаточной производительности пользовательского приложения с указанием степени их влияния на общее время работы приложения.

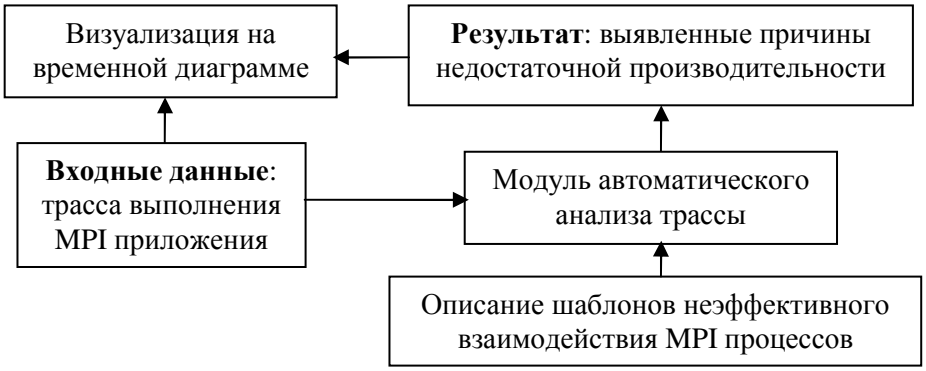


Рис. 1. Схема работы системы

С помощью разработанного языка описания шаблонов взаимодействия MPI процессов в системе реализовано выявление следующих причин недостаточной производительности: поздняя посылка данных, поздний прием данных, прием сообщений в неверном порядке, 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», задержка перед барьерной синхронизацией и других. Описание некоторых из перечисленных причин недостаточной производительности приведено в [1].

Для проверки разработанной системы было осуществлено повышение производительности MPI программы, реализующей метод Гаусса-Зейделя для численного решения задачи Дирихле для уравнения Лапласа.

Литература

1. Карпенко С.Н., Дергунов А.В. Модели и программные средства повышения производительности MPI-приложений // Технологии Microsoft в теории и практике программирования: Материалы конференции. Н.Новгород: Изд. Нижегородского госуниверситета, 2009. С. 188-192.

Вычислительная технология распознавания цветных изображений по критериям сопряженности

Р.К. Захаров, В.А. Фурсов

Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет)

В работе рассматриваются методы, алгоритмы и информационная технология распознавания цветных изображений по критериям сопряженности и с использованием разложений по базису Карунена-Лоэва, в частности, приводятся алгоритмы отбора информативных признаков по критериям сопряженности. Рассматриваются возможности применения для этих алгоритмов различных типов декомпозиции.

1. Постановка задачи

Предполагается, что имеется M изображений каждого из K объектов. Каждое изображение представляется вектором $\mathbf{x}=[x_1,x_2,...,x_N]^T$ размерности N, где $x_1,x_2,...,x_N$ – признаки. Векторы, соответствующие изображениям одного объекта, составляют класс. Совокупность векторов признаков всех классов образует обучающую выборку. Решение задачи распознавания состоит в конструировании решающей функции $f:R^N\mapsto\{0,1,2,...,K\}$, которая каждому вектору $\mathbf{x}$ ставит в соответствие некоторый класс. Для уменьшения числа неправильных классификаций вводится также класс с номером 0, соответствующий отказу в распознавании.

Из множества $\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_M\}$ векторов каждого класса составляется $N\times M$ -матрица

$$\mathbf{X}=[\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_M]. \qquad (1)$$

Этой матрице ставятся в соответствие так называемая информационная $M\times M$ -матрица:

$$\mathbf{A}=\mathbf{X}^T\mathbf{X} \qquad (2)$$

и ковариационная $N\times N$ -матрица

$$\mathbf{B}=\mathbf{X}\mathbf{X}^T. \qquad (3)$$

Предполагается, что $rank\mathbf{A}=M$. Известно также, что собственные значения $\lambda_i(\mathbf{A}),\ i=\overline{1,M}$ матрицы $\mathbf{A}$ совпадают с ненулевыми собственными значениями матрицы $\mathbf{B}$, а собственные векторы матрицы $\mathbf{B}$, соответствующие ненулевым собственным значениям, образуют ортогональный базис (разложение Карунена-Лоэва).

2. Исследуемые решающие правила

Наряду с широко известными решающими правилами, основанными на вычислении евклидовых расстояний и меры близости Махаланобиса в работе исследуются новые, развиваемые авторами, подходы. В частности, исследуются алгоритмы распознавания, основанные на использовании в качестве мер близости показателей сопряженности. Соответствующие решающие правила строятся на следующей основе. В рассмотрение вводится

так называемый показатель сопряженности с подпространством, натянутым на векторы признаков образов объектов из заданного класса:

$$R_k = \frac{\mathbf{x}^T \mathbf{X}_k \left[\mathbf{X}_k^T \mathbf{X}_k \right]^{-1} \mathbf{X}_k^T \mathbf{x}}{\mathbf{x}^T \mathbf{x}} . \qquad (4)$$

Здесь $\mathbf{x}$ – вектор признаков неизвестного образа, предъявленный для установления близости к k-му классу, а $\mathbf{X}_k$ – $N \times M$-матрица, составленная из векторов образов, принадлежащих k-му классу.

Наряду с указанным, в работе рассматривается также показатель сопряженности с нуль-пространством транспонированной матрицы $\mathbf{X}_k$, который вычисляется как

$$S_k = \frac{\mathbf{x}^T \mathbf{T}_k \mathbf{T}_k^T \mathbf{x}}{\mathbf{x}^T \mathbf{x}} . \qquad (5)$$

Здесь $\mathbf{T}_k$- матрица, составленная из собственных векторов, соответствующих нулевым собственным значениям матрицы $\mathbf{B}_k = \mathbf{X}_k \mathbf{X}_k^T$, а $\mathbf{X}_k$ – $N \times M$-матрица, та же, что и в (5).

С использованием указанных показателей сопряженности в предположении, что для каждого (k-го) класса сформирована одна из следующих $N \times N$-матриц $\mathbf{Q}_{(*)}$:

$$\mathbf{Q}_{k,R} = \mathbf{X}_k \left[\mathbf{X}_k^T \mathbf{X}_k \right]^{-1} \mathbf{X}_k^T \qquad (6)$$

или

$$\mathbf{Q}_{k,S} = \mathbf{T}_k \mathbf{T}_k^T , \qquad (7)$$

соответствующая решающая функция $f(\mathbf{x})$ строится следующим образом. Вектор $\mathbf{x}$ принадлежит m-му классу, то есть $f(\mathbf{x}) = m, \quad m = 1, 2, \ldots K$,

если $R_m = \max_k R_k$, где $R_k = \frac{\mathbf{x}^T \mathbf{Q}_{k,R} \mathbf{x}}{\left(\mathbf{x}^T \mathbf{x} \right)}$, (8)

либо $S_m = \min_k S_k$, где $S_k = \frac{\mathbf{x}^T \mathbf{Q}_{k,S} \mathbf{x}}{\left(\mathbf{x}^T \mathbf{x} \right)}$. (9)

При использовании порогового значения T_0, решающая функция дополняется правилом

$f(\mathbf{x}) = 0$, если $R_m \le 1 - T_0$ или $S_m \ge T_0$. (10)

3. Применение к распознаванию цветных изображений.

В работе изучаются вопросы построения процедур распознавания цветных изображений с использованием решающих правил, основанных на показателях сопряженности. В частности, исследуются вопросы построения эффективных алгоритмов отбора информативных признаков по показателям сопряженоти (4), (5). Один из основных вопросов, который возникает при построении процедур распознавания цветных изображений: насколько информативными оказываются компоненты,описывающие цветовые составляющие изображеинй.

В работе исследуется качество распознавания при различных вариантах представления многокомпонентных изображений, в т.ч. при их разложении по ортогональным базисам

(Карунена-Лоэва). Для различных вариантов представлений изображений сравниваются результаты, полученные при использовании различных решающих правил. В частности, проясняется вопрос зависимости качества распознавания от типа цветовых пространств, которые используются для представления дополнительных цветовых компонент векторов признаков.

4. Применение к задаче кластеризации цветных изображений.

Известно, что решение о принадлежности образа классу может оказаться ошибочным в случае, если векторы одного класса обучающей выборки сильно отличаются друг от друга. Известный путь преодоления этой проблемы состоит в разбиении обучающих классов на подклассы – кластеризации. В работе исследуется эффективность применения для формирования кластеров в качестве меры близости показателей сопряженности.

Для класса представленного в обучающей выборке множеством $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_M\}$ векторов образов алгоритм состоит в следующей последовательности шагов: выбор двух наиболее удаленных (по косинусу угла между векторами) образа, например, $\mathbf{x}_1$ и $\mathbf{x}_2$ (инициализация матриц $\mathbf{X}_1 = [\mathbf{x}_1]$ и $\mathbf{X}_2 = [\mathbf{x}_2]$); произвольный выбор вектора $\mathbf{x}_i$ из числа оставшихся и вычисление показателя сопряженности со столбцовыми пространствами матриц $\mathbf{X}_1$ и $\mathbf{X}_2$; добавление этого вектора $\mathbf{x}_i$ в качестве нового столбца к матрице, соответствующей ближайшему классу. Алгоритм может применяться к каждому из полученных кластеров для дальнейшего разбиения на подклассы.

При обработке цветных изображений большое влияние на качество кластеризации также может оказывать тип цветового пространства в котором представляются изображения.

5. Задачи исследования параллельных алгоритмов

Применение расширенных матриц признаков, включающих дополнительные цветовые составляющие, приводит к существенному возрастанию вычислительных затрат. Поэтому в работе исследуются различные схемы декомпозиции, для построения параллельных и распределенных алгоритмов.

Экспериментальные исследования проводятся на вычислительном кластере. Цель экспериментов состоит в проверке условий изоэффективности при различной вычислительной сложности задачи. Результатом будут рекомендации по выбору числа вычислительных узлов в зависимости от объема обучающей выборки для каждого типа решающего правила.

Литература

1. Zhao W., Chellappa R., Rosenfeld A., Phillips P.J. Face Recognition // A Literature Survey, ACM Computing Surveys, 2003. P. 399-458.
2. Turk M.A., Pentland A.P. Face Recognition Using Eigenfaces, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Maui, Hawaii, USA, 3-6 June 1991. P. 586-591.

3. Belhumeur P.N., Hespanha J.P., Kriegman D.J. Eigenfaces vs. Fisherfaces: Recognition using Class Specific Linear Projection, Proc. of the 4th European Conference on Computer Vision, ECCV'96, 15-18 April 1996, Cambridge, UK, P. 45-58.
4. Fursov V.A., Kozin N.E. Stage-wise learning of radial neural networks // Proceedings of The 12th ISPE International Conference on Concurrent Engineering: Research and Applications, Focus Symposium Recursive Dynamics and Iterated Mappings in Service Modeling and Design, Ft. Worth/Dallas, USA, 25 – 29 July, pp. 391-396.
5. Fursov V.A., Kozin N.E. Algorithm for parallel learning of radial neural networks, Proceedings of The IASTED International Conference on Automation, Control and Applications (ACIT-ACA 2005), Novosibirsk, June 20-24, Russia. 2005. P. 481-485.

Применение параллельных вычислений для решения задачи анализа качества программного кода

С.В. Звездин

Разработка программного обеспечения является сложным процессом, на который человеческий фактор имеет большое влияние. Поэтому характеристики качества удобно выражать лингвистически. В этом случае для построения модели оценки качества кода используется математический аппарат нечеткой логики и нечеткие продукционные модели на основе нейронной сети Ванга-Менделя [1].

Одним из режимов работы модуля по оценке качества является анализ программного кода "на лету". Поэтому работа алгоритма может быть оптимизирована с точки зрения быстродействия путем распараллеливания алгоритма для работы на центральном и графических [1] процессорах.

Поскольку система анализа программного кода реализована на платформе Microsoft .NET 4.0, то для использования параллельных вычислений применяется библиотека TPL [2]. Для вычислений на графических процессорах были использованы библиотеки CUDA.NET, GPU.NET и Accelerator [3]. Сравнение эффективности двух алгоритмов производилось на испытательных площадках, оборудованных центральными процессорами Intel Core 2 Duo P8600 (2x2.4GHz) и Intel Core 2 Quad Q6600 (4x2.4GHz), и видео-адаптером NVIDIA 8800GT (112 потоковых процессоров). Результаты экспериментов приведены на рис. 1.

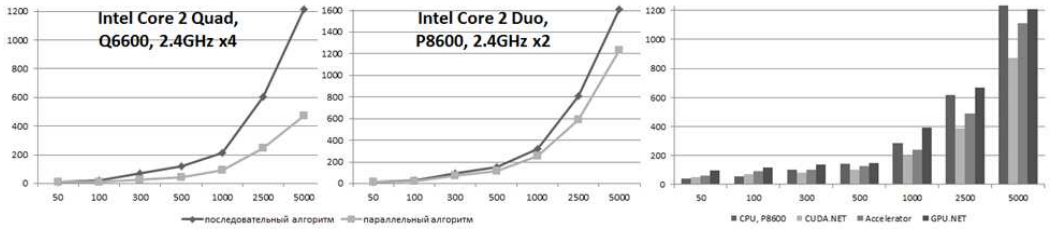


Рис. 1. Затраты времени на анализ программного кода (ось абсцисс - количество оцениваемых программных элементов; ось ординат - время работы алгоритма в мс.)

