Российская академия наук
Суперкомпьютерный консорциум университетов России

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ (ПаВТ'2013)

Труды международной научной конференции

г. Челябинск, 1–5 апреля 2013 г.

Челябинск,
Издательский центр ЮУрГУ
2013

УДК 004.75
П 18

Параллельные вычислительные технологии (ПаВТ'2013): труды международной научной конференции (г. Челябинск, 1–5 апреля 2013 г.). Челябинск: Издательский центр ЮУрГУ, 2013. 637 с.

ISBN 978-5-696-04396-8

Данный сборник содержит статьи, включенные в программу Международной научной конференции «Параллельные вычислительные технологии 2013». Конференция проходила с 1 по 5 апреля 2013 года в Национальном исследовательском Южно-Уральском государственном университете. Подробную информацию о конференции можно найти в сети Интернет по адресу `http://agora.guru.ru/pavt`.

Отпечатано с авторских оригиналов.

Одобрено Советом факультета Вычислительной математики и информатики ЮУрГУ

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

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

Конференция проводится при поддержке
Российского фонда фундаментальных исследований

ISBN 978-5-696-04396-8

ПЛАТИНОВЫЕ СПОНСОРЫ

Группа компаний РСК
Корпорация Intel
Группа компаний Т-Платформы

ЗОЛОТЫЕ СПОНСОРЫ

Корпорация NVIDIA
Корпорация Hewlett-Packard
Компания ТЕСИС
Корпорация IBM

СЕРЕБРЯНЫЕ СПОНСОРЫ

Компания CADFEM
Компания Adaptive Computing

СПОНСОР СЕКЦИИ

Компания Делкам-Урал

ИНФОРМАЦИОННАЯ ПОДДЕРЖКА

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

НАБЛЮДАТЕЛЬНЫЙ СОВЕТ

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

ПРОГРАММНЫЙ КОМИТЕТ

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

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

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

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

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

ОРГАНИЗАЦИОННЫЙ КОМИТЕТ

Председатель организационного комитета:
Шестаков А.Л, ректор ЮУрГУ, г. Челябинск

Зам. председателя организационного комитета:
Соколинский Л.Б., декан факультета ВМИ ЮУрГУ, г. Челябинск

Члены организационного комитета:
Брызгалов П.А., научный сотрудник НИВЦ МГУ, г. Москва
Долганина Н.Ю., начальник отдела поддержки и обучения пользователей ЛСМ ЮУрГУ, г. Челябинск
Иванова Е.В., программист ЛСМ ЮУрГУ, г. Челябинск
Костенецкий П.С., директор СКЦ ЛСМ ЮУрГУ, г. Челябинск
Пан К.С., программист ЛСМ ЮУрГУ, г. Челябинск
Радченко Г.И., зам. руководителя ЛСМ ЮУрГУ, г. Челябинск
Репина К.В., программист ЛСМ ЮУрГУ, г. Челябинск
Силкина Н.С., зам. декана факультета ВМИ ЮУрГУ, г. Челябинск
Скрипник Д.С., лаборант кафедры системного программирования ЮУрГУ, г. Челябинск
Соболев С.И., научный сотрудник НИВЦ МГУ, г. Москва
Халюченко Л.И., программист ЛСМ ЮУрГУ, г. Челябинск
Худякова Е.С., программист ЛСМ ЮУрГУ, г. Челябинск
Цымблер М.Л., начальник отдела интеллектуального анализа данных и виртуализации ЛСМ ЮУрГУ, г. Челябинск
Шамакина А.В., начальник отдела распределенных вычислений и встроенных систем ЛСМ ЮУрГУ, г. Челябинск

ЭКСПЕРТЫ ПаВТ

Авербух В.Л., ИММ УрО РАН, г. Екатеринбург
Адинец А.В., НИВЦ МГУ, г. Москва
Аксенов А.А., ТЕСИС, г. Москва
Антонов А.С., НИВЦ МГУ, г. Москва
Баландин Д.В., ННГУ, г. Нижний Новгород
Бандман О.Л., ИВМиМГ СО РАН, г. Новосибирск
Баркалов К.А., ННГУ, г. Нижний Новгород
Болдарев А.С., ИПМ РАН, г. Москва
Бородулин К.В., ЮУрГУ, г. Челябинск
Варламов Д.А., ИПХФ РАН, г. Черноголовка
Водопьянов В.В., УГАТУ, г. Уфа
Волохов В.М., ИПХФ РАН, г. Черноголовка
Вшивков В.А., ИВМиМГ СО РАН, г. Новосибирск
Гасилов В.А., ИПМ РАН, г. Москва
Горнов А.Ю., ИДСТУ СО РАН, г. Иркутск
Городняя Л.В., ИВМиМГ СО РАН, г. Новосибирск
Губайдуллин И.М., ИНиК РАН, г. Уфа
Долганина Н.Ю., ЮУрГУ, г. Челябинск
Дордопуло А.И., ЮНЦ РАН, г. Таганрог
Дорохов В.А., ЮУрГУ, г. Челябинск
Жуматий С.А., НИВЦ МГУ, г. Москва
Затуливетер Ю.С., ИПУ РАН, г. Москва
Иванова Е.В., ЮУрГУ, г. Челябинск
Иванова О.Н., ЮУрГУ, г. Челябинск

Карпенко А.П., МГТУ, г. Москва
Кетков Ю.Л., ННГУ, г. Нижний Новгород
Козинов Е.А., ННГУ, г. Нижний Новгород
Корж О.В., МГУ, г. Москва
Коротченко А.Г., ННГУ, г. Нижний Новгород
Костенецкий П.С., ЮУрГУ, г. Челябинск
Крюков В.А., ИПМ РАН, г. Москва
Легалов А.И., СФУ, г. Красноярск
Линд Ю.Б., БашНИПИнефть, г. Уфа
Линёв А.Н., ННГУ, г. Нижний Новгород
Лукащук С.Ю., УГАТУ, г. Уфа
Лымарь Т.Ю., ЮУрГУ, г. Челябинск
Марчевский И.К., МГТУ, г. Москва
Медведев А.А., ЮУрГУ, г. Челябинск
Мееров И.Б., ННГУ, г. Нижний Новгород
Миниахметов Р.М., ЮУрГУ, г. Челябинск
Михайленко К.И., ИМ УНЦ РАН, г. Уфа
Мортиков Е.В., НИВЦ МГУ, г. Москва
Оленев Н.Н., ВЦ РАН, г. Москва
Пан К.С., ЮУрГУ, г. Челябинск
Пивушков А.В., ИПХФ РАН, г. Черноголовка
Подлазов В.С., ИПУ РАН, г. Москва
Половинкин А.Н., ННГУ, г. Нижний Новгород
Посыпкин М.А., ИСА РАН, г. Москва
Радченко Г.И., ЮУрГУ, г. Челябинск
Репина К.В., ЮУрГУ, г. Челябинск
Розенберг В.Л., ИММ УрО РАН, г. Екатеринбург
Романов С.Ю., НИВЦ МГУ, г. Москва
Сальников А.Н., ВМК МГУ, г. Москва
Свешников В.М., ИВМиМГ СО РАН, г. Новосибирск
Сенин А.В., ННГУ, г. Нижний Новгород
Соболев С.И., НИВЦ МГУ, г. Москва
Стегайлов В.В., ОИВТ РАН, г. Москва
Степаненко В.М., НИВЦ МГУ, г. Москва
Сухорослов О.В., ИСА РАН, г. Москва
Толстых М.А., ИВМ РАН, г. Москва
Файзуллин Р.Т., ОмГТУ, г. Омск
Федянина Р.С., ЮУрГУ, г. Челябинск
Халюченко Л.И., ЮУрГУ, г. Челябинск
Харченко С.А., ТЕСИС, г. Москва
Худякова Е.С., ЮУрГУ, г. Челябинск
Чеверда В.А., ИВМиМГ СО РАН, г. Новосибирск
Шалаев В.И., МФТИ, г. Москва
Шамакина А.В., ЮУрГУ, г. Челябинск
Шишков А.В., ННГУ, г. Нижний Новгород

Правда, искажающая истину.
Как следует анализировать Top500?

С.М. Абрамов

Институт программных систем имени А.К. Айламазяна Российской академии наук

После каждого выпуска рейтинга Top500 [1] выполняются подсчеты и публикуются суждения, вида: «Подавляющее большинство суперкомпьютеров списка Top500 используются в индустрии». Или другие подобные подсчеты и суждения о долях в списке Top500: (i) разных типов процессоров; (ii) различных типов интерконнекта; (iii) производителей суперкомпьютеров; (iv) стран и т. п. Часто на базе подобных подсчетов и суждений принимаются серьезные решения, в том числе и на правительственном уровне. В данной работе показано: все, что сказано в подобных суждениях — это правда, однако эта правда серьезно искажает истину: искажает истинное положение в суперкомпьютерной отрасли. В работе дается анализ причины серьезного отличия «правды» от «истины», приводятся методика корректного анализа данных Top500 и результаты такого анализа.

1. Введение

Начиная с июня 1993 года, два раза в год публикуется список пятисот самых мощных суперкомпьютеров мира — мировой рейтинг Top500 [1]. Всего за истекшие 20 лет опубликовано сорок выпусков Top500. Каждая публикация рейтинга является серьезным новостным событием, а также поводом для анализа состояния и тенденций суперкомпьютерной отрасли[1].

После выхода новой редакции рейтинга (или одновременно с этим) многие выполняют различные подсчеты и публикуют суждения, основанные на результатах таких подсчетов. Довольно часто подсчеты посвящены вычислению различных долей в списке Top500. Например, вычисляют, какие доли приходятся на различные области применения суперкомпьютеров в рейтинге Top500. Или, какие доли приходятся на суперкомпьютеры, использующие те или иные микропроцессоры. Анализируют и другие распределения долей: доли различных архитектур, доли производителей суперкомпьютеров, доли стран и т. п.…

Среди прочих, таким анализом занимаются и сами издатели рейтинга — на портале Top500 публикуют одновременно и сам список, и плакат, посвященный выходу в свет новой редакции рейтинга. Обратим внимание на плакат, выпущенный в ноябре 2012 года[2], рассмотрим диаграмму «Installation Type» с этого плаката — Рис. 1. Мы видим диаграмму из 40 столбцов — каждый столбец соответствует одному выпуску рейтинга, два столбца (июнь и ноябрь) приходятся на один год. Столбец состоит из частей разных цветов; размер частей определяется долями различных сегментов применения суперкомпьютеров из соответствующего рейтинга Top500. Различают шесть значений для задания сегментов применения: «Vendor», «Research», «Industry», «Government», «Classified» и «Academic».

Действительно, легко взять полную Excel-таблицу[3] списка Top500 за ноябрь 2012 года и посчитать, сколько суперкомпьютеров в нем в колонке «Segment» имеют то или иное значение области применения. Результат представлен ниже (Таблица 1). Доли, посчитанные в третьей колонке, естественно, в точности соответствуют длинам цветных частей в правого столбца диаграммы «Installation Type» (Рис. 1). Тем самым, будет справедливым сделать следующее суждение:

---

[1] Здесь и далее используется широкое толкование суперкомпьютерной отрасли, что включает исследование, разработку, изготовление, эксплуатацию суперкомпьютерных технологий и охватывает аппаратные решения, программное обеспечение — системное, инструментальное, прикладное,— и суперкомпьютерные сервисы.

[2] http://s.top500.org/static/lists/2012/11/TOP500_201211_Poster.pdf

[3] http://s.top500.org/static/lists/2012/11/TOP500_201211.xls

§1 *В ноябре 2012 года большая часть (49.40%) суперкомпьютеров использовалось в индустрии (Segment=Industry). При этом индустриальное использование превосходило применение суперкомпьютеров для науки — 44.2% = 24.60% + 19.60% (Segment=Research и Segment= Academic).*

По результатам подобного подсчета для Top500 за ноябрь 2009 года (обратите внимание на Рис. 1 на столбец, соответствующий ноябрю 2009 года) можно сказать еще сильнее:

§2 *В ноябре 2009 года **подавляющая** часть (62.40%) суперкомпьютеров использовалась в индустрии (Segment=Industry). При этом индустриальное использование **значительно** (**почти вдвое**) превосходило применение суперкомпьютеров для науки — 34.0% = 18.20% + 15.80% (Segment=Research и Segment= Academic).*

Таблица 1. Распределение суперкомпьютеров по различным сегментам применения — по сведениям Top500 за ноябрь 2012 года

Значения в колонке «Segment»	Количество систем	Доля
Vendor	12	2.40%
Research	123	24.60%
Industry	247	49.40%
Government	16	3.20%
Classified	4	0.80%
Academic	98	19.60%
ВСЕГО	**500**	**100.00%**

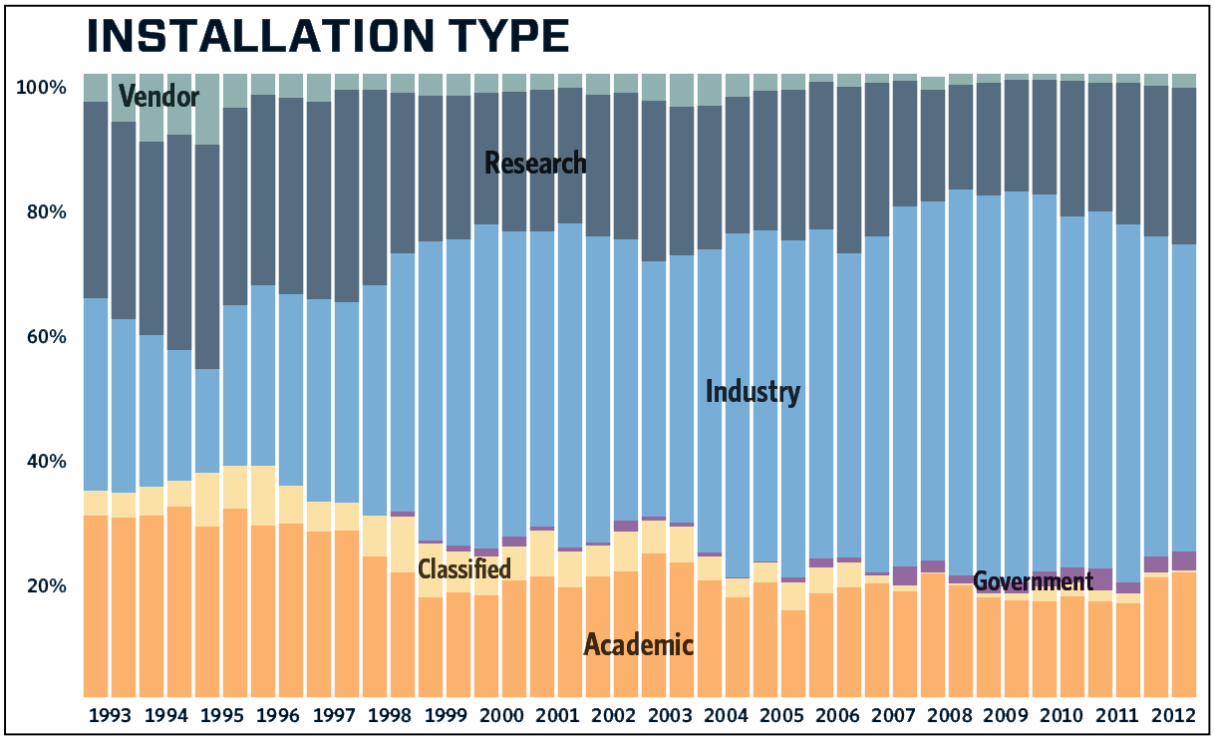


Рис. 1. Диаграмма «Installation Type» с плаката рейтинга Top500, выпущенного в ноябре 2012 года

Подобные вычисления и суждения (§1 и §2) легко могут быть построены — для этого не нужно быть большим специалистом, достаточно начальных навыков владения Excel-ем. Более того, график «Installation Type» просто входит в официальный плакат рейтинга Top500 и очень наглядно иллюстрирует распределение суперкомпьютеров по сегментам применения и то, как с течением времени меняется это распределение.

И подобные суждения (§1 и §2) и график «Installation Type» широко обсуждаются в различных публикациях, которые читают и специалисты, и обыватели, и лица, принимающие решения. Как результат, суждения, подобные §1 и §2, находим в правительственной переписке самого высокого уровня, посвященной суперкомпьютерам. Естественно, в контексте §1 и §2 следующие управленческие решения *на первый взгляд кажутся вполне разумными*:

§3 Государственная поддержка должна стимулировать создание суперкомпьютеров в большей степени (почти в два раза) не в научных российских центрах, а в промышленных.

§4 В деле развития российской суперкомпьютерной отрасли представляется правильным перераспределить ресурсы, роли и ответственность с переносом центра тяжести к министерствам и ведомствам, связанным с индустрией, а не с наукой.

§5 При создании суперкомпьютеров следует стремиться к таким долям государственного финансирования и привлекаемых из индустрии внебюджетных средств (ВБС): около 35% от государства, около 65% ВБС от индустрии — см. §2.

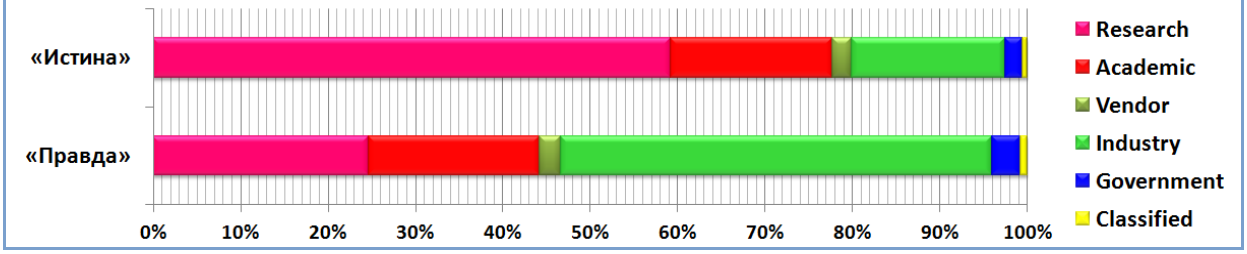
Ключевым обстоятельством в данной статье является следующее: график «Installation Type» (Рис. 1), суждения §1 и §2 — все они являются правдой, но **эта правда существенным образом искажает истинное положение дел в суперкомпьютерной отрасли**. И как результат — сплошь и рядом приводит к ошибочным управленческим решениям.

Если посчитать истинные доли, приходящиеся на различные сегменты применения суперкомпьютеров[1] (Таблица 2), то мы увидим, что различие[2] между «правдой» (колонка А) и «истиной» (колонка В) оказывается весьма значительной — в разы. И так же от §1 разительно отличается истинное суждение:

§6 В ноябре 2012 года подавляющая доля — 77.67% = 59.23% + 18.44% — была у использования суперкомпьютеров в науке (Segment=Research и Segment= Academic), что многократно (в 4,4 раза) больше доли (17.56%) использования суперкомпьютеров в индустрии (Segment=Industry).

Таблица 2. Истинное распределение долей различных сегментов применения суперкомпьютеров — по сведениям Top500 за ноябрь 2012 года

Сегмент применения	(A) «Правда»: доля систем (Таблица 1)	(B) «Истина»: истинная доля	Степень искажения истины
Research	24.60%	59.23%	↓2.41
Academic	19.60%	18.44%	↑1.06
Vendor	2.40%	2.22%	↑1.08
Industry	49.40%	17.56%	↑2.81
Government	3.20%	2.00%	↑1.60
Classified	0.80%	0.55%	↓2.41
ВСЕГО	**100.00%**	**100.00%**	



[1] Как следует вычислять истинные доли — будет обсуждено в последующих разделах.

[2] Степень искажения истины — самая правая колонка, вычисляется как max(A,B)/min(A,B),— что указывает во сколько раз «правда» (A) приуменьшает (знак «↓» перед числом) или преувеличивает (знак «↑» перед числом) «истину» (B).

Серьезное отличие (в разы) «правды» от «истины» влечет опасность и недопустимость использования графика «Installation Type» (Рис. 1), суждений §1 и §2 для обоснования любых управленческих решений: на их основе легко сделать ложные выводы и, как результат — вредные управленческие решения, например §3–§5.

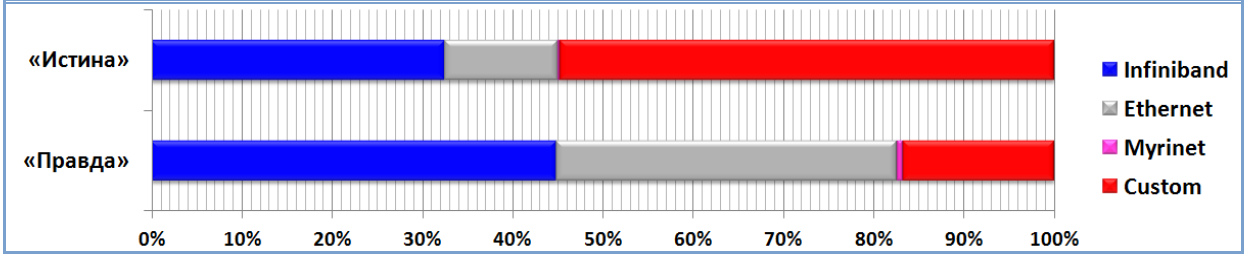
Распределение долей вычисляют не только для сегментов применения суперкомпьютеров. В общем случае, если суперкомпьютеры некоторой редакции Top500 каким-то образом разбиты на категории, то можно двумя способами посчитать доли этих категорий: (A) по общепринятой процедуре — доли числа суперкомпьютеров (среди всех 500 систем) соответствующей каждой категории; (B) истинные доли категорий — методика подсчета обсуждается в разделе 3.

На основе данных Top500 за ноябрь 2012 года были построены таблицы таких долей для следующих категорий:

– Таблица 3: используемая технология интерконнекта: Infiniband, Ethernet, Myrinet и Custom[1].
– Таблица 4: компания-производитель: IBM, Hewlett-Packard, Cray Inc. и Others[2].

Таблица 3. Распределение долей между разными технологиями интерконнектов, используемых в суперкомпьютерах — по сведениям Top500 за ноябрь 2012 года

Интерконнект	Число систем	(A) «Правда»: доля систем в Top500	(B) «Истина»: истинная доля	Степень искажения истины
Infiniband	224	44.80%	32.51%	↑1.38
Ethernet	189	37.80%	12.60%	↑3.00
Myrinet	3	0.60%	0.21%	↑2.79
Custom	84	16.80%	54.68%	↓3.25
Всего	**500**	**100.00%**	**100.00%**	



Во всех рассмотренных случаях очень часто «правда» сильно (в разы) отличается от «истины»: Таблица 2–Таблица 4, см. правую колонку. Вот несколько примеров:

– Таблица 2: в колонке «A» доля сегмента «Industry» преувеличена в 2.8 раза; доля сегмента «Research» преуменьшена в 2.4 раза.
– Таблица 3: в колонке «A» доля технологии «Infiniband» преувеличена в 1.38 раза, доля технологии «Ethernet» преувеличена в 3 раза, доля коммерчески недоступных решений — «Custom»,— преуменьшена в 3.25 раз.
– Таблица 4: в колонке «A» доля компании Hewlett-Packard преувеличена в 2.61 раза, доля компании Cray Inc. преуменьшена в 2.8 раза.

Какой должна быть корректная методика вычисления истинных долей? По какой причине «правда» так сильно отличается от «истины»? — Все это будет обсуждено ниже:

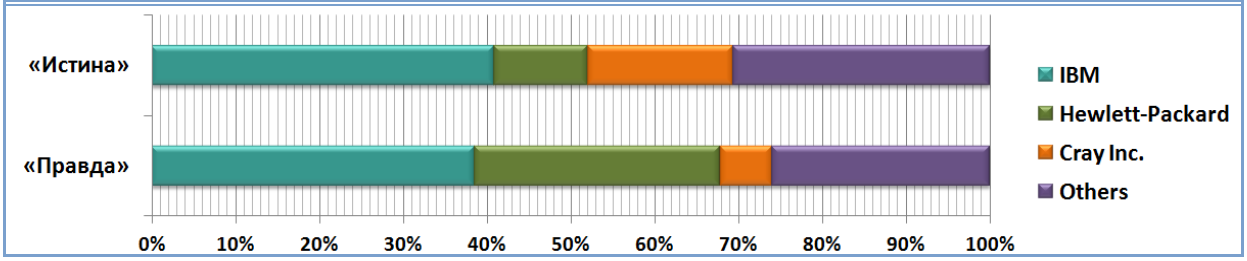
– В разделе 2 обсуждаются основные определения и понятия.
– В разделе 3 определяется методика вычисления истинных долей.
– В разделе 4 обсуждается причина сильного отличия «правды» от «истины».

[1] Интерконнект, коммерчески недоступный на рынке. Недоступный, по крайней мере, как отдельное изделие. Интерконнект, который невозможно купить. Если надо использовать такой, то придется (аналог) разработать самостоятельно.
[2] Все остальные компании-поставщики.

– В разделах 5–8 исследуются истинные доли для таких понятий как «сегменты применения суперкомпьютеров», «технологии процессоров, используемых в суперкомпьютерах», «компании-производители», «технологии интерконнекта».
– В разделе 9 формулируется заключение по результатам работы.

Таблица 4. Распределение долей между разными компаниями-производителями суперкомпьютеров — по сведениям Top500 за ноябрь 2012 года

Компания-производитель	Число систем	(A) «Правда»: доля систем в Top500	(B) «Истина»: истинная доля	Степень искажения истины
IBM	193	38.60%	40.84%	↓1.06
Hewlett-Packard	146	29.20%	11.18%	↑2.61
Cray Inc.	31	6.20%	17.39%	↓2.80
Others	130	26.00%	30.59%	↓1.18
Всего	**500**	**100.00%**	**100.00%**	



2. Высокопроизводительные вычисления, суперкомпьютеры

Для того чтобы исключить неверное толкование, приведем используемые нами определения некоторых терминов.

2.1 Производительность

Среди важнейших технических характеристик компьютеров традиционно выделяют ***производительность*** — количество операций с плавающей точкой, выполняемых вычислителем за секунду[1]. Различают

– ***пиковую производительность*** — максимальное число операций в секунду, которое может выполнить установка в идеальном случае — в принципе;
– ***реальную производительность на некоторой задаче*** — реальное количество операций, выполненных при решении задачи, деленное на реальное время решения задачи.

Пиковую производительность оценивают теоретически, исходя из состава оборудования компьютера. Реальную производительность измеряют опытным путем, решая на системе некоторую задачу. На разных задачах реальная производительность одного и того же компьютера могут быть разными.

Для сравнения производительности различных суперкомпьютеров между собой чаще всего используют реальную производительность на задаче Linpack[2]. В последнее время набирают популярность и другие тесты реальной производительности суперкомпьютеров, например, основанные на задачах с интенсивной обработкой данных[3].

[1] 1 Gflops — гигафлопс, 10^9 операций в секунду, 1 Tflops — терафлопс, 10^{12} операций в секунду, 1 Pflops — петафлопс, 10^{15} операций в секунду, 1 Eflops — эксафлопс, 10^{18} операций в секунду.
[2] Решение системы линейных уравнений с большим числом неизвестных. Используется в мировом рейтинге суперкомпьютеров Top500 [1].
[3] Например, задача поиска в большом графе в ширину — используется как тест в другом мировом рейтинге суперкомпьютеров — Graph500 [2].

2.2 Суперкомпьютеры

Отметим интересный факт: если в известной сетевой энциклопедии Wikipedia [3] попытаться просмотреть термин «High-performance computing — высокопроизводительные вычисления», то, как результат, будет перенаправление на страницу «Supercomputer — Суперкомпьютер». Это верно и для англоязычной и для русскоязычной версии Wikipedia. Тем самым, подчеркивается синонимичность понятий «высокопроизводительный компьютер» и «суперкомпьютер». Дадим формальное определение.

К вычислительным системам высокой производительности — к ***суперкомпьютерам***,— отнесем вычислительные машины, значительно превосходящие по своей реальной производительности большинство существующих компьютеров.

То есть, в каждый момент времени, если среди всех существующих компьютеров отобрать самые мощные — например, ***пятьсот самых производительных***,— то они и определят термин «суперкомпьютер» на данный момент времени.

Значит, начиная с июня 1993 года, можно установить тесную связь между понятием «суперкомпьютер» и рейтингом Top500. В принципе можно сказать, что вычислительная система является суперкомпьютером, если она была включена[1] в некоторый выпуск рейтинга Top500 — и только в этом случае.

Тем самым, каждую редакцию Top500 можно рассматривать как исчерпывающее описание текущего состояния суперкомпьютерных технологий. А всю совокупность выпусков рейтинга можно рассматривать как исчерпывающую хронологию суперкомпьютерной отрасли за последние 20 лет.