В результате применения параллельных вычислений на центральном процессоре наблюдается прирост быстродействия на многоядерных процессорах. Применение вычислений на графических процессорах обосновано только в случае анализа очень крупных проектов, содержащих большое количество программного кода (в противном случае можно наблюдать не увеличение, а снижение производительности из-за накладных расходов).

Литература

1. Звездин С.В. Метод определения качественных характеристик программного кода // Научная перспектива. Раздел "Технические науки". №10. 2010. С. 86-88.

2. Leijen D., Schulte W., Burckhardt S. The Design of a Task Parallel Library // ACM SIGPLAN Notices - OOPSLA'09. V. 44. №10. New York: ACM, 2009.

3. Banzhaf W., Harding S. Accelerating evolutionary computation with graphics processing units // GECCO'09 Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. New York: ACM, 2009.

[1] Работа алгоритма на графическом процессоре позволяет также снизить нагрузку на центральный процессор, что положительно влияет на производительность рабочего места в целом.

Эффективная реализация алгоритма поиска лиц людей на изображении для архитектуры NVIDIA CUDA

И.И. Зиновьев

Владимирский государственный университет имени А.Г. и Н.Г. Столетовых

Комплекс алгоритмов для обнаружения объектов, предложенный в 2001 году Полом Виолой (Paul Viola) и Майклом Джонсом (Michael Jones) [1], логически состоит из двух компонентов: обучение классификатора и непосредственно обнаружение объектов на изображении. На практике основные требования к скорости работы предъявляются ко второму алгоритму. В данной работе рассматривается реализация этого алгоритма для архитектуры NVIDIA CUDA, которая позволяет в несколько раз ускорить его работу по сравнению с реализацией на центральном процессоре, представленной в библиотеке OpenCV.

Алгоритм Viola-Jones реализован как одно CUDA-ядро. Это позволило минимизировать пересылки данных через глобальную память графического ускорителя. Правильный выбор размера блока потоков и количества используемых регистров позволяет достичь значения коэффициента использования мультипроцессора порядка 2/3, что является достаточным для нивелирования влияния латентности при работе с памятью.

С целью эффективного использования особенностей работы различный видов памяти графического ускорителя исходные для алгоритма данные, а именно каскад классификаторов и анализируемое изображение, размещаются в текстурной и константной памяти. Это позволяет реализовать эффективный алгоритм масштабирования путем выборки из текстуры с аппаратной билинейной фильтрацией. Из-за большого числа обращений к памяти при работе рассматриваемого алгоритма разделяемая память используется как управляемый L1 кэш. Применение технологии «zero copy», позволяет сократить накладные расходы при пересылке данных между хостом и ускорителем практически к нулю за счет их асинхронного выполнения.

Вычисление интегрального изображения является ключевым шагом в работе алгоритма Viola-Jones. Для облегчения процесса расчета интегрального изображения в данной работе предлагается использовать частичное интегральное изображение, которое вычисляется независимо для участков исходного изображения, уже размещенных в разделяемой памяти мультипроцессоров графического ускорителя. Благодаря этому работа отдельных мультипроцессоров становится независимой, что исключает временные затраты на синхронизацию их работы.

В ходе экспериментов было достигнуто ускорение работы алгоритма на графическом ускорителе от 2 до 5 раз в сравнении с однопоточным вариантом, работающем на центральном процессоре. Такой результат в несколько раз превышает тот, что представлен автором работы [2]. Полученная реализация алгоритма показывает хорошую масштабируемость, что позволяет рассчитывать на еще большее ускорение на более современных графических ускорителях.

В данной работе предложен оригинальный вариант эффективной реализации алгоритма Viola-Jones для архитектуры графических ускорителей производства NVIDIA. Следует отметить, что сам алгоритм не слишком приспособлен для реализации на графической карте. Большое число обращений к памяти, простои потоков при работе каскада классификаторов негативно сказываются на производительности. Но даже при этом полученная реализация алгоритма на современных графических ускорителях работает в несколько раз быстрее, чем аналоги на центральном процессоре.

Литература

1. Paul Viola, Michael J. Jones. Robust real-time face detection // International journal of computer vision 57(2), 137–154, 2004

2. J. P. Harvey. GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA URL: https://ritdml.rit.edu/bitstream/handle/1850/10894/35445_pdf_00B0B24A-DFD8-11DE-9A30-D21AD352ABB1.pdf?sequence=1. (дата обращения: 11.10.2010)

Инфраструктура для параллельного поиска объектов разных классов на изображениях*

Н.Ю. Золотых, Е.А. Козинов, В.Д. Кустикова, И.Б.Мееров, А.Н. Половинкин

Нижегородский государственный университет им. Н.И. Лобачевского

В работе ставится задача создания исследовательской инфраструктуры для поиска объектов заданных классов на изображениях. Инфраструктура должна позволять решать задачу в пакетном режиме. При этом исследователь формирует задание в виде конфигурационного файла с описанием анализируемого изображения и набора классов объектов для поиска.

Предлагается подход, ориентированный на эффективное использование кластерных систем. Инфраструктура построена на основе системы управления кластером «Метакластер»[4]. Программная инфраструктура предоставляет интерфейс, который позволяет выполнять поиск объектов N разных классов на изображении, принудительную остановку поиска, получение состояния решения задачи, получение результатов поиска. Задача поиска объектов конкретного класса решается с использованием алгоритма машинного обучения Latent SVM [1]. Алгоритм поиска, предложенный авторами работы [1], был реализован в ННГУ и продемонстрировал сравнимые показатели качества детектирования объектов [2] на данных VOC 2007 [6]. Программная реализация интегрирована в библиотеку с открытым кодом OpenCV [5].

Для проведения экспериментов использовалась база данных конкурса PASCAL VOC 2007 [6], содержащая фотографии объектов двадцати классов (aeroplane, bicycle, bird, bottle и др.). Проведены две группы экспериментов с использованием разработанной инфраструктуры: последовательный поиск объектов каждого из двадцати классов для коллекции изображений и параллельный поиск объектов всех двадцати классов. Полученные результаты экспериментов показали преимущество выполнения параллельного поиска объектов нескольких классов на изображении перед последовательным поиском.

Литература

1. 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. P. 1627–1645.
2. Золотых Н.Ю., Козинов Е.А., Кустикова В.Д., Мееров И.Б., Половинкин А.Н. Об одном подходе к решению задачи поиска объектов на изображениях // 10-я Международная конференция «Высокопроизводительные вычисления на кластерных системах» (HPC-2010): Материалы конференции (Пермь, ноябрь 1-3, 2010) / Пермь: Изд. ПГТУ, 2010. С.270-277.
3. P.N. Druzhkov, V.L. Eruhimov, E.A. Kozinov, V.D. Kustikova, I.B. Meyerov, A.N. Polovinkin, N.Yu. Zolotykh. On some new object detection features in OpenCV library // 10th International Conference on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-10-2010), December 5-12, 2010 St. Petersburg, Conference Proceedings, Vol. II, P. 91-93.
4. Гергель В.П., Сенин А.В. Разработка системы управления интегрированной средой высокопроизводительных вычислений Метакластер // Вестник ННГУ. 2010 (принято к печати).
5. Официальная страница библиотеки OpenCV. URL:http://sourceforge.net/projects/opencvlibrary (дата обращения: 12.12.2010).
6. Официальная страница конкурса PASCAL Visual Object Classes Challenge. URL:http://pascallin.ecs.soton.ac.uk/challenges/VOC (дата обращения: 12.12.2010).

* Авторы благодарят В. Л. Ерухимова (компания ITSeez) за ценные замечания и полезные обсуждения. Работа выполнена при поддержке федеральной целевой программы «Научные и научно-педагогические кадры инновационной России», госконтракт 02.740.11.5131.

Моделирование фотонно-кристаллических волноводов с помощью параллельного метода конечных объёмов

Т.З. Исмагилов, А.И. Кузьмин
Новосибирский Государственный Университет

Фотонные кристаллы это новый вид оптических материалов обладающих периодически изменяющейся диэлектрической проницаемостью. Такие материалы впервые были предложены в 80х годах 20го века и с тех пор внимание к ним продолжает расти. Основным свойством фотонных кристаллов является наличие полной запрещённой зоны. Это означает, что электромагнитные волны из некоторого диапазона не могут распространяться внутри фотонного кристалла ни в каком направлении. Данное свойство используется для создания различных устройств на основе фотонных кристаллов.

Для численного моделирования фотонно-кристалических структур рассматривается схема Годунова второго порядка со специальным способом монотонизации градиентов. Численный алгоритм распараллеливается с помощью метода геометрической декомпозиции. Параллельный алгоритм применяется для моделирования распространения электромагнитных волн в фотонно-кристаллическом волноводе с изгибом. Рассматриваемый фотонный кристалл имеет квадратную решётку с периодом $a = 0.059$. В качестве структурных элементов выбраны цилиндры радиуса 0.01062. Диэлектрическая проницаемость цилиндров равна 11.56. Изучаются зависимости коэффициентов прохождения от частоты для различных конфигураций изгиба с поворотом на 90 градусов. Рассматриваются 4 конфигурации. Для каждой проводятся расчёты для трёх

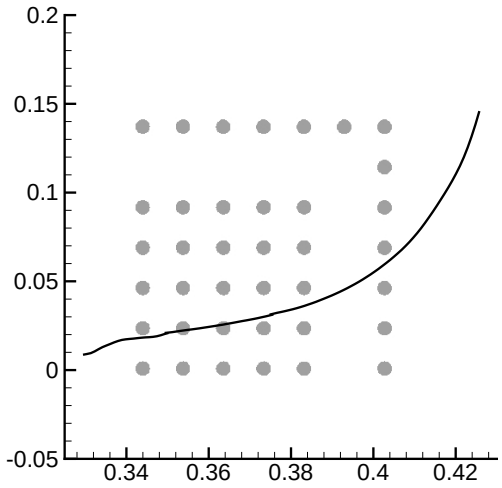


Рис. 1. Спектр прохождения

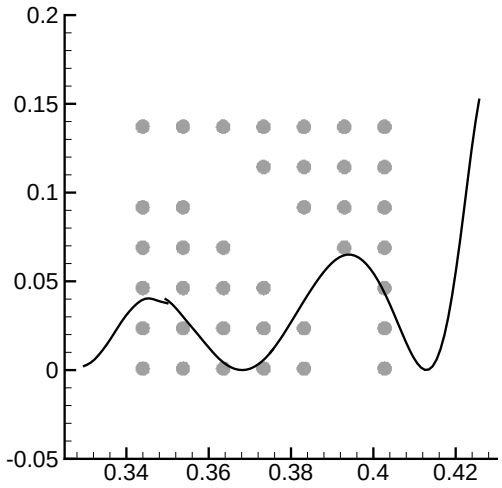


Рис. 2. Спектр прохождения

сигналов для различных наборов частот. Используя результаты расчётов вычисляются спектры отражения и прохождения. Примеры спектров отражения для диапазона частот от $0.33 \times 2\pi/a$ до $0.425 \times 2\pi/a$ для двух конфигураций изгиба показаны на **Рис. 1** и **Рис. 2**. Находятся частоты и конфигурации для которых отражения нет. Полученные результаты хорошо соотносятся друг с другом и с результатами других исследователей. Приводятся результаты тестов на масштабируемость реализации параллельного алгоритма.

Параллельные вычисления при моделировании мышечной активности организма человека

Д.С. Казакова[1], Ю.Б. Линд[2]

ГОУ ВПО «Башкирский государственный университет»[1], ООО «БашНИПИнефть»[2]

Изучение процессов, происходящих в сокращающейся мышце, является одной из наиболее важных и актуальных задач биомеханики, поскольку с мышечной активностью связано функционирование всех физиологических процессов в организме человека.

В настоящее время отсутствует единая математическая модель, которая могла бы описать все стадии процессов сокращения и расслабления в мышце, т.к. расчеты по такой модели требуют большого объема вычислений. Применение параллельных вычислений позволяет рассмотреть весь комплекс процессов, протекающих в мышце.