2.3 Top500 — источник знаний о суперкомпьютерной отрасли

Редакции рейтинга Top500 публикуются дважды в год (в июне и ноябре) начиная с июня 1993 года. Рейтинг основан на реальной производительности суперкомпьютеров на задаче Linpack. На сегодня в открытом доступе [1] имеются данные 40 выпусков рейтинга — с июня 1993 года по ноябрь 2012 года. Можно выгрузить каждый выпуск рейтинга в виде Excel-таблицы. В этом случае предоставляется самая полная информация. Если свести все 40 Excel-таблиц вместе, то получим таблицу с 40×500 = 20 000 строками и с 40 колонками[2] (полями). Профессиональный анализ выпусков списка Top500 позволяет строить весьма достоверные суждения о состоянии и перспективах суперкомпьютерных технологий в мире и в России.

Обратим внимание, что при проведении анализа иногда приходится совместно обрабатывать несколько полей одной записи. Так, совместная обработка полей «Segment» и «Application Area» позволяет установить область использования суперкомпьютера более точно, чем это указано в поле «Segment». Чтобы точнее понять устройство интерконнекта, имеет смысл обрабатывать тоже два поля: «Interconnect» и «Interconnect Family». Для точного определения используемого процессора надо рассмотреть шесть полей: «Processor», «Processor Family», «Processor Generation», «Processor Technology», «Proc. Frequency», «Cores per Socket».

Понятно, что вручную выполнить тонкий анализ такого количества данных (20 000 записей с 40 полями) невозможно. Поэтому автор в 2009 году в инициативном порядке создал и до сих пор развивает программу Top500 Analyzer [4] для анализа рейтинга Top500. Все иллюстрации (за исключением Рис. 1 и Рис. 7) и все данные для расчетов в данной работе подготовлены при помощи этой программы.

---

[1] Либо ее технические показатели позволяли ее включить в рейтинг, но это не было сделано по некоторым причинам.

[2] Приведем имена этих полей: Accelerator, Accelerator Cores, Application Area, Architecture, Computer, Continent, Cores, Cores per Socket, Country, Efficiency(%), First Appearance, First Rank, Interconnect, Interconnect Family, Manufacturer, Measured Size, Mflops/Watt, Name, Nhalf, Nmax, Operating System, OS Family, Power, Previous Rank, Proc. Frequency, Processor, Processor Cores, Processor Family, Processor Generation, Processor Technology, Rank, Region, RMax, Rpeak, Segment, Site, System Family, System Model, Year.

2.4 О частичной неполноте и о частичной недостоверности Top500

Время от времени случаются публикации [5], указывающие на ***частичную недостоверность данных*** в рейтинге Top500: в рейтинг попадают установки до момента, когда они реально создаются, или установки, которые прекратили свое существование. Бывает.

Кроме того, всегда и во всех странах существуют суперкомпьютеры, которые не включают в рейтинг Top500 из соображений государственной безопасности или по каким-то другим причинам. Значит можно говорить о ***частичной неполноте данных*** в рейтинге Top500.

Однако можно предполагать, что эти обстоятельства:

- не существенны;
- более-менее равномерно влияют на различные категории суперкомпьютеров.

Тем самым, выводимые из данных Top500 относительные оценки оказываются весьма достоверными.

Это подобно тому, что вполне достоверно можно сравнивать между собою айсберги, основываясь на неполной информации — анализируя только их надводные (видимые) части.

3. Методика вычисления истинных долей

Почему правильное суждение (абсолютная правда) «В ноябре 2012 года большая часть (247 из 500) суперкомпьютеров использовалась в индустрии (Segment=Industry)» не может использоваться для вычисления «истинной доли» индустриального применения суперкомпьютеров в лоб — по формуле 247/500 = 49.40%?

Совсем небольшое размышление приводит к правильному ответу: суперкомпьютеры нельзя мерить штуками.

Пять одних суперкомпьютеров могут сильно отличаться от пяти других, в любом смысле: в стоимостном (при оценке долей рынка), в смысле технической сложности (при оценке доли в общем количестве процессоров/ядер или доли в общем числе портов интерконнекта») и т. п.

Вычисляя «истинные доли», следует оперировать не количеством суперкомпьютеров «в штуках», а такими числовыми характеристиками суперкомпьютеров, которые наиболее верно отражают **наиважнейшую характеристику суперкомпьютеров**, как изделий. Точно так же, например, когда сравнивают торговые флоты разных стран, измеряют размеры флотов не «в штуках», а в суммарном тоннаже.

Самая важная числовая характеристика суперкомпьютеров очевидна (даже просто в силу самого определения понятия «суперкомпьютер», см. раздел 2.2) — это реальная производительность. Конечно, лучше было бы при этом оперировать реальной производительностью на некоторых целевых (интересующих того или иного заказчика), задачах. Но если таких сведений нет, то будем довольствоваться Linpack-производительностью, сведения о которой имеются в записях Top500 — поле RMax.

3.1 Linpack-производительность, как истинная мера при измерении долей

Реальная производительность — в частности, Linpack-производительность,— главная, определяющая характеристика суперкомпьютеров. По ней разграничиваются суперкомпьютеры от «просто компьютеров». Кроме того, по сравнению со «штуками», Linpack-производительность гораздо точнее[1] коррелирует с такими характеристиками, как:

- Научно-технический уровень системы.
- Стоимость системы — что важно для правильной оценки распределения долей рынка.
- Объемы различных подсистем и смежные технические параметры. Например, размер подсистемы интерконнекта (количество портов), количество процессоров или ядер и т. п.

Таким образом, мы приходим к методике расчета истинных долей через ***вычисление доли суммарной Linpack-производительности***.

[1] Разницу можно оценить в два порядка — до ~250 раз,— как увидим далее.

3.2 Формальное описание метода вычисления истинных долей

Пусть $n \in [1..40]$ — номер редакции Top500, $i \in [1..500]$ позиция, занятая некоторым суперкомпьютером в рейтинге, обозначим $RMax(n,i)$ — Linpack-производительность данной системы в *n*-ой редакции Top500.

Рассмотрим некоторую категорию суперкомпьютеров, например, все суперкомпьютеры индустриального использования: Segment=Industry. Пусть $\mathcal{C} = \{\dots i \dots\} \subseteq [1..500]$ множество всех позиций, которые суперкомпьютеры из данной категории занимают в *n*-ой редакции Top500.

В *n*-ой редакции Top500 ***истинную долю суперкомпьютеров заданной категории*** определим как долю суммарной Linpack-производительности суперкомпьютеров данной категории в суммарной Linpack-производительности всего списка:

$$\frac{\sum_{i\in\mathcal{C}} RMax(n,i)}{\sum_{i\in[1..500]} RMax(n,i)}$$

Рассмотрим некоторый ***подсписок*** в *n*-ой редакции Top500, заданный множеством позиций $\mathcal{I} = \{\dots i \dots\} \subseteq [1..500]$. Например, «первую сотню» Top1–100: $\mathcal{I} = [1..100]$.

В *n*-ой редакции Top500 ***истинную долю суперкомпьютеров заданной категории в указанном подсписке*** определим как долю суммарной Linpack-производительности суперкомпьютеров данной категории из подсписка в суммарной Linpack-производительности всего подсписка:

$$\frac{\sum_{i\in(\mathcal{C}\cap\mathcal{I})} RMax(n,i)}{\sum_{i\in\mathcal{I}} RMax(n,i)}$$

4. Причина сильного отличия «правды» от «истины»

Используя обозначения раздела 3.2, посчитаем «правду» — долю категории $\mathcal{C}$ по традиционной методике, «в штуках»:

$$p_1 = \frac{\sum_{i\in\mathcal{C}} 1}{\sum_{i\in[1..500]} 1} = \sum_{i\in\mathcal{C}} 1/500 = \sum_{i\in\mathcal{C}} 0.2\%$$

Таким образом, при такой методике в общую копилку доли p_1 категории $\mathcal{C}$ каждый суперкомпьютер вносит один и тот же вклад — 0.2%,— вне зависимости от того, крупный это суперкомпьютер или небольшой, дорогой или бюджетный и т. п.

Введем обозначение для доли Linpack-производительности одного суперкомпьютера $RMax(n,i)$ в суммарной Linpack-производительности всего списка:

$$pRMax(n,i) = \frac{RMax(n,i)}{\sum_{i\in[1..500]} RMax(n,i)}$$

Тогда истинную долю категории $\mathcal{C}$ можно записать таким образом:

$$p_2 = \frac{\sum_{i\in\mathcal{C}} RMax(n,i)}{\sum_{i\in[1..500]} RMax(n,i)} = \sum_{i\in\mathcal{C}} pRMax(n,i)$$

Сравним между собою «правду» $p_1 = \sum_{i\in\mathcal{C}} 0.2\%$ и «истину» $p_2 = \sum_{i\in\mathcal{C}} pRMax(n,i)$. Ясно, что если бы все суперкомпьютеры не очень сильно отличались бы между собою по Linpack-производительности, то все $pRMax(n,i)$ были бы близки к 0.2%, а «правда» p_1 не сильно бы отличалась от «истины» p_2.

Однако суперкомпьютеры в одном и том же рейтинге Top500 имеют огромный разброс в Linpack-производительности $RMax(n,i)$ и, как следствие, огромный разброс $pRMax(n,i)$ — от 10.849% до 0.047% для 40-ой редакции рейтинга Top500 — разница в 230 раз!

Такое гигантское расслоение суперкомпьютеров по параметру Linpack-производительности определяет огромное отличие «правды» от «истины». Это расслоение делает осмысленным введение отдельных уровней (слоев, классов) суперкомпьютеров.

4.1 Различные уровни суперкомпьютерных систем

В работе [6] были введены 4 уровня суперкомпьютеров — Top1–20, Top21–100, Top101–250, Top251–500,— цитата:

- ***суперЭВМ в крупнейших национальных центрах*** — единичные установки в стране, соответствующие местам 1–20 в мировом рейтинге Top500;
- ***суперЭВМ в крупнейших региональных и отраслевых центрах*** — два–четыре десятка установок в стране, соответствующих местам 21–100 в мировом рейтинге Top500;
- ***суперЭВМ в крупных региональных и корпоративных центрах*** — от четырех десятков до сотни установок в стране, соответствующих местам 101–250 в мировом рейтинге Top500;
- ***суперЭВМ предприятий и научных учреждений*** — одна–три сотни установок в стране, соответствующих местам 251–500 в мировом рейтинге Top500.

В работе [7] предлагается и обосновывается выделение из первого уровня отдельной группы ***сверхвысокопроизводительных систем*** — Top1–10.

Везде далее обсуждаются эти пять уровней суперкомпьютеров: Top1–10, Top11–20, Top21–100, Top101–250 и Top251–500.

4.2 Резкое расслоение в суперкомпьютерной отрасли по Linpack-производительности

Для оценки глубины расслоения суперкомпьютерной отрасли рассмотрим разницу в Linpack-производительности у суперкомпьютеров разных уровней (по данным редакции Top500 за ноябрь 2012 года, Таблица 5). Разница по Linpack-производительности самой мощной и самой слабой системы в классе Top1–20 (20 систем) составляет 16.7 раза[1], в классе Top21–100 (80 систем) — 4.3 раза, в классе Top101–250 (150 систем) — всего 2.0 раза, в классе Top251–500 (250 систем) — 1.4 раза.

Таким образом, системы из Top1–20 (и особенно — системы Top1–10) радикально отличаются от других, а в классах Top21–100, Top101–250 и Top251–500 расположены системы, не принципиально отличающиеся друг от друга по производительности.

Таблица 5. Разница в Linpack-производительности между суперкомпьютерами разных уровней — по данным редакции Top500 за ноябрь 2012 года

Места в Top500	Linpack-производительность, max–min (Tflops)	Разница Linpack-производительности, max/min (разы)	Разница Linpack-производительности от Top1 (разы)
Top1–10	17 590–1 515	11.6	1–12
Top11–20	1 359–1 050	1.3	13–17
Top21–100	1 043–244	4.3	17–72
Top101–250	240–111	2.2	73–159
Top251–500	111–76	1.4	159–230

Наглядно оценить резкое расслоение суперкомпьютерной отрасли сегодня позволяют график функции $f(i) = \frac{RMax(40,i)}{RMax(40,1)}$, где $i \in [1..500]$ (Рис. 2), а так же график функции $g(n) = \frac{\sum_{i\in[1..n]} RMax(40,i)}{\sum_{i\in[1..500]} RMax(40,i)}$, где $n \in [1..500]$ (Рис. 3), иллюстрирующий, какую долю суммарной Linpack-

[1] Top1–10 — 11.6 раза, Top11–20 — 1.3 раза.

производительности всего списка Top500 обеспечивают первые n систем из списка. Видно, что глубокое расслоение суперкомпьютерной отрасли обеспечивает почти точное выполнение принципа Вильфредо Парето[1].

Подчеркнем, что анализируя приведенные данные (Таблица 5, Рис. 2, Рис. 3), уместно помнить, что отличия (сильные или слабые) суперкомпьютеров по Linpack-производительности влекут подобные же (сильные или слабые) отличия по цене, технической сложности, объему оборудования в различных подсистемах суперкомпьютеров.

Например, скорее всего (Рис. 3) суммарная стоимость первых 50 суперкомпьютеров в Top500 примерно равна суммарной стоимости остальных 450 суперкомпьютеров.

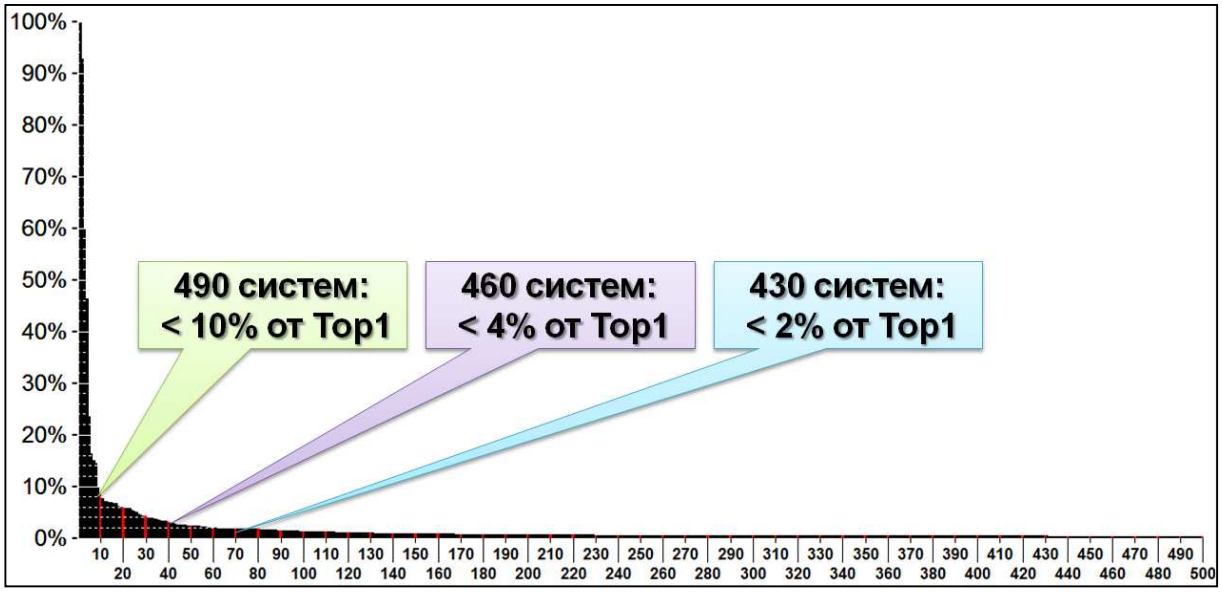


Рис. 2. Относительная Linpack-производительность в Top500 i-той системы, $\boldsymbol{i} \in [\mathbf{1}..\mathbf{500}]$ — за 100% принята Linpack-производительность Top1. По данным Top500 за ноябрь 2012 года

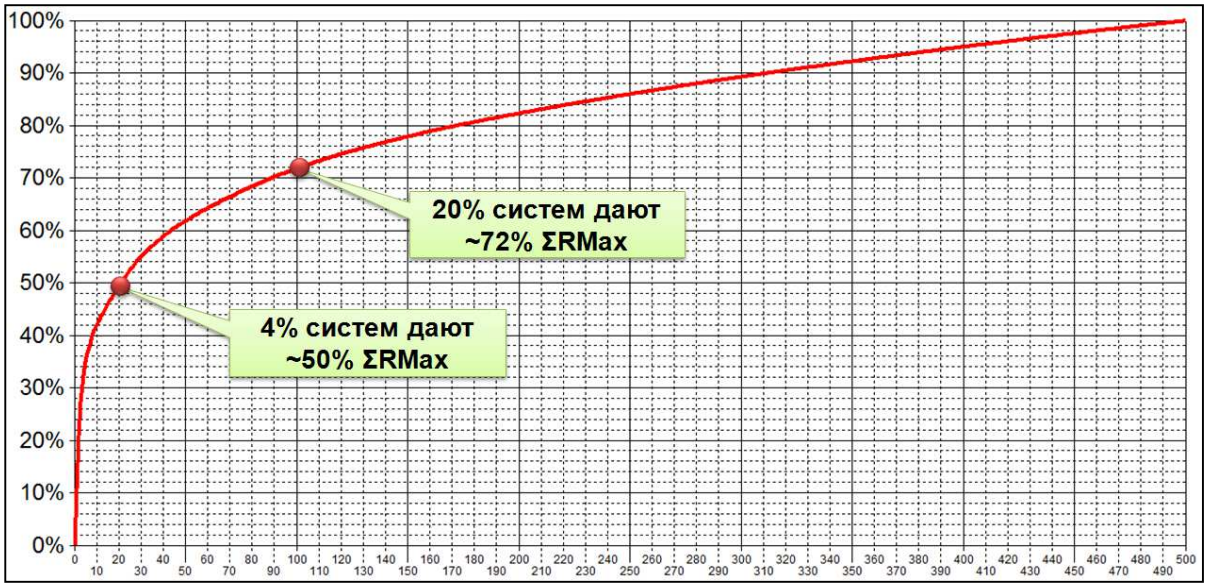


Рис. 3. Какую долю суммарной Linpack-производительности всего списка Top500 обеспечивают первые $\boldsymbol{n} \in [\mathbf{1}..\mathbf{500}]$ систем из списка. По данным Top500 за ноябрь 2012 года

[1] Принцип Вильфредо Парето часто формулируют так: «20% усилий дают 80% результата».

5. Восстановление истины: сегменты применения суперкомпьютеров

Выведя и обосновав (раздел 3) методику вычисления истинных долей различных категорий, обсудив (раздел 4) причины серьезного различия истинных долей от долей, рассчитанных «в штуках», далее, в этом и последующих разделах мы проведем исследование долей по различным категориям. Исследования будут выполняться при помощи программы Top500 Analyzer [4].

5.1 Анализ сегментов применения суперкомпьютеров

Начнем с анализа сегментов применения суперкомпьютеров. Все суперкомпьютеры разбиваются по категориям, в зависимости от указанных значений поля «Segment» в Top500. В данном поле всегда указывают одно из шести значений — «Research», «Academic», «Vendor», «Industry», «Government», «Classified». Соответственно получаем шесть категорий суперкомпьютеров. Диаграмма (Рис. 1) долей «в штуках» этих категорий входит в официальный плакат рейтинга Top500, выпущенного в ноябре 2012 года. Ниже (Рис. 4) показаны для сравнения диаграммы, построенные программой Top500 Analyzer. Левая часть рисунка — доли «в штуках»,— в точности совпадает с диаграммой с официального плаката. Правая диаграмма показывает истинные доли сегментов применения суперкомпьютеров.

Сравнивая левую и правую части рисунка, мы видим, что в левой части доля категории «Research» существенно занижалась в каждом выпуске рейтинга, а доля категории «Industry» — существенно преувеличивалась.

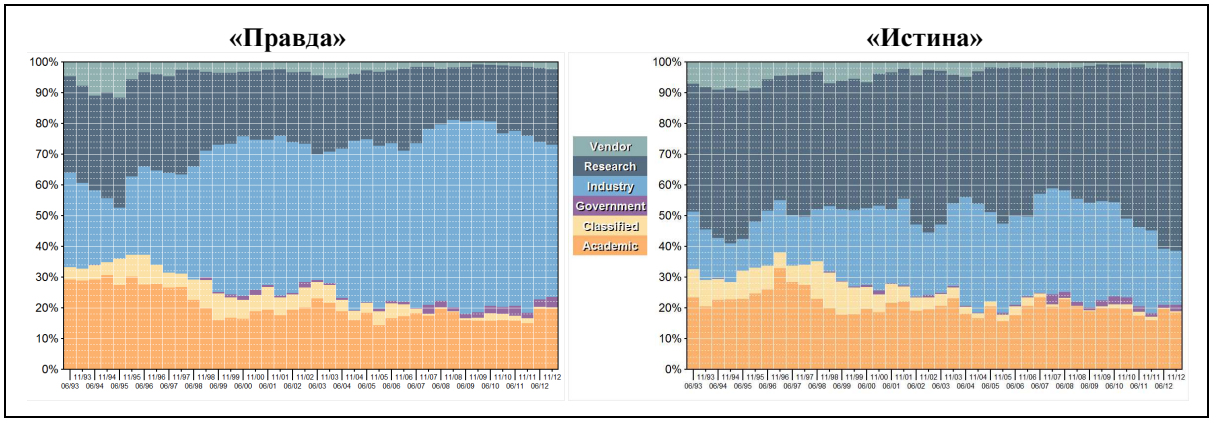


Рис. 4. Доли сегментов применения суперкомпьютеров по данным всех 40 редакций рейтинга Top500 (от июня 1993 года по ноябрь 2012 года). Слева: доли «в штуках» (доли от общего числа суперкомпьютеров), справа истинные доли (доли в Linpack-производительности)

5.2 Анализ областей использования суперкомпьютеров

Понятие «сегмент применения суперкомпьютеров» определяется напрямую значением одного поля «Segment» в рейтинге Top500. В программе Top500 Analyzer кроме этого поддерживается понятие «область применения суперкомпьютера», которое определяется за счет анализа двух полей: «Segment» и «Application Area». В результате все суперкомпьютеры программой Top500 Analyzer относятся в одну из 4 категорий: «RnD»[1] — использование для фундаментальных исследований и НИОКР; «Industry» — использование в промышленности и в других областях реальной экономики (например, в индустрии развлечений и т. п.); «Gov.Mil» — использование для государственных и военных нужд; «Unknown» — недостаточно информации для отнесения в категории «RnD», «Industry» или «Gov.Mil».

[1] От английского: Research and Development — исследования и разработки (НИР и ОКР).

При помощи программы Top500 Analyzer построены диаграммы (Рис. 5) данных категорий. Левая часть рисунка — доли областей использования суперкомпьютеров «в штуках», правая диаграмма — истинные доли областей использования суперкомпьютеров.

Сравнивая левую и правую части рисунка, легко заметить, что в левой части доля категории «RnD» существенно занижалась в каждый момент времени, а доля категории «Industry» — существенно преувеличивалась. Более того, правая часть рисунка явно выявляет тенденцию последних лет на сокращение истинной доли индустриального использования суперкомпьютеров.

При помощи программы Top500 Analyzer для редакции Top500 за ноябрь 2012 года построим распределение (Рис. 6) областей использования суперкомпьютеров по пяти уровням суперкомпьютеров: Top1–10, Top11–20, Top21–100, Top101–250 и Top251–500. Поясним структуру этой диаграммы. Левые пять столбцов иллюстрируют истинные доли (доли суммарной Linpack-производительности — RMax) областей использования для пяти отдельных уровней суперкомпьютеров: от Top1–10 до Top251–500. Предпоследний столбец на Рис. 6 иллюстрируют истинные доли областей использования для всего списка Top500 (ноябрь 2012 года) — что, по сути, совпадает с правым столбцом в правой части Рис. 5. Последний столбец на Рис. 6 иллюстрируют доли «в штуках» областей использования для всего списка Top500 (ноябрь 2012 года) — что, по сути, совпадает с правым столбцом в левой части Рис. 5.

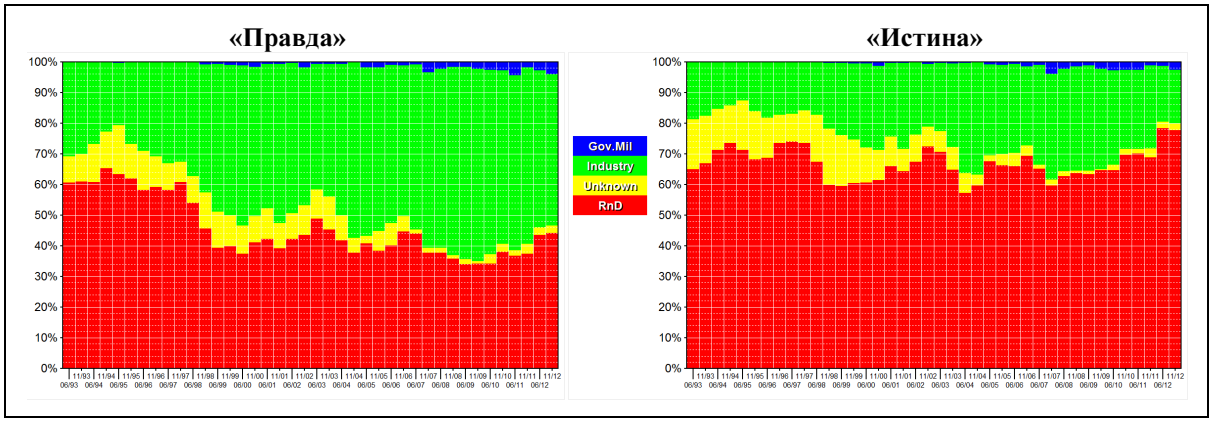


Рис. 5. Доли областей использования суперкомпьютеров по данным всех 40 редакций рейтинга Top500 (от июня 1993 года по ноябрь 2012 года). Слева: доли «в штуках» (доли от общего числа суперкомпьютеров), справа истинные доли (доли в Linpack-производительности)

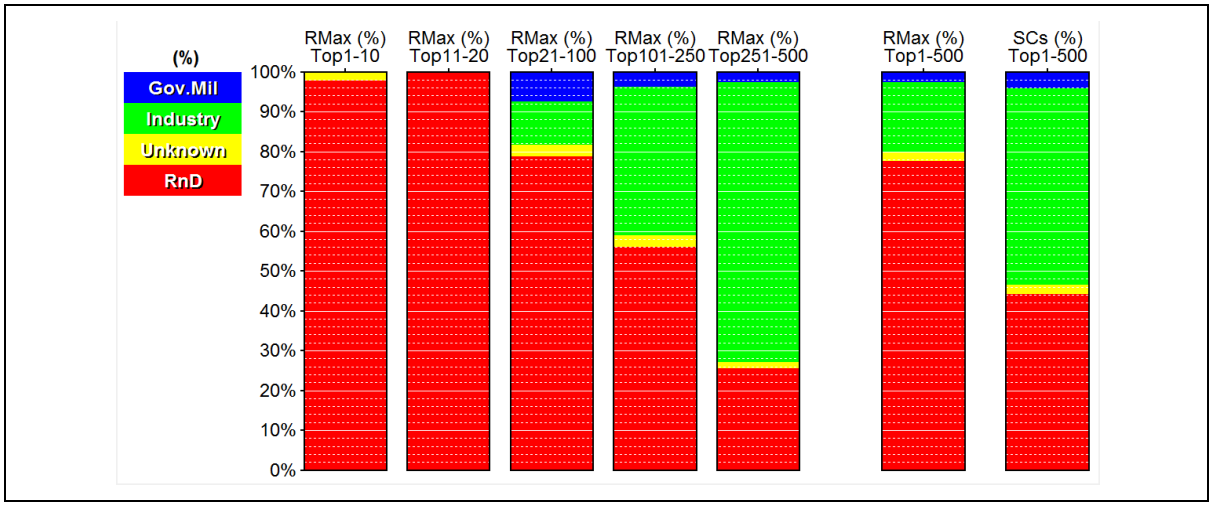


Рис. 6. Распределение областей использования суперкомпьютеров по уровням Top1–10, Top11–20, Top21–100, Top101–250, Top251–500. По данным Top500 за ноябрь 2012 года

Распределение по уровням (см. левые пять столбцов на Рис. 6) позволяет понять резкое отличие «истины» и «правды» (см. два правых столбца на Рис. 6) за счет явного изображения «ареалов обитания» каждой категории на различных уровнях суперкомпьютеров. Так, видно, что в индустрии совсем не применяются суперкомпьютеры первого и второго уровня, использование систем третьего и четвертого уровня незначительно, и только самые слабые и самые многочисленные системы — пятый уровень, 250 систем с производительностью в 150–230 раз хуже, чем у Top1,— характерны для категории «Industry».

6. Восстановление истины: типы микропроцессоров, используемых в суперкомпьютерах

На плакатах рейтинга Top500 кроме диаграммы «Installation Type» (доли сегментов применения) традиционно размещают диаграмму «Chip Technology» — диаграмму долей различных технологий процессоров, используемых в суперкомпьютерах. Данная диаграмма на плакате за ноябрь 2012 года (Рис. 7) состоит из 40 столбцов — каждый столбец соответствует одному выпуску рейтинга, два столбца (июнь и ноябрь) приходятся на один год. Каждый столбец состоит из частей разных цветов; размер частей определяется долями различных технологий процессоров из соответствующего рейтинга Top500. Различают восемь значений (категорий) для обозначения технологий процессоров: «Alpha», «IBM», «HP», «Intel», «MIPS», «SPARC», «AMD» и «Proprietary».

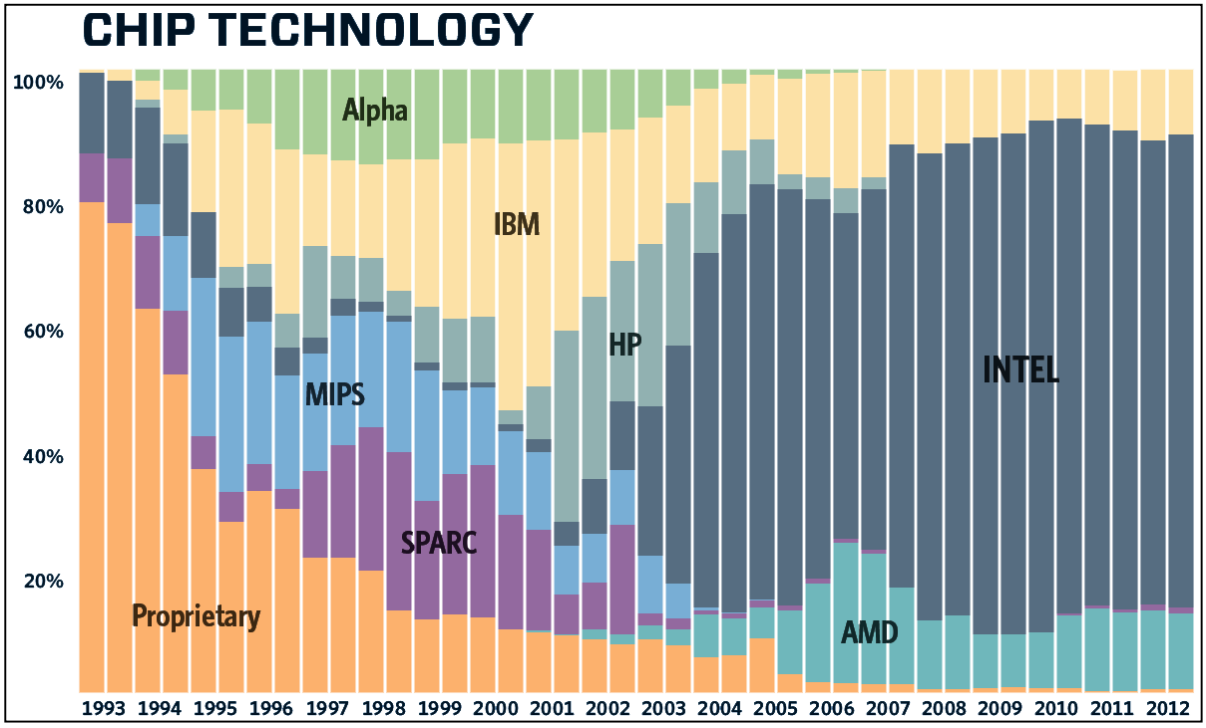


Рис. 7. Диаграмма «Chip Technology» с плаката рейтинга Top500, выпущенного в ноябре 2012 года

Рассматривая эту диаграмму, легко сделать весьма ошибочные суждения, например:

§7 К ноябрю 2012 года в суперкомпьютерах Top500 на процессоры Intel приходится подавляющая доля (76%). Отрыв от ближайших преследователей весьма значительный: почти в 7 раз от IBM (11%) и почти в 6 раз от AMD (13%).

Для выявления истинного положения с помощью программы Top500 Analyzer построим диаграммы долей различных технологий процессоров, используемых в суперкомпьютерах (Рис. 8). Для вычисления технологии процессора для каждой записи в Top500 анализируется 4 поля. «Processor», «Processor Family», «Processor Generation», «Processor Technology». Левая часть рисунка — доли «в штуках»,— в точности совпадает с диаграммой «Installation Type» с

официального плаката. Правая диаграмма показывает истинные доли различных технологий процессоров, используемых в суперкомпьютерах.

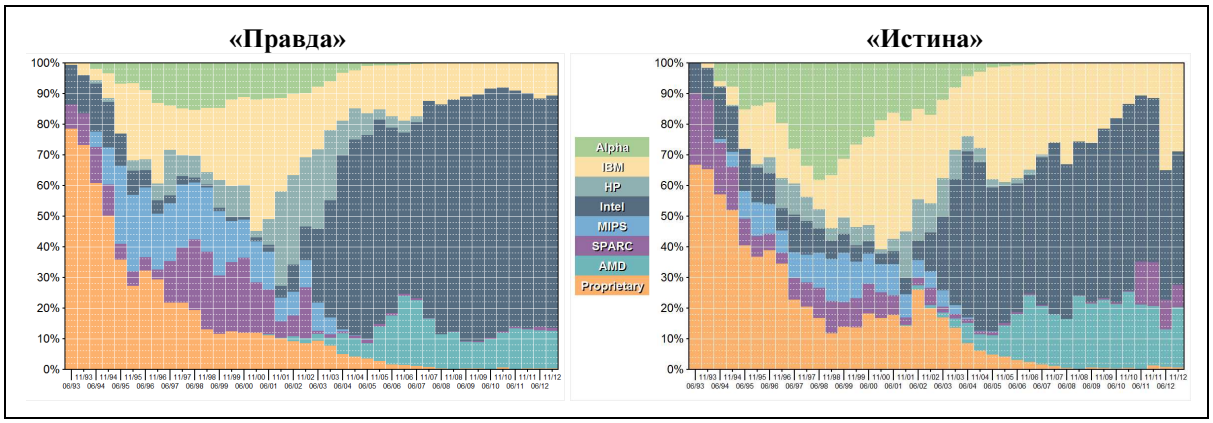


Рис. 8. Доли различных технологий процессоров, используемых в суперкомпьютерах по данным всех 40 редакций рейтинга Top500 (от июня 1993 года по ноябрь 2012 года). Слева доли «в штуках» (доли от общего числа суперкомпьютеров), справа истинные доли (доли в Linpack-производительности)

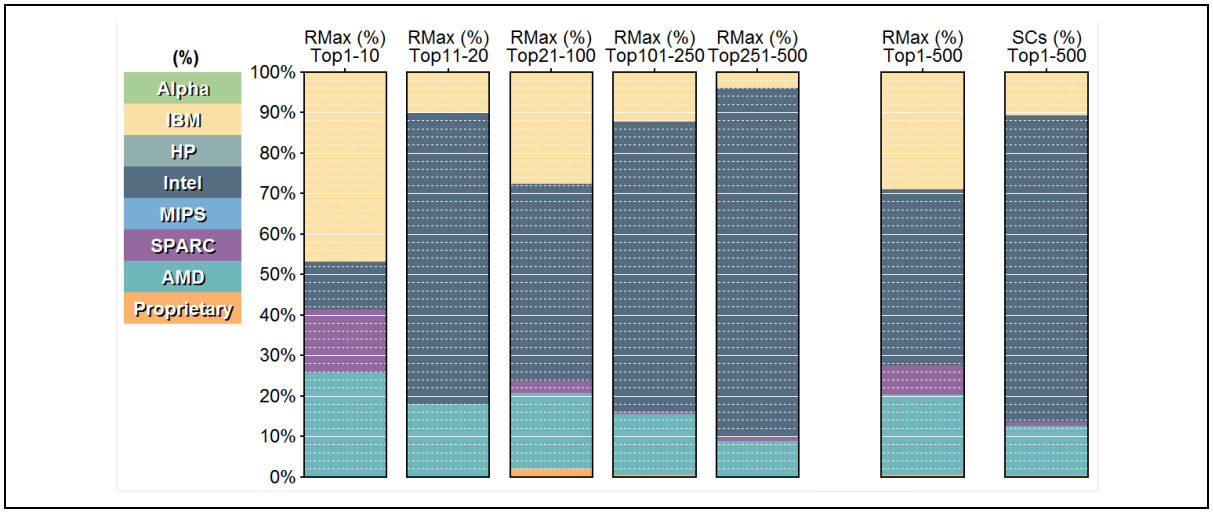


Рис. 9. Распределение технологий процессоров по уровням Top1–10, Top11–20, Top21–100, Top101–250, Top251–500. По данным Top500 за ноябрь 2012 года

Налицо явная и весомая разница между «правдой» и «истиной». Разберемся с этой разницей на примере редакции Top500 за ноябрь 2012 года. При помощи программы Top500 Analyzer построим распределение (Рис. 9) технологий процессоров по пяти уровням суперкомпьютеров: Top1–10, Top11–20, Top21–100, Top101–250 и Top251–500.

Видно, что для категории «Intel» ареал распространения в левых пяти колонках напоминает треугольник, с вершиной слева и с основанием — справа. То есть, процессоры Intel тем лучше представлены в суперкомпьютерах, чем к более слабому уровню они относятся — где суперкомпьютеров по количеству много, а по производительности они слабые. Для категорий «IBM», «AMD» и «SPARC» ареалы распространения в левых пяти колонках смещены к старшим уровням суперкомпьютеров — где суперкомпьютеров по количеству мало, а по производительности они мощные. В результате на официальном плакате истинные доли категорий «IBM», «AMD» и «SPARC» сильно приуменьшены, а доля категории «Intel» серьезно преувеличена. Истинное суждение (поправляющее заблуждение §7) будет таким:

§8 *К ноябрю 2012 года в суперкомпьютерах Top500 на процессоры Intel приходится значительная доля (44%). Однако, отрыв от ближайших преследователей не такой уж и большой: в 1.5 раз от IBM (29%), в 2.2 раза от AMD (20%). Заметная доля (7%) приходится на процессоры SPARC.*

7. Восстановление истины: компании-производители суперкомпьютеров

Для суперкомпьютеров, вошедших в Top500, проанализируем показатель «компания-производитель»; соответствующее поле в записях Top500 — «Manufacturer». Безусловные лидерские позиции здесь принадлежат трем компаниям, поэтому в программе «Top500 Analyzer» введем четыре категории[1]: «Cray» — суперкомпьютер изготовлен компанией Cray Inc.; «IBM» — суперкомпьютер изготовлен компанией IBM; «HP» — суперкомпьютер изготовлен компанией Hewlett-Packard, «Other» — суперкомпьютер изготовлен любой иной компанией.

С помощью программы Top500 Analyzer построим диаграммы долей компаний-производителей (Рис. 10). Как обычно, левая часть рисунка — доли «в штуках», правая диаграмма показывает истинные доли компаний-производителей. Опять налицо явная и серьезная разница между «правдой» и «истиной». Среди прочего видно, что в последнее пятилетие истинная доля категории «HP» существенно (в разы) преувеличивается, а доля категории «Cray» существенно преуменьшается. Основываясь на вычислении долей «в штуках» за ноябрь 2008 года можно сделать следующее утверждение, которое, несомненно, является правдой:

§9 По данным редакции Top500 за ноябрь 2008 года компания Hewlett-Packard поставила больше всех других компаний суперкомпьютеров (42%), вошедших в данную редакцию Top500. Ближайшие конкуренты: IBM (37%, отставание в 1.13 раза) и Cray (5%, отставание в 8.4 раза). Все остальные компании-производители, вместе взятые, серьезно уступают лидеру (16%, отставание в 2.6 раза).

Истинное положение дел в ноябре 2008 года серьезно отличается от утверждения §9:

§10 По данным редакции Top500 за ноябрь 2008 года суперкомпьютеры компании IBM обеспечили 38% всей суммарной Linpack-производительности списка Top500. Это серьезно превышает доли ближайших конкурентов. Так, доля суперкомпьютеров компании Hewlett-Packard — 25% (отставание в 1.5 раза), компании Cray — 15%, всех остальных компаний-производителей, вместе взятых—22%.

Сравнивая утверждения §9 и §10, отметим, что в утверждении §9 истинная доля категории «HP» была серьезно (в 1.76 раз) преувеличена, истинная доля категории «Cray» была серьезно (в 3 раза) преуменьшена, истинная доля категории «Others» была преуменьшена в 1.4 раза. Кроме того, был просто неверно указан лидер отрасли.

Уместно напомнить, что отличия суперкомпьютеров по Linpack-производительности влекут подобные же отличия по цене, технической сложности, объему оборудования в различных подсистемах суперкомпьютеров. Тем самым, утверждение §10 дает лучшее представление о распределении между компаниями долей (в денежном исчислении) рынка суперкомпьютеров. Именно такая информация важна для потенциальных инвесторов.

Подобные (§9) мнимые признаки абсолютного лидерства в принципе дают компании аргументы для настойчивого продвижения своих решений, даже в тех сегментах, где, на самом деле, позиции компании весьма слабы. Например, это позволяет всерьез обращаться к лицам, принимающим решения, с предложением построить для России суперкомпьютер высшей производительности (Top1–5), аргументируя данное предложение своим лидерством в суперкомпьютерной отрасли. Для правильной оценки подобных предложений важно знать истинные позиции той или иной компании, причем на различных уровнях суперкомпьютерной отрасли.

На примере редакции Top500 за ноябрь 2012 года разберем распределение (Рис. 11) компаний-производителей по пяти уровням суперкомпьютеров: Top1–10, Top11–20, Top21–100, Top101–250 и Top251–500.

[1] Заинтересованный читатель-программист легко может в программе To500 Analyzer [4] изменить эти установки.

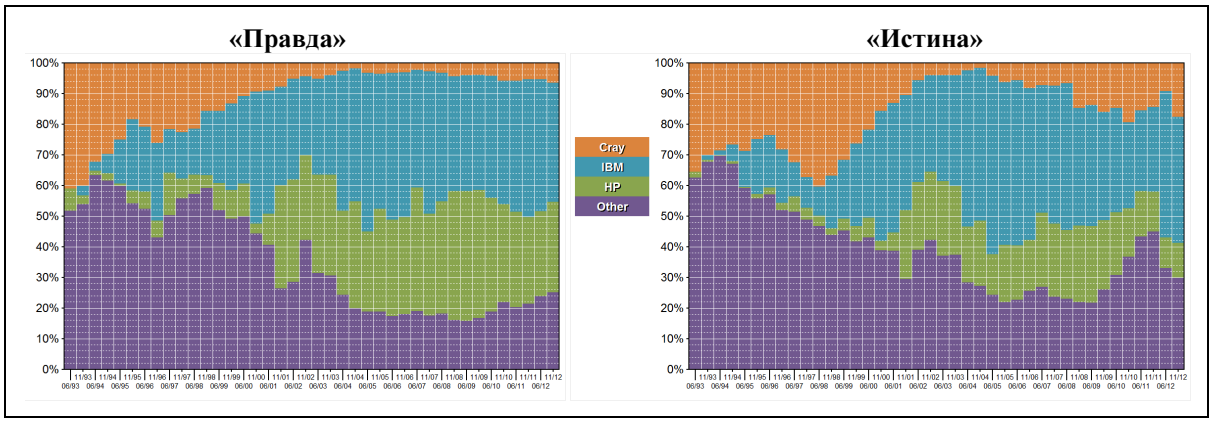


Рис. 10. Доли компаний-производителей суперкомпьютеров по данным всех 40 редакций рейтинга Top500 (от июня 1993 года по ноябрь 2012 года). Слева доли «в штуках» (доли от общего числа суперкомпьютеров), справа истинные доли (доли в Linpack-производительности)

Видно, что для категории «HP» ареал распространения в левых пяти колонках напоминает треугольник, с вершиной в третьем уровне и с основанием в пятом. Суперкомпьютеры Hewlett-Packard вообще отсутствуют в высших двух уровнях (Top1–10, Top11–20), слабо представлены на третьем уровне (Top21–100), заметно присутствуют на 4–5 уровнях (Top101–500) — там, где суперкомпьютеров по количеству много, а по производительности они слабые.

Для категорий «Others» и особенно «Cray» ареалы распространения смещены к старшим уровням суперкомпьютеров — где суперкомпьютеров по количеству мало, а по производительности они мощные. Для категории «IBM» ареал распространения занимает сравнимые доли на всех пяти уровнях.

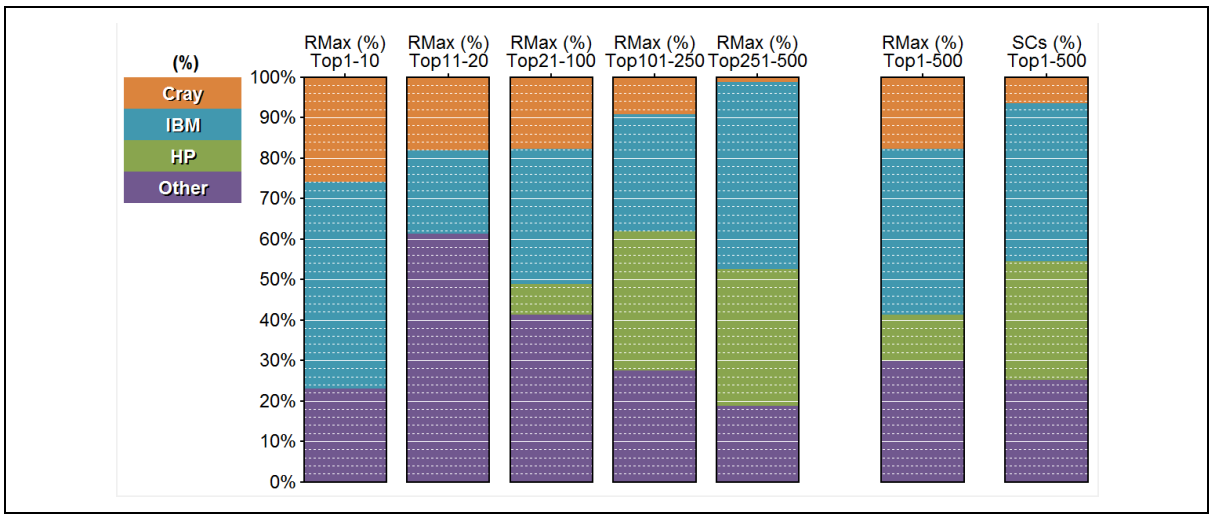


Рис. 11. Распределение долей компаний-производителей суперкомпьютеров по уровням Top1–10, Top11–20, Top21–100, Top101–250, Top251–500. По данным Top500 за ноябрь 2012 года

8. Восстановление истины: технологии интерконнекта

Для суперкомпьютеров, вошедших в Top500, проанализируем показатель «используемая технология интерконнекта». Для этого следует анализировать два поля: «Interconnect» и «Interconnect Family». В программе Top500 Analyzer введем шесть категорий для обозначения технологии интерконнекта.

Пять категорий из шести явно указывают используемую сетевую технологию: «Infiniband», «Ethernet», «Myrinet», «SCI» и «Quadrics». Все эти пять сетевых технологий являются коммерчески доступными: любой разработчик суперкомпьютеров может[1] приобрести отдельно

[1] В период производства соответствующих изделий.

соответствующие сетевые изделия — сетевые адаптеры, коммутаторы, кабели или даже микросхемы для адаптеров и коммутаторов,— и на этой базе разрабатывать свои собственные суперкомпьютеры.

Шестая категория — «Custom»,— объединяет сетевые технологии, которые коммерчески недоступны, как отдельные сетевые решения. Поясним: можно купить целиком суперкомпьютер IBM Blue Gene, но невозможно купить отдельно интерконнект, который используется в суперкомпьютерах IBM Blue Gene, и на базе такого интерконнекта разработать свой собственный суперкомпьютер. По факту, в категорию «Custom» попадают различные решения, которые, по сравнению с остальными, имеют более высокие технические показатели и расширенные функциональные возможности. При этом, данные технологии невозможно купить отдельно. Значит, если будет стоять задача разработки российского суперкомпьютера с подобным интерконнектом, то подобный интерконнект (аналог) придется разрабатывать самостоятельно, из-за невозможности покупки готового решения. При принятии решения о такой разработке, естественно, встает вопрос:

§11 Надо ли тратить ресурсы на разработку российской технологии интерконнекта, подобной технологиям, представленным в категории «Custom»? Или коммерчески доступных технологий интерконнекта вполне достаточно для создания всех необходимых отечественных суперкомпьютеров?

Давайте разберемся. С помощью программы Top500 Analyzer построим диаграммы долей технологий интерконнекта (Рис. 12). Как обычно, левая часть рисунка — доли «в штуках», правая диаграмма показывает истинные доли технологий интерконнекта.

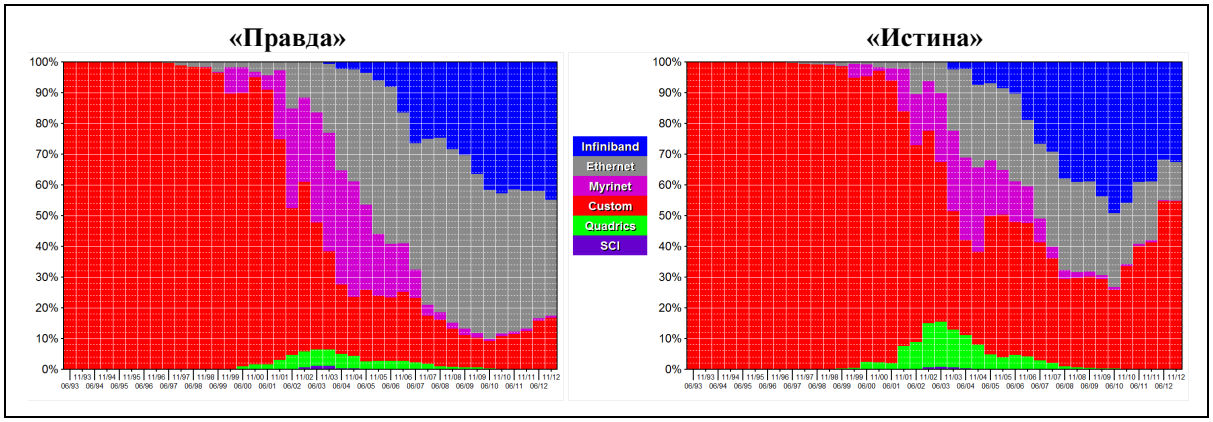


Рис. 12. Доли технологий интерконнекта в суперкомпьютерах по данным всех 40 редакций рейтинга Top500 (от июня 1993 года по ноябрь 2012 года). Слева доли «в штуках» (доли от общего числа суперкомпьютеров), справа истинные доли (доли в Linpack-производительности)

Опять имеем явную и серьезную разницу между «правдой» и «истиной». Среди прочего видно, что (на отрезке за последние годы) на левой диаграмме истинная доля категории «Ethernet» существенно (в разы) преувеличивается, а доля категории «Custom» существенно преуменьшается. Основываясь на вычислении долей «в штуках», можно сделать следующие утверждения, которые, несомненно, являются правдой:

§12 По данным редакции Top500 за ноябрь 2012 года технологии Infiniband и Ethernet использовались в большинстве суперкомпьютеров, вошедших в данную редакцию списка (45%+38%=83%). Доли категорий «Custom» (16.7%) и Myrinet (0.3%) незначительны.

§13 Немногим ранее ситуация была еще радикальнее. По данным редакции Top500 за июнь 2010 года технологии Infiniband и Ethernet использовались в подавляющем большинстве суперкомпьютеров, вошедших в данную редакцию списка (41%+49%=90%). Доли категорий «Custom» (9%), Myrinet (0.5%) и Quadrics Myrinet (0.5%) — незначительны.

На базе утверждений §12 и §13 легко принять глубоко ошибочное решение по вопросу §11:

§14 Нецелесообразно тратить ресурсы на разработку российской технологии интерконнекта, подобной технологиям, представленным в категории «Custom». При

разработке отечественных суперкомпьютеров вполне можно обойтись коммерчески доступными решениями Ethernet и Infiniband.

Истинное положение дел (и в ноябре 2012 года, и в июне 2010 года) серьезно — многократно,— отличается от утверждений §12 и §13. На это уже указывалось выше (Таблица 3). При этом не только многократно искажены доли технологий интерконнекта, но и неверно указана лидирующая категория: истинный и абсолютный лидер по данным Top500 за ноябрь 2012 года — категория «Custom» (55%). Еще раз упомянем: доля суперкомпьютера по Linpack-производительности коррелирует с технической сложностью, объемом оборудования в различных подсистемах суперкомпьютера (например, с числом портов интерконнекта).

Для правильной оценки роли той или иной технологии интерконнекта важно знать и распределение долей по уровням суперкомпьютерной отрасли. На примере редакции Top500 за ноябрь 2012 года разберемся с этим распределением (Рис. 13). Видно, что для категории «Ethernet» ареал распространения напоминает треугольник, с вершиной на третьем уровне и с основанием на пятом. Суперкомпьютеры в интерконнектом на базе Ethernet вообще отсутствуют в высших двух уровнях (Top1–10, Top11–20), слабо представлены на третьем уровне (Top21–100), заметно присутствуют на 4–5 уровнях (Top101–500) — где суперкомпьютеров по количеству много, а по производительности они слабые. Ареалы распространения категории «Infiniband» можно описать так: очень малое присутствие (менее 10%) на первом уровне, значительное присутствие (70%–50%–50%–40%) на втором–пятом уровнях. У категории «Custom» ареал распространения смещен к старшим уровням суперкомпьютеров — где суперкомпьютеров по количеству мало, а по производительности они мощные. Именно на данных решениях и строятся рекордные установки, обладание которыми стратегически важно для России.

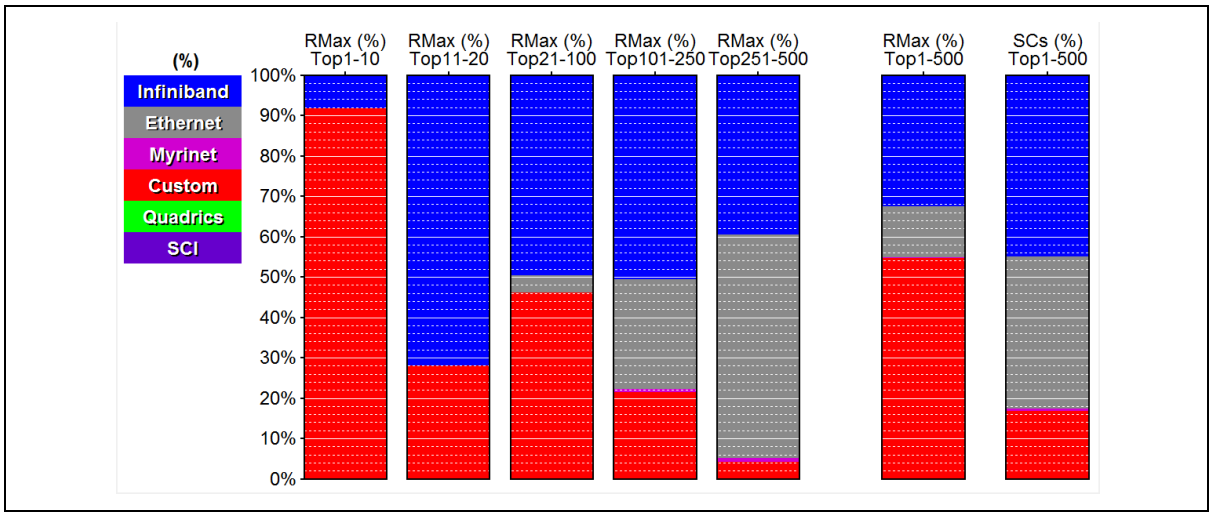


Рис. 13. Распределение долей технологий интерконнекта в суперкомпьютерах по уровням Top1–10, Top11–20, Top21–100, Top101–250, Top251–500. По данным Top500 за ноябрь 2012 года

Тем самым, обоснованное решение по вопросу §11 будет таким:

§15 Технологи категории «Custom» обеспечивают подавляющую долю Linpack-производительности (55% по данным редакции Top500 от ноября 2012 года). А если говорить про самые мощные суперкомпьютеры — Top1–10 и Top10–20,— про суперкомпьютеры, которые России вряд ли будут проданы и которые предстоит построить самостоятельно, то эти системы практически всегда строятся на технологиях категории «Custom». При этом, технологии категории «Custom» не продаются как отдельные продукты. Таким образом, в России, безусловно, необходимо проводить разработку собственных технологий интерконнекта из категории «Custom».