Используя химическую реакцию сокращения саркомера – элементарной сократительной единицы мышцы [1], на основе закона действующих масс построена математическая модель - задача Коши для системы обыкновенных нелинейных дифференциальных уравнений:

$$\begin{cases} dx_0/dt = -k_1 x_0 (x_1)^2 + \bar{k}_1 (x_2)^2 + k_6 x_6 - \bar{k}_6 (x_8)^2 (x_9)^2 + k_7 x_7 - \bar{k}_7 (x_8)^2; & dx_1/dt = -2k_1 x_0 (x_1)^2 + 2\bar{k}_1 (x_2)^2; \\ dx_2/dt = 2k_1 x_0 (x_1)^2 - 2\bar{k}_1 (x_2)^2 - 2k_2 (x_2)^2 + \bar{k}_2 (x_3)^2 x_4; & dx_3/dt = 2k_2 (x_2)^2 - 2\bar{k}_2 (x_3)^2 x_4 - 2k_3 (x_3)^2 + 2\bar{k}_3 x_5; \\ dx_4/dt = k_2 (x_2)^2 - \bar{k}_2 (x_3)^2 x_4; & dx_5/dt = k_3 (x_3)^2 - \bar{k}_3 x_5 - k_4 x_5 + \bar{k}_4 x_6; \\ dx_6/dt = k_4 x_5 - \bar{k}_4 x_6 - k_5 x_6 + \bar{k}_5 x_7 (x_9)^2 - k_6 x_6 + \bar{k}_6 (x_8)^2 (x_9)^2; & dx_7/dt = k_5 x_6 - \bar{k}_5 x_7 (x_9)^2 - k_7 x_7 + \bar{k}_7 (x_8)^2; \\ dx_8/dt = 2k_6 x_6 - 2\bar{k}_6 (x_8)^2 (x_9)^2 + 2k_7 x_7 - 2\bar{k}_7 (x_8)^2; & dx_9/dt = 2k_5 x_6 - 2\bar{k}_5 x_7 (x_9)^2 + 2k_6 x_6 - 2\bar{k}_6 (x_8)^2 (x_9)^2; \end{cases} \quad (1)$$

с начальными условиями: $x_i(t_0) = x_i^0$, (2)

где x_i, i=1,…,9 – концентрации веществ-участников акта мышечного сокращения, $k_i, \bar{k}_i$ (i=1,…,7) – кинетические константы скоростей прямой и обратной стадий, t – время протекания реакции.

При решении обратной задачи нахождения оптимальных параметров системы (1)-(2) наиболее эффективным оказался параллельный вариант генетического алгоритма (рис. 1).

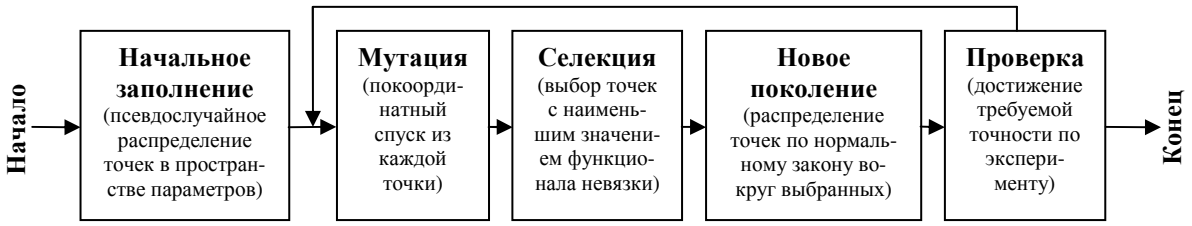


Рис. 1. Генетический алгоритм.

Распараллеливание вычислительного процесса производится на стадии начального заполнения, когда точки в пространстве параметров распределяются по процессорам МВС. Время автономной работы процессоров значительно превышает время межпроцессорных взаимодействий на этапе селекции, что обусловливает эффективность данного алгоритма.

Планируется оценить влияние внешних и внутренних факторов на полученные оптимальные параметры системы с целью коррекции мышечной деятельности человеческого организма. Разработанный программный продукт планируется внедрить в работу медицинских учреждений и спортивно-оздоровительных организаций.

Литература

1. Быстрай Г.П., Охотников С.А. Термодинамика нелинейных биологических процессов. Переход к хаосу. – Екатеринбург: Изд-во Урал ун-та, 2008, – 154 с.

Моделирование течения в осесимметричном тупике CFD-кодом

А.А. Казанцев, В.Р. Анисонян

ЭНИМЦ Моделирующие системы

Расчеты 3D-моделирования с использованием методов вычислительной гидродинамики выполнены для течения в осесимметричном тупике. С использованием распараллеливания MPI проведено тестирование возможностей пакета программ OpenFOAM. Проведено сравнение решения с экспериментальными данными.

1. Выбор вычислительного инструмента и решателя

Для решения задачи моделирования течения в осесимметричном тупике [1] (**Рис. 1**) был выбран сопряженный решатель chtMultiRegionFoam открытого пакета программ OpenFOAM [2].

На рисунке 2 представлена расчетная сетка содержащая 112 тысяч ячеек. Вычисления проводились как на одном ядре, так и параллельно на 4-х ядрах по методике [3]. Полученные результаты показывают ускорение по времени счета при параллельном вычислении в 2,5 раза.

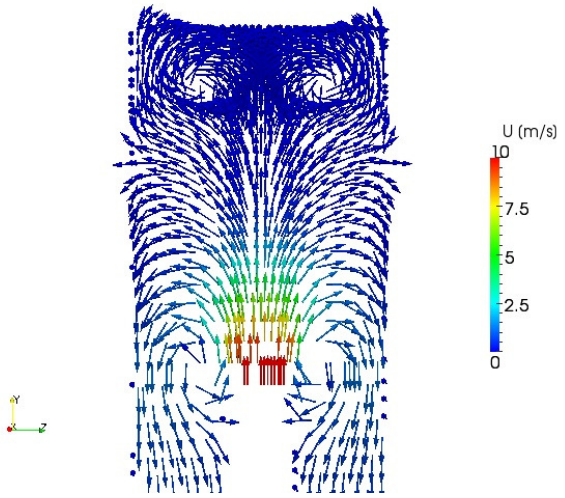


Рис.1. Поле скоростей

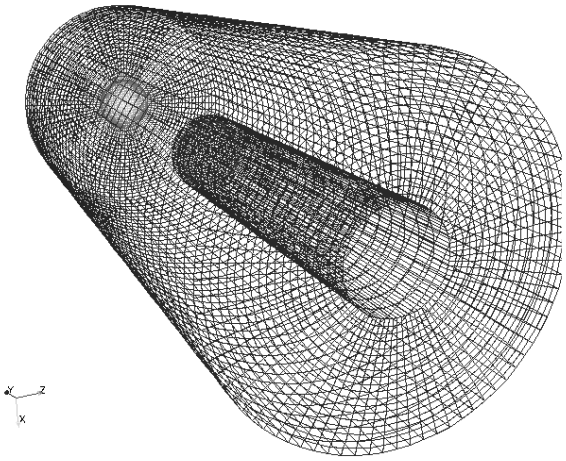


Рис. 2. Расчетная сетка

Выводы

Длительность расчета 1-й секунды, при расчете на 4-х ядрах, занимает около 8 часов, а при расчете на 1-ом ядре 20 часов. Выполнение расчета на 8-ядерном компьютере, но при распараллеливании так же на 4 ядра, дало возможность сделать вывод, что при увеличении количества ядер, ускорение расчета увеличивается не линейно, это связанно с ограничением по пропускной способности общей шины компьютера.

Литература

1. Казанцев А. А. Моделирование 3D течения CFD кодом OpenFOAM / А.А. Казанцев, В.Р. Анисонян // Известия вузов. Ядерная энергетика. – 2010. - № 4. – С.. (в печати)
2. OpenFOAM, Интернет адрес www.opencfd.co.uk
3. Казанцев А.А., Руководство программиста для кода OpenFOAM (перевод с английского), Интернет адрес www.os-cfd.ru

Разработка методов молекулярно-динамического моделирования для применения на гибридных вычислительных системах

А.М. Казённов[1,2], И.В. Морозов[1,2], Р.Г. Быстрый[1,2]

Объединенный институт высоких температур РАН[1], Московский физико-технический институт НИУ[2]

Высокоэнергетичные воздействия на конденсированное вещество, такие как короткие лазерные импульсы, ударные волны и потоки заряженных частиц, приводят к образованию метастабильных состояний, а при достаточной энергии воздействия, к ионизации вещества мишени и формированию области неидеальной плазмы. Динамика релаксационных процессов в таких средах плохо поддается теоретическому описанию из-за существенной роли столкновительных и коллективных эффектов. Широкое применение здесь находит прямое численное моделирование методом молекулярной динамики (МД) [1-3], который предъявляет высокие требования к производительности вычислительных систем.

В настоящее время все большее число пакетов МД моделирования поддерживают вычисления на графических ускорителях (ГПУ). Однако, пока реализованы лишь простейшие потенциалы взаимодействия, такие как потенциал Леннарда-Джонса. Для моделирования конденсированных веществ требуются более сложные модели взаимодействий, в частности, полуэмпирический потенциал погруженного атома (ППА) [4] для металлов. Моделирование неидеальной электрон-ионной плазмы требует кулоновского потенциала на больших расстояниях и достаточно сложной квантово-механической модели на малых расстояниях.

В данной работе рассматриваются особенности реализации потенциала ППА и дальнодействующего кулоновского потенциала на графических ускорителях Nvidia с применением технологии CUDA. Проведено сравнение эффективности распараллеливания потенциалов Леннарда-Джонса, Кулона и ППА на ГПУ. Показаны ограничения на размер моделируемой системы, связанные с объемом внутренней памяти ГПУ. В качестве примера рассмотрена задача моделирования кристаллизации переохлажденного расплава алюминия.

Максимальный прирост производительности при использовании ГПУ Nvidia GeForce 480GTX по сравнению с выполнением аналогичной программы но одном ядре центрального процессора Intel Xeon E5520 составил 180 раз для кулоновского взаимодействия и около 50 раз для короткодействующих потенциалов Леннарда-Джонса и ППА. Эти цифры показывают, что применение ГПУ для задач МД представляется достаточно перспективным. Следует, однако, отметить, что создание программ, эффективно использующих вычислительные возможности ГПУ, это достаточно трудоемкий процесс, требующий учета специфики архитектуры ГПУ.

Некоторые из разработанных в настоящей работе алгоритмов (метод ППА для ГПУ) были включены в свободно распространяемый пакет HOOMD-blue [5]. Для сравнения также использовался широко применяемый пакет LAMMPS [6].

Литература

1. Kuksin, A. Y., Norman, G. E., Stegailov, V. V., Yanilkin, A. V., and Zhilyaev, P. A., International Journal of Fracture 162 (2010) 127.
2. Kuksin, A. Y., Morozov, I. V., Norman, G. E., Stegailov, V. V., and Valuev, I. A., Mol. Simulat. 31 (2005) 1005.
3. Морозов И.В., Норман Г.Э. Столкновения и плазменные волны в неидеальной плазме // ЖЭТФ. 2005. Т. **127**. №2. С. 412–430.
4. Daw, M. S., Foiles, S. M., and Baskes, M. I., Mater. Sci. Rep. 9 (1993) 251.
5. http://codeblue.umich.edu/hoomd-blue
6. http://lammps.sandia.gov

Усовершенствованный метод матрицы переноса для подсчета гамильтоновых цепей на прямоугольных решетках и цилиндрах

А.М. Караваев

Петрозаводский государственный университет

Задача подсчета числа гамильтоновых цепей на прямоугольных сеточных графах имеет ряд важных практических приложений [1, 2] и является актуальной задачей физики полимеров. Гамильтонова цепь является достаточно точной моделью плотноупакованной макромолекулы. Задача состоит в том, чтобы рассчитать количество гамильтоновых цепей для заданной прямоугольной решетки $P_m \times P_n$ и заданного цилиндра $C_m \times P_n$.

Традиционно считается, что проводить расчеты на цилиндре $C_m \times P_n$ сложнее, чем на решетке $P_m \times P_n$. Использование нашей технологии дает возможность проводить вычисления для цилиндров с меньшими затратами ресурсов, чем для решеток, и получать, в связи с этим, более точные результаты. Технология заключается в усовершенствовании стандартного метода матрицы переноса, применительно к поставленной задаче.

Помимо нашей технологии, реализованной в алгоритме, к новым результатам также следует отнести получение точного решения задачи для решеток размером до 17×100 и цилиндров размером до 18×100 (ранее были известны лишь результаты для решеток размером 17×17, а вычисления для цилиндров для $m > 6$ нами проводились впервые), мы получили ряд новых рекуррентных соотношений и уточнили универсальную константу связности для гамильтоновых цепей на решетках и цилиндрах: $\mu = 1.47281 \pm 0.00002$. Порядки полученных соотношений указаны в таблице 1. Все полученные данные доступны для скачивания на нашем сайте [3].

Все вычисления для решеток и цилиндров для $m \geqslant 17$ проводились на кластере Московского государственного института электронной техники в г. Зеленограде с использованием, в основном, 168 вычислительных ядер (21 узел по 8 ядер и 4 Гб ОП).

Таблица 1. Порядки рекуррентных соотношений для решеток и цилиндров с фиксированным параметром m. Символом «*» отмечены новые результаты

m	2	3	4	5	6	7	8	9	10
$P_m \times P_n$	3	6	13	27	70	*183	*431		
$C_m \times P_n$		3	7	12	*24	*50	*93	*252	*452

Литература

1. Cloizeaux J., Jannink G. Polymers in Solution: Their Modelling and Structure. Oxford : Clarendon Press, 1990. 928 P.

2. N. Madras, G. Slade, The Self-Avoiding Walk. Birkhauser, Boston, MA, 1993.

3. FlowProblem [электронный ресурс]. Copyright © 2008–2010. Artem M. Karavaev. URL: `http://www.flowproblem.ru`. Загл. с экрана. — Яз. англ.

Возможности построения идеальных системных сетей многопроцессорных вычислительных систем

М.Ф. Каравай, В.С. Подлазов

Институт проблем управления им. В.А.Трапезникова РАН

В настоящее время создание системных сетей многопроцессорных вычислительных сетей с минимальными задержками доставки пакетов данных является актуальной научной задачей [1]. Одним из методов снижения задержек является создание идеальных системных сетей [2], в которых источники и приемники соединяются прямыми канальными соединениями без промежуточной буферизации пакетов.

Метод создания таких системных сетей был ранее разработан авторами [3-4]. Он заключается в построении распределенного полного коммутатора произвольного размера в фиксированном схемном базисе (с коммутаторами и мультиплексорами/демультиплексорами одного любого размера). Бесконфликтная маршрутизация в нем осуществляется методом червоточины. Достоинством предложенного метода является неблокируемость и самомаршрутизируемость системной сети на произвольной перестановке пакетов данных.

Неблокируемость означает возможность бесконфликтно осуществлять произвольную перестановку пакетов данных между абонентами при параллельной передаче пакетов от всех абонентов, а самомаршрутизируемость – возможность прокладки маршрута при перестановке пакетов каждым абонентом самостоятельно независимо от других абонентов.

В докладе рассмотрен метод построения функционально идеальной системной сети на любое число абонентов в виде многокаскадного распределенного полного коммутатора с резервированными каналами. Он имеет квадратичную схемную сложность, но содержат значительно меньшее (в $\sim N^{1/2}$ раз, где N – число абонентов) число портов и проводников, чем цельный полный коммутатор.

Рассмотренные системные сети ориентированы на две основных области применения – в многоядерных СБИС и в отказоустойчивых МВС реального времени. В первом случае квадратичная схемная сложность не играет особой роли, т.к. сложность распределенного коммутатора много меньше сложности даже одного ядра, но существенно важно значительное сокращение числа проводников. Во втором случае особую роль выполняет полная однородность системной сети, которая позволяет иметь любое число резервных процессоров, и встроенная возможность резервирования каналов.

Для создания подлинно идеальной системной сети необходимо сокращение схемной сложности до сложности сетей Клоза или m-ичных гиперкубов ($m > 2$) при сохранении свойства неблокируемости и самомаршрутизируемости. Используемые в суперкомпьютерах сети Клоза [5] не обеспечивают выполнения одновременно обоих этих свойств. В настоящее время авторы ведут работу в указанном направлении.

Литература

1. Горбунов В.С. Архитектура хорошо масштабируемого вычислительного кластера // Труды международной научно-технической конференции «Суперкомпьютерные технологии: разработка, программирование, применение» (СКТ-2010) Дивноморское. Сент. 2010. т.1. С. 48-54.
2. Kumar A., Peh L-S., Kundu P., Jha N.K. Toward ideal on-chip communication using express virtual channels // IEEE Micro. 2008. Jan/Feb. P. 80-90.
3. Подлазов В.С., Соколов В.В. Метод однородного расширения системных сетей многопроцессорных вычислительных систем // Проблемы управления. 2007. № 2. С. 22–27.
4. Каравай М.Ф., Подлазов В.С. Метод инвариантного расширения системных сетей многопроцессорных вычислительных систем // АиТ. 2010. №. 12. С. 166-176.
5. Scott S., Abts D., Kim J., and Dally W. The black widow high-radix Clos network // Proc. 33rd Intern. Symp. Comp. Arch. (ISCA'2006). 2006. http://cva.stanford.edu/people/jjk12/isca06.pdf.

Анализ последовательных программ с помощью средств УБТ

Н.А. Катаев

Московский Государственный Университет имени М.В. Ломоносова, Институт Прикладной Математики имени М.В. Келдыша РАН

Исследование проводится в рамках Системы Автоматизированного Распараллеливания Фортран Программ (САПФОР) [1], разрабатываемой в Институте Прикладной Математики им. Келдыша РАН. САПФОР относится к диалоговым распараллеливающим системам.

В рамках проводимого исследования предполагается разработать анализатор последовательных программ, написанных на языках Fortran и C/C++. На вход анализатора поступает последовательная программа (далее “программа”), возможно содержащая в тексте специальные аннотации, добавленные пользователем и уточняющие свойства программы. На выходе должна быть построена база данных САПФОР, содержащая структуру программы и информацию о зависимостях по данным в каждом цикле исходной программы, необходимые для построения параллельной версии программы. В качестве основного средства анализа используется Универсальная Библиотека Трансляции (УБТ)[2].

Прежде чем поступить на вход анализатора, внутреннее представление программы должно быть сохранено на xml-файл с помощью модифицированного компилятора GCC [2]. Далее средствами УБТ происходит загрузка данного представления в анализатор, выполняется анализ программы, строится база данных САПФОР. Анализ программ выполняется на основе внутреннего представления GIMPLE компилятора GCC. Единство данного представления для различных языков программирования позволяет создать единую систему анализа программ, написанных на языках Fortran и C/C++. Средства УБТ позволяют обнаруживать циклы, реализованные в исходной программе с помощью оператора GOTO, а низкий уровень операторов GIMPLE позволяет статически более точно оценить время выполнения витков цикла.

В силу нацеленности базы данных САПФОР на высокоуровневые языки программирования (для удобства пользователя), использование низкоуровневого представления влечет за собой определенные сложности. Одному оператору исходной программы в общем случае соответствует группа операторов GIMPLE. Для восстановления исходного вида оператора используется граф потока данных программы, строящийся УБТ. Для корректного распределения данных в языке Fortran DVM (C DVM) необходимо знать размерности линеаризованных в GIMPLE представлений массивов. Особенности представления COMMON-блоков языка Fortran и необходимость анализа наличия тесной вложенности циклов добавляют дополнительные трудности.

Анализатор был испытан на тестах Якоби и SOR, для которых существующие редукционные зависимости были заданы с помощью аннотаций в тексте программы. Была выполнена ручная проверка корректности баз данных. Полученные базы данных были поданы на вход эксперта САПФОР, построившего параллельные версии указанных тестов.

Проводимое исследование показывает возможность использования анализатора на основе УБТ в качестве компонента САПФОР. Использование GIMPLE представления компилятора GCC в качестве основы анализа позволяет расширить применимость САПФОР для языков Fortran 95 и C/C++. Дополнительно, разрабатываемые в процессе исследования средства анализа GIMPLE представления могут быть использованы для восстановления исходного кода программы по GIMPLE представлению.

Выражаю благодарность В.Е. Владиславлеву за помощь в освоении УБТ.

Литература

1. Система Автоматизированной Параллелизации Фортран Программ.
   URL: http://www.keldysh.ru/dvm/SAPFOR/ (дата обращения: 02.01.2011).
2. Optimizing Technologies.
   URL: http://www.optimitech.com/ (дата обращения: 07.01.2011).

Параллельная реализация алгоритма колонии пчел для поиска оптимального мэппинга на кластерную архитектуру

М.С. Колесин

МГУ им. М.В. Ломоносова

1. Алгоритм колонии пчел

Алгоритм колонии пчел является одним из т.н. *эволюционных алгоритмов*, которые в настоящее время активно применяются для решения различных задач оптимизации. Ключевое отличие этих алгоритмов от других заключается в том, что на каждом шаге они используют *популяцию* решений вместо одного решения. По ходу обработки популяции решений в течение одной итерации на выходе также получается популяция решений. В случае одного глобального оптимального значения, популяция должна сходится к нему. Когда же у задачи есть несколько оптимальных решений, эволюционный алгоритм может использоваться для заключения их всех в своей конечной популяции. Данный алгоритм находит применение в различных областях, [2].

2. Задача нахождения оптимального мэппинга

Как известно, время работы параллельной складывается из времени, затраченного на вычисления, времени, затраченного на обмен информацией между процессами, а так же времени на синхронизацию процессов.

В рамках данной работы рассматривается влияние мэппинга (способа назначения процессов на физические процессоры) только на время обменов.

Задачу нахождения оптимального мэппинга можно сформулировать как минимизацию значения функции $E[f] = \sum_{i,j=1}^{N} C(p_i, p_j) S(i, j)$, где перестановка $(p_1 \quad p_2 \quad \dots \quad p_N)$ - это мэппинг, а c_{ij} - это объем информации, которым обмениваются MPI-процессы с номерами i и j во время выполнения задачи, s_{ij} - «стоимость» обмена информацией между процессорами с номерами i и j. Матрица C определяется только MPI-приложением, а матрица S – только характеристиками кластера, такими, как топология, используемая среда передачи данных между узлами и т.п. Таким образом, имеем пространство поиска размером $N!$, где N – число процессов MPI-приложения. Применимость алгоритма колонии пчел для решения этой задачи была исследована в работе [1].

В алгоритме колонии пчел заложен очевидный параллелизм – различные решения из популяции можно обрабатывать параллельно, обмениваться информацией и переходить к новой итерации работы алгоритма. Такая схема распараллеливания с поправкой на особенности архитектуры была использована в двух реализациях – при помощи технологий MPI на системе Bluegene/P и NVIDIA CUDA на GPU NVIDIA Tesla C1060.

Литература

1. Колесин М. С. Исследование возможностей применения алгоритма колонии пчел для решения задачи нахождения оптимального мэппинга параллельной задачи на архитектуру BlueGene/P. // Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009.

2. Teodorovic D., Davidovic T., Selmic M. Bee Colony Optimization: The Applications Survey. // http://www.mi.sanu.ac.rs/~tanjad/BCO-ACM-Trans-Ver2.pdf

Организация виртуальных персональных компьютеров студентов на базе суперкомпьютера

П.С. Костенецкий, А.И. Семенов

Южно-Уральский государственный университет

Южно-Уральский государственный университет регулярно обновляет свои вычислительные мощности, производя замену суперкомпьютеровприблизительно раз в три года. Так как суперкомпьютеры предыдущего поколения, освобождающиеся при замене, сохраняют свою работоспособность и за счет надежности кластерной архитектуры обеспечивают достаточно высокую отказоустойчивость, стала очевидной необходимость утилизации вычислительной мощности таких суперкомпьютеров.Так в 2010 г. ЮУрГУ приобрел суперкомпьютер «СКИФ-Аврора ЮУрГУ» [1], превосходящий по производительности вычислительный кластер «СКИФ Урал» [2] более чем в 6 раз. На освободившемся суперкомпьютере был реализован пилотный проект «Персональный виртуальный компьютер» (ПВК).

Система ПВК используется на новом факультете ЮУрГУ «Вычислительная математика и информатика».В рамках системы для каждого студента создается персональный виртуальный компьютер на базе ОС Windows 7, доступ к которому осуществляется с домашнего компьютера, ноутбука,нетбукаи т.п. (см. рис. 1). В результате, в качестве компьютерного класса может быть использована любая учебная аудитория ЮУрГУ с рабочими местами, оснащенными электрическими розетками.

Рис. 1. Принцип работы системы «Персональный виртуальный компьютер».

Система основана на виртуальной серверной инфраструктуре,построенной потехнологии MicrosoftHyper-V.В решении использована выделенная система хранения данных, подключенная к узлам кластера по технологии iSCSIoverInfiniBand. На виртуальной серверной инфраструктуре создана инфраструктура виртуальных рабочих столов по технологииCitrixXenDesktopVDIEdition [3].

Использование ПВК снижает затраты на создание компьютерных классов и обновление парка ПК университета, обеспечивает студентов доступом к ПВК с лицензионным программным обеспечением, а также повышает безопасность данных за счет централизованного хранения всех компьютеров на едином вычислительном комплексе университета.

Литература

1. Вычислительный кластер «СКИФ-Аврора ЮУрГУ». URL: http://supercomputer.susu.ru/computers/skif_avrora/ (дата обращения: 30.12.2010).
2. Вычислительный кластер «СКИФ Урал». URL: http://supercomputer.susu.ru/computers/skif_ural (дата обращения: 30.12.2010).
3. *Sheppard B., Rutherfoord A. et al.* Application Streaming Delivery and Profiling Best Practices. Citrix Systems, 2009. 8 p.

Библиотека для динамической адаптации программ к гетерогенным архитектурам вида CPU + GPU

М.А. Кривов[1], С.А. Гризан[2]

Московский Государственный Университет им. М.В. Ломоносова[1],
Сибирский Федеральный Университет[2]

За последние два года архитектура высокопроизводительных вычислительных систем претерпела ряд существенный изменений. Появившаяся в 2007 году технология NVidia CUDA позволила задействовать колоссальные вычислительные мощности существующих графических ускорителей, сделав возможным создание «домашних» суперкомпьтеров. Выпуск специализированных видеоплат серии NVidia Tesla позволил оснащать узлы классических кластеров данными ускорителями, в результате чего широкое распространение также стали получать так называемые гетерогенные суперкомпьютеры.

При реализации алгоритмов для систем с гетерогенными архитектурами с использованием современных технологий перед программистом встаёт ряд проблем, связанных с неоднородностью вычислителей. Одной из основных проблем является необходимость выбора оптимального вычислителя для каждой части задачи. Другими словами, без знания специфики архитектуры не всегда понятно, какую часть задачи на каком процессоре лучше считать. Согласно идеологии NVidia, на графические ускорители нужно переносить всё, что удаётся распараллелить под модель SIMT, оставляя центральному процессору только последовательные части программы. Однако, как показано в работах [1-2], при решении ряда задач, оптимально подходящих для видеокарт, центральный процессор класса Intel Xeon может показать схожую производительность. Соответственно, если же алгоритм «плохо» отображается на архитектуру видеокарт, то их использование может лишь замедлить программу.