9. Заключение

При написании данной статьи целью было продемонстрировать читателю:

- насколько традиционный и широко распространенный способ анализа Top500 — подсчет долей в количестве суперкомпьютеров,— искажает истинное положение дел в суперкомпьютерной отрасли (разделы 1, 5–8);
- как важно разработать и грамотно применять правильные методики анализа данных Top500 (раздел 3);
- как легко на базе неверного поверхностного анализа совершаются ошибки в управленческих решениях с серьезными последствиями;
- насколько сильно сегодня расслоение (по реальной производительности) в суперкомпьютерном мире (раздел 4) — кажется, пока еще даже профессионалы (чисто психологически) не всегда в полной мере осознают всю глубину данного расслоения;
- насколько важно для профессионального анализа Top500 обладать правильно построенным инструментарием.

Хочется надеться, что эти цели хотя бы частично были достигнуты. Конечно, формат статьи не позволяет продемонстрировать все возможности программы Top500 Analyzer. Так, распределений долей категорий (например, Рис. 13) по уровням (Top1–10, Top11–20, Top21–100, Top101–250, Top251–500) в программе можно просмотреть за каждую редакцию Top500. И можно это сделать в режиме анимации: один год (две редакции Top500) — за секунду. При таком просмотре можно разглядеть эпохи появления, расцвета и угасания той или иной категории, рассмотреть, как разные категории конкурируют за доли на том или ином уровне, как происходит их миграция с уровня на уровень.

Что касается дальнейшего развития работ, то есть планы по ряду улучшений в программе Top500 Analyzer. Будет хорошо, если найдутся коллеги, которые помогут в этом — советом или делом.

Конечно, было бы интересно применить методику анализа и программу Top500 Analyzer к рейтингу Graph500 [2] и к национальному рейтингу 50 самых мощных систем в СНГ [8]. Однако в последнем случае серьезными препятствиями являются:

- невозможность выгрузки редакций этого рейтинга в виде Excel-таблицы или в виде иного файла, с возможностью (приемлемого по сложности) разбора рейтинга по записям и по полям;
- предположительно малое число формализованных полей в записях рейтинга.

Завершая, хочется поблагодарить сотрудников ИПС имени А.К.Айламазяна РАН — Е.П. Лилитко и М.Г. Химшиашвили,— которые помогали автору при создании данной статьи.

Литература

1. TOP500 Supercomputer Sites — мировой рейтинг пятисот самых производительных (на тесте Linpack) вычислительных машин мира // Электронный ресурс в сети Интернет: http://www.top500.org/
2. Graph500 —мировой рейтинг самых производительных (на задаче поиска в большом графе в ширину) вычислительных машин мира // Электронный ресурс в сети Интернет: http://www.graph500.org
3. Википедия — свободная энциклопедия, которую может редактировать каждый // Электронный ресурс в сети Интернет, русскоязычный вариант — http://ru.wikipedia.org и англоязычный вариант — http://en.wikipedia.org.
4. Абрамов С.М. Top 500 Analyzer — программа для анализа данных рейтинга Top500 // Электронный ресурс в сети Интернет http://skif.pereslavl.ru/psi-info/rcms-skif/top500analyzer
5. Воейков Д. Рейтинг Top 500. Соревнование с гандикапом // PC Week/RE №27–28 (633–634) 22 июля–11 августа 2008, http://www.pcweek.ru/themes/detail.php?ID=112308

6. Абрамов С.М. Суперкомпьютерные технологии России: объективные потребности и реальные возможности // Журнал «CAD/cam/cae Observer» #2 (54), 2010, с. 1–11.
7. Абрамов С.М., Лилитко Е.П. Состояние и перспективы развития вычислительных систем сверхвысокой производительности // Труды Шестой Международной конференции «Параллельные вычисления и задачи управления». Москва, 24–26 октября 2012 г. (PACO-2012), Институт проблем управления им. В.А. Трапезникова РАН. Том 1, М.: ИПУ РАН, 2012, с. 10–32. ISBN 978-5-91450-122-5 (т. 1).
8. Top50 суперкомпьютеров — рейтинг 50 вычислительных систем, установленных на территории СНГ и показавших наибольшую производительность на тесте Linpack // Электронный ресурс в сети Интернет: http://top50.supercomputers.ru

Алгоритмы решения обратных задач гравиметрии о нахождении поверхностей раздела сред на многопроцессорных вычислительных системах*

Е.Н. Акимова[1,2], В.В. Васин[1,2], В.Е. Мисилов[1]

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

Для решения трехмерной структурной обратной задачи гравиметрии о нахождении поверхностей раздела в многослойной среде предложены новые линеаризованные итерационные методы градиентного типа (наискорейшего спуска и минимальной ошибки) с переменными демпфирующими множителями. На основе методов типа Ньютона, Левенберга–Марквардта и линеаризованных методов градиентного типа решения задач гравиметрии для двухслойной и трехслойной среды разработаны эффективные параллельные алгоритмы, численно реализованные на многопроцессорных системах различного типа: многопроцессорном комплексе МВС-ИММ, многоядерном процессоре Intel и графических процессорах NVIDIA, входящими в состав суперкомпьютера «Уран». Проведено исследование эффективности и оптимизация параллельных алгоритмов. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях».

1. Введение

Важнейшими задачами исследования структуры земной коры являются обратные задачи гравиметрии о восстановлении поверхностей раздела между средами в двухслойной и многослойной среде [1, 2]. Задачи гравиметрии описываются нелинейными интегральными уравнениями Фредгольма первого рода, т.е. являются существенно некорректными задачами. При разработке методов решения задач используются идеи итеративной регуляризации [3]. После дискретизации задачи сводятся к системам нелинейных уравнений большой размерности (до нескольких сотен тысяч). Необходимость повышения точности результатов решения задач, в частности, использование более мелких сеток, существенно увеличивает время вычислений.

Одним из путей уменьшения времени расчетов и повышения эффективности решения геофизических задач является распараллеливание алгоритмов и использование многопроцессорных вычислительных систем (МВС). В Институте математики и механики УрО РАН (г. Екатеринбург) установлены суперкомпьютеры МВС-1000, МВС-ИММ и «Уран», которые успешно используются при решении прикладных задач.

В настоящее время для решения прикладных задач активно используются многоядерные гибридные вычислительные системы с графическими процессорами (видеокартами), которые по сравнению с суперкомпьютерами представляют собой более дешевую многопроцессорную технику с низким энергопотреблением. Установленный в ИММ УрО РАН суперкомпьютер «Уран» включает в себя гибридный вычислительный кластер на основе видеоускорителей NVIDIA Tesla и многоядерных CPU.

В данной работе предложены новые линеаризованные итерационные методы градиентного типа (наискорейшего спуска и минимальной ошибки) с переменными демпфирующими множителями для решения трехмерной нелинейной обратной задачи гравиметрии о восстановлении поверхностей раздела в многослойной среде. На основе методов типа Ньютона, Левенберга–Марквардта и линеаризованных методов градиентного типа решения задач гравиметрии для двухслойной и трехслойной среды разработаны эффективные параллельные алгоритмы, численно реализованные на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA и многоядерном

* Работа выполнена при поддержке УрО РАН в рамках программ фундаментальных исследований Президиума РАН № 15 (проект 12-П-1-1023) и № 18 (проект 12-П-15-2019) и при поддержке РФФИ (проект 12-01-00105-а).

процессоре Intel. Проведено исследование эффективности и оптимизация параллельных алгоритмов. Решены задачи с модельными и реальными данными. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях».

2. Алгоритмы решения структурных обратных задач гравиметрии о нахождении границ раздела сред постоянной плотности

2.1 Обратная задача гравиметрии о нахождении поверхности раздела в двухслойной среде

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

Предполагается, что нижнее полупространство состоит из двух слоев постоянной плотности, разделенных искомой поверхностью S (рис. 1).

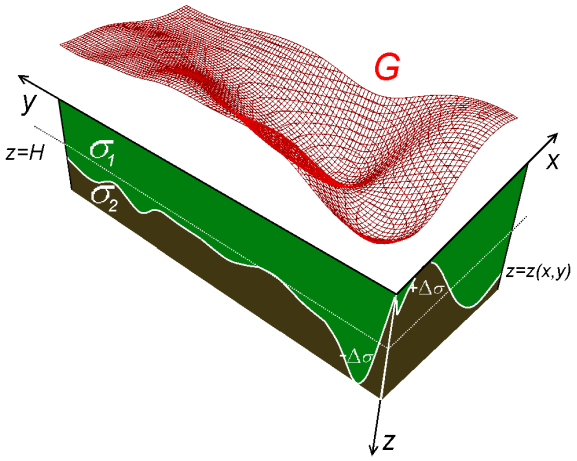


Рис. 1. Модель двухслойной среды

В предположении, что гравитационная аномалия создана отклонением искомой поверхности S от горизонтальной плоскости $z = H$ (ось z направлена вниз), в декартовой системе координат функция $z = z(x, y)$, описывающая искомую поверхность раздела, удовлетворяет нелинейному двумерному интегральному уравнению Фредгольма первого рода

$$A[z] \equiv f\Delta\sigma \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \left\{ \frac{1}{\left[(x-x')^2 + (y-y')^2 + z^2(x',y')\right]^{1/2}} - \frac{1}{\left[(x-x')^2 + (y-y')^2 + H^2\right]^{1/2}} \right\} dx'dy' = G(x,y), \tag{1}$$

где f – гравитационная постоянная, $\Delta\sigma$ – скачок плотности на границе раздела сред, $G(x,y)$ – аномальное гравитационное поле, $z = H$ – асимптотическая плоскость для данной границы раздела, т.е. $\lim\limits_{\substack{|x|\to\infty \\ |y|\to\infty}} |z(x,y) - H| = 0.$

Предварительная обработка гравитационных данных, связанная с выделением аномального поля из общих гравитационных данных, выполняется по методике, предложенной в работе [4].

Обратная задача гравиметрии является существенно некорректной задачей, решение которой обладает сильной чувствительностью к погрешности правой части, полученной в результате измерений и предварительной обработки геофизических данных. Поэтому при ее решении используются методы итеративной регуляризации.

После дискретизации уравнения (1) на сетке $n = M \times N$, где задана $G(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам имеем систему нелинейных уравнений

$$A_n[z] = F_n. \tag{2}$$

2.2 Методы решения обратной задачи гравиметрии для двухслойной среды

Для решения системы нелинейных уравнений (2) используется методы типа Ньютона и Гаусса-Ньютона в регуляризованном варианте:

итеративно регуляризованный метод Ньютона [5]

$$z^{k+1} = z^k - \left[A_n'\left(z^k\right) + \alpha_k I \right]^{-1} \left[A_n\left(z^k\right) + \alpha_k z^k - F_n \right]; \tag{3}$$

модифицированный метод Ньютона

$$z^{k+1} = z^k - \left[A_n'\left(z^0\right) + \alpha_k I \right]^{-1} \left[A_n\left(z^k\right) + \alpha_k z^k - F_n \right]; \tag{4}$$

метод Левенберга–Марквардта (МЛМ) [3, 6]

$$z^{k+1} = z^k - \left[\left(A_n'\left(z^k\right)\right)^T A_n'\left(z^k\right) + \alpha_k I \right]^{-1} \left(A_n'\left(z^k\right)\right)^T \left(A_n\left(z^k\right) - F_n\right); \tag{5}$$

либо модифицированный метод Левенберга–Марквардта (ММЛМ) [7]

$$z^{k+1} = z^k - \left[\left(A_n'\left(z^0\right)\right)^T A_n'\left(z^0\right) + \alpha_k I \right]^{-1} \left(A_n'\left(z^k\right)\right)^T \left(A_n\left(z^k\right) - F_n\right). \tag{6}$$

Здесь $A_n(z^k)$ и F_n – конечномерные аппроксимации интегрального оператора и правой части в уравнении (1), $A_n'(z^k)$ – производная оператора A в точке z^k, I – единичный оператор, α_k – последовательность положительных параметров регуляризации, k – номер итерации.

Нахождение очередного приближения z^{k+1} метода Ньютона (3) либо его модифицированного варианта (4) сводится к решению СЛАУ

$$A_n{}^k z^{k+1} = F_n{}^k, \tag{7}$$

где $A_n{}^k = A_n'\left(z^k\right) + \alpha_k I$ – плохо обусловленная несимметричная заполненная $n \times n$ матрица для метода (3) либо симметричная $n \times n$ матрица $A_n{}^k = A_n'\left(z^0\right) + \alpha_k I$ для метода (4). Вектор $F_n{}^k$ размерности n имеет вид $F_n{}^k = A_n{}^k z^k - \left(A_n\left(z^k\right) + \alpha_k z^k - F_n\right)$.

Заметим, что в общем случае решения задачи методом Ньютона предварительно СЛАУ (7) приводится к виду с симметричной матрицей

$$D^k z^{k+1} \equiv \left[(A_n^k)^T A_n^k + \alpha'_k I \right] z^{k+1} = (A_n^k)^T F_n^k \equiv b, \tag{8}$$

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

Нахождение очередного приближения z^{k+1} метода Левенберга–Марквардта (5) либо его модифицированного варианта (6) сводится к решению СЛАУ

$$\tilde{A}_n^{\,k} z^{k+1} = \tilde{F}_n^{\,k}, \tag{9}$$

с симметричной положительно-определенной $n \times n$ матрицей $\tilde{A}_n^k = \left(A_n'\left(z^k\right)\right)^T A_n'\left(z^k\right) + \alpha_k I$ либо $\tilde{A}_n^k = \left(A_n'\left(z^0\right)\right)^T A_n'\left(z^0\right) + \alpha_k I$ и вектором правой части размерности n вида $\tilde{F}_n^k = A_n^k z^k - \left(A_n'\left(z^k\right)\right)^T \left(A_n\left(z^k\right) - F_n\right)$.

Условием останова итерационных процессов (3)−(6) является выполнение условия $\left\|A_n z - F_n\right\| / \left\|F_n\right\| < \varepsilon$ при достаточно малом $\varepsilon > 0$.

На каждом шаге метода Ньютона, метода Левенберга–Марквардта и их модифицированных вариантов для решения СЛАУ используются следующие итерационные методы градиентного типа:

метод простой итерации (МПИ)

$$z^{k+1} = z^k - \frac{1}{\lambda_{\max}}[(B + \alpha E)z^k - b], \tag{10}$$

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

$$z^{k+1} = z^k - \frac{(B(Bz^k - b), Bz^k - b)}{\left\|B(Bz^k - b)\right\|^2}(Bz^k - b), \tag{11}$$

где методы (10) и (11) применяются для $B = B^T \geq 0$;

метод минимальной ошибки (ММО)

$$z^{k+1} = z^k - \frac{\left\|Bz^k - b\right\|^2}{\left\|B^T(Bz^k - b)\right\|^2} B^T(Bz^k - b); \tag{12}$$

метод наискорейшего спуска (МНС)

$$z^{k+1} = z^k - \frac{\left\|B^T(Bz^k - b)\right\|^2}{\left\|BB^T(Bz^k - b)\right\|^2} B^T(Bz^k - b); \tag{13}$$

где B – матрица СЛАУ, возникающая при реализации методов (3)−(6) на каждом шаге;

либо метод сопряженных градиентов (МСГ) в регуляризованном варианте

$$z^{k+1} = z^k - \gamma_k(D^k z^k - b) + \beta_k(z^k - z^{k-1}), \tag{14}$$

где γ_k и β_k вычисляются по известным формулам [8].

Условием останова итерационных процессов (10)−(14) является условие $\left\|Bz - b\right\| / \left\|b\right\| < \varepsilon$ при достаточно малом ε.

2.2 Обратная задача гравиметрии о нахождении поверхностей раздела в многослойной среде и методы ее решения

Предполагается, что нижнее полупространство состоит из нескольких слоев постоянной плотности, разделенных искомыми поверхностями S_l, где L – число границ раздела (рис. 2). Гравитационный эффект от такого полупространства равен сумме гравитационных эффектов от всех поверхностей раздела.

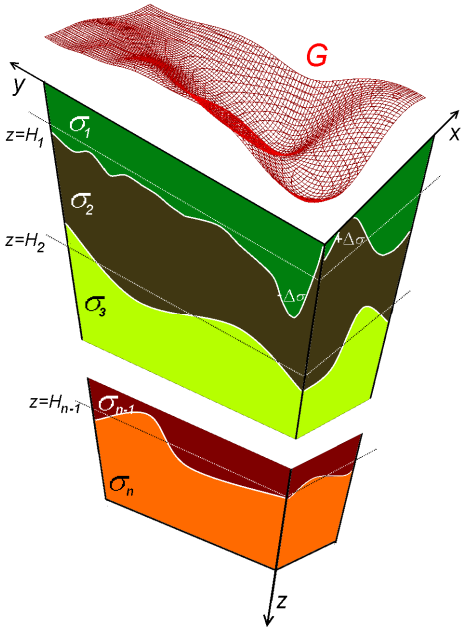


Рис. 2. Модель многослойной среды

Пусть поверхности раздела задаются уравнениями $z_l = z_l(x, y)$, скачки плотности на них равны $\Delta\sigma_l$, поверхности имеют горизонтальные асимптотические плоскости $z_l = H_l$, т.е. $\lim\limits_{\substack{|x|\to\infty \\ |y|\to\infty}} |z_l(x,y) - H_l| = 0.$

Поле от полупространства с точностью до постоянного слагаемого равно [2]

$$A(z) \equiv f\sum_{l=1}^{L}\Delta\sigma_l \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{(x-x')^2+(y-y')^2+z_l^2(x,y)}} - \frac{1}{\sqrt{(x-x')^2+(y-y')^2+H_l^2}}\right)dx\,dy = \Delta g(x',y',0), \tag{15}$$

где f – гравитационная постоянная, L – число границ раздела.

После дискретизации уравнения (15) на сетке $n = M \times N$, где задана правая часть $\Delta g(x, y)$, и аппроксимации интегрального оператора $A(z)$ по квадратурным формулам имеем вектор правой части $F(x, y)$ размерности $M \times N$, результирующий вектор решения $z(x, y) = [z_1(x, y), ..., z_L(x, y)]$ размерности $L \times M \times N$, матрицу производной оператора $A'(z^k)^T$ размерности $L \times M^2 \times N^2$ и результирующую систему нелинейных уравнений

$$\tilde{A}_n[z] = \tilde{F}_n. \tag{16}$$

Задача является недоопределенной, т.к. по заданной функции $\Delta g(x, y)$ мы пытаемся найти несколько неизвестных функций $z_l = z_l(x, y)$, что влечет неединственность решения.

В этом случае требуется либо находить близкое начальное приближение к решению, либо использовать адаптивные методы, которые за счет подходящей настройки параметров переводят итерации в область локальной сходимости.

Для решении системы нелинейных уравнений (2), возникающей после дискретизации уравнения (1) для двухслойной среды, кроме итеративно регуляризованных методов Ньютона и Левенберга–Марквардта, успешно использовались линеаризованные методы градиентного типа с дополнительными демпфирующими множителями γ, предложенными в работе [9]: например, метод наискорейшего спуска в этом случае имеет вид:

$$z^{k+1} = z^k - \gamma \frac{\left\|S(z^k)\right\|^2}{\left\|A'(z^k)S(z^k)\right\|^2} S(z^k) \equiv T(z^k), \quad S(z^k) = A'(z^k)^T (A(z^k) - F); \quad (17)$$

В работе [9] (см. также [3, 10]) показано, что при $\gamma < \gamma_0$, где γ_0 определяется исходными данными задачи, оператор шага $T(z^k)$ в процессе (17) являются псевдосжимающим, что влечет строго монотонную сходимость итераций

$$\left\|z^{k+1} - z\right\|^2 \le \left\|z^k - z\right\|^2 - \nu \left\|z^{k+1} - z^k\right\|, \quad \nu > 0.$$

Введение дополнительного множителя $0 < \gamma < \gamma_0 < 1$ позволяет расширить область допустимых начальных приближений и на первых шагах процесса перевести итерационные точки в область локальной монотонной сходимости, тогда как при $\gamma = 1$ процесс может расходиться (см. [10]).

В настоящей работе для решении системы нелинейных уравнений (16), возникающей после дискретизации уравнения (15) решения задачи гравиметрии в многослойной среде, предлагаются следующие линеаризованные итерационные методы градиентного типа с переменными демпфирующими множителями γ_i :

линеаризованный метод наискорейшего спуска (ЛМНС)

$$z_i^{k+1} = z_i^k - \gamma_i \frac{\left\|S(z^k)\right\|^2}{\left\|A'(z^k)S(z^k)\right\|^2} S_i(z^k), \quad S(z^k) = A'(z^k)^T (A(z^k) - F); \quad (18)$$

либо линеаризованный метод минимальной ошибки (ЛММО)

$$z_i^{k+1} = z_i^k - \gamma_i \frac{\left\|A(z^k) - F\right\|^2}{\left\|S(z^k)\right\|^2} S_i(z^k), \quad (19)$$

где $\gamma_i \in [0,1]$, $z_i -$ i - компонента результирующего вектора $z(x, y)$; $k -$ номер итерации.

Демпфирующие множители γ_i, зависящие от номера компоненты вектора z^k, выбираются специальным образом путем нормировки аномальных полей для каждой границы раздела, выделенных из общего гравитационного поля $\Delta g(x, y)$.

В качестве начального приближения используются горизонтальные асимптотические плоскости $z_l^0 = H_l$ $(l = 1,..., L)$.

Условием останова итерационных процессов (18)−(19) является выполнение условия $\left\|A(z) - F\right\| / \left\|F\right\| < \varepsilon$ при достаточно малом ε.

3. Распараллеливание и численная реализация итерационных методов решения обратных задач

Параллельные алгоритмы решения нелинейной обратной задачи гравиметрии о восстановлении поверхностей раздела для трехслойной среды на основе линеаризованных методов градиентного типа, а также параллельные алгоритмы решения нелинейной обратной задачи гравиметрии о восстановлении поверхности раздела для двухслойной среды на основе методов градиентного типа, итеративно регуляризованных методов Ньютона и Левенберга–Марквардта и их модифицированных вариантов численно реализованы на многопроцессорном комплексе кластерного типа с распределенной памятью МВС-ИММ, многоядерном процессоре Intel и графических процессорах NVIDIA, входящими в состав суперкомпьютера «Уран» (ИММ УрО РАН). Параллельные алгоритмы реализованы на МВС-ИММ и «Уран» с помощью технологии MPI, на графических процессорах NVIDIA с помощью технологии CUDA и библиотеки CUBLAS, на многоядерном процессоре с помощью технологии OpenMP.

При реализации на МВС и многоядерном процессоре распараллеливание итерационных методов градиентного типа, методов типа Ньютона и Левенберга–Марквардта основано на разбиении матрицы A СЛАУ горизонтальными полосами на m блоков, а вектора решения z и вектора правой части b СЛАУ на m частей так, что $n = m \times L$, где $n -$ размерность системы уравнений, $m -$ число процессоров, $L -$ число строк матрицы в блоке (рис. 3). На текущей итерации каждый из m процессоров вычисляет свою часть вектора решения. В случае умножения матрицы A на вектор z каждый из m процессоров умножает свою часть строк матрицы A на вектор z. В случае матричного умножения $A^T A$ каждый из m процессоров умножает свою часть строк транспонированной матрицы A^T на всю матрицу A. Host-процессор (ведущий) отвечает за пересылки данных и также вычисляет свою часть вектора решения.

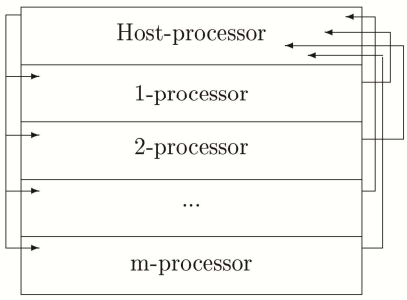


Рис. 3. Схема распределения данных по процессорам

Заметим, что при выполнении векторно-матричных операций в итерационных методах с помощью технологии OpenMP создаются параллельные потоки и эффективно распределяется работа между ними.

При реализации на графических процессорах NVIDIA распараллеливание линеаризованных итерационных методов наискорейшего спуска и минимальной ошибки решения нелинейной задачи гравиметрии для трехслойной среды основано на принципах распараллеливания итерационных методов решения линейной задачи гравиметрии, описанных в работе [11], в сочетании с использованием библиотеки CUBLAS.

Для оптимизации работы с памятью при вычислениях используется следующий прием. Для сеток довольно большой размерности, когда данные могут не входить в память видеокарты, наилучшим по быстродействию оказывается метод вычисления элементов матрицы A' «на лету», т.е. вычисление значения элемента матрицы происходит в момент обращения к этому элементу без сохранения его в память видеокарты. Это позволяет существенно снизить количество обращений к памяти видеокарты и заметно ускорить процесс вычислений по сравнению с хра-

нением матрицы A' в памяти Host-процессора и порционной загрузкой в видеоускоритель для вычислений.

Параллельные алгоритмы решения структурных обратных задач гравиметрии о восстановлении поверхностей раздела сред встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях» [12], установленный в Отделе некорректных задач анализа и приложений Института математики и механики УрО РАН. В настоящее время на Веб-портале предусмотрен запуск программ для решения задач гравиметрии на МВС-ИММ и суперкомпьютере «Уран», включающем в себя вычислительный кластер на основе видеоускорителей NVIDIA Tesla.

МВС-ИММ состоит из 128 AMD Opteron (2.6 Ггц), интерфейса GiEthernet и 256 Гб ОП.

Суперкомпьютер «Уран» состоит из 1784 Xeon (3.0 ГГц), интерфейса GiEthernet, 5328 Гб ОП и 30 вычислительных узлов с NVIDIA Tesla, содержащих по 8 GPU и по 2 шестиядерных CPU.

Специализированный Веб-портал предоставляет возможность пользователю через Веб-интерфейс выбирать тип многопроцессорного вычислителя с указанием числа процессорных узлов (МВС, NVIDIA Tesla, Multi-Core CPU), вид задачи и метод ее решения, загружать входные данные, получать выходные данные и графическое изображение результатов решения с помощью графических пакетов Surfer и gnuplot. Для каждой задачи выводится время счета.

4. Результаты численных экспериментов

Результаты численных расчетов решения задачи гравиметрии с реальными данными для двухслойной среды на многопроцессорных системах приводятся в работах [12, 13].

Здесь мы рассмотрим решение обратной задачи гравиметрии в трехслойной среде, разделенной двумя искомыми поверхностями S_1 и S_2 с модельными данными на площади S, имеющей размеры 90×100 км^2. Суммарное гравитационное поле на площади S находилось путем решения прямой задачи гравиметрии по формуле (15) с известными точными решениями $z_1(x,y)$ и $z_2(x,y)$:

$$z_1(x,y) = 5 - 2\mathrm{e}^{-(x/10-3.5)^2-(y/10-2.5)^2} - 3\mathrm{e}^{-(x/10-5.5)^2-(y/10-4.5)^2};$$

$$z_2(x,y) = 20 - 10\mathrm{e}^{-(x/25-2.2)^2-(y/25-1.75)^2}.$$

Расстояния до асимптотических плоскостей принимались равными $H_1 = 5$ км и $H_2 = 20$ км. Соответствующие скачки плотности принимались равными $\Delta\sigma_1 = 0.25$ г/см^3 и $\Delta\sigma_2 = 03$ г/см^3, гравитационная постоянная $f = 6.67\cdot10^{-8}$ см^3/г·с^2. При этом шаги сетки составили $\Delta x = \Delta y = 1.0$ км.

После дискретизации уравнения (15) на сетке имеем вектор правой части $F(x,y)$ размерности 9000, результирующий вектор решения $z(x,y) = \left[z_1(x,y), z_2(x,y)\right]$ размерности 18000, матрицу производной оператора $A'(z^k)^T$ размерности 18000×9000 и систему нелинейных уравнений вида (16).

Задача решалась на МВС-ИММ с помощью технологии MPI и вычислительном кластере NVIDIA Tesla с применением технологии OpenMP и технологии CUDA.

Для решения задачи использовались параллельные итерационные линеаризованные методы наискорейшего спуска и минимальной ошибки с демпфирующими множителями. Переменные множители находились путем специальной нормировки гравитационных полей для каждой поверхности раздела S_1 и S_2, выделенных из суммарного гравитационного поля Δg по методике [4].

На рис. 4 изображены точные решения структурной обратной задачи гравиметрии в трехслойной среде. На рис. 5 изображено суммарное гравитационное поле Δg, полученное путем решения прямой задачи гравиметрии для области S по формуле (15).