В данной работа предложен подход, основная идея которого заключается в использовании динамических параметров, непосредственное значение которых будет определяться во время работы программы и, более того, будет изменяться в зависимости от типа вычислений и возможностей используемой аппаратной платформы. Для программиста это означает, что больше не требуется знание специфичных особенностей архитектуры. Если требуется определить значение какого-либо параметра, то вместо него можно указать специальную конструкцию-«заглушку» и, при необходимости, определить начальное значение.

Для подбора оптимальных значений определённых подобным образом параметров возможно два режима работы программы. В первом из них перед выполнением основных вычислений производится «обучение» на небольших входных данных. Второй режим ориентирован на итеративные программы, в которых одно и тоже вычислительное ядро выполняется много раз. Таким образом, каждую итерацию или запуск в режиме обучения можно рассматривать как один шаг оптимизационного метода, используемого для минимизации функции времени работы.

Данный подход был протестирован на задаче перемножения матриц размерности 1024 на 1024. При работе на системе с 4-ядерным центральным процессором (Intel Core 2 Quad Q6600) и двумя графическими ускорителями разной архитектуры (NVidia Tesla C1060 и NVidia Tesla C2050) за 7 итераций время работы приложения было уменьшено более чем в 8 раз.

Список литературы

1. Bordwekar R., Bondhugula U., Rao R., Believe it or Not! Multy-core CPUs Can Match GPU Performance for FLOP-intensive Application! IBM Research Report RC24982 (W1004-095), April 23, 2010.

2. Bordwekar R., Bondhugula U., Rao R., Can CPUs Match GPUs on Performance with Productivity?: Experiences with Optimizing a FLOP-intensive Application on CPUs and GPU, IBM Research Report RC25033 (W1008-020), August 5, 2010.

Принципы построения грид инфраструктуры для Национальной нанотехнологической сети*

А.П. Крюков, А.П. Демичев, В.А. Ильин, Л.В. Шамардин
НИИЯФ МГУ, 119991, Москва, Ленинские горы, д.1

Использование архитектурного подхода REST. В настоящее время общепринятым является Открытая архитектура грид-сервисов, которая реализована в виде набора спецификаций и стандартов, получивший название Инфраструктура ресурсов веб-сервисов (Web Service Resource Framework, WSRF). Однако, WSRF - это исключительно сложный набор протоколов и стандартов. Использование простого и ясного архитектурного стиля REST дало замечательный результат для построения Веб-сервисов. Авторами было предложено использовать стиль REST для реализация грид-сервисов в рамках концепции OGSA. Для этого был предложен ряд изменений, которые необходимы для расширения Веб-сервисов до грид-сервисов.

Управление циклом существования ресурсов. В гриде, построенном согласно принципам OGSA, постоянно происходит создание и уничтожение ресурсов, и, как следствие, связанных с ними логических сервисов. Поэтому необходим единый механизм для информирования о времени жизни сервисов/ресурсов, а так же предсказуемого изменения этого времени жизни. Было предложено использовать нестандартный заголовок протокола HTTP Termination-Time.

Свойство идемпотентности REST запросов. Действия, выполняемые методами GET и HEAD не должны выполнять никаких функций помимо собственно возвращения представления ресурсов. Действия, выполняемые методами PUT и DELETE должны быть идемпотентными, то есть побочные эффекты, вызываемые запросом с таким методом должны быть одинаковы для любого количества повторений одного и того же запроса.

Обработка ошибок. RESTful-грид-сервис использует для индикации ошибок стандартные коды статусов HTTP с документами, поясняющими детали произошедшей ошибки. Они должны использовать коды состояния HTTP со значениями 4xx и 5xx.

Обработка потока заданий (workflow). Задания в ГридННС являются композитными объектами, которые состоят из совокупности задач, порядок выполнения которых описывается с помощью направленного ациклического графа. Задания и задачи представляются в виде JSON-объектов.

Модель управления данными. ГридННС является вычислительным гридом, ориентированным на суперкомпьютерные вычисления. Для управления данными используются GridFTP серверы

Модель аутентификации ГридННС. Модель построена на основе PKI с использованием цифровых сертификатов стандарта X.509. В рамках инфраструктуры был развернут центр выдачи сертификатов.

Виртуальные организации. Работа пользователей в ГридННС происходит в рамках виртуальных организаций. Все пользователи ГридННС должны быть зарегистрированы хотя бы в одной ВО. Такая организация работ решает две основные проблемы: авторизация доступа пользователей к ресурсам и разделение их прав доступа.

Экспериментальное исследование производительности сервиса Pilot. Для проверки указанных принципов в ГридННС был реализован грид-сервис распределения заданий - Pilot. Для оценки его производительности были проведены замеры времени запуска заданий. Для сравнения использовался сервис WMS ППО gLite. Время обработки задачи для сервиса Pilot составляет 7.5 секунд, в то время как для gLite WMS — 99 секунд,. Проведенные эксперименты показали эффективность сервиса, его устойчивую и надежную работу.

1. А.Демичев, В.Ильин, А.Крюков и Л.Шамардин: Реализация программного интерфейса грид-сервиса Pilot на основе архитектурного стиля REST.// Вычисленные методы и программирование, с.62-65, т.11, 2010.

* Работа была частично выполнена при финансовой поддержке РФФИ (грант № 10-07-00332), и Министерства образования и науки РФ (контракты № 01.647.11.2004, №. 02.740.11.0388 и НШ № 4142.2010.2).

Использование высокопроизводительного вычислительного кластера для решения трехмерной многоконтактной задачи механики деформируемого твердого тела

Н.Н. Куриков

Актуальной задачей проектирования нового двигателя и модернизации имеющейся конструкции является повышение его удельной мощности за счет снижения массы. Такая цель может быть достигнута, в частности, снижением массы компонент шатунно-поршневой группы. Отдельно в качестве объекта исследования можно выделить конструкцию прицепного шатуна. При этом конечной целью ставится нахождение его оптимальной формы, обеспечивающей минимальную массу объекта при сохранении его прочностных и жесткостных характеристик.

Задание правильного внешнего воздействия на шатун требует принятия во внимание податливостей деталей шатунно-поршневой группы, находящихся в непосредственной близости с ним – необходимо решение нелинейной контактной задачи взаимодействия нескольких тел.

В работе представлено решение нелинейной контактной задачи с помощью комплекса Abaqus на вычислительном кластере.

Конечно-элементная сетка состоит из 226171 элементов (четырех- и десятиузловые тетраэдры), построенных на 287041 узле. Общее количество контактных элементов в модели – 25014. Тип контакта: жесткий контакт поверхность-поверхность, наиболее подходящий для общих случаев взаимодействия. Учтены все инерционные нагрузки.

КЭ расчеты осуществлялись с помощью решателя Abaqus/Standard версии 6.9.2 (при использовании прямого решателя систем уравнений) в статической постановке, без учета нелинейных геометрических эффектов. Была проведена серия расчетов с использованием различного количества ядер (2, 6, 12, 24) для одной и той же КЭ сетки при одном и том же положении механизма. Минимальное количество использованных ядер, равное 2, обусловлено размерностью задачи и конфигурацией вычислительного кластера (11 узлов, оснащенных двумя четырехъядерными процессорами Intel Quad Core и восемью гигабайтами оперативной памяти каждый). Задача требовала порядка 10-11 гигабайт оперативной памяти, в связи с чем решение ее было возможным не менее, чем на двух узлах. Количество уравнений в задаче составляло порядка 1370000, на каждой итерации осуществлялось (1…2)e12 операций с плавающей запятой.

В серии расчетов оценивалось общее время решения задачи, среднее время счета одной итерации, ускорение. Приводится сравнение этих параметров для различного числа использованных ядер.

Литература

1. Abaqus version 6.9 Documentation
2. Биргер И. А. и др. Авиационные поршневые двигатели. Под ред. И. Ш. Неймана. М., Государственное издательство обороной промышленности, 1950 г.
3. Василевский Б. И. Исследование напряженно-деформированного состояния шатунов двигателей внутреннего сгорания. Автореферат на соискание ученой степени кандидата технических наук. Ленинград, 1978 г.
4. Ливенцев Ф. Л. Двигатели со сложными кинематическими схемами. Л., Машиностроение, 1973 г.
5. Научно-технический журнал “Мир нефтепродуктов. Вестник нефтяных компаний”. М., Издательский центр “Техинформ МАИ”. Номер 2, 2009 г.

Параллельный алгоритм фрактального поиска в базе данных*

Т.Ю. Лымарь, Н.Ю. Староверова

Южно-Уральский государственный университет

Основной принцип применения теории фракталов – это поиск простых частей, подобных целому, применив к которым определенную итерационную функцию, можно получить всю систему [1]. Выделение так называемых самоподобных частей (доменов) называют фрактальным анализом.

Обычным применением теории фракталов в информационных технологиях является обработка графической информации, что вполне естественно и понятно, однако можно рассмотреть с точки зрения данной теории и системы баз данных. Совокупность большого количества записей таблиц, может стать источником дополнительной, гораздо более ценной информации, которую нельзя получить на основе одной конкретной записи. Вполне возможно, что анализируемая информация может быть самоподобна и может отображать определенную зависимость не только между записями таблиц, но и внутри самой записи [2].

Целью данного исследования является рассмотрение реляционных таблиц с точки зрения теории фракталов, определение подходящих методов фрактального анализа и разработка алгоритма поиска доменов в реляционных таблицах. Возможным применением полученных результатов является поиск функциональных зависимостей в таблицах и сжатие хранимых данных.

В ходе начатого исследования разработан последовательный алгоритм поиска доменов, работа которого состоит из следующих этапов:

1. Формируется множество изначально возможных структур доменов, выбирается минимальный набор, применив к которому некоторую функцию, можно восстановить всю таблицу;
2. По списку возможных структур доменов вычисляется реальное количество доменов из таблицы для каждого сочетания;
3. Выбираются те домены, минимальное число которых позволит «собрать» всю таблицу.

Алгоритм реализован для СУБД Oracle в качестве тестовой системы. Основными источниками данных для формирования минимального множества возможных доменов являются достаточно обширный в данной СУБД словарь данных и инструмент сбора статистики. В качестве основного критерия для выделения домена использовалось количество различных значений в атрибутах, входящих в домен.

Распараллеливание реализованного алгоритма проводится на основе использования фрактальной кластеризации [3] при выявлении доменов:

1. На выделенном узле формируется начальное множество D возможных доменов;
2. Элементы множества рассылаются остальным узлам;
3. Каждый узел-получатель пытается соответственно кластеризовать свой фрагмент анализируемой таблицы, в случае неприменимости полученного домена к имеющимся записям, создается новый элемент множества D, для которого процедура рекурсивно повторяется.

Литература

1. Мандельброт Б. Фрактальная геометрия природы. М.: Институт компьютерных исследований, 2002, 656 с.
2. Traina C. Jr., Traina A., Wu L., Faloutsos C. Fast feature selection using fractal dimension. URL: http://seer.lcc.ufmg.br/index.php/jidm/article/viewFile/4/1 (дата обращения: 27.01.2010).
3. Daniel B., Ping Ch. Using the Fractal Dimension to Cluster Data sets. URL: http://cms.dt.uh.edu/Faculty/ChenP/paper/KDD00.pdf (дата обращения: 27.01.2010).

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 09-07-00241-а).

Моделирование элементов энергонезависимой памяти типа RRAM на высокопроизводительных вычислительных системах методом Монте-Карло *

А. Макаров[1,2], В. Свердлов[1], Д. Крыжановский[2], М. Гиркин[2], S. Selberherr[1]

Institute for Microelectronics, TU Wien, Вена, Австрия[1],
Волгоградский Государственный Технический Университет, Волгоград, Россия[2]

Концепции памяти, основанные на хранении заряда (такие как Flash память, DRAM и другие), близки к физическому пределу масштабирования. В последние несколько лет рост интереса к минимизации устройств (например MP3 плееров и мобильных телефонов) способствует ускорению разработки новых типов памяти. Кроме хорошей масштабируемости новые типы памяти должны характеризоваться низким рабочем напряжением, низким энергопотреблением, высокой скоростью работы, высокой надежностью и простой структурой [1]. Наиболее перспективными в этой связи являются типы памяти, основанные на явлении резистивного переключения (CBRAM, PCRAM, RRAM). Данные типы памяти обладаю наиболее простой структурой (метал - изолятор - метал) и, как следствие, хорошей масштабируемостью. Кроме того, RRAM также характеризуется быстрым временем переключения ($< 10ns$), низким рабочим напряжением ($< 2V$) и большой плотностью элементов. В качестве изоляторного слоя в памяти типа RRAM используются оксиды металлов, такие как TiO_x, HfO_2, Cu_xO, NiO, ZnO, и перовскиты, такие как $SrTiO_3$, $SrZrO_3$, $Pr_{1-x}Ca_xMnO$. В недавних исследованиях были предложены несколько моделей резистивного переключения RRAM, однако фундаментального понимания механизма переключения до сих пор недостигнуто [2].

В данной работе мы представляем результаты стохастического моделирования памяти RRAM. Мы ассоциируем резистивное переключение в данном типе памяти с формированием и разрывом проводящих нитей. Проводящая нить формируется из локализованных вакансий кислорода (V_o) или доменов V_o. Формирование и разрыв проводящей нити происходит вследствие окислительно-востановительной реакции в оксидном слое под приложенным напряжением. Электрическая проводимость обусловлена прыжками электронов между V_o.

Для моделирования резестивного переключения методом Монте-Карло, в соответствии с изложенной выше концепцией, мы описываем динамику ионов кислорода и электронов в оксидном слое как: 1) прыжок электрона на вакансию с электрода; 2) прыжок электрона с вакансии на электрод; 3) прыжок электрона между вакансиями; 4) формировании вакансии кислорода за счет выхода иона кислорода в междоузлие; 5) аннигиляция вакансии кислорода ионом кислорода.

Представленные результаты моделирования воспроизводят результаты экспериментальных исследований.

Литература

1. Kryder, M.H., Kim C.S. After Hard Drives - What Comes Next? // IEEE Trans. Magn., 45 (2009); 3406-3413.

2. Makarov A., Sverdlov V., and Selberherr S. Stochastic Model of the Resistive Switching Mechanism in Bipolar Resistive Random Access Memory: Monte Carlo Simulations // Journal of Vacuum Science & Technology B, 29 (2011), 1; 01AD03-1 - 01AD03-5.

*Работа выполнена при поддержке European Research Council грант No. 247056 MOSILSPIN.

Технология моделирования систем по частям

М.Н. Максимов

ООО "Информационные и инновационные технологии"[1]

В статье кратко описана технология моделирования сложных систем по частям и новый численный метод, на базе которого она построена. Сложные системы обычно имеют иерархическую структуру, которую можно представить в виде набора связанных между собой функциональных блоков. Разработанная технология позволяет моделировать функциональный блоки системы параллельно друг другу и сшивать их решения после каждой итерации в не зависимости от их количества и способа соединения между собой.

На сегодняшний день для увеличения быстродействия САПР используют многопроцессорные ЭВМ (суперкомпьютеры). К сожалению, их применение ограничивается конечностью возможностей внутреннего параллелизма известных алгоритмов. Поэтому актуальным является разработка новых математических методов и алгоритмов, позволяющих более полно распараллеливать решения задач САПР и тем самым добиться увеличения их быстродействия на многопроцессорных системах за счёт более эффективного использования вычислительных ресурсов.

Одним из основных результатов разработанной технологии является прямой численный метод решения разреженных СЛАУ по функциональным блокам. Основные черты метода опишем на примере моделирования некоторой сложной системы.

Итак, пусть имеется система, которую можно представить в виде набора функциональных блоков, соединённых друг с другом в N узлах (причём N<<R, где R общее количество узлов в системе). Поведение каждого из функциональных блоков описывается системой дифференциальных уравнений. При моделировании от систем дифференциальных уравнений методами численного интегрирования переходят к дискретным моделям (к СЛАУ) блоков. Далее формируют «большую системную матрицу» (порядка R) путём включения в неё СЛАУ блоков. Причём если блоки функционально связаны, то они в матрице частично перекрываются (складываются). Полученную «большую системную матрицу» решают прямыми или чаще итерационными методами. Это стандартный путь.

При моделировании системы по частям решение получаю следующим образом:

1. Вместо формирования «большой системной матрицы» формируется так называемая «матрица сшивания», ранг которой не превышает N.
2. СЛАУ функциональных блоков решаются (любыми известными методами) параллельно друг другу относительно заданных значений векторов b в правой части;
3. С помощью решений, полученных на предыдущем шаге, формируем правую часть матрицы сшивания и решаем её (количество переданных данных между процессорами на 2-3 и 3-4 шагах минимально).
4. Далее опять параллельно решаем СЛАУ функциональных блоков, но с учётом решения, полученного на предыдущем шаге. В результате получаем решение, которое совпадает (с учётом ошибки округления) с решением «большой» СЛАУ для заданного суммарного вектора b*. Если необходимо модифицируем вектора b блоков и ищем новые решения, перейдя к шагу 2.

Таким образом, скорость решения СЛАУ по функциональным блокам на многопроцессорной системе зависит от максимального размера СЛАУ блока, размера матрицы сшивания и скорости обмена данными между процессорами, и практически не зависит от количества блоков и их способа соединения между собой, т.е. слабо зависит от размерности решаемой задачи.

Разработанный численный метод лежит на стыке прямых и итерационных методов решения разреженных СЛАУ. Доказано, что если коэффициенты матрицы сшивания вычислены точно, то метод является прямым, но, так как из-за ошибки округления, точно их вычислить проблематично, то метод можно отнести и к блочным итерационным методам, все собственные числа матрицы перехода которого имеют порядок приблизительно равный квадратному корню из порядка ошибки округления, т.е. существенно меньше единицы.

[1] Создано при поддержке Южного Федерального Университета

Методы обеспечения отказоустойчивости для ряда задач в распределенном окружении

А.А. Московский[1], Е.О. Тютляева[2]

ЗАО «РСК СКИФ»[1], ИПС РАН им. А.К. Айламазяна[2]

Увеличение масштабов современных вычислительных систем приводит к увеличению вероятности отказа отдельных элементов системы. В ряде случаев вычислительные алгоритмы, использующие методы типа Монте-Карло, генетические алгоритмы и т.п., допускают потерю части промежуточных результатов вычислений, что может произойти, например, из-за отказа одного узла в большом вычислительном кластере. В статье предлагаются методы для реализации таких алгоритмов и обеспечения их работоспособности при условии программных и аппаратных сбоев на вычислительных узлах.

В рамках данной работы создан отдельный тип для организации отказоустойчивых вычислений — неготовая переменная (англ. future) с заданным временем жизни. Этот тип обладает всеми свойствами неготовой переменной, поддерживает корректную работу в распределенной памяти, но в шаблонных параметрах данного типа можно указать время жизни такой переменной. Добавление концепции времени жизни означает, что после истечения указанного времени результат переменной ожидаться не будет и программа будет корректно завершена, даже если результат вычисления каких-то функций, ожидаемых в данной переменной, не получен. Программист имеет возможность обрабатывать полученный результат согласно логике задачи, т.е. он может оставить данный результат невостребованным, запустить аналогичную или перезапустить эту же задачу с ожиданием результата в другой неготовой переменной со временем жизни, при этом увеличив время жизни и т.п. Данный подход позволяет получить результат за детерминированное время для тех задач, которые позволяют получить корректный результат даже если одна или несколько вычислительных подзадач не завершили свою работу. Достоинством такого подхода является гарантированная завершаемость при отказах, простота применения и минимальные накладные расходы. Недостатком такого подхода является сложность применения в случае неравновесных гранул параллелизма, к примеру при рекурсивном вычислении.

В качестве альтернативы был реализован метод сохранения задач, при котором все исполняющиеся задачи сохраняются в локальных map-структурах участвующих в вычислении узлов. В случае перехвата отказа одного из вычислительных узлов, все оставшиеся активными узлы удаляют адрес данного узла из списка доступных для проведения вычисления и выполняют повторную посылку тех задач, которые были на него отправлены. Недостатком такого подхода является наличие высоких накладных расходов на сохранение задач и требуемой оперативной памяти на поддержание списка задач. Достоинством является движение в сторону локальной синхронизации, т.к. каждый узел помнит те задачи, для которых было проведено локальное назначение.

Для испытания разработанных примитивов была модифицирована библиотека шаблонных классов C++ T-Sim, в которую была добавлена корректная обработка сбоев и модифицирован планировщик для работы в нестабильном окружении. С использованием разработанных примитивов на языке T-Sim был реализован пример редукционного (монотонного) объекта, а также нескольких альтернативных механизмов перезапуска заданий.

Чтобы гарантированно выявлять сбои, а не ждать завершения работы задачи, направленной на аварийный узел, в тех случаях, когда больше никакие задачи на этот узел не направляются и таким образом невозможно выявить сбой в работе узла во время естественного хода вычисления, планируется также реализовать дополнительную стратегию планирования, которая будет через заданные промежутки времени опрашивать состояние всех участвующих в вычислении узлов.

Эффективность и масштабируемость параллельных алгоритмов расчета электростатического поля для численных методов типа Хокни*

А. В. Позднеев

Факультет ВМК МГУ имени М. В. Ломоносова

Данная работа продолжает цикл исследований автора, посвященных построению эффективных методов для математического моделирования в масс-спектрометрии. Поведение заряженных частиц в ионных ловушках моделируется методом частиц в ячейке. При этом на каждом шаге интегрирования уравнений движения требуется выполнять решение уравнения Пуассона. Следует подчеркнуть, что на движение ионов оказывают влияние как сила со стороны статического поля ловушки, так и поле зарядов, индуцированных на ее стенках, и кулоновские силы межионных взаимодействий. В современных масс-спектрометрах предъявляются жесткие требования к точности расчета электрических полей. Все это обусловливает крайнюю важность разработки параллельных алгоритмов расчета электрических полей на сверхгустых сетках.

В предыдущей работе автором было выполнено исследование алгоритма для метода типа Хокни на системе Blue Gene/P. Алгоритм состоял из следующих шагов: (1 и 5) серия двумерных быстрых преобразований Фурье (БПФ), (2 и 4) межпроцессные коммуникации, (3) решение систем с трехдиагональными матрицами (СТМ). Была показана его масштабируемость до тысяч процессоров и десятков миллиардов точек сетки. Однако до половины времени расчета занимали межпроцессные коммуникации типа «каждый–каждому», которые осуществлялись функцией `MPI_Alltoallv()` над коммуникатором `MPI_COMM_WORLD` [1].

Целью данной работы является модификация алгоритма, представленного в работе [2]. Основная идея состоит в следующем. Двумерные БПФ выполняются за три шага: сначала осуществляются одномерные БПФ по одному измерению, затем производятся межпроцессные обмены, потом — одномерные БПФ по другому измерению. Важная особенность такой схемы заключается в том, что межпроцессные обмены проходят в рамках непересекающихся подмножеств процессоров. Принципиальным преимуществом представляемого параллельного алгоритма является возможность решать задачу для граничных условий первого рода, а не только для периодических. Во-вторых, по одному из направлений осуществляется решение СТМ, что позволило впервые разработать эффективный алгоритм расчета поля в коаксиальной [3] трехмерной ионной ловушке типа ‘O-trap’. Масштабируемость алгоритмов исследуется на суперкомпьютерах, установленных в МГУ имени М. В. Ломоносова.

Литература

1. Позднеев А.В. Исследование масштабируемости прямого метода решения уравнения Пуассона на вычислительной системе Blue Gene/P // Сб. тр. междун. суперкомп. конф. «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи». — Новороссийск: Изд-во МГУ, 20–25 сентября 2010. — С. 123-132.

2. Chatelain P. et al. Billion vortex particle direct numerical simulations of aircraft wakes // Comput. Methods Appl. Mech. Engrg. — 2008. — Vol. 197. — Pp. 1296–1304.

3. Christopher I., Knorr G., Shoucri M., Bertrand P. Solution of the Poisson Equation in an Annulus // J. Comput. Phys. — 1997. — Vol. 131, no. 2. — Pp. 323–326.

*Работа выполняется в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» (государственные контракты П1317 от 09.06.2010, П958 от 20.08.2009 и 02.740.11.0196 от 07.07.2009).

Моделирование процесса окислительной регенерации закоксованных катализаторов и оптимизация процесса на основе параллельных вычислений

Л.В. Сайфуллина, М.Р. Еникеев

Башкирский государственный университет

Описан процесс окислительной регенерации в виде систем алгебраических и дифференциальных уравнений, выражающих основные законы сохранения. Предложен алгоритм численного решения систем уравнений математического описания. Приведена программная реализация вычислительного алгоритма для ЭВМ. При помощи разработанных математических моделей исследованы возможные перегревы в слое катализатора от значений основных определяющих параметров процесса.

Изучению кинетики регенерации промышленных катализаторов от углеродистых отложений окислением последним кислородом воздуха посвящено большое число расчетных и экспериментальных работ, однако авторы ряда работ делают неоправданные допущения. Математическую модель процесса получим на основе уравнений балансов: уравнения материальных балансов по кислороду и по коксу и уравнение теплового баланса для элементарного слоя за время от τ до $\tau+d\tau$ [1]. При выборе кинетической модели учитывалось, что удаляемый кокс имеет сложную структуру и представляет собой в основном продукты реакции уплотнения, содержащие связанный водород. Схема химических превращений, описывающая окисление коксовых отложений, и соответствующие ей кинетические уравнения скоростей стадий имеет вид, рассматриваемый в работе [2].

Таким образом, математическая модель процесса выжигания кокса из слоя катализатора в различных кинетических режимах представляет собой систему дифференциальных уравнений с учетом соотношений для расчета скоростей стадий реакций.