На рис. 6 изображены восстановленные поверхности раздела. При решении задачи методами ЛМНС и ЛММО относительные нормы невязок $\varepsilon = \|A(z) - F\| / \|F\|$ по сравнению с начальной нормой невязки $\varepsilon_0 = 1$ уменьшились на три порядка и составили $\varepsilon_{ЛМНС} \approx 0.005$ и $\varepsilon_{ЛММО} \approx 0.0055$ (100 итераций). Относительные погрешности $\delta_i = \|z_i^T - z_i^{np}\| / \|z_i^T\| \cdot 100\%$ для решений $z_1(x, y)$ и $z_2(x, y)$ составили 4.5 % и 3 %, соответственно.

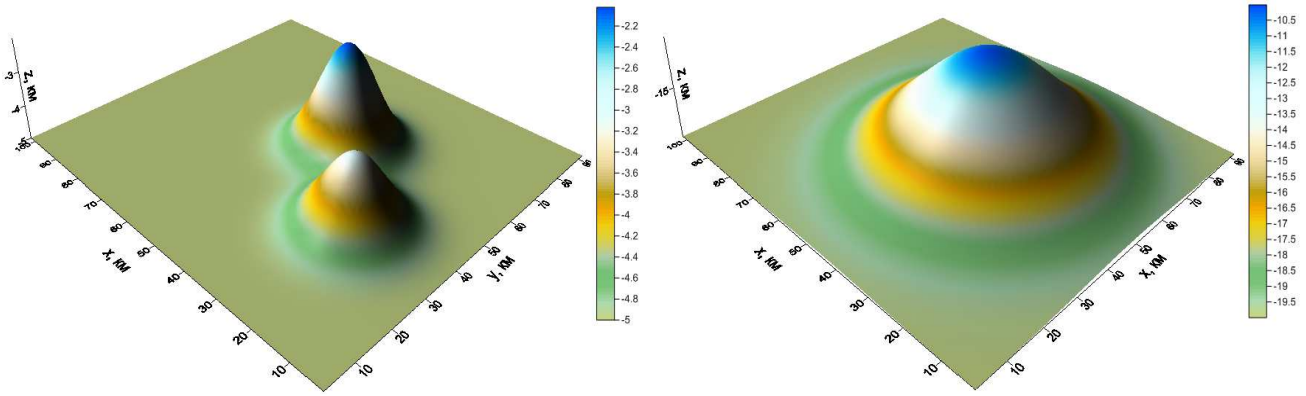


Рис. 4. Точные решения $z_1(x, y)$ и $z_2(x, y)$ для трехслойной среды

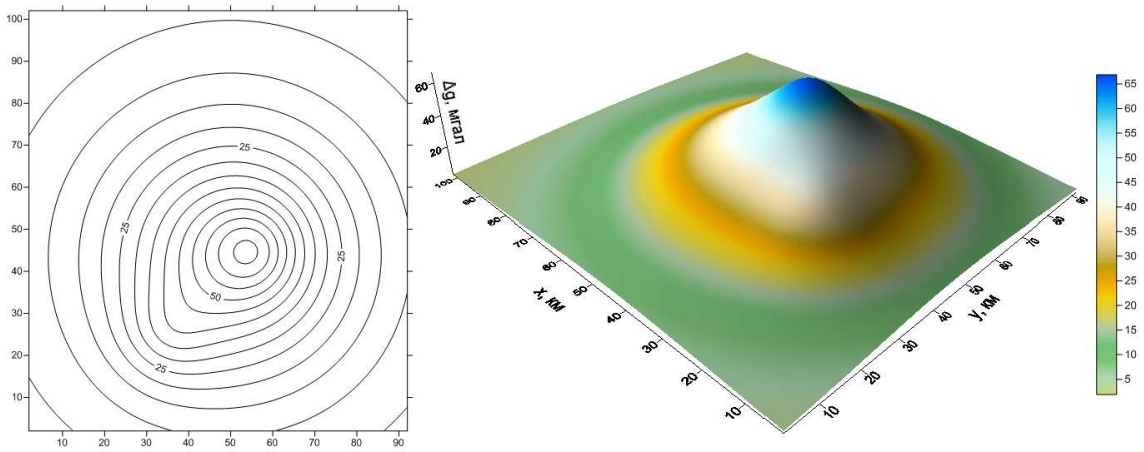


Рис. 5. Суммарное гравитационное поле

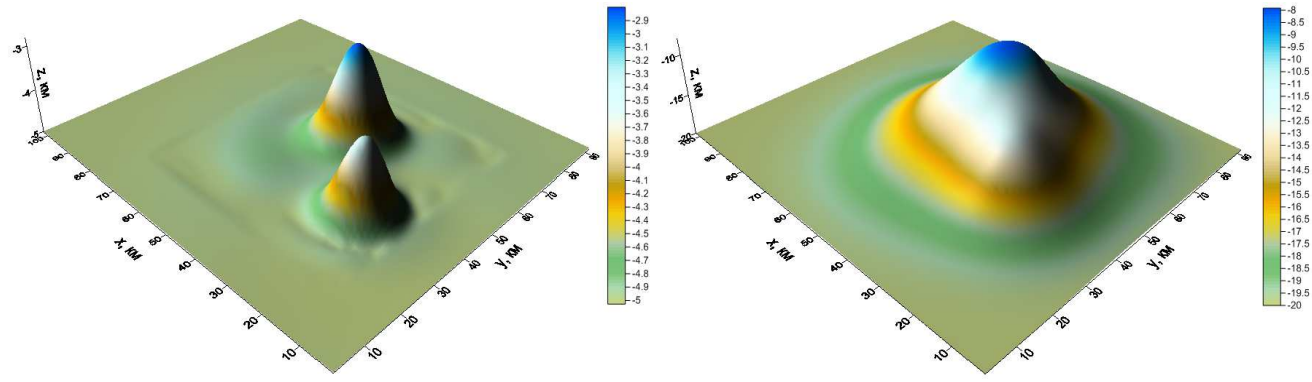


Рис. 6. Приближенные решения $\hat{z}_1(x, y)$ и $\hat{z}_2(x, y)$ для трехслойной среды

В табл. 1 и 2 приводятся времена решения нелинейной задачи гравиметрии для трехслойной среды на многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA Tesla и многоядерном процессоре Intel Xeon линеаризованными методами наискорейшего спуска (100 итераций) и минимальной ошибки (100 итераций).

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

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

где T_m – время выполнения параллельного алгоритма на МВС-ИММ либо на многоядерном процессоре с числом процессоров или ядер m ($m > 1$), T_1 – время выполнения последовательного алгоритма на одном процессоре либо на одном ядре, T_2 – время решения задачи на видеоускорителе. T_m представляет собой совокупность чистого времени счета и накладных расходов.

Таблица 1. Времена решения задачи гравиметрии линеаризованным методом наискорейшего спуска

Вычислитель	Время T_m, мин.	Ускорение S_m либо S	Эффективность E_m
Intel Xeon (1 ядро)	18.94	–	–
Intel Xeon (2 ядра)	9.55	1.98	0.99
Intel Xeon (3 ядра)	6.52	2.90	0.97
Intel Xeon (4 ядра)	4.88	3.88	0.97
Intel Xeon (5 ядер)	3.98	4.76	0.95
Intel Xeon (6 ядер)	3.41	5.55	0.92
Intel Xeon (7 ядер)	3.14	6.03	0.86
NVIDIA Tesla (448 ядер)	0.49	38.7	–

Таблица 2. Времена решения задачи гравиметрии линеаризованным методом минимальной ошибки

Вычислитель	Время T_m, мин.	Ускорение S_m либо S	Эффективность E_m
Intel Xeon (1 ядро)	18.57	–	–
Intel Xeon (2 ядра)	9.31	1.99	0.99
Intel Xeon (3 ядра)	6.40	2.90	0.97
Intel Xeon (4 ядра)	4.80	3.87	0.97
Intel Xeon (5 ядер)	3.92	4.74	0.95
Intel Xeon (6 ядер)	3.37	5.51	0.92
Intel Xeon (7 ядер)	3.11	5.97	0.85
NVIDIA Tesla (448 ядер)	0.46	40.4	–
МВС–ИММ (1 проц.)	19.98	–	–
МВС–ИММ (2 проц.)	10.98	1.82	0.91
МВС–ИММ (3 проц.)	8.02	2.49	0.83
МВС–ИММ (5 проц.)	5.57	3.58	0.71
МВС–ИММ (8 проц.)	5.15	3.87	0.48

5. Заключение

Для решения трехмерной структурной обратной задачи гравиметрии о восстановлении поверхностей раздела в многослойной среде предложены новые линеаризованные методы наискорейшего спуска и минимальной ошибки с переменными демпфирующими множителями. На основе итеративно регуляризованных методов типа Ньютона, Левенберга–Марквардта и линеаризованных методов градиентного типа для решения задач гравиметрии для двухслойной

и трехслойной среды разработаны параллельные алгоритмы, численно реализованные на многопроцессорных вычислительных системах различного типа: многопроцессорном комплексе МВС-ИММ, графических процессорах NVIDIA Tesla и многоядерном процессоре Intel Xeon с высокой эффективностью распараллеливания с использованием новых вычислительных технологий. Проведено исследование эффективности и оптимизация параллельных алгоритмов. Решены модельные задачи для трехслойной среды. Параллельные алгоритмы встроены в разработанную систему удаленных вычислений «Специализированный Веб-портал решения задач на многопроцессорных вычислителях».

Результаты вычислений показывают, что использование линеаризованных итерационных методов градиентного типа при решении структурных обратных задач гравиметрии для двухслойной и трехслойной среды позволяет получать корректные решения. Применение параллельных алгоритмов при решении обратных задач гравиметрии на многопроцессорных вычислительных системах существенно уменьшает время счета.

Литература

1. Нумеров Б.В. Интерпретация гравитационных наблюдений в случае одной контактной поверхности // ДАН СССР. 1930. № 21. С. 569–574.
2. Мартышко П.С., Ладовский И.В., Цидаев А.Г. Построение региональных геофизических моделей на основе комплексной интерпретации гравитационных и сейсмических данных // Физика земли. 2010. № 11. С. 23–35.
3. Васин В.В., Еремин И.И. Операторы и итерационные процессы фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН. 2005.
4. Мартышко П.С., Пруткин И.Л. Технология разделения источников гравитационного поля по глубине // Геофизический журнал. 2003. Т. 25. № 3. С. 159–68.
5. Bakushinsky A., Goncharsky A. Ill-Posed Problems: Theory and Applications. London: Kluwer Akad. Publ. 1994.
6. Hanke M. A regularization Levenberg–Maquardt scheme with applications to inverse groundwater filtration problems // Inverse Problems. 1997. Vol. 13. P. 79–95.
7. Васин В.В. Метод Левенберга–Марквардта для аппроксимации решений нерегулярных операторных уравнений // Автоматика и телемеханика. 2012. № 3. С. 28─38.
8. Фаддеев В.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. М.: Гос. издат. физ.-мат. литературы. 1963.
9. Васин В.В. О сходимости методов градиентного типа для нелинейных уравнений // ДАН. 1998. Т. 359. № 1. С. 5–7.
10. Vasin V.V., Skorik G.G. Iterative processes of gradient type with applications to gravimetry and magnetometry inverse problems // J. Inverse and Ill-Posed Problems. 2010. Vol. 18. № 8. P. 855─876.
11. Акимова Е.Н., Белоусов Д.В. Распараллеливание алгоритмов решения линейной обратной задачи гравиметрии на МВС-1000 и графических процессорах // Вестник ННГУ. 2010. № 5. Ч. 1. С. 193–200.
12. Акимова Е.Н., Белоусов Д.В., Мисилов В.Е. Алгоритмы решения обратных геофизических задач на многопроцессорных вычислительных системах // Труды межд. конференции «Параллельные вычислительные технологии», Новосибирск, 26–30 март. 2012 г. Челябинск: ЮУрГУ, 2012. С. 28–41.
13. Акимова Е.Н. Параллельные алгоритмы решения обратных задач гравиметрии и магнитометрии на МВС-1000 // Вестник ННГУ. 2009. № 4. С. 181–189.

Параллельный алгоритм моделирования идеального квантового алгоритма Гровера*

Д.Ю. Андреев[1,2], О.В. Корж[1], С.В. Коробков[1], А.Ю. Чернявский[1,3]

Московский государственный университет им. М.В. Ломоносова[1],
Вычислительный центр им. А. А. Дородницына РАН[2],
Физико-технологический институт РАН[3]

Одной из задач, решение которой предполагается получить с помощью экзафлопсного суперкомпьютера, является построение суперкомпьютера на новых принципах, который позволит получить существенный прогресс в скорости вычислений. В данной статье представлено моделирование работы идеального квантового компьютера на суперкомпьютере Ломоносов. Предложен эффективный алгоритм распараллеливания вычислений при одно-, двух- и трехкубитных преобразованиях с использованием библиотеки DISLIB. В качестве примера моделирования рассматривается квантовый алгоритм Гровера и квантовое преобразование Фурье.

1. Введение

Квантовая информатика является относительно молодой и очень бурно развивающейся областью современной науки. Из важнейших направлений квантовой информатики можно выделить квантовые вычисления (с которыми связана данная работа), квантовую криптографию и моделирование квантовых систем. При успешном создании квантового компьютера реализуемые на нем алгоритмы позволят решать некоторые важные задачи существенно быстрее, нежели на классических компьютерах. Так алгоритм Шора[1] позволяет раскладывать числа на простые множители за полиномиальное время, а алгоритм Гровера[2] позволяет решать любые переборные задачи за корень из классического времени. Для создания полномасштабного квантового компьютера, несомненно, необходимо моделирование его работы. Особенно важно моделирование на основе реальных физических систем с учетом квантового шума, который является основным препятствием на пути реализации квантовых битов (кубитов) и вентилей. Отметим, что с ростом числа кубитов, необходимые вычислительные ресурсы растут экспоненциально. В связи с этим фактом для решения многих задач даже с не очень большим числом кубитов невозможно обойтись без использования суперкомпьютеров. Данная работа посвящена моделированию идеальных квантовых схем и является основополагающим шагом к решению важных практических задач квантовой информатики. В работе рассматривается моделирование алгоритма Гровера, а также важнейшей подпрограммы квантовых алгоритмов – квантового преобразования Фурье, используемого в алгоритме Шора и других алгоритмах, основанных на выделении периода функций[3,4].

С точки зрения параллельных вычислений базовые алгоритмы квантовых вычислений относятся к классу DIC (Data Intensive Computing). Главным свойством задач этого класса является существенное преобладание чтения-записи данных по сравнению с количеством вычислений. При проведении каждой одно-, двух- и т.д. кубитных операции происходит изменение всего вектора состояния. При этом доступ к данным осуществляется в произвольном порядке: порядок зависит от последовательности номеров кубитов, для которых выполняется преобразование. Квантовые вычисления могут быть эффективно смоделированы на машинах с общей памятью, однако очевидно, что размер современных машин с общей памятью не позволит выполнить моделирование существенного количества кубитов. В данной статье предлагается рассмотреть моделирование квантовых вычислений на суперкомпьютере с распределенным хранением данных. Для реализации параллельной версии программы предлагается использовать метод активных сообщений, и в частности, библиотеку DISLIB[5]. Преимуществом такого подхода является возможность оптимизации пересылок данных между процессорами за счет исполь-

* Работа выполнена про поддержке грантов РФФИ 12-07-31229 и 12-01-31274.

зования низкоуровневых протоколов передачи данных в коммуникационных сетях. Высокоуровневый интерфейс библиотеки позволяет скрыть от пользователя особенности протоколов нижнего уровня, используемых на той или иной суперкомпьютерной архитектуре. Использование стандартного протокола MPI для такого рода задач представляется неэффективным, поскольку этот протокол является неэффективным для коммуникационно-сложных задач[6].

Статья включает в себя краткий обзор существующих подходов к моделированию квантовых вычислений на суперкомпьютерах, краткое описание квантовых алгоритмов, для которых проводится моделирование (алгоритма Гровера и квантового преобразования Фурье), описание программной реализации с использованием протокола активных сообщений. В заключительном разделе приводится описание результатов экспериментов для суперкомпьютера Ломоносов. Приводится анализ масштабируемости разработанных методов.

2. Обзор существующих подходов

При моделировании квантовых вычислений на суперкомпьютере количество необходимой памяти растет как степень от числа кубитов. Соответственно, возможности симуляции квантового компьютера на классических устройствах весьма ограничены. Так на современных персональных компьютерах доступное число кубитов составляет порядка 25-28, а для моделирования 35 идеальных кубитов уже требуется более 1 терабайта оперативной памяти.

Использование суперкомпьютера для моделирования квантовых вычислений требует специализированного программного обеспечения, которое позволит легко эмулировать выбранную квантовую схему. Естественно, для эффективного использования такое программное обеспечение должно быть параллельным. В статье [7] представлен алгоритм для моделирования на компьютере с общей памятью. В работе используется специальные структуры данных для хранения разреженных матриц, при помощи которых представляется эволюция квантовых состояний в ходе вычислений. В результате такого подхода было достигнуто моделирование 32 кубитов на довольно небольших ресурсах: 16 гигабайт памяти и 64 процессора. Но данный метод применим только для систем с общей памятью, где нет необходимости в пересылках между различными вычислительными узлами.

Для использования больших кластеров с распределенной памятью требуется осуществлять обмены между вычислительными узлами. Для этого наиболее часто используется протокол MPI. Примером реализации программного обеспечения удовлетворяющего данным условиям является набор программ QCMPI [8], написанный на языке Fortran с использованием MPI. Эта программа имеет большой набор операторов: стандартный оператор Паули, преобразование Адамара, контролируемое отрицание, фазовый сдвиг, и квантовое преобразование Фурье. Программная система использует библиотеки линейной алгебры BLAS[9], LAPACK[10], SCALAPACK[11]. В случае моделирования квантовых вычислений целесообразно использовать более частные алгоритмы для разреженных матриц.

В статье [12] описывается аналогичный подход к реализации модели квантового компьютера. В этой статье показано, что при моделировании 37 кубитного квантового компьютера можно добиться эффективности в 1 квантовую операцию за 10 секунд.

Также возможно использование реконфигурируемых вычислителей [13] для моделирования квантовых алгоритмов. Но и в этом случае размерность моделируемой системы ограничена памятью вычислителя.

В 2010 году было выполнено моделирование алгоритма Шора на суперкомпьютере Jugene в Суперкомпьютерном центре Юлиха [14]. Пиковая производительность этого суперкомпьютера на тот момент составляла порядка одного петафлопса, объем оперативной памяти составлял порядка 140 терабайт. Удалось выполнить моделирование 42 кубитов.

В случае же реализации подхода с хранением разреженных матриц и работой с такими матрицами с использованием специализированных протоколов обмена данными можно добиться увеличения размерности моделируемого квантового компьютера.

3. Квантовый алгоритм Гровера и квантовое преобразование Фурье

Задачей данной работы является суперкомпьютерное моделирование многокубитных квантовых схем для алгоритма Гровера и квантового преобразование Фурье. Кратко опишем формализм квантовых вычислений и рассматриваемые схемы.

3.1 Общая схема квантовых вычислений

Подробное описание формализма квантовых вычислений можно найти, например, в книгах [3,4].

Состояние одного кубита квантового компьютера определяется нормированным на единицу вектором пространства двумерного комплексного пространства $\mathbb{C}^2$:

$$a|0\rangle + b|1\rangle, |a|^2 + |b|^2 = 1,$$

где через вектора $|0\rangle$ и $|1\rangle$ обозначаются базисные вектора рассматриваемого пространства.

Состояние n-кубитной квантовой системы лежит тензорном произведении однокубитных пространств $(\mathbb{C}^2)^{\otimes n} = \underbrace{\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{n}$, соответствующие вектора имеют вид

$$\sum_{i_1, i_2, \ldots, i_n = 0}^{1} c_{i_1, i_2, \ldots, i_n} |i_1 i_2 \ldots i_n\rangle. \quad (1)$$

В каждый момент времени состояние меняется под действием некоторого унитарного преобразования. Обычно, преобразования действуют лишь на малое число кубитов (в данной работе рассматриваются одно- и двухкубитные преобразования). С формальной точки зрения такие преобразования имеют вид $U \otimes I$, где U – матрица, действующая на выбранные кубиты или кубит, I - единичная матрица, действующая на остальные кубиты. Квантовый алгоритм представляет собой последовательное применение таких преобразований. Входные данные подаются либо в виде начального состояния (вектора) системы, либо в виде выбора применяемых преобразований. После окончания работы алгоритма производится так называемое измерение, дающее для состояний вида (1) n-битный ответ $\{i_1 i_2 \ldots i_n\}$ с вероятностью $|c_{i_1, i_2, \ldots, i_n}|$. Таким образом квантовый алгоритм должен увеличивать коэффициент $c_{i_1, i_2, \ldots, i_n}$ при правильном ответе задачи.

Последовательность квантовых операций, которую и представляет из себя алгоритм, удобно представлять графически в виде так называемых квантовых схем, пример которой приведен на рис. 1.

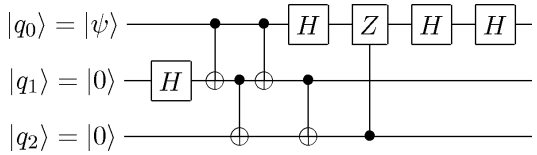


Рис. 1. Пример квантовой схемы. Каждая линия соответствует одному кубиту

3.2 Алгоритм Гровера

Пусть задана булева функция $f(x), x \in \{0,1\}^n$. Необходимо найти x, такой что $f(x) = 1$. Очевидно, что классический алгоритм в среднем решает задачу за 2^{n-1} обращений к функции f (оракулу), причем улучшить порядок $O(2^n)$ невозможно. Алгоритм Гровера позволяет решить задачу за время $O(2^{n/2})$. Важно отметить, что для работы алгоритма Гровера необходим так называемый квантовый оракул, переводящий базисное квантовое состояние $|x\rangle$ в

$(-1)^{f(x)} | x\rangle$. Несложно показать, что такой оракул может быть легко создан в виде квантовой схемы, для любой булевой функции, заданной в виде схемы из функциональных элементов. В силу ограниченности статьи мы не будем приводить квантовую схему алгоритма Гровера и объяснение действия его работы - соответствующую информацию можно найти в [3,4]. Приведем лишь особенности реализации алгоритма.

Важным подпрограммой (или подсхемой) алгоритма Гровера является инверсия относительно нулевого состояния $2|0^n\rangle\langle 0^n| - I$. В работе данное преобразование реализуется на основе методов раздела 4.3 книги [4] при помощи преобразований Тоффоли и контролируемого преобразования $Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. Само преобразование Тоффоли реализуется в виде схемы на рис. 2, где $V = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ - корень из операции NOT.

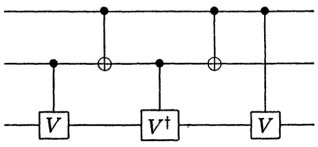


Рис. 2. Реализация элемента Тоффоли

Оракул в представленных тестовых задачах реализуется в виде обращения знака элемента вектора, соответствующего правильному ответу, однако, при необходимости может быть заменен на произвольную квантовую схему, соответствующую классической схеме вычисления некоторой булевой функции. Также в большинстве тестов инверсия производится аналогично оракулу (обращением знака при соответствующих амплитудах).

3.3 Квантовое преобразование Фурье

Квантовое преобразование Фурье является квантовым аналогом дискретного преобразования Фурье. Оно играет ключевую роль в алгоритме Шора, который позволяет разложить число N на простые множители за время $O(\log^2 N \log^3(\log N))$, что дает экспоненциальный рост скорости относительно наилучшего известного на данный момент классического алгоритма. Кроме того на основе квантового преобразования Фурье строятся алгоритмы моделирования квантовых систем на квантовом компьютере [7].

Само преобразование имеет вид $| j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{j \cdot k} | k\rangle$, $\omega = \frac{2\pi i}{2^n}$. Для соответствия с видом (1) следует отметить, что под j и k подразумеваются двоичные записи этих чисел. Такое преобразование задается рекурсивной схемой, приведенной на рис. 3.

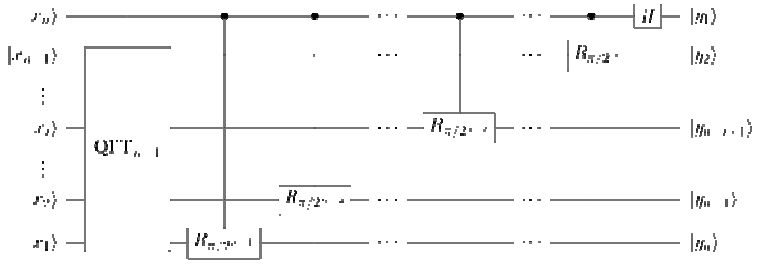


Рис. 3. Схема квантового преобразования Фурье

4. Реализация параллельной версии алгоритма с помощью библиотеки DISLIB

Особенность квантовых алгоритмов с точки зрения реализации для параллельной вычислительной системы – необходимость постоянного нерегулярного доступа к данным, которые хранятся распределенно. Поэтому использование механизма односторонних активных сообщений представляется перспективным.

Основной проблемой в реализации является размер данных, хранимых для преставления текущего вектора состояний. Для выполнение одной операции одно-, двух-, трехкубитного преобразования требуется хранить два массива комплексных чисел двойной точности. Размер каждого массива – $2^{nqubits}$ элементов, где nqubits – количество моделируемых кубитов. В случае моделирования алгоритма Гровера размер массивов $2^{2*nqubits}$ элементов.

На рис. 4 показан пример хранения вектора состояний. В примере показано хранение данных для случая двух процессоров и 4 кубитов.

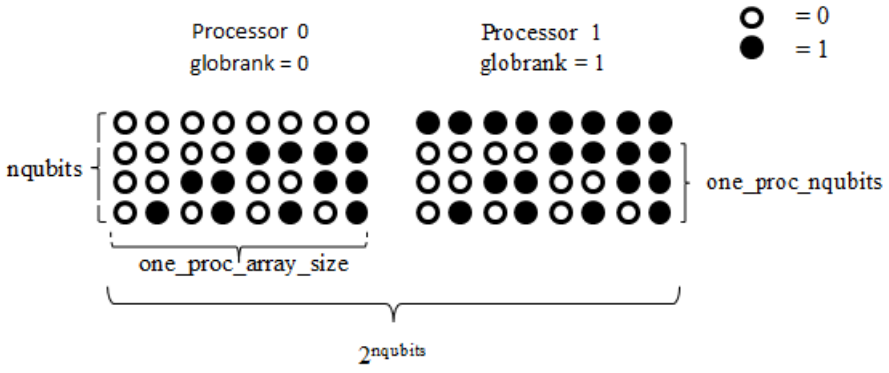


Рис. 4. Расположение данных по процессорам

Битовая длина номера элемента в массиве равна количеству используемых кубитов. При этом номер элемента можно представить как номер процессора, на котором находятся данные, и номер элемента в локальном массиве на каждом процессоре. В показанном выше примере номер процессора соответствует первому биту. Таким образом, можно определять номер элемента в глобальном массиве по номеру элемента в локальном массиве и номеру процессора. Однако это накладывает ограничение, что количество используемых процессоров должно быть степенью двойки.

При реализации однокубитной операции происходит изменение всех элементов массива состояния. При этом для расчета нового значения элемента используется текущее значение и значение элемента, индекс которого в глобальном массиве состояний отличается на один бит. Причем отличающийся бит находится в позиции, которая соответствует номеру кубита, по которому осуществляется преобразование. При этом для части номеров кубитов нужный элемент находится на текущем процессоре, а часть элементов необходимо передать по сети с других процессоров. На рис. 5 показаны необходимые обращения к данным для различных номеров кубитов. Можно видеть, что для приведённого примера обмен данными между процессорами потребуется только в случае, если номер кубита, по которому проводится преобразование, равен одному. В случае использования модели активных сообщений такая структура данных определяет, какому процессору нужно послать значение текущего обрабатываемого элемента. Соответственно идея распараллеливания однокубитной операции заключается в следующем. Для каждого элемента массива нужно посчитать сумму двух слагаемых: свое текущее значение, умноженное на некоторый коэффициент, и значение элемента с противоположным битом, также умноженное на некоторый коэффициент. Поскольку не определен порядок, в котором будет происходить суммирование, потребуется дополнительный массив, куда будут записаны результаты промежуточного суммирования. Если на процессоре не оказалось нужных данных для суммирования, то мы отправляем свой элемент на процессор, номер которого отличатся в бите,

соответствующем номеру активного кубита. И соответственно ждем от этого процессора его элемент. При этом получение элемента может произойти раньше, чем отправка. Для этого необходимо выполнять барьерную синхронизацию после отправки и получения всех элементов одной итерации. В Листинге 1 приведен код функции, реализующей отправку сообщений. В Листинге 2 показана функция-обработчик для этой функции отправки.

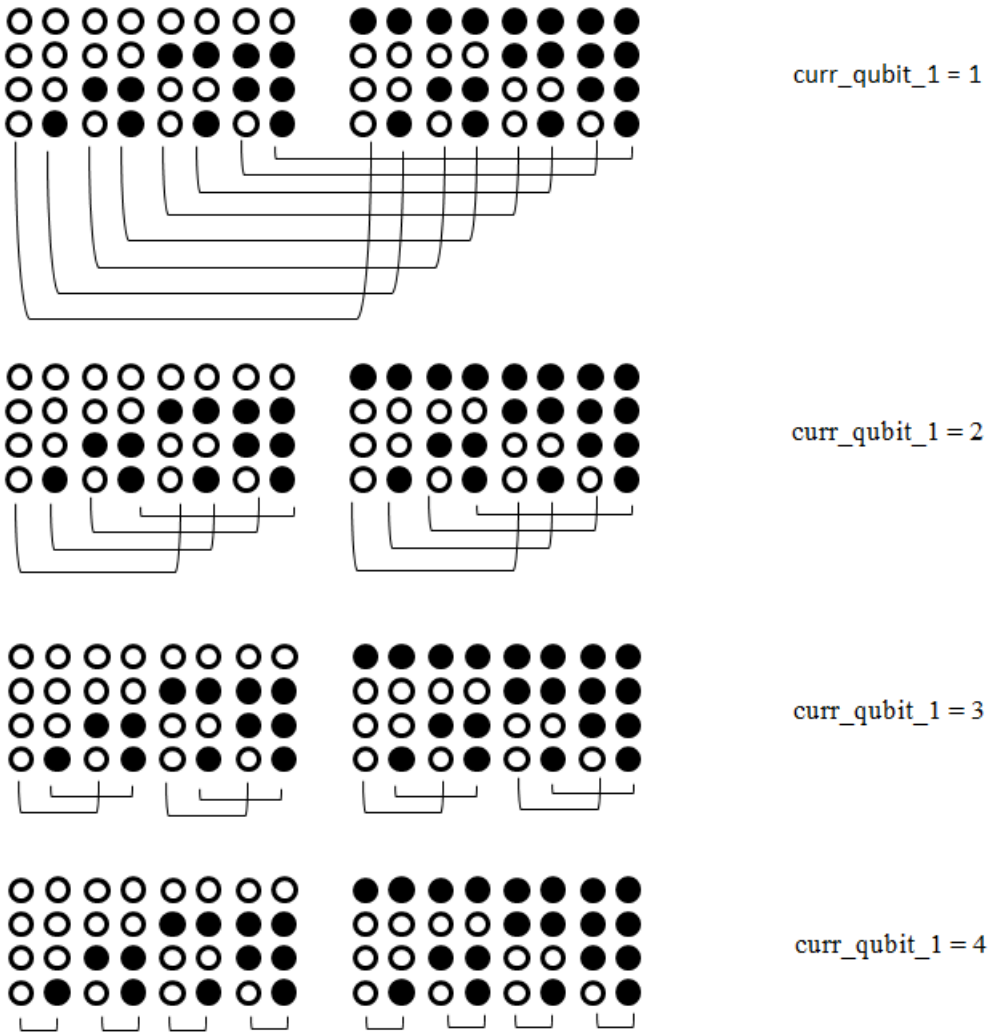


Рис. 5. Обмен данными при различных номерах активного кубита

Часто получается так, что матрица коэффициентов разреженная, таким образом можно не пересылать элементы, которые заведомо будут умножаться в итоговой сумме на ноль. Также вектор исходных состояний часто равен {1,0,0,…,0}, поэтому так же можно экономить на посылке нулевых элементов. Такая оптимизация нарушает симметричность посылок. Однако это допустимо в модели активных сообщений. И не влияет на производительность. Если в процессе не выполнилась отправка, то получение активируется вызовом барьера.

Для моделирования двух- и трехкубитных преобразований используется тот же принцип распараллеливания вычислений. Массив состояний равномерно разделяется по процессорам. Далее при изменении элементов определяется, какие являются локальными, а какие необходимо отправить / получить. Применяются аналогичные оптимизации с пересылками нулевых элементов и элементов, которые должны быть умножены на нулевые коэффициенты. С помощью определенных выше простейших функций реализуются набор операторов, таких как преобразование Адамара, фазовый сдвиг, CNOT и др.

```
// one qubit evolution function, current modification qubit is curr_qubit_1
// current matrix for transformation is in curr_U1
void one_qubit_evolution(int curr_qubit_1)
{
    long long tmp_send_proc_ind; // receiving processor number
    long long tmp_U_ind; // local index with opposite bit
    elem_ind tmp_send; // data for sending
    // output array initialization
    for(long long i=0;i<one_proc_array_size;i++) {out[i] = 0;}
    shmem_barrier_all();
    // modification of all local array elements
    for(long long i=0;i<one_proc_array_size;i++)
    {
        // required element is non-local
        if ((nqubits - curr_qubit_1) >= one_proc_nqubits)
        {
            // data for sending
            tmp_send.elem = in[i]; tmp_send.ind = i;
            // receiving processor number estimation
            tmp_send_proc_ind =
            globrank ^ (1L << (nqubits - curr_qubit_1 - one_proc_nqubits));
            // if bit in curr_qubit_1 position is 0
            if (tmp_send_proc_ind > globrank)
            {
                // current element addition
                out[i] += curr_U1[0][0] * in[i];
                // sending if element and coefficient are non-zero
                if (((abs(real(tmp_send.elem)) > 0.0000001)||(abs(imag(tmp_send.elem)) > 0.0000001))
                && (((abs(real(curr_U1[1][0]))) > 0.0000001)||(abs(imag(curr_U1[1][0]))) > 0.0000001))
                    shmem_send(&tmp_send, 1, sizeof(elem_ind), tmp_send_proc_ind);
            }
            else // if bit in curr_qubit_1 position is 1
            {
                // current element addition
                out[i] += curr_U1[1][1] * in[i];
                // sending if element and coefficient are non-zero
                if (((abs(real(tmp_send.elem)) > 0.0000001)||(abs(imag(tmp_send.elem)) > 0.0000001))
                && (((abs(real(curr_U1[1][0]))) > 0.0000001)||(abs(imag(curr_U1[1][0]))) > 0.0000001))
                    shmem_send(&tmp_send, 1, sizeof(elem_ind), tmp_send_proc_ind);
            }
        }
        else //all elements are at the current processor
        {   //local index with opposit bit
            tmp_U_ind = i ^ (1L<< (nqubits - curr_qubit_1));
            //if bit in curr_qubit_1 position is 1
            if ( i > tmp_U_ind )
                out[i] += curr_U1[1][1] * in[i] + curr_U1[1][0] * in[tmp_U_ind];
            //if bit in curr_qubit_1 position is 0
            else out[i] += curr_U1[0][0] * in[i] + curr_U1[0][1] * in[tmp_U_ind];
        }
    }
    // all processes are finished
    shmem_barrier_all();
}
```

Листинг 1. Параллельная функция для однокубитного преобразования

```
typedef struct elem_ind {
    complexd elem;
    long long ind;
}e_i;

// handler for one_qubit_evolution function
void one_qubit_evolution_hhnl(int from,void* data,int sz)
{
    // received data
    elem_ind * temp = (elem_ind *)data;

    // if current modification bit is 1
    if ((globrank - from) > 0 )
    out[temp->ind] += curr_U1[1][0] * temp->elem;
    // if current modification bit is 0
    else out[temp->ind] += curr_U1[0][1] * temp->elem;
}
```

Листинг 2. Хэндлер для функции однокубитного преобразования

В Листинге 3 приведена реализация алгоритма Гровера, последовательно вызывающая приведенные выше операторы. Особенность реализации алгоритма – массив состояний имеет размер $2^{2*nqubits}$ элементов. Это вдвое уменьшает количество доступных для моделирования кубитов. Аналогичным образом реализовано квантовое преобразование Фурье.

```
const double pi = asin(1.0)*2;
//Number of algorithm steps
double r = pi*(sqrt((double)(1<<(2*nqubits))))/4;

// Grover transformation
void Grover()
{
    Hn(); // n-qubits Hadamar transformation
    for (int i=1; i< r; i++)
    {
        Oracle(); // Oracle function
        Hn(); // n-qubits Hadamar transformation
        PhaseInverse(); // phase inverse transformation
        Hn(); // n-qubits Hadamar transformation
    }
}
```

Листинг 3. Реализация квантового алгоритма Гровера

Реализованы идеальные квантовые алгоритмы. В дальнейшем планируется реализация возможности добавления шумов. Пользователю предоставляется библиотека, реализующая базовые квантовые преобразования. Далее пользователь имеет возможность конструировать различные алгоритмы из этих преобразований. Библиотека легко расширяема. Переносимость библиотеки определяется возможностью установки библиотеки DISLIB на кластере. В дальнейшем предполагается разработка графического интерфейса для задания квантовой схемы с помощью графических нотаций.

5. Результаты тестирования алгоритмов на суперкомпьютере Ломоносов

На первом этапе тестирования реализованного алгоритма на суперкомпьютере Ломоносов требовалось провести оценку максимально возможного числа кубитов, которые получится моделировать. Это число ограничено размером данных, которые поместятся в оперативную память процессора. Теоретическая оценка показывает, что в текущей конфигурации суперкомпьютера и при использовании разработанных алгоритмов максимально возможный размер моделирования составляет 40 кубитов. На 32 узлах возможно моделирование 33 кубитов. Максимально возможное число кубитов, доступных для моделирования на одном процессоре – 29.

Далее интерес представляла оценка среднего времени выполнения одного квантового преобразования. Тестирование проводилось для преобразования Адамара. Проводилось n-кубитное преобразование и время работы делилось на количество кубитов. В таблице 1 приведено время работы программы в секундах для моделирования от 24 до 33 кубитов. Значение SF означает, что задача данного размера не может быть смоделирована на указанном количестве процессоров из-за недостаточного объема памяти.

Таблица 1. Время работы одного квантового преобразования

	24	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**
4	0.0319	0.0635	0.1275	0.2522	0.5032	1.1414	2.2330	SF	SF	SF
8	0.0102	0.0318	0.6473	0.1261	0.2518	0.5105	1.1423	2.2604	SF	SF
16	0.0083	0.0171	0.3213	0.0652	0.1293	0.2517	0.0503	1.1443	2.2896	SF
32	0.0043	0.0085	0.0160	0.0335	0.0628	0.1267	0.2504	0.5047	1.0718	2.3216

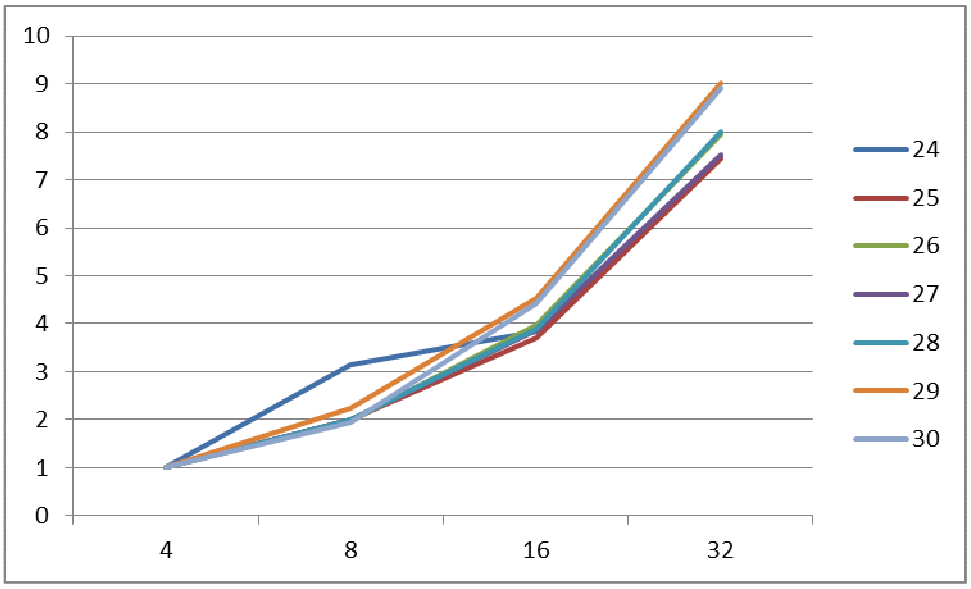


Рис. 6. График ускорения для однокубитного преобразования

На Рис. 6 показан график ускорения, полученного в этом эксперименте. Полученное ускорение практически линейное для всех рассмотренных размеров задачи. Таким образом, можно говорить о хорошей масштабируемости данной задачи как по отношению к размеру задачи, так и по отношению к увеличению количества процессоров. Эксперименты по достижению предела масштабируемости не проводились ввиду того, что для этого потребуется существенное время счета и фактически монопольный доступ на суперкомпьютер.

Далее был рассмотрен эксперимент по моделированию квантового преобразования Фурье. Результаты работы программы приведены в Таблице 2. В таблице приведено значение времени (в секундах) в зависимости от количества процессоров и количества моделируемых кубитов.

Таблица 2. Время работы квантового преобразования Фурье

QFT	**28**	**29**	**30**	**31**	**32**	**33**
8	17.4491	38.0417	90.2237	186.8465	SF	SF
16	8.5838	18.0346	40.3449	82.3250	192.9130	SF
32	4.3957	9.3806	19.5471	42.4178	97.0880	203.6334

Таблица 3. Время работы квантового алгоритма Гровера

Grover	**14**	**15**	**16**
1	10971.4132	SF	SF
2	5582.2765	SF	SF
4	2821.2334	17628.4555	SF
8	1452.7525	8876.5264	SF
16	746.7995	4499.1933	13932.3842
32	383.9903	2308.4533	6737.3245
64	198.3245	1274.4882	3367.0641

При моделировании квантового преобразования Фурье также была получена существенная масштабируемость приложения благодаря использованию механизма активных сообщений. При этом время моделирования пропорционально зависит от размера задачи.

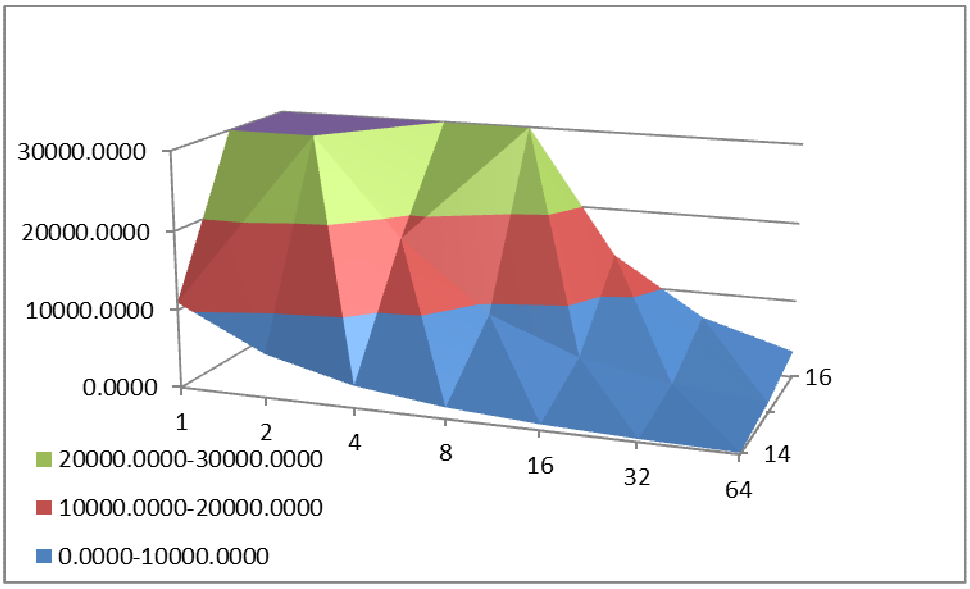


Рис. 6. График времени работы преобразования Гровера в зависимости от количества процессоров и размера задачи

В таблице 3 показано время работы программы по моделированию квантового алгоритма Гровера. Эта задача существенно более ресурсоемкая, как по времени работы, так и по количеству требуемой памяти. Максимально для моделирования использовалось 64 вычислительных узла, при этом максимально возможное количество моделируемых кубитов составило 16 (т.е. моделировалась работа 32-х кубитного квантового компьютера). Также в эксперименте наблюдается хорошая масштабируемость задачи (Рис.6).

Авторами также проводилось моделирование больших размеров задачи на большем числе вычислительных узлов. Максимально моделируемое число кубитов составило 39. Для этого использовалось 2048 вычислительных узлов суперкомпьютера Ломоносов. Однако на момент выхода этой статьи результаты эксперимента находятся в стадии верификации, поэтому численные результаты моделирования будут опубликованы позже.

6. Заключение

В работе представлено моделирование некоторых базовых алгоритмов квантовых вычислений на суперкомпьютере. Для моделирования было разработано программное обеспечение, использующее механизм передачи активных сообщений для межпроцессорных коммуникаций. Использована библиотека DISLIB. Проведено моделирование преобразования Адамара, квантового преобразования Фурье и квантового алгоритма Гровера. Во всех экспериментах показана хорошая масштабируемость разработанной параллельной программы. Дальнейшее развитие работы предполагает проведение моделирования задач большей размерности. Также предполагается моделирование квантовых алгоритмов с шумами.

Литература

1. Shor, Peter W. (1997), "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", SIAM J. Comput. 26 (5): 1484–1509
2. Grover L.K.: A fast quantum mechanical algorithm for database search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) p. 212
3. Ожигов Ю.И. Квантовые вычисления, М., Макс Пресс. 2003, -152с.
4. Нильсен М., Чанг И. Квантовые вычисления и квантовая информация: Пер с англ. МИР, 2006.

5. Корж А. А. Масштабирование Data-Intensive приложений с помощью библиотеки DISLIB на суперкомпьютерах Blue Gene/P и “Ломоносов”// Труды конференции “Научный сервис в сети Интернет-2011”, М: -изд.МГУ,сс.126-131.
6. Корж А.А. Результаты моделирования бенчмарка NBP UA на тысячи ядер суперкомпьютера BlueGene /P с помощью PGAS-расширения OpenMP // Вычислительные методы и программирование, том 11, раздел 2, 2010, сс.31-41.
7. J. Robert Burger, “New approachs to quantum computer simulaton in a classical supercomputer”, CoRR, Vol. Quant-ph/0308158 (2003)
8. Frank Tabakin, Bruno Juliá-Díaz, ”QCMPI: A parallel environment for quantum computing”, Computer Physics Communications №180, p. 948. 2009
9. Altschul S., Gish W., Miller W., Myers E., Lipman D. Basic local alignment search tool. Journal of Molecular Biology, vol. 215 (3), pp. 403–410, 1990
10. Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., Sorensen D. LAPACK Users' Guide (Third ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
11. ScaLAPACK web page http://www.netlib.org/scalapack/ (дата обращения 01.12.2012г.)
12. Guido Arnold, Thomas Lippert, Nikolas Pomplun, Marcus Richter, “Large Scale Simulation of Ideal Quantum Computers on SMP-Clusters”, PARCO , pp. 447-454, 2005
13. Goran Negovetic, Marek Perkowski, Martin Lukac, Andrzej Buller, “Evolving quantum circuits and an FPGA-based Quantum Computing Emulator”, International Worksho on Boolean Problems, 2002.
14. World record: German supercomputer simulates quantum computer http://phys.org/news189231849.html (дата обращения 01.12.2012г.)

Системный подход к суперкомпьютерному образованию

А.С. Антонов[1], Вл.В. Воеводин[1], В.П. Гергель[2], Л.Б. Соколинский[3]

Московский государственный университет имени М.В. Ломоносова[1], Нижегородский государственный университет имени Н.И. Лобачевского[2], Южно-Уральский государственный университет[3]

Система суперкомпьютерного образования создаётся в рамках выполнения проекта Комиссии при Президенте РФ по модернизации и технологическому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения». В данной статье даётся краткая характеристика создаваемой Системы, рассматривается её структура и основные компоненты. Главный акцент делается на системность реализуемого подхода, позволяющую заложить основу для внедрения суперкомпьютерного образования в практику образовательных учреждений России.

1. Введение

В 2010-2012 годах был реализован проект Комиссии при Президенте РФ по модернизации и технологическому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения» («Суперкомпьютерное образование»). В рамках данного проекта впервые были предприняты усилия по созданию в России целостной системы суперкомпьютерного образования.

Стремительное внедрение суперкомпьютерных технологий в повседневную практику ставит принципиально новые задачи перед системой высшего образования. Требуется массовая подготовка не просто компьютерно-грамотных пользователей, а именно специалистов в области параллельных вычислений и суперкомпьютерных технологий. При реализации данного проекта основное внимание было уделено не столько подготовке определённого количества специалистов в данной области, сколько созданию целостной системы, способной в течение длительного времени готовить соответствующие кадры.

В данной статье системность реализуемого подхода будет продемонстрирована на различных областях реализации данного проекта:

- создание Системы научно-образовательных центров суперкомпьютерных технологий (Системы НОЦ СКТ), охватывающей всю территорию России;
- систематизация знаний в области суперкомпьютерных технологий;
- реализация системы обучения в области суперкомпьютерных технологий;
- система сертификации знаний в области суперкомпьютерных технологий;
- система проведения мероприятий в области суперкомпьютерных технологий;
- система издания учебной и учебно-методической литературы в области суперкомпьютерных технологий;
- система мероприятий по популяризации.

2. Система НОЦ СКТ

В результате выполнения проекта «Суперкомпьютерное образование» Комиссии при Президенте РФ по модернизации и технологическому развитию экономики России к концу 2012 года реализована национальная Система научно-образовательных центров суперкомпьютерных технологий. В Систему НОЦ СКТ входят 8 региональных НОЦ, созданных в 7 федеральных округах России (см. **Рис. 1**).

Рис. 1. География Системы научно-образовательных центров суперкомпьютерных технологий

Научно-образовательные центры, входящие в систему НОЦ СКТ, создаются на базе структурных подразделений учреждений высшего профессионального образования, входящих в Суперкомпьютерный консорциум университетов России, которые обладают значительным опытом выполнения научно-исследовательских работ и ведения образовательной деятельности в области СКТ.

Основными задачами Системы НОЦ СКТ являются:

- подготовка, переподготовка и повышение квалификации специалистов по приоритетным и перспективным направлениям суперкомпьютерных технологий и специализированного программного обеспечения;
- повышение эффективности научных исследований;
- осуществление инновационной деятельности в научной и образовательной сферах совместно с организациями науки, промышленности и бизнеса.

Структура Системы НОЦ СКТ представлена на следующей схеме (**Рис. 2**).

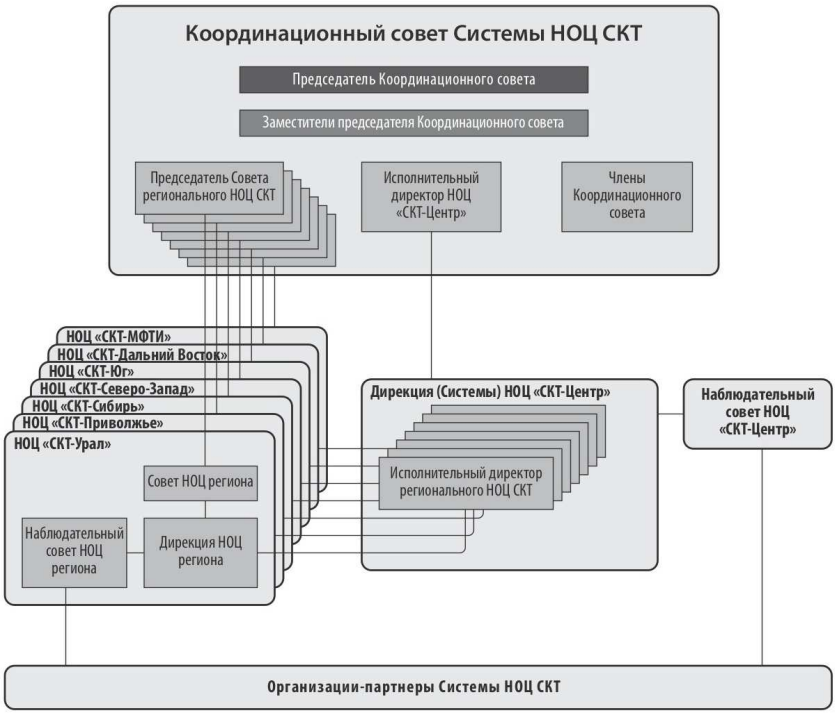


Рис. 2. Структура Системы НОЦ СКТ

Председателем Координационного совета Системы НОЦ СКТ является ректор МГУ имени М.В. Ломоносова академик В.А. Садовничий.

3. Систематизация знаний в области СКТ

Центральное место в разработке учебно-методического обеспечения системы подготовки, переподготовки и повышения квалификации кадров в области суперкомпьютерных технологий занимает разработка Свода знаний и умений в области СКТ. Это те компетенции, которыми должны обладать учащиеся после того, как закончат соответствующий факультет или курс, пройдут переподготовку или специализированное обучение в рамках спецгрупп. Именно Свод помогает понять, как должен строиться и на что должен опираться учебный процесс.

Главное в Своде — это описать предметную область «Суперкомпьютеры и параллельные вычисления», с тем, чтобы потом четко определиться, чему учить и как организовать учебный процесс для каждой конкретной целевой группы обучаемых.

Структура созданного Свода знаний согласована с рекомендациями международных профессиональных сообществ ACM и IEEE Computer Society.

Свод знаний, который должен быть освоен для успешной деятельности в рамках определенных отраслей науки, техники и бизнеса, определяется набором отдельных областей знаний, представляющих собой отдельные части изучаемой специальности. Далее, области делятся на меньшие структуры, называемые разделами, которые представляют собой отдельные тематические модули внутри области. Каждый раздел, в свою очередь, состоит из набора тем, представляющих собой нижний уровень этой иерархии в определяемой специальности. Каждая тема сопровождается указанием, является ли она обязательной или факультативной, а также рекомендуемым объемом учебного времени, необходимым для ее изучения.

Важно подчеркнуть, что подобная структура областей, разделов и тем определяет именно свод знаний, необходимый для освоения специальности, а не перечень учебных курсов. И именно этот свод знаний может служить основой для разработки учебных планов и определения необходимых учебных курсов.

Применение указанной методики (подготовка свода знаний, рекомендации по учебным планам и составу необходимых учебных курсов) для конкретной специальности или направления осуществляется специально формируемыми для этой цели рабочими группами экспертов и, как правило, такая деятельность занимает достаточно большой период времени.

Дополняет Свод знаний и умений в области СКТ комплекс созданных в рамках выполнения проекта учебных курсов. Комплекс учебных курсов покрывает основные направления развития суперкомпьютерных технологий и параллельных вычислений. При этом рассматриваются не только теоретические аспекты, но и практические вопросы использования параллельных и распределенных вычислительных технологий для решения задач в различных областях фундаментальных и прикладных исследований. Данные учебные курсы позволяют сформировать у обучаемого понятийный аппарат параллельного программирования, набор практических умений и навыков в области суперкомпьютерных технологий. Все учебные курсы прошли учебно-методическую экспертизу и получили заключение Учебно-методического совета по прикладной математике, информатике и информационным технологиям Учебно-методического объединения классических университетов.

4. Система обучения в области СКТ

Одной из важнейших задач реализации проекта является подготовка специалистов, реализуемая сразу по нескольким направлениям.

В рамках проекта реализуются программы массовой подготовки специалистов начального уровня по суперкомпьютерным технологиям. Данное мероприятие охватило все федеральные округа Системы НОЦ СКТ и более 45 вузов России. Программы обучения в большинстве вузов были направлены на получение базовых знаний и освоение наиболее востребованных технологий параллельного программирования, что нужно для успешного вхождения в область СКТ.

Примеры учебных программ подготовки специалистов начального уровня по суперкомпьютерным технологиям:

- «Параллельное программирование и высокопроизводительные вычисления»;
- «Суперкомпьютерные технологии, параллельные вычисления и их приложения»;
- «Высокопроизводительные вычисления в прикладном численном моделировании»;
- «Основы применения параллельных вычислений на основе Windows2008 HPC, Visual Studio 2011 и Intel Parallel 2011»;
- «Многопоточные вычисления на основе технологий MPI и OpenMP»;
- «Многопоточные вычисления на основе технологий CUDA и OpenCL»;
- «Суперкомпьютерные технологии с использованием прикладных пакетов»;
- «Суперкомпьютерные технологии для гибридных кластерных систем»;
- «Параллельные алгоритмы в электродинамике».

За время выполнения проекта успешно реализованы программы переподготовки и повышения квалификации профессорско-преподавательского состава. Успешно прошли переподготовку и повышение квалификации преподаватели из более 40 вузов всех федеральных округов России.