Построенная математическая модель может быть использована для исследования и расчета ряда регенерационных устройств – изотермического и адиабатического реактора. Для изотермического реактора $\partial T / \partial l = 0$, а если рассматривать режим с постоянной в ходе всей регенерации температурой, то $\partial T / \partial \tau = 0$ и описание процесса регенерации в изотермическом реакторе заметно упрощается.

Оптимизация осуществлялась с помощью генетического алгоритма. Проблема выбора критерия оптимизации связана, прежде всего, с задачей охраны окружающей среды, так как выбрасываемый в атмосферу газ должен содержать такое количество монооксида углерода, которое не превышает санитарно допустимую норму. Поэтому в качестве критерия оптимизации рассматривается условие минимизации среднего за время работы t_k содержания CO, образующегося в ходе регенерации:

$$max\left\{ J = -\frac{1}{t_k}\int_0^{t_k} x_4(\tau_k, t)dt \right\}$$

Литература

1. Жоров Ю.М. Моделирование физико-химических процессов нефтепереработки и нефтехимии. – М.: Химия, 1978. – 376 с., ил. – (Процессы и аппараты химической и нефтехимической технологии).

2. Балаев А.В., Дробышевич В.И., Губайдуллин И.М., Масагутов Р.М. Исследование волновых процессов в регенераторах с неподвижным слоем катализатора // Распространение тепловых волн в гетерогенных средах. Н-ск: Наука, Сиб. отд-ние, 1988.- С. 233-246.

Подход к решению междисциплинарных задач на основе системы многотельной динамики

Е.С. Сергеев[1], В.В. Гетманский[1], О.В. Шаповалов[1], А.С. Горобцов[1]

Волгоградский государственный технический университет[1]

Существует два подхода для численного решения смешанных физических моделей: непосредственное связывание (direct-coupled, fully-coupled) и последовательное связывание (sequential-coupled)[1] физических явлений. В данной работе рассмотрены модели программ, решающих задачи междисциплинарного моделирования, основанных на последовательном связывании. В разрабатываемом программном комплексе ФРУНД[2] используется подход, сочетающий расчет динамики систем тел с различными дополнительными видами расчета для каждого тела. К настоящему времени имеются 2 расчетных модуля: динамики, теплопроводности, каждый из которых обладает различной вычислительной сложностью. Модуль расчета динамики механической системы выполняется как основной и передает на каждой итерации определенные рассчитанные параметры во вспомогательные модули.

В случае увеличения размерности задачи до нескольких миллионов тел время расчета сильно возрастает, и для его уменьшения необходимо проводить декомпозицию сетки на части с учетом оптимальности распределения вычислительной нагрузки на доступные вычислительные ресурсы. В результате декомпозиции формируются списки тел модели, входящие в каждую подмодель и списки граничных тел, соединенных с рассеченными шарнирами. Тестирование метода распределения нагрузки проводилось на примере решения задачи теплопроводности для детали, представленной объемной сеткой из 11 млн. узлов. Прямой последовательный расчет на одном ядре CPU занял 130 секунд, при равномерном распределении по 4 ядрам CPU время сократилось почти в 4 раза. При использовании для расчета GPU вместо некоторых ядер CPU с балансировкой сетки в соответствии с производительностью вычислителей было получено ускорение расчета, результаты тестов приведены в таблице 1.

	1 CPU	4 CPU	3 CPU + 1 GPU 9800GTX+	3 CPU + 1 GPU Tesla c1060	2 CPU + 2 GPU Tesla c1060
Время расчета, с	130	40,4	35,1	19,9	10,2

Таблица 1. Ускорение на одном гибридном узле кластера за счет балансировки нагрузки

Для экспериментов использовались два разных гибридных узла вычислительного кластера: Core Quad Q6600 + NVIDIA 9800GTX+ и Core i7 920 + 2 x Tesla c1060. По сравнению с параллельным расчетом использование балансировки позволило получить ускорение в 1.15 раз на NVIDIA 9800GTX+ и в 2 раза NVIDIA Tesla. При использовании одновременно двух GPGPU Tesla вместо двух ядер было получено ускорение в 4 раза. В перспективе планируется получить результаты на гибридном кластере.

Литература

1. Paul Lethbridge, Multiphysics Analysis, The Industrial Physicist, Dec 2004/Jan 2005. - P. 21-23.
2. Горобцов А.С. Перенос системы многотельной динамики на вычислительный кластер / А.С. Горобцов, Е.С. Сергеев, В.В. Гетманский //Научно-технические ведомости СПбГПУ. серия “Информатика. телекоммуникации. управление” / СПбГПУ. – Санкт-Петербург – 2010. -№3. - С. 93-99.

Моделирование живых систем с использованием высокопроизводительных вычислений*

В.М. Соловьев, Л.Ю. Коссович, И.В. Кириллова
Саратовский государственный университет

При построении оптимальных компьютерных моделей на основе механики деформируемого твердого тела (ДТТ), позволяющих создавать трёхмерные модели живых (биологических) систем, приходится сталкиваться с проблемой расчета систем со сложной геометрической конфигурацией и нерегулярной физической структурой. Например, в медицине при высокотехнологичных методах лечения перелома бедренной кости с учётом действия сил различных мышц на костные отломки. Такая сложная задача может быть решена на основе высокопроизводительного вычислительного программно-информационного комплекса (ПИК). Программно-информационный комплекс может использоваться для оперативной выработки лечебных рекомендаций. Он позволит выполнить расчеты с использованием приближенных численных методов – методов конечных элементов (МКЭ). Для использования МКЭ необходима адекватная трехмерная геометрическая модель и «решатель», специально разработанный для моделирования живых систем. Таким образом, весь процесс моделирования, например, переломов бедренной кости в ПИК может быть сведен к трем фазам: препроцессингу – подготовке адекватной трехмерной модели перелома бедренной кости; процессу моделирования с использованием «решателя», созданного на основе МКЭ; постпроцессингу – визуализации и интерпретации результатов моделирования.

Все три фазы могут быть реализованы в ПИК, на основе имеющихся вычислительных комплексах инженерных расчетов (CAE-систем) с проприетарным (proprietary) или свободно распространяемым (open source) программным обеспечением (ПО). В случае проприетарного ПО CAE-системы должны предусматривать возможность подключения автономных решателей (solvers), реализующих параллельные вычисления. Для свободно распространяемого ПО CAE-системы должны собираться (компилироваться) на основе создаваемых решателей. Причем вся структура программных модулей ПИК должны адаптироваться под решаемую задачу.

При создании такого ПИК необходимо учитывать, что трехмерная геометрическая модель должна быть ориентирована на численные методы с использованием МКЭ. Она должна быть реализована для множества трехмерных элементарных геометрических объектов, в совокупности адекватно представляющих живую систему, как твердое тело.

Однако на «негладких» поверхностях геометрической модели трудно обеспечить стыковку на линиях излома. Кроме того, в слоистых материалах, к которым относятся биологические объекты, возникают дополнительные трудности. Так, при моделировании каждого слоя бедренной кости, трудно стыковать, например, две слоистые структуры со слоями разной толщины. Поэтому вопросы «аккуратного» моделирования на фазе препроцессинга очень актуальны. Они могут быть решены только путем использования соответствующих технологий твердотельного моделирования в современных CAD-системах. При моделировании перелома бедренной кости с применением МКЭ основой моделей являются матричные вычисления. Используя перемещения в качестве параметров, вариационные принципы и соотношения трехмерной теории упругости, получают матрицы жесткости как линейные операторы, действующие на поле перемещений.

С другой стороны, поверхностный элемент в процессе деформации порождает многообразие перемещений точек поверхности и углов поворота, которые можно использовать в качестве параметров. В сочетании с инженерными теориями, основанными на гипотезах поведения, можно построить матрицы жесткости поверхностных элементов, как линейные операторы, действующие на перемещения и углы поворота поверхности. Применяя обычную технику МКЭ, требуется решить эти две задачи, используя соответствующие решатели, спроектированные для высокопроизводительных вычислений.

* Работа выполнена при финансовой поддержки Министерства образования и науки РФ, шифр 2009-04-1.4-20-05-014.

Применение coarray Fortran для реализации методов тензорного анализа сетей

Ю.Н. Сохор

Тензорные методы расчета технических устройств были предложены в 30-60-х годах прошлого столетия Г.Кроном [1] для расчета сложных технических систем. Техническая система описывается дифференциальными уравнениями в частных производных и обыкновенными дифференциальными уравнениями. В результате аппроксимации дифференциальных уравнений, строятся алгебраические уравнения, которым ставится в соответствие электрическая резистивная сеть. Введение различных координатных систем и их преобразование по правилам тензорной математики, позволяет свести расчет большой системы к расчету систем меньших размеров.

В данной работе, для реализации тензорных методов, была применена технология программирования coarray Fortran (CAF). В статье Р.Намрича [2] отмечается, что CAF представляет согласование тензорных обозначений с расширенным синтаксисом массивов Фортрана. Название технологии coarray связано с ее «тензорным» происхождением, так как другое название тензорного исчисления - ковариантное исчисление. Ковариантный тензор – аналог комассивов (coarray) в программировании. Технология может использоваться для архитектур с распределенной и (или) общей памятью. В настоящей работе расчеты выполнялись на 32-х ядерной архитектуре Intel ® Manycore Testing Lab [3] с использованием компиляторов Intel Fortran 12.0 и g95 для Linux.

Программа, написанная на CAF, копируется (отображается) в оперативной памяти N раз. Данные каждого отображения (image) доступны друг другу, если объявлены в программе в соответствии с синтаксисом coarray. Пересылка данных от одного отображения на другое выполняется также по синтаксическим правилам индексации комассивов, например, запись: `U(i) = U(j)[k]` соответствует пересылке j-го элемента массива U из k-го отображения, в i-й элемент массива U локального отображения. Можно считать, что в настоящее время это наиболее простой способ программирования межпроцессорного обмена данными.

В ходе исследований рассматривалась задача параллельного расчета эквивалентных узловых подсхем. Электрическая схема собиралась из разного количества подсхем по 1012 или 2047 сопротивлений. Число подсхем варьировалось от 1 до 30, каждая подсхема рассчитывалась на одном отображении. Тестовые расчеты для слабосвязанных схем показали, это для схемы, состоящей из подсхем с размером 1012 сопротивлений каждая, при увеличении числа рассчитываемых подсхем в 30 раз, время вычисления увеличивается линейно всего в 1,5 - 1,6 раза. Предварительные расчеты показали более сильную, по сравнению с кластерными расчетами, зависимость времени расчета от связности подсхем. Результаты исследований публикуются на сайте [4].

Литература

1. Kron G. Diakoptics. Macdonald, London, 1963
2. Robert W.Numrich. Parallel Numerical Algorithms Based on Tensor Notation and Co-Array Fortran Syntax // Parallel Computing, 31, p.588, 2005
3. Веб страница Intel® Manycore Testing Lab – http://software.intel.com/en-us/articles/intel-many-core-testing-lab
4. Исследование технических систем тензорными методами. Материалы сайта http://diakoptics.narod.ru.

Программный комплекс поиска оптимальных управлений на множествах простой структуры

О.В. Фесько

Институт программных систем имени А.К. Айламазяна РАН,
г. Переславль-Залесский

При исследовании различных динамических систем с управлением большое значение имеет поиск простых оптимальных законов управления, реализуемых на практике. В связи с этим предлагается алгоритм поиска решения задачи оптимального управления в виде кусочно-линейной функции управления с подвижными моментами переключения.

Рассматривается задача оптимального управления вида

$$\dot{x}(t) = f(t, x(t), u(t)), x(t_0) = x_0, \;\; t \in [t_0, t_1], \;\; F\left(x(t_1)\right) \to \min, \tag{1}$$

где $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$, $x_i(t), i = \overline{1,n}$ – кусочно-гладкие, управление $u(t)$ – кусочно-линейно разрывно с m штук моментов переключения.

Были найдены условия, при которых исходную задачу (1) можно свести к задаче условной конечномерной минимизации функции многих переменных [1]. Тогда алгоритм решения задачи (1) будет состоять в поочередном применении на каждой итерации численных алгоритмов: метода Рунге–Кутта/Рунге–Кутта–Фельберга для решения задачи Коши и комбинации методов Ньютона-Рафсона и градиентного спуска с модификациями для минимизации многомерной многоэкстремальной функции.

Алгоритм решения приведенной задачи (1) был реализован в виде параллельной программы на языке T++. Гранулой параллелизма в данной задаче была выбрана комбинация градиентного метода и метода Ньютона-Рафсона, т.е. независимый поиск минимума в подобластях, полученных при декомпозиции основной области. В таблице 1 представлены результаты ускорения при решении одной задачи оптимального управления с двумя моментами переключения (при разбиении основной области на 32 подобласти).

Таблица 1. Анализ эффективности параллельной версии программы

Число узлов, p	1	2	4	6	8	10	12
Время, t_p (в сек.)	2182.976	1319.838	826.408	526.569	454.782	412.992	352.079
Ускорение, t_1/t_p	1	1.65	2.64	4.15	4.8	5.29	6.2

Для автоматизации процесса проведения расчетов был написан графический интерфейс пользователя программного комплекса. Среди поддерживаемых функций комплекса можно выделить следующие: аутентификация пользователя на суперЭВМ; задание правых частей управляемой системы, терминального функционала и входных параметров задачи; выбор типа искомого управления (в виде кусочно-постоянных или кусочно-линейных функций); запуск и прерывание счета задачи на кластерном вычислительном устройстве; обмен входными/выходными данными по протоколу sftp; вывод результатов в виде текстовых файлов и графиков.