Рис. 3. Группа повышения квалификации в НОЦ «СКТ-Центр»

Примеры программ переподготовки и повышения квалификации профессорско-преподавательского состава:

- «Суперкомпьютерные системы и приложения»;
- «Суперкомпьютерные технологии с использованием прикладных пакетов»;
- «Основы суперкомпьютерных технологий»;
- «Современные параллельные вычислительные технологии»;
- «Перспективные технологии распределенных вычислений»;
- «Параллельные вычислительные системы и технологии, используемые при разработке программ»;
- «Многопроцессорные вычислительные системы и параллельное программирование»;
- «Распараллеливание алгоритмов и программ».

Выполнено крайне сложное мероприятие проекта – целевая интенсивная подготовка в области суперкомпьютерных технологий в рамках специальных групп, ориентированная на глубокое изучение конкретных разделов суперкомпьютерных технологий. Обучение спецгрупп было организовано во всех федеральных округах Системы НОЦ СКТ.

В числе уже реализованных программ обучения:

- «Суперкомпьютерное моделирование: технологии, инструменты и приложения (основная и углубленная программа)»;
- «Технологии, используемые при организации высокопроизводительных вычислений»;
- «Решения прикладных задач механики деформируемого твердого тела в высокопроизводительных система»;
- «Виртуализация и построение виртуальных кластеров на базе Microsoft HyperV и VMWare ESXI, разработка многопоточных приложений для виртуальных кластеров»;
- «Распараллеливание алгоритмов и программ»;
- «Моделирование биологических систем на GPU»;
- «Суперкомпьютерные технологии для решения естественнонаучных задач».

Ещё один вид обучения, реализованный в рамках выполнения проекта – это дистанционное образование. Проект Интернет-университета суперкомпьютерных технологий имеет уникальный характер. Отличительные особенности: привлечение ведущих специалистов страны для проведения занятий, реализация классической формы обучения на новой технологической основе, доступность обучения за счёт использования сети Интернет. Все эти характеристики проекта позволяют обеспечить массовую и оперативную подготовку специалистов в области суперкомпьютерных технологий.

На базе Интернет-университета суперкомпьютерных технологий проводилось дистанционное обучение по следующим курсам:

- «Основы параллельных вычислений»;
- «Введение в параллельные алгоритмы»;
- «Параллельное программирование с OpenMP»;
- «Основы параллельного программирования с использованием MPI».

В рамках проекта «Суперкомпьютерное образование» начальную подготовку в области СКТ с использованием технологий дистанционного образования получили слушатели из более 100 городов России.

5. Система сертификации знаний в области СКТ

Система сертификации знаний в области СКТ разрабатывается для инженерно-технических, естественнонаучных и социально-гуманитарных направлений. Разработка системы сертификации знаний входит обязательным условием формирования национальной системы подготовки высококвалифицированных специалистов в области СКТ.

Сертификация призвана зафиксировать обладание человеком определенным уровнем знаний по конкретной предметной области. Предметной областью может быть как вся область суперкомпьютерных технологий, так и некоторая её подобласть. Сертификация в некоторой области выполняется по трем уровням знаний: Introduction, Basic, Master (Начальный, Базовый, Мастер).

Сертификация знаний производится на основе обязательного выполнения человеком набора тестов в автоматизированном режиме с помощью электронной системы тестирования. Это условие необходимое, но для высших уровней знаний отдельных областей может потребоваться выполнение дополнительных заданий.

Набор тестов для каждого уровня знаний каждой области отражает все разделы Свода знаний по сертифицируемой области. Свод знаний по сертифицируемой области формируется в соответствии со Сводом знаний и умений по суперкомпьютерным технологиям, разработанным в рамках данного проекта, но не обязательно должен являться его подмножеством.

Для каждого уровня каждой области фиксируется процедура прохождения сертификации. Иногда на начальных уровнях возможна полностью автоматизированная сертификация, выполняемая через Интернет без какого-либо контроля над способом прохождения тестирования. В

большинстве случаев при прохождении тестирования требуется обязательное присутствие преподавателя.

Процедура включения сертификации по некоторой конкретной области СКТ в общую систему сертификации предполагает выполнение следующих шагов.

1. Формирование Свода знаний по сертифицируемой области на основе общего Свода знаний и умений по суперкомпьютерным технологиям.

2. Выделение сертифицируемых уровней знаний.

3. Указание в Своде знаний сертифицируемой области разделов, необходимых для каждого выделенного уровня знаний.

4. Формирование наборов тестов для каждого выделенного уровня знаний для организации электронного тестирования.

5. Описание процедуры прохождения сертификации для каждого уровня.

6. Утверждение Консорциумом Свода знаний по сертифицируемой области, уровней знаний и процедуры прохождения сертификации. В случае положительного решения со стороны Консорциума, на выдаваемом сертификате размещается логотип Консорциума, ставится подпись от Консорциума, каждый сертификат получает уникальный номер, сохраняемый в базе Консорциума.

В июле 2012 года в рамках проведения в Московском государственном университете имени М.В.Ломоносова международной Летней Суперкомпьютерной Академии система сертификации знаний была опробована при проведении сертификация базового уровня знаний по параллельным вычислениям и суперкомпьютерным технологиям. В результате проведения тестирования группа выпускников Академии получила первые официальные сертификаты.

С 2011 г. реализуется похожая программа сертификации по параллельному программированию при активном использовании программных инструментов компании Интел.

6. Система проведения мероприятий

В последние годы сформировалась система основных национальных суперкомпьютерных конференций:

- Международная научная конференция «Параллельные вычислительные технологии (ПаВТ)», организаторы – Российская академия наук, Суперкомпьютерный консорциум университетов России (http://agora.guru.ru/pavt).
- Международные суперкомпьютерные конференции серии «Научный сервис в сети Интернет», организаторы – Российская академия наук, Суперкомпьютерный консорциум университетов России (http://agora.guru.ru/abrau).
- Суперкомпьютерный форум «Суперкомпьютерные технологии в образовании, науке и промышленности», учредитель – Суперкомпьютерный консорциум университетов России (http://agora.guru.ru/hpc2012).
- Сибирская конференция по параллельным и высокопроизводительным вычислениям (http://ssspc.math.tsu.ru).

Система молодёжных суперкомпьютерных школ в России состоит из серии мероприятий, организованных Суперкомпьютерным консорциумом университетов России на базе ведущих вузов:

- Северного (Арктического) федерального университета (февраль);
- Санкт-Петербургского национального исследовательского университета информационных технологий, механики и оптики (апрель);
- Московского государственного университета имени М.В. Ломоносова (июнь – июль);
- Московского физико-технического института (август);
- Нижегородского государственного университета имени Н.И. Лобачевского (ноябрь);
- Томского государственного университета (декабрь).

7. Система издания учебной и учебно-методической литературы

Успешно реализуется комплексная программа по разработке и экспертизе учебной и учебно-методической литературы в области суперкомпьютерных технологий для бакалавриата и магистратуры. В формируемую в рамках проекта серию «Суперкомпьютерное образование» входят монографии, учебники и учебные пособия, написанные ведущими российскими и зарубежными специалистами по основным разделам Свода знаний в области суперкомпьютерных технологий. Всего в серии – более 25 книг российских и зарубежных авторов.

Рис. 4. Обложки книг серии «Суперкомпьютерное образование»

От 10 до 50 экземпляров каждой книги серии «Суперкомпьютерное образование» бесплатно передается в 43 университета России. Всего за время проекта в российские университеты передается 37500 книг данной серии.

8. Система мероприятий по популяризации области СКТ

Разработана и реализована система мероприятий по популяризации достижений и перспектив использования суперкомпьютерных технологий с привлечением СМИ: выступления с лекциями в проекте «Академия» на телеканале «Культура», публикации в профильных Интернет-изданиях, на сайте Комиссии при Президенте РФ (i-russia.ru), серии публикаций в центральных и региональных СМИ, участие в работе конференций, фестивалей, выставок, организация экскурсий школьников в суперкомпьютерные центры и многое другое.

Результаты проекта постоянно отображаются на страницах Интернет-центра (http://hpc-education.ru). В разделах центра можно познакомиться со Сводом знаний и умений в области суперкомпьютерных технологий и принять участие в работе над его созданием. Здесь представлена деятельность Научно-образовательных центров в области СКТ, созданных в федеральных округах России, новые учебные курсы и программы, разработанные с учетом рекомендаций Свода знаний и умений и международного опыта, существующая и разрабатываемая нормативная база в области СКТ, в частности, рекомендации по расширению ФГОС, а также информация о международных, федеральных, региональных учебно-научных мероприятиях и событиях. Вся информация, размещаемая на станицах этого ресурса, проходит строгую экспертную оценку.

Деятельность Интернет-центра наглядно демонстрирует тот вклад, который внесли участники Суперкомпьютерного консорциума университетов России в создание национальной системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий, а также их четкое понимание ответственности за подготовку высококвалифицированных

специалистов и формирование прочного научного фундамента, столь необходимого для эффективного использования суперкомпьютерных технологий на практике.

8. Развитие системы суперкомпьютерного образования

В рамках проекта Комиссии при Президенте РФ по модернизации и технологическому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения» создана основа функционирования системы суперкомпьютерного образования в России. Однако эта система требует дальнейшего развития и совершенствования.

Дальнейшей целью должно быть обеспечение устойчивого развития национальной системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения на основе развития инфраструктуры НОЦ СКТ, расширения спектра образовательных программ с учетом потребностей инновационной экономики и решения задач по приоритетным направлениям развития науки, техники и технологий, совершенствования учебных планов и программ подготовки, переподготовки и повышения квалификации специалистов.

Предполагается создание специализированного математического и программного обеспечения для реализации услуг в области распределенных и облачных вычислений, применения дистанционных технологий в организации совместной образовательной деятельности НОЦ СКТ, самостоятельной работы студентов и слушателей, обучающихся по образовательным программам в области СКТ, организации дополнительного образования учащихся старших классов на базе специализированных школ для одаренных детей при вузах.

Подготовка преподавателей вузов, реализующих образовательные программы по направлениям инженерного, естественнонаучного и социально-гуманитарного образования, является необходимым элементом развития системы суперкомпьютерного образования. Должны быть разработаны предложения по модификации (обновлению) существующих и разработке новых учебных курсов и программ в области СКТ для укрупненной группы специальностей и направлений подготовки по социально-гуманитарным направлениям, включая: экономические, гуманитарные и социальные направления подготовки магистров.

Для кадрового обеспечения социально-экономического развития регионов России на основе модифицированных учебных планов и программ в соответствии с разработанной системой сертификации знаний будут реализованы образовательные программы подготовки специалистов начального уровня и целевой интенсивной подготовки в области СКТ.

Важными направлениями развития проекта являются расширение спектра образовательных программ в области инженерного образования, приоритетных направлений науки, техники и технологий и образовательных программ с учетом потребностей высокотехнологичных отраслей экономики России. Должны быть разработаны предложения по расширению федеральных государственных образовательных стандартов третьего поколения для специальностей, отвечающих приоритетным направлениям развития науки, техники и технологий, включая: «Индустрия наносистем», «Науки о жизни», «Рациональное природопользование», «Транспортные и космические системы», «Энергоэффективность, энергосбережение, ядерная энергетика». Должны быть разработаны предложения и рекомендации по модификации (обновлению) учебных планов и программ переподготовки и повышения квалификации в области СКТ для системы инженерного образования по направлениям «Авиационная и ракетно-космическая техника», «Металлургия, машиностроение и материалообработка», «Энергетика, энергетическое машиностроение и электротехника».

Учебно-методическое обеспечение образовательных программ является важным фактором повышения качества учебного процесса и в целом качества образования специалиста. Планируется разработка новых учебных пособий и учебников по СКТ для выбранных направлений подготовки специалистов в области инженерного образования, социально-гуманитарного образования, естественнонаучного образования.

Привлечение зарубежных ученых и научно-образовательных центров является важным фактором повышения качества суперкомпьютерного образования в России. Развитие международного сотрудничества в области суперкомпьютерного образования (СКО) предполагается по

направлениям: разработка совместных с зарубежными университетами образовательных программ в области СКТ; разработка и реализация программ академического обмена с ведущими зарубежными научно-образовательными центрами для студентов, аспирантов и молодых ученых. В рамках этого сотрудничества будут публиковаться альманахи «Суперкомпьютерное образование в мире». Планируется привлечение зарубежных специалистов для формирования международной системы сертификации знаний в области СКТ.

Одним из ключевых факторов успешной реализации проекта является информационная и просветительская деятельность. Популяризация научных знаний и достижений науки и техники в области СКТ являются основными механизмами формирования общественного мнения и положительного имиджа СКТ в обществе, воспитания научного мировоззрения молодежи и привлечения ее в науку. С этой целью предусматривается создание системы информационного сопровождения проекта с привлечением печатных и электронных СМИ и коллективных научных мероприятий. На базе НОЦ СКТ планируется создание специализированных школ по СКТ при вузах для одаренных детей, реализация программ дополнительного образования детей и создание системы интеллектуальных конкурсов студентов в области СКТ

Для выполнения поставленных в проекте задач может быть задействован уникальный потенциал профессорско-преподавательского состава и ученых университетов-членов Суперкомпьютерного консорциума университетов России, а также университетов и научных центров, входящих в состав Системы НОЦ СКТ.

9. Заключение

Реализация системы суперкомпьютерного образования в рамках выполнения проекта Комиссии при Президенте РФ по модернизации и технологическому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения» позволяет рассчитывать на постоянный приток высококвалифицированных специалистов, обладающих необходимыми знаниями и навыками для внедрения передовой суперкомпьютерной техники в самые разные отрасли экономики.

Усилия, предпринятые в рамках данного проекта, были высоко оценены как в России, так и в мире. Так, в 2011 году объединённая команда МГУ имени М.В. Ломоносова и ННГУ имени Н.И. Лобачевского победила в международном конкурсе по разработке учебных материалов по параллельному программированию «Informatics Europe Curriculum Best Practices Award». Также большое внимание было уделено презентациям проекта на стендах российских участников ведущих международных суперкомпьютерных конференций International Supercomputing Conference и Supercomputing.

Литература

1. Воеводин Вл.В., Гергель В.П., Соколинский Л.Б., Демкин В.П., Попова Н.Н., Бухановский А.В. Развитие системы суперкомпьютерного образования в России: текущие результаты и перспективы // Вестник Нижегородского университета, 2012. № 4. С.203-209.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ-Петербург, 2002. - 608 с.
3. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем.- М.: Издательство Московского университета, 2010. - 544 с.
4. Воеводин Вл.В., Гергель В.П. Суперкомпьютерное образование: третья составляющая суперкомпьютерных технологий // Вычислительные методы и программирование: новые вычислительные технологии. - Москва: НИВЦ МГУ, 2010. Т. 11. № 2. С. 117-122.
5. Антонов А.С., Артемьева И.Л., Бухановский А.В., Воеводин Вл.В., Гергель В.П., Демкин В.П., Коньков К.А., Крукиер Л.А., Попова Н.Н., Соколинский Л.Б., Сухинов А.И. Проект «Суперкомпьютерное образование»: 2012 год// Научный сервис в сети Интернет: поиск новых решений: Труды Всероссийской научной конференции (17-22 сентября 2012 г., г. Новороссийск).-М.: Изд-во МГУ, 2012. С. 4-8.

Использование языка Fortran DVMH для решения задач гидродинамики на высокопроизводительных гибридных вычислительных системах*

В.А. Бахтин, М.С. Клинов, В.А. Крюков,
Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов

ФГБУН Институт прикладной математики им. М.В. Келдыша РАН

В 2011 году для новых гетерогенных и гибридных суперкомпьютерных систем в Институте прикладной математики им. М.В. Келдыша РАН была предложена модель DVMH (DVM for Heterogeneous systems), разработаны языки программирования высокого уровня, представляющие собой стандартные языки Фортран и Си, расширенные директивами отображения программы на параллельную машину, оформленными в виде специальных комментариев (или прагм). В статье анализируется эффективность разработанных на языке Fortran DVMH параллельных программ для решения задач гидродинамики «Каверна» и «Контейнер». Приводятся результаты расчетов при использовании нескольких тысяч ядер и более 1200 GPU-ускорителей.

1. Введение

Будущее высокопроизводительных компьютерных технологий неразрывно связано с массивным параллелизмом и с гетерогенностью. Создаются процессоры, содержащие всё большее количество ядер. Жесткие ограничения по энергопотреблению приводят к тому, что основные вычислительные мощности обеспечиваются многоядерными GPU-ускорителями достаточно специфичной архитектуры, адаптация программного обеспечения к которой - сложная наукоёмкая задача.

Разрыв между существующим программным обеспечением и возможностями новых суперкомпьютеров носит принципиальный характер и является существенной проблемой на пути эффективного использования современной вычислительной техники в научных исследованиях.

Разработанная в Институте прикладной математики им. М.В. Келдыша РАН высокоуровневая модель параллельного программирования - DVMH (DVM for Heterogeneous systems) существенно упрощает разработку параллельных программ для кластеров с гетерогенными узлами, использующих в качестве ускорителей графические процессоры.

2. Модель DVMH. Язык Fortran DVMH

При разработке модели DVMH за основу была взята модель DVM[1], в которую были добавлены следующие возможности:

1) Определение фрагментов программы, которые следует выполнять на том или ином ускорителе.

Такими фрагментами программ (называемых вычислительными регионами, или просто регионами) могут быть отдельные DVM-циклы или их последовательность.

2) Определение требуемых регионам данных.

Для каждого региона указываются требуемые ему данные и вид их использования (входные, выходные, локальные).

3) Задание свойств цикла и правил отображения витков цикла на ускоритель.

Для каждого DVM-цикла можно задать конфигурацию блока нитей (в терминологии CUDA). Если конфигурация блока нитей не задана в программе, то она определяется автоматически.

* Исследование выполнено при финансовой поддержке грантов РФФИ № 11-01-00246, 12-01-33003 мол_а_вед, 12-07-31204-мол_а и гранта Президента РФ НШ-4307.2012.9.

4) Управление перемещением данных между оперативной памятью универсального процессора и памятями ускорителей.

Перемещение данных осуществляется, в основном, автоматически в соответствии с запусками регионов на ускорителях и информацией об используемых ими данных. Для фрагментов программ, которые выполняются на универсальном процессоре (вне вычислительных регионов), имеются специальные средства для задания, какие данные с ускорителя им нужны и какие данные ими были скорректированы.

2.1 Организация вычислений, спецификации потоков данных

Вычислительный регион выделяет часть программы (с одним входом и одним выходом) для возможного выполнения на одном или нескольких вычислителях.

!DVM$ REGION [clause {, clause}]

<region inner>

!DVM$ END REGION

Регион может быть исполнен на одном или сразу нескольких ускорителях и/или на хост-системе, при этом на хост-системе может быть исполнен любой регион, а на возможность использования каждого типа ускорителей могут накладываться свои дополнительные ограничения на содержание региона.

Например, с использованием CUDA-устройства может быть исполнен любой регион без использования операций ввода/вывода, вызовов внешних процедур, рекурсивных вызовов.

Для управления тем, на каких вычислителях регион может исполняться, следует использовать клаузу TARGETS (см. ниже).

Вложенные (статически или динамически) регионы не допускаются.

DVM-массивы распределяются между выбранными вычислителями (с учетом их заданных весов и быстродействия вычислителей), нераспределенные данные размножаются. Витки вложенных в регион параллельных DVM-циклов делятся между выбранными для региона вычислителями в соответствии с правилом отображения параллельного цикла, заданного в директиве параллельного DVM-цикла. Количество и типы используемых каждым MPI-процессом ускорителей можно задать с помощью переменных окружения, а по умолчанию каждым процессом будут использованы все найденные поддерживаемые ускорители.

В качестве клауз может быть задано:

1) IN(subarray_or_scalar {, subarray_or_scalar}), OUT(subarray_or_scalar {, subarray_or_scalar}), INOUT(subarray_or_scalar {, subarray_or_scalar}), LOCAL(subarray_or_scalar {, subarray_or_scalar}), INLOCAL(subarray_or_scalar {, subarray_or_scalar})

Указание направления использования подмассивов и скаляров в регионе. IN - по входу в регион нужны актуальные данные. OUT - в регионе значение переменной изменяется, причем это изменение может быть использовано далее. INOUT(subarray_or_scalar {, subarray_or_scalar}) - сокращенная запись одновременно двух clause: IN(subarray_or_scalar {, subarray_or_scalar}), OUT(subarray_or_scalar {, subarray_or_scalar}). LOCAL - в регионе значение переменной изменяется, но это изменение не будет использовано далее. INLOCAL(a) - сокращенная запись одновременно двух clause: IN(a), LOCAL(a). Если для переменной указано IN, и не указано OUT или LOCAL, то считается, что в такую переменную в регионе вообще нет записей и она не меняется в процессе его исполнения.

После выбора набора исполнителей региона автоматически определяются и выполняются операции по выделению памяти для подмассивов и скаляров (если отсутствовал представитель или присутствовал не являющийся объемлющим), операции по обновлению входных данных (если не было актуального представителя). По выходу из региона обновления данных не происходят.

Указание всех используемых переменных в регионе не обязательно. При этом используемые, но не указанные в клаузах переменные включаются в регион в автоматическом режиме компилятором Fortran DVMH по правилам:

а) Все используемые массивы считаются используемыми полностью (не выделяются подмассивы);

б) Всякая переменная, которая используется на чтение получает атрибут IN;

в) Всякая переменная, которая используется на запись получает атрибут INOUT;

г) Всякая переменная, направление использования которой не поддаётся определению, получает атрибут INOUT;

д) атрибуты LOCAL и OUT в автоматическом режиме не проставляются.

2) TARGETS(target_name {, target_name})

где target_name это CUDA | HOST

Указание списка типов вычислителей, на которых предполагается исполнять регион. Такая клауза может быть только одна в директиве. Действительное исполнение региона будет происходить на всех используемых конкретным MPI-процессом вычислителях указанных в директиве типов, для которых регион был подготовлен, а если таковых нет, то на хост-системе. Количество и типы используемых каждым MPI-процессом ускорителей можно задать с помощью переменных окружения, а по умолчанию все вычислительные ресурсы каждого узла будут использованы процессами равномерно.

3) ASYNC - указание возможности асинхронного исполнения региона.

При запуске региона в любом режиме (синхронный, асинхронный) ожидание завершения ранее запущенного региона возникает, если клаузами IN, OUT, LOCAL, INOUT, INLOCAL задается необходимость изменить данные, используемые этим (ранее запущенным) регионом или необходимость использовать (запись или чтение) данные, изменяемые этим (ранее запущенным) регионом (OUT, INOUT, LOCAL, INLOCAL).

Управление не перейдет на следующий за синхронным регионом оператор, пока текущий регион не закончит исполнение. Управление может перейти на следующий за асинхронным регионом оператор, не дожидаясь его завершения (или даже его старта).

В полный цикл исполнения региона входит:

1) освобождение места для новых переменных на ускорителях (возможна автоматическая актуализация переменных на хосте),
2) выделение памяти для новых переменных на ускорителях,
3) закачка необходимых актуальных данных на вычислители,
4) исполнение исполняемых операторов на вычислителях.

<region inner> - это нуль или более следующих друг за другом конструкций:

1) Параллельный DVM-цикл

Параллельный DVM-цикл - важнейшая часть вычислительного региона.

!DVM$ PARALLEL clause {, clause}

<DVM-loop nest>

В качестве клауз кроме клауз DVM-цикла могут быть также заданы:

а) PRIVATE(array_or_scalar {, array_or_scalar})

Объявляет переменную приватной (локальной для каждого витка цикла), при этом ее объявление в объемлющем цикл регионе не обязательно (более того, если по-другому она не используется, то объявление ее в регионе излишне).

б) CUDA_BLOCK(X [, Y [, Z]])

Указание размера блока нитей для вычислителя CUDA. Может указываться целочисленное выражение - тогда блок полагается одномерным, может указываться два или три целочисленных выражения через запятую - соответственно блок будет полагаться указанной размерностью.

2) Последовательная группа операторов

Каждый оператор последовательной группы операторов исполняется на всех вычислителях, выбранных для исполнения региона, кроме случая модификации в нем распределенных данных - тогда действует правило собственных вычислений.

3) Хост-секция

!DVM$ HOSTSECTION

<hostsection inner>

!DVM$ END HOSTSECTION

Объявляет специального вида секцию исполнения на хосте.

<hostsection inner> - это часть программы с одним входом и одним выходом, которая будет исполняться на хост-системе. Всякие изменения переменных в этой секции могут быть потеряны. Такие секции предлагается использовать в отладочных целях для промежуточного контроля значений переменных по ходу исполнения региона. Операции вывода разрешены, вызовы внешних процедур разрешены.

2.2 Управление перемещением данных, актуальностью

Для фрагментов программ, которые выполняются на хосте (вне вычислительных регионов), управление перемещением данных между оперативной памятью универсального процессора и памятями ускорителей задается при помощи специальных директив актуализации:

!DVM$ GET_ACTUAL[(subarray_or_scalar {, subarray_or_scalar})]

делает все необходимые обновления для того, чтобы в хост-памяти были самые новые данные в указанном подмассиве или скаляре. В случае отсутствия параметров все имеющиеся новые данные с ускорителей переписываются в память хост-системы;

!DVM$ ACTUAL[(subarray_or_scalar {, subarray_or_scalar})]

объявляет тот факт, что указанный подмассив или скаляр самую новую версию имеет в хост-памяти. При этом пересекающиеся части всех других представителей указанных переменных автоматически устаревают и перед использованием будут (по необходимости) обновлены. В случае отсутствия параметров все имеющиеся представители переменных в памятях ускорителей объявляются устаревшими.

Использование директив ACTUAL и GET_ACTUAL без параметров не рекомендуется в силу повышения вероятности ошибок (ACTUAL), а также опасности излишних перемещений данных (GET_ACTUAL).

2.3 Компилятор с языка Fortran DVMH

Компилятор с языка Fortran DVMH преобразует исходную программу в параллельную программу на языке Fortran с вызовами функций системы поддержки времени выполнения (библиотека Lib-DVM). Кроме того, компилятор создает для каждой исходной программы еще два модуля: один - на языке C CUDA[2] и второй - на языке Fortran CUDA[3].

В частности, для параллельного цикла из региона компилятор генерирует функцию-обработчик на языке C CUDA и ядро для вычислений на GPU на языке Fortran CUDA, а также процедуру-обработчик на языке Fortran для выполнения на хост-машине. Обработчик - подпрограмма, осуществляющая обработку части параллельного цикла на конкретном устройстве. Она принимает в качестве аргументов описатель устройства и части параллельного цикла. Обработчик запрашивает порцию для исполнения (границы циклов и шаг), конфигурацию параллельной обработки (количество нитей), запрашивает инициализацию редукционных переменных, а после выполнения порции - передаёт результат частичной редукции в систему поддержки. В случае CUDA-обработчика, он для обработки частей цикла вызывает специальным образом сгенерированное ядро на языке Fortran CUDA. CUDA-ядро выполняется на GPU, производя вычисления, составляющие тело цикла.

По умолчанию, предполагается, что регион может исполняться на всех типах вычислителей и компилятор генерирует обработчики для хост-машины и CUDA-вычислителя. Пользователь может указать посредством клаузы TARGETS директивы REGION, на каких вычислителях предполагается исполнять регион. Согласно его указаниям компилятор генерирует тот или иной обработчик.

Для взаимодействия между узлами система поддержки использует библиотеку MPI.

Основная работа по реализации модели выполнения параллельной программы (например, распределение данных и вычислений) осуществляется динамически. Это позволяет обеспечить динамическую настройку DVMH-программ при запуске (без перекомпиляции) на конфигурацию параллельного компьютера (количество процессоров, ускорителей, их производительность и тип, а также латентность и пропускную способность коммуникационных каналов). Тем самым программист получает возможность иметь один вариант программы для выполнения на последовательных ЭВМ и параллельных ЭВМ различной конфигурации.