Литература

1. 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.

Методы декомпозиции для решения трехмерных задач трехфазной фильтрации жидкости на графических вычислительных системах

А.В. Цепаев

Учреждение Российской академии наук Институт механики
и машиностроения Казанского научного центра РАН

Рассматривается трехфазная изотермическая фильтрация нефти, воды и газа, подчиняющаяся линейному закону Дарси. Считается, что пласт, нефть и вода несжимаемы и отсутствует массообмен между нефтяной и газовой фазой. Гравитационные силы не учитываются. Тогда справедлива следующая система уравнений:

$$\mathrm{div}(\mathbf{q}_w) + m\partial S_w / \partial t = 0,$$

$$\mathrm{div}(\mathbf{q}_o) + m\partial S_o / \partial t = 0,$$

$$\mathrm{div}(\rho_g \mathbf{q}_g) + m\partial(\rho_g S_g) / \partial t = 0,$$

$$q_\alpha = -(f_\alpha k / \mu_\alpha)\mathrm{grad}(p), (\alpha = o, w, g).$$

Здесь p = p(x,y,z) – давление, индексы “o” ,”w”, “g” соответствуют нефти, воде и газу, $\mathbf{q}_\alpha$ - вектор скорости фильтрации фазы α, S_α - насыщенность пласта фазой α, $S_o + S_w + S_g = 1$, f_α, - относительные фазовые проницаемости, k – абсолютная проницаемость, μ_α - динамическая вязкость фазы, ρ_g - плотность газа в пластовых условиях, m - пористость.

Для поставленной задачи разработаны два метода декомпозиции области. Первый метод – для определения поля давлений, второй - для определения поля насыщенностей. Декомпозиция сеточных систем уравнений по давлению и насыщенности основана на независимом решении уравнений на сгущающихся участках и новом типе согласования этих решений с решением на грубой сетке. Согласование для уравнений по давлению достигается за счет введения дополнительных грубых сеток на сгущающихся участках. Согласование для уравнений по насыщенности достигается за счет сочетания элементов явной и неявной схем.

На основе предложенных методов декомпозиции построены алгоритмы для решения задачи на графических вычислительных системах.

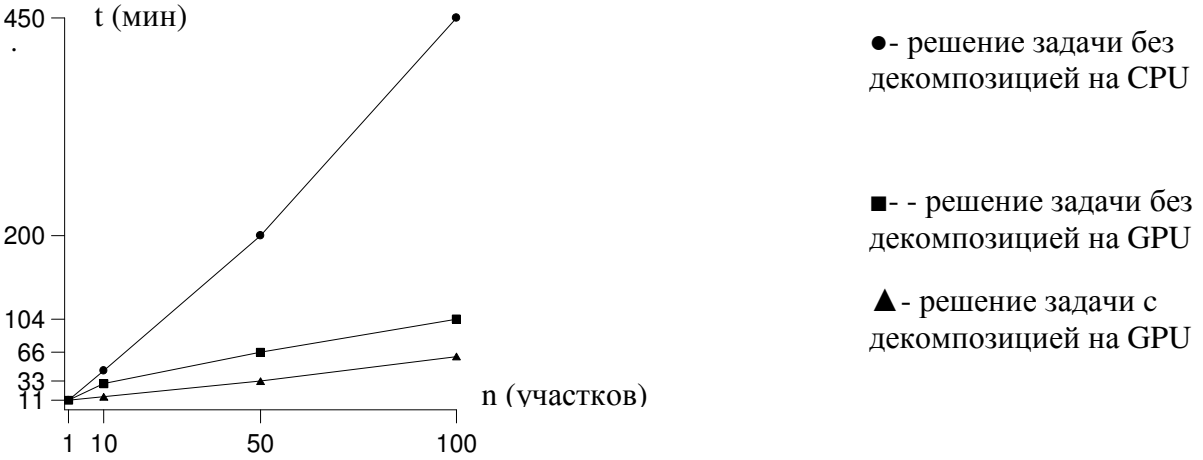


Рис.1. Время решения задачи с декомпозицией области по давлению и насыщенности на CPU и GPU

Применение суперкомпьютеров в задаче о переносе излучения*

М.А. Чащин, Е.Ф. Леликова, Л.И. Рубина, О.Н. Ульянов

Институт математики и механики Уральского отделения РАН

Авторы на протяжении ряда лет разрабатывают методику численного исследования радиационного переноса, сочетающую определение интенсивности излучения в каждой спектральной линии и вычисление населенностей уровней всех веществ смеси. Моделирование взаимодействия излучения и вещества в данной методике основывается на совместном решении интегро-дифференциального спектрального уравнения переноса излучения при соответствующих граничных условиях и системы уравнений кинетики населенностей уровней.

Методика имеет две важные особенности: методика направлена на расчет взаимодействия смеси веществ с излучением методом, который, фактически, является методом line-by-line; методика объединяет два подхода МАПИ и МПЛЧ, один из которых (МПЛЧ – Метод Полиномов Лагранжа-Чебышева) использует для представления решения полиномы Лагранжа с узлами Чебышева и обеспечивает приемлемое время расчетов, второй---(МАПИ – Метод Аналитического Представления Интенсивности) позволяет выбрать некоторые параметры для расчета по программам метода МПЛЧ, оценить и гарантировать в некотором смысле точность результатов расчета.

Ранее авторами изучался перенос излучения в плоском слое смеси веществ, содержащем несколько подслоев, имеющих разные плотности и температуры, с учетом нескольких десятков резонансных линий [1-2].

В данной работе представлено развитие методики для случая, когда в модели учитывается несколько сотен спектральных линий, и для случая, когда математическая модель дополнена уравнением энергобаланса для электронной температуры.

Предложенные алгоритмы, разработанные программы и модули позволили проводить численное моделирование для новых классов задач — задач с учетом до тысячи спектральных линий и задач с учетом энергобаланса — за приемлемое время с приемлемой точностью. Эти классы задач с физической точки зрения существенно более содержательны, с вычислительной — принципиально более сложные.

Проведены серии численных расчетов по моделированию взаимодействия излучения с веществом на суперкомпьютерах МВС-100К (МСЦ РАН, г. Москва) и «Уран» (ИММ УрО РАН, г. Екатеринбург).

Литература

1. 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.
2. 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.

* Работа выполнена в рамках Программы фундаментальных исследований Президиума РАН «Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация» при поддержке УрО РАН (проект 09-П-1-1003) и проекта «Моделирование фундаментальных и прикладных задач математической физики с применением супервычислителей» Региональной целевой программы развития вычислительных, телекоммуникационных и информационных ресурсов УрО РАН (проект РЦП-11-П8).

Параллельная реализация схемы усвоения данных на базе локального ансамблевого фильтра Калмана с преобразованием ансамбля

А.В. Шляева, В.Г. Мизяк

ГУ «Гидрометцентр России»

Усвоение данных является задачей нахождения оптимального анализа по имеющимся его оценкам (первому приближению, являющемуся, как правило, краткосрочным прогнозом, и наблюдениям), содержащим неизвестную ошибку. В усвоении с помощью ансамблевых фильтров Калмана на каждом шаге усвоения матрица ковариаций ошибок первого приближения аппроксимируется статистикой, полученной по ансамблю первых приближений. Как правило, используется ансамбль из 40-100 участников. Учитывая сложность современных моделей атмосферы, использование для усвоения методов на базе ансамблевого фильтра Калмана влечет за собой распараллеливание задачи по участникам ансамбля. Идея локальных ансамблевых фильтров заключается в том, что локализация приводит к уменьшению размерности, в которой решается задача (и к увеличению количества задач, которые необходимо решить). Одним из достоинств локализации является возможность эффективного распараллеливания усвоения, так как анализ можно вычислять параллельно в разных точках сетки. В данной работе описывается реализация локального ансамблевого фильтра Калмана с преобразованием ансамбля (Local Ensemble Transform Kalman Filter, LETKF), предложенного в работе [1] и представляющего собой детерминированный фильтр квадратного корня, использующий локализацию. Алгоритм распараллелен с использованием технологии MPI. Поскольку LETKF позволяет вычислять анализ в каждой из точек сетки независимо от остальных, алгоритм распараллелен по данным: каждый из вычислительных узлов получает свою область, в которой проводятся расчет анализа. Реализованная схема усвоения LETKF проверена [2] на полулагранжевой двумерной модели мелкой воды на сфере с форсингом [3]. Исследовалось усвоение с помощью данной схемы в непрерывном цикле усвоения. В качестве наблюдений использовались псевдонаблюдения, сгенерированные из данных реанализа 2 NCEP/NCAR. Фильтр стабильно работает на временах порядка года, при этом нормированная ошибка анализа (по сравнению с данными псевдонаблюдений в случайно выбранных точках) квазистационарна. Были проведены исследования влияния выбора различных параметров фильтра на качество усвоения, показавшие, в частности, что для исследованной задачи количество участников ансамбля может составлять 40–60. Проводится исследование данной схемы на трехмерной модели циркуляции атмосферы. В работе приводятся также результаты распараллеливания при использовании различного количества участников ансамбля. Обсуждается эффективность распараллеливания. Работа выполнена при поддержке гранта РФФИ 10-05-01066-а.

Литература

1. Hunt, B.R., Kostelich, E.J. and Szunyogh, I., 2007: Efficient data assimilation for spatiotemporal chaos: a Local Ensemble Transform Kalman Filter. Physica D, 230, 112 – 126.

2. Shlyaeva A.V., Tolstykh M.A., 2009. Local Ensemble Transform Kalman Filter for Semi-Lagrangian Barotropic Model of Atmosphere. The 5th WMO Symposium on Data Assimilation Extended Abstracts, 2009, Melbourne, p.216.1-216.5.

3. M.A. Tolstykh. Vorticity-divergence semi-Lagrangian shallow-water model on the sphere based on compact finite differences, J. Comput. Phys., 2002, v. 179, pp. 180-200.

Применение методов декомпозиции области при гидродинамическом моделировании нефтегазовых месторождений

А.А. Яппарова[1], Д.Ф. Марьин[1], И.Ф. Сайфуллин[2]

Уфимский государственный авиационный технический университет[1],
ООО «РН-УфаНИПИнефть»[2]

Методы пространственной декомпозиции (в англоязычной литературе Domain Decomposition Methods) – это методы решения дифференциальных уравнений в частных производных, прежде всего эллиптического и параболического типов, на основе идеологии «разделяй и властвуй». Основная идея методов пространственной декомпозиции для решения краевых задач состоит в разделении исходной задачи на подзадачи в подобластях меньшего размера. Задачи в подобластях решаются независимо друг от друга, что делает методы пространственной декомпозиции подходящими для параллельных вычислений, однако необходимо согласование решений на границах. Классические методы пространственной декомпозиции, делятся на два широких класса: методы с перекрытием и без перекрытия подобластей [1, 2].

Идея использования методов пространственной декомпозиции при гидродинамическом моделировании возникла из необходимости расчета моделей реальных нефтегазовых месторождений. Средствами библиотеки MPI были реализованы два метода пространственной декомпозиции: метод Шварца и метод Дирихле-Неймана, в качестве надстройки над существующим гидродинамическим симулятором NGT BOS.

Двумерной декомпозицией исходная модель месторождения разбивается на подмодели с сохранением информации о граничных ячейках. Каждая подмодель запускается на расчет независимо, выполняется согласование граничных условий между подобластями.

В методе Шварца решение в одной подобласти используется для получения новых значений граничных условий Дирихле в другой подобласти. Для метода Шварца были проведены численные эксперименты для исследования зависимости времени расчета модели от размера перекрытия.

В отличие от метода Шварца, методы без перекрытия требуют обеспечения непрерывности функций и их первых производных. Итерационный алгоритм Дирихле-Неймана предусматривает обмен значениями неизвестных в граничных ячейках и потоков через границы подобластей.

Полученные в ходе численных экспериментов результаты показывают, что при расчетах достаточно больших моделей со скважинами, расположенными вдали от границ между подобластями, реализованные методы декомпозиции являются достаточно эффективными. При сравнении результатов параллельных расчетов с результатами последовательного расчета наибольшее расхождение наблюдается около границ секторов, вокруг скважин и на границах движущихся фронтов.

Литература

1. Mathew T. Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations // Lecture Notes in Computational Science and Engineering. 2008. Volume 61. 764 p.
2. Toselli A., Widlund O. DomainDecomposition Methods – Algorithms and Theory // Springer Series in Computational Mathematics. 2004. Volume 34. 450 p.

Индекс по фамилиям

Содержание

Всероссийская конференция молодых ученых «Параллельные и распределенные вычисления»

Электронное издание

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ
(ПаВТ’2011)

Труды международной конференции
(Москва, 28 марта – 1 апреля 2011 г.)

Издательский центр Южно-Уральского государственного университета

Подписано в печать 24.02.2011. Формат 60×84 1/16.
Усл. печ. л. 85,09. Заказ 46.