2.4 Пример программы Якоби на языке Fortan DVMH

Проиллюстрируем возможности языка Fortan DVMH на примере программы для алгоритма Якоби (см. **Рис. 1**).

```
      PROGRAM JAC
      PARAMETER (L=8, ITMAX=10)
      REAL A(L,L), EPS, MAXEPS, B(L,L)
!DVM$ DISTRIBUTE ( BLOCK, BLOCK ) :: A
!DVM$ ALIGN  B(I,J) WITH A(I,J)
!     arrays A and B  with block distribution
      PRINT *,  '**********  TEST_JACOBI   **********'
      MAXEPS  =  0.5E - 7
!DVM$ REGION
!DVM$ PARALLEL (J,I) ON A(I, J)
!     nest of two parallel loops, iteration (i,j) will
!     be executed on device, which is owner of element A(i,j)
      DO J = 1, L
        DO I = 1, L
          A(I, J) = 0.
          IF(I.EQ.1 .OR. J.EQ.1 .OR. I.EQ.L .OR. J.EQ.L) THEN
             B(I, J) = 0.
          ELSE
             B(I, J) = ( 1. + I + J )
          ENDIF
        END DO
      END DO
!DVM$ END REGION
      DO IT = 1, ITMAX
         EPS = 0.
!DVM$    REGION
!DVM$    PARALLEL (J, I) ON A(I, J), REDUCTION ( MAX( EPS ))
!        variable EPS is used for calculation of maximum value
         DO J = 2, L-1
           DO I = 2, L-1
              EPS = MAX ( EPS, ABS( B( I, J) - A( I, J)))
              A(I, J) = B(I, J)
           END DO
         END DO
!DVM$    PARALLEL (J, I) ON B(I, J), SHADOW_RENEW (A)
         DO J = 2, L-1
           DO I = 2, L-1
             B(I, J) = (A( I-1, J ) + A( I, J-1 ) + A( I+1, J) + A( I,
J+1 )) / 4
           END DO
         END DO
!DVM$    END REGION
!DVM$    GET_ACTUAL(EPS)
         PRINT 200,  IT, EPS
200      FORMAT(' IT = ',I4, '   EPS = ', E14.7)
         IF ( EPS . LT . MAXEPS ) EXIT
      END DO
!DVM$ GET_ACTUAL(B)
      OPEN (3, FILE='JAC.DAT', FORM='FORMATTED', STATUS='UNKNOWN')
      WRITE (3,*)   B
      CLOSE (3)
      END
```

Рис. 1. Программа Якоби на языке Fortan DVMH

В результате выполнения директивы
!DVM$ DISTRIBUTE (BLOCK, BLOCK) :: A

массив A будет распределен между вычислителями. Количество и тип используемых вычислителей задается при запуске программы с помощью переменных окружения и параметров командной строки.

Директива

!DVM$ ALIGN B(I,J) WITH A(I,J)

задает совместное распределение двух массивов A и B. Элементы массива B будут распределены на тот же вычислитель, где будут размещены соответствующие элементы массива A.

Директива

!DVM$ PARALLEL (J,I) ON A(I, J)

задает распределение вычислений. Витки цикла будут выполняться на том вычислителе, где распределены соответствующие элементы массива A.

Клауза REDUCTION (MAX(EPS)) организует эффективное выполнение редукционной операции - глобальной операции с расположенными на различных вычислителях данных (нахождение максимального значения).

Клауза SHADOW_RENEW (A) указывает на необходимость подкачки удаленных данных (теневых граней) с других вычислителей перед выполнением цикла.

Поскольку никакие дополнительные клаузы в директивах REGION не заданы, компилятор определяет направления использования переменных автоматически - INOUT(A,B,EPS).

При выполнении первого вычислительного региона (цикла инициализации) для распределенных частей массивов A и B на ускорителях будет выделена необходимая пямять.

При входе во второй вычислительный регион (в итерационном цикле) осуществляется проверка: присутствуют ли актуальные представители для массивов A и B на вычислителе? Поскольку такие представители уже присутствуют, то никакие дополнительные операции копирования актуальных данных на вычислители не выполняются.

При выходе из вычислительного региона обновление данных в памяти хоста не производится. Перед выводом массива B в файл, требуется скопировать последние изменения массива из памяти вычислителя при помощи директивы GET_ACTUAL(B).

С использованием языка Fortran DVMH были разработаны прикладные программы решения задач гидродинамики.

3. Разработка параллельных программ на языке Fortran DVMH для задач гидродинамики «Каверна» и «Контейнер»

3.1 Задача «Каверна»

Программа «Каверна» предназначена для моделирования циркуляционного течения в плоской квадратной каверне с движущейся верхней крышкой в двумерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода.

Последовательная версия программы занимает 496 строк.

В ходе разработки параллельной программы для данной задачи были проведены следующие действия:

1) Добавлены директивы распределения данных:

CDVM$ DISTRIBUTE ro(BLOCK,BLOCK)

CDVM$ ALIGN (i,j) WITH ro(i,j) :: ux, uy, p, E, ro1, ux1, uy1, E1, p1

CDVM$ ALIGN (i,j) WITH ro(i,j) :: SFro, SFux, SFuy, SFE, tmp1, tmp2

CDVM$ ALIGN (i) WITH ro(*,*) :: hx,hy

2) Вставлены директивы PARALLEL перед 28-ю гнездами циклов. Из них:
   а) 8 параллельных циклов имеют спецификацию PRIVATE;
   б) 2 цикла спецификацию REDUCTION;
   в) 7 циклов спецификацию SHADOW_RENEW.
3) Вставлены директивы начала и конца вычислительного региона в 7-ми местах программы.
4) Вставлены директивы объявления данных актуальными в 6-ти местах программы.
5) Вставлены директивы запроса актуальных данных в 5-ти местах программы.

6) Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным (данным, не расположенным на устройстве, которое должно выполнить оператор)

7) Для того чтобы виток цикла целиком мог выполняться на одном устройстве, в 4-х местах программы циклы были разбиты на два.

8) В 4-х местах программы сделаны тесно-гнездовые циклы.

9) Для 6-ти гнезд циклов изменен порядок вычисления витков циклов, что позволило обрабатывать элементы массивов согласно их расположению в памяти ЭВМ.

10) Устранены OUTPUT зависимости между витками 4-х циклов.

Таким образом, было изменено 45 строк (или 9% от количества строк последовательной программы), добавлено 117 строк (или 23,5% от количества строк последовательной программы), текст параллельной программы занимает 613 строк.

3.2 Задача «Контейнер»

Программа Контейнер предназначена для численного моделирования течения вязкой тяжелой жидкости под действием силы тяжести в прямоугольном контейнере с открытой верхней стенкой и отверстием в одной из боковых стенок в трехмерной постановке в широком диапазоне как параметров задачи, так и параметров численного метода. Последовательная версия программы занимает 828 строки.

В ходе разработки параллельной программы для данной задачи были проведены следующие действия.

1) Добавлены директивы распределения данных:

```
CDVM$ DISTRIBUTE ro(BLOCK,BLOCK,BLOCK)
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: ux, uy, uz, p, E
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: ro1, ux1, uy1,uz1, p1, E1
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: SFro, SFux, SFuy, SFuz, SFE
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: F1x, F2x, F1y, F2y, F1z, F2z
CDVM$ ALIGN (i,j,k) WITH ro(i,j,k) :: F3x, F3y, F3z
```

2) Вставлены директивы PARALLEL перед 21-м гнездом циклов. Из них:
   а) 9 параллельных циклов имеют спецификацию PRIVATE;
   б) 4 цикла спецификацию REDUCTION;
   в) 5 циклов спецификацию SHADOW_RENEW.
3) Вставлены директивы начала и конца вычислительного региона в 5-ти местах программы.
4) Вставлены директивы объявления данных актуальными в 4-х местах программы.
5) Вставлены директивы запроса актуальных данных в 3-х местах программы.
6) Вставлена одна директива REMOTE_ACCESS для доступа к удаленным данным.
7) Для того чтобы виток цикла целиком мог выполняться на одном устройстве, в 3-х местах программы циклы были разбиты на два.
8) В 6-ти местах программы сделаны тесно-гнездовые циклы.
9) Изменен порядок вычисления витков циклов для 12-ти гнезд циклов.
10) Устранены OUTPUT и FLOW зависимости между витками в 1 цикле.

Таким образом, при распараллеливании было изменено 37 строк (или 4,4% от количества строк последовательной программы), добавлено 114 строк (или 13,7% от количества строк последовательной программы), текст параллельной программы занимает 942 строки.

Для разработанных параллельных программ было проведено исследование эффективности.

4. Анализ эффективности разработанных на языке Fortran DVMH параллельных программ при запусках на большом числе узлов и GPU

В следующих подразделах приводятся времена выполнения программ (в секундах), которые были получены на суперкомпьютерном комплексе МГУ «Ломоносов»[5]. Для компиляции кода, выполняемого на хосте, использовались компиляторы Intel версии 13.0, для компиляции кода, выполняемого на ускорителях, использовался компилятор CUDA Fortran компании Port-

land Group версии 12.9 и NVIDIA CUDA C версии 4.0. Для взаимодействия между узлами использовалась библиотека Intel MPI версии 4.0.3.

4.1 Программа «Каверна»

Ускорение выполнения программы «Каверна» на одном GPU по сравнению с выполнением на 1 ядре центрального процессора в зависимости от размера сетки было опубликовано в [4].

В таблицах 1 и 2 приведены времена выполнения 200 итераций программы «Каверна» на сетке 3200x3200 на разном числе ядер и GPU.

Таблица 1. Время выполнения программы «Каверна» на сетке 3200x3200 на разном числе ядер

1	2	4	8	16	32	64	128	256	400	512	1024
1241,83	631,47	332,36	182,95	100,05	75,8	40,20	21,33	11,74	7,11	6,44	3,48

Таблица 2. Время выполнения программы «Каверна» на сетке 3200x3200 на разном числе GPU

1	2	4	8	16	32	64	128	256	400
73,07	39,34	19,94	11,65	7,17	4,80	3,96	3,45	3,32	3,19

При использовании 1024 ядер программа «Каверна» ускорилась в 357 раз по сравнению с выполнением на 1 ядре. При использовании 1 ускорителя программа ускоряется в 17 раз по сравнением с выполнением программы на 1 ядре. Максимальное ускорение, полученное с использованием ускорителей - 390 раз по сравнением с выполнением программы на 1 ядре.

4.2 Программа «Контейнер»

В таблицах 3 и 4 приведены времена выполнения программы «Контейнер» на разном числе ядер и GPU.

Таблица 3. Время выполнения программы «Контейнер» на разном числе ядер

Сетка, количество итераций	4	8	16	32	64	128	256	512	1024	2048
200x200x200 itmax=200	754,01	384,92	206,47	113,64	49,87	29,90	14,52	8,63	5,63	6,01
400x400x400 itmax=100	-	1202,32	630,15	317,36	164,02	85,68	43,10	22,53	13,54	7,66
800x800x800 itmax=50	-	-	-	-	576,16	318,75	151,78	79,68	41,26	21,91
1600x1600x1600 itmax=20	-	-	-	-	-	-	-	235,64	117,88	62,28

Таблица 4. Время выполнения программы «Контейнер» на разном числе GPU

Сетка, количество итераций	1	2	4	8	16	32	64	128	256	512	1024	1280
200x200x200 itmax=200	166,95	86,05	45,77	26,82	14,95	8,99	6,12	3,99	3,26	3,01	3,60	4,32
400x400x400 itmax=100			168,80	89,17	47,15	26,09	14,17	8,39	4,86	3,20	2,88	3,26
800x800x800 itmax=50						92,20	51,80	30,32	13,58	7,56	4,67	4,17
1600x1600x1600 itmax=20									37,38	20,14	10,74	8,95

Для сеток 200x200x200 и 400x400x400 при использовании большого числа графических процессоров задача перестает ускоряться и даже замедляется. Это связано с тем, что при увеличении числа используемых GPU существенно сокращается объем данных, обрабатываемых на одном GPU, что не позволяет полностью загрузить аппаратуру. Накладные расходы на подготовку и запуск вычислительных ядер, копирование теневых граней превышают эффект от распараллеливания программы.

Для сетки 200x200x200 при использовании 4 GPU программа ускоряется 16,47 раз по сравнению с выполнением на 4-х ядрах.

Для сетки 400x400x400 при использовании 8 GPU программа ускоряется 13,48 раз по сравнению с выполнением на 8-х ядрах.

Для сетки 800x800x800 при использовании 64 GPU программа ускоряется 11,12 раз по сравнению с выполнением на 64-х ядрах.

Для сетки 1600x1600x1600 при использовании 512 GPU программа ускоряется 11,7 раз по сравнению с выполнением на 512-х ядрах.

Одним из факторов при выборе задач «Каверна» и «Контейнер» для распараллеливания на языке Fortran DVMH было наличие у этих программ разработанных версий в модели SHMEM/CUDA. Данные об ускорении этих программ, полученные при использовании GPU, были опубликованы еще в 2010 году [6].

Было проведено сравнение эффективности параллельных программ в модели DVMH и модели SHMEM/CUDA. Для этого использовался следующий подход. Осуществлялся запуск исходной задачи на 1-м GPU (сетка 150x150x150), замерялось время ее выполнения. Затем в 2 раза увеличивалась сложность решаемой задачи (размер вычислительной сетки) и задача запускалась на 2 раза большем числе GPU и т.д. В таблицах 5 и 6 приведены времена выполнения 200 итераций SHMEM/CUDA и DVMH-версиий программы «Контейнер» на разном числе GPU.

Таблица 5. Время и эффективность выполнения SHMEM/CUDA-программы «Контейнер» на разном числе GPU

Число GPU	1	2	4	8	16	32	64	128	256	512	1024
время, с	87,12	87,824	88,8	89,296	90,216	90,992	91,496	91,576	91,976	92,468	92,74
Эффективность	100,0%	99,2%	98,1%	97,6%	96,6%	95,7%	95,2%	95,1%	94,7%	94,2%	93,9%

Таблица 6. Время и эффективность выполнения DVMH-программы «Контейнер» на разном числе GPU

Число GPU	1	2	4	8	16	32	64	128	256	512	1024
время, с	71,93	74,77	76,12	76,75	80,56	80,76	82,76	82,91	82,03	90,56	88
Эффективность	100,0%	96,2%	94,5%	93,7%	89,3%	89,1%	86,9%	86,8%	87,7%	79,4%	81,7%

Современные графические процессоры позволяют настраивать режим работы кэша L1 для каждого SM. По умолчанию 16 KB используется для L1, а 48 KB - для общей памяти. В системе поддержки выполнения DVMH-программ задан режим cudaDeviceSetCacheConfig(cudaFuncCachePreferL1), в котором 48 KB используется для кэша L1, а 16 KB - для общей памяти. Разработчики SHMEM/CUDA версии программы не учли эту возможность. В результате DVMH-программа выполняется на 1-ом GPU в 1,2 раза быстрее чем SHMEM/CUDA-программа.

При увеличении числа GPU эффективность DVMH-программы падает. Одна из причин – «лишние» обмены теневыми гранями. Обмен теневыми гранями – это достаточно дорогостоящая операция: необходимо скопировать требуемые теневые грани из памяти ускорителя в память хоста, запустить соответствующие обмены между узлами кластера, а затем скопировать полученные значения в память ускорителя. При определенных условиях можно обновить теневые грани за счет дополнительных вычислений. Такой механизм реализован для DVM-программ (SHADOW_COMPUTE). В настоящее время ведется доработка компилятора Fortran DVMH и системы поддержки выполнения программ для реализации такой возможности при использовании ускорителей.

5. Выводы

Появление новых гетерогенных и гибридных компьютерных архитектур, в частности, на основе многоядерных вычислительных ускорителей, позволило значительно повысить производительность суперкомпьютеров, что сделало актуальным разработку и оптимизацию прикладного программного обеспечения для соответствующих вычислительных систем.

Оценивая современное состояние методов разработки эффективных приложений для высокопроизводительных систем, следует отметить, что имеющиеся средства программирования

являются по своей сути низкоуровневыми и требуют значительных затрат от разработчика, без гарантии достижения требуемого уровня качества создаваемого прикладного обеспечения. Под качеством здесь в первую очередь понимается сокращение времени решения прикладных задач без потери точности их решения, а также простота сопровождения ПО и его переноса на новые архитектуры.

Разработанный в Институте прикладной математики им. М.В. Келдыша РАН подход к созданию прикладного программного обеспечения существенно упрощает создание прикладных программ для суперкомпьютерных систем с ускорителями. Язык Fortran DVMH обеспечивает высокий уровень переносимости прикладного ПО на системы с другими архитектурами графических процессоров, поскольку перенос не требует изменения программы.

Проведенное исследование характеристик разработанных приложений «Каверна» и «Контейнер» показало, что эффективность программ, разработанных в высокоуровневой гибридной модели DVMH, очень мало отличается от эффективности программ, написанных с использованием низкоуровневой технологии CUDA.

Литература

1. DVM-система [Электронный ресурс] - : [web-сайт] - Режим доступа: http://www.keldysh.ru/dvm. - 01.12.2012
2. CUDA [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.nvidia.ru/object/what_is_cuda_new_ru.html. – 01.12.2012.
3. CUDA Fortran. Programming Guide and Reference. Release 2012. [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.pgroup.com/lit/whitepapers/pgicudaforug.pdf. – 01.12.2012.
4. Бахтин В. А., Давыдов А. А., Крюков В. А., Четверушкин Б. Н., Шильников Е. В. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. //ДОКЛАДЫ АКАДЕМИИ НАУК, 2011, том 441, № 6, С. 734–736.
5. Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы. - Москва: Издательский дом "Открытые системы", 2012. – 7.
6. Давыдов А.А., Четверушкин Б.Н., Шильников Е.В. // Моделирование течений несжимаемой жидкости и слабосжимаемого газа на многоядерных гибридных вычислительных системах. Ж. Вычисл. матем. и матем. физ. – 2010. – Т. 50, № 12. – С. 2275–2284.

Моделирование на суперЭВМ динамики плазменных электронов в ловушке с инверсными магнитными пробками и мультипольными магнитными стенками*

Е.А. Берендеев[1], А.В. Иванов[2], Г.Г. Лазарева[3], А.В. Снытников[3]

Новосибирский государственный университет[1], Институт ядерной физики СО РАН[2], Институт вычислительной математики и математической геофизики СО РАН[2]

Рассмотрена задача моделирования динамики плазменных электронов в ловушке с инверсными магнитными пробками и мультипольными магнитными стенками. Модель построена на основе модифицированного метода частиц в ячейках. Сложный характер исследуемых процессов и необходимая высокая точность потребовали разработки параллельного алгоритма, позволяющего за разумное время рассчитывать траектории миллиардов частиц. Для равномерной и полной загрузки вычислительных узлов выполнена смешанная эйлерово-лагранжевая декомпозиция с учётом динамического шага по времени. Такой подход позволяет достичь высокой масштабируемости параллельного алгоритма и существенно ускорить вычисления.

1. Введение

Наиболее эффективным методом получения мощных нейтральных пучков для установок управляемого термоядерного синтеза является нейтрализация пучков отрицательных ионов в плазменной ловушке-мишени. В ИЯФ СО РАН предложена линейная осесимметричная ловушка с инверсными пробками (с обратным магнитным полем) для нейтрализации пучка отрицательных ионов[1]; для проведения экспериментальных исследований создана ловушка мишенной плазмы длиной 130 см с апертурой 10 см. Проект направлен на решение проблемы минимизации потерь плазмы в широко апертурные проходные отверстия в торцах, в которых находятся инверсные магнитные пробки, а также через цилиндрические мультипольные магнитные стенки ловушки на её вакуумную камеру.

Полноценное исследование физических процессов в плазме может быть проведено только при комплексном подходе, сочетающем как экспериментальные исследования, так и исследования вычислительными методами, адекватно описывающими эти процессы. Для того, чтобы избежать упрощений и получить качественно правильную физическую картину, необходимо построить максимально полную математическую модель.

Общепринято, что хорошей исходной моделью полностью ионизированной плазмы является система уравнений, состоящая из уравнения Больцмана для функций распределения ионов и электронов [2]

$$\frac{\partial f_\alpha}{\partial t}+\vec{v}\frac{\partial f_\alpha}{\partial \vec{r}}+\vec{F}_\alpha\frac{\partial f_\alpha}{\partial \vec{p}}=St\{f_\alpha\},\ \vec{F}_\alpha=q_\alpha(\vec{E}+\frac{1}{c}[\vec{v},(\vec{H}+\vec{H}_0)]), \tag{1.1}$$

и системой уравнений Максвелла с самосогласованными электромагнитными полями

$$rot\vec{H}=\frac{4\pi}{c}\vec{j}+\frac{1}{c}\frac{\partial\vec{E}}{\partial t}=\frac{4\pi}{c}\sum_\alpha q_\alpha\int f_\alpha\vec{v}d\vec{v}+\frac{1}{c}\frac{\partial\vec{E}}{\partial t}, \tag{1.2}$$

$$rot\vec{E}=-\frac{1}{c}\frac{\partial\vec{H}}{\partial t}, \tag{1.3}$$

$$div\vec{E}=4\pi\rho=4\pi\sum_\alpha q_\alpha\int f_\alpha d\vec{v}, \tag{1.4}$$

* Работа выполнена при поддержке Интеграционного Проекта СО РАН № 105 и грантов РФФИ № 11-01-00178, 11-01-00249, 12-07-00065.

$$div\vec{H} = 0\,. \qquad (1.5)$$

Здесь индексом α обозначается сорт частиц (ионы и электроны); $f_\alpha(\vec{r},\vec{v},t)$ - функция распределения частиц сорта α; q_α - заряд частицы; $\vec{j}$ - плотность тока ρ - плотность заряда; $\vec{E}$ - напряжённость электрического поля; $\vec{H}$ - напряжённость магнитного поля; $\vec{H}_0$ - магнитное поле ловушки; $St\{f_\alpha\}$ - функция, описывающая следующие физические процессы:

- ионизация атома водорода,
- ионизация и диссоциация молекулы H_2
- диссоциативное возбуждение и диссоциативная рекомбинация H^{2+}
- диссоциативная рекомбинация D^{2+}
- перезарядка протонов на атомах водорода.

Наиболее универсальным и широко применяемым методом для решения этих уравнений является метод частиц в ячейках [3]. Общая схема этого метода состоит в том, что плазма представляется набором достаточно большого числа модельных частиц, движущихся в соответствии с законами классической механики в самосогласованном электромагнитном поле. При этом каждая модельная частица характеризует движение многих реальных частиц и становится носителем некоторого набора характеристик среды, таких как заряд, масса, импульс, кинетическая энергия и т.д. Для точного описания физических эффектов, происходящих во всей ловушке, необходимо использовать до 10^9-10^{13} модельных частиц и 10^6-10^9 узлов сетки (в двумерном случае). Несмотря на внутренний параллелизм метода частиц – траектории модельных частиц могут быть вычислены независимо друг от друга – построение параллельного масштабируемого алгоритма представляет собой нетривиальную задачу и зависит от особенностей рассматриваемых физических процессов[4].

Одной из таких особенностей является сложная конфигурация магнитного поля в ловушке. На рисунке 1 показана геометрия кольцевых магнитов с железными экранами с магнитным полем, представляющим ловушку мишенной плазмы. При этом магнитное поле ловушки в отдельных областях меняется в пределах величины 50 Гс – 7 кГс.

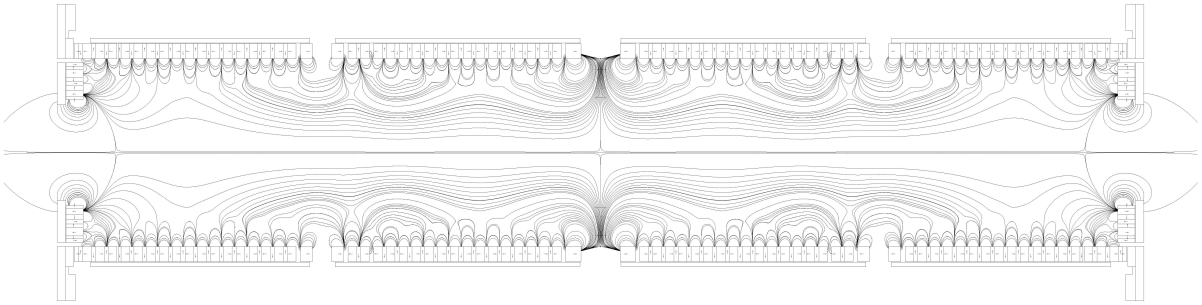


Рис. 1. Геометрия магнитной системы плазменной ловушки-мишени и силовые линии магнитного поля.

Такая разница в величине магнитного поля (до 2-х порядков) существенно влияет на ларморовский радиус частиц, что в свою очередь приводит к уменьшению временного шага при вычислении траекторий. В настоящей работе также рассматривается проблема адаптивного шага по времени для частиц, находящихся в различных областях ловушки с учётом распределения частиц на вычислительные ядра.

2. Решение основных уравнений

В осесимметричной ловушке с кольцевым магнитным полем отсутствует азимутальный компонент поля, а также отсутствует стационарное азимутальное электрическое поле. Соответственно, в такой ловушке не может возникать нормальный к стенкам стационарный дрейф плазмы в скрещенных полях. Благодаря этому, наиболее оптимальной является двумерная постановка задачи в цилиндрических R-Z координатах, с учётом всех трёх компонент скоростей и импульсов частиц.

2.1 Решение уравнения Больцмана

Решение уравнения Больцмана, используя метод расщепления, можно свести к решению уравнения Власова

$$\frac{\partial f_\alpha}{\partial t} + \vec{v}\frac{\partial f_\alpha}{\partial \vec{r}} + \vec{F}_\alpha \frac{\partial f_\alpha}{\partial \vec{p}} = 0\,, \tag{2.2.1}$$

и корректировке траекторий частиц с учётом процессов ионизации и диссипации, используя методы Монте-Карло[5].

$$\frac{Df_\alpha}{Dt} = St\{f_\alpha\} \tag{2.2.2}$$

Решение уравнения Власова производится в лагранжевых координатах – характеристики этого уравнения описывают движения модельных частиц:

$$\frac{d\vec{p}}{dt} = q_\alpha(\vec{E} + \frac{1}{c}[\vec{v},\vec{H}])\,,\ \frac{d\vec{r}}{dt} = \vec{v}\,. \tag{2.2.3}$$

Здесь $\vec{H}$ включает в себя самосогласованное магнитное поле частиц и внешнее поле ловушки.

Для того, чтобы избежать особенностей у оси симметрии, в настоящей работе используется схема Бориса [6] – решение уравнения Власова в декартовых координатах с последующим локальным преобразованием решения в цилиндрические координаты. В этом случае для определения траектории частиц в декартовых координатах можно использовать следующую схему:

$$\frac{\vec{p}_i^{\,m+1/2} - \vec{p}_i^{\,m-1/2}}{\tau} = q_\alpha\left(\vec{E}_i^{\,m} + \frac{1}{\mathrm{c}}\left[\frac{\vec{v}_i^{\,m+1/2} + \vec{v}_i^{\,m-1/2}}{2}, \vec{H}_i^{\,m}\right]\right), \tag{2.2.4}$$

Здесь τ - шаг по времени; верхний индекс указывает на момент времени, в который вычисляется искомая функция. Индекс i указывает на номер частицы, для которой производятся вычисления.

2.2 Решение уравнений Максвелла

Уравнения Максвелла решаются в эйлеровых переменных. Необходимые для их решения плотности заряда и плотности тока вычисляются по скоростям и координатам отдельных частиц:

$$\rho(\vec{r},t) = \sum_j q_j R(\vec{r},\vec{r}_j(t))\,, \tag{2.2.1}$$

$$\vec{j}(\vec{r},t) = \sum_j q_j \vec{v}_j(t) R(\vec{r},\vec{r}_j(t))\,. \tag{2.2.2}$$

Здесь q_j - заряд частицы с номером j; функция $R(\vec{r},\vec{r}_j(t))$ (функция ядра) характеризует форму, размер частицы и распределение в ней заряда.

В настоящей работе плотности тока и плотности заряда вычисляются по формулам, аналогичным [7], но адаптированным для цилиндрической системы координат. При таком подходе разностный аналог уравнения (1.4) выполняется автоматически, что позволяет существенно ускорить вычисления.

Для нахождения электрических и магнитных полей используется схема, предложенная Лэнгдоном и Лазинским в 1976 году [8], в которой поля определяются из разностных аналогов законов сохранения Фарадея и Ампера:

$$\frac{\vec{H}^{m+1/2} - \vec{H}^{m-1/2}}{\tau} = -c\,rot_h\vec{E}^{\,m}\,, \tag{2.2.3}$$

$$\frac{\vec{E}^{m+1} - \vec{E}^{m}}{\tau} = -4\pi\,\vec{j}^{\,m+1/2} + c\,rot_h\vec{H}^{\,m+1/2}\,. \tag{2.2.4}$$

Таким образом, схема решения задачи разбивается на три этапа. На первом (лагранжневом) этапе по схеме (2.2.4) вычисляются скорости и координаты частиц. Здесь же определяются компоненты плотности тока $\vec{j}^{m+1/2}$ и плотности заряда ρ^{m+1}. На втором этапе происходит корректировка траекторий частиц, добавляются и удаляются частицы, в соответствии с учётом процессов ионизации и диссипации, используя методы Монте-Карло. Также удаляются частицы, покинувшие ловушку. На третьем (эйлеровом) этапе по схеме (2.2.3)-(2.2.4) решаются уравнения Максвелла, т.е. определяются значения функций $\vec{H}^{m+1/2}$ и $\vec{E}^{m+1}$ в узлах сетки. Значения электрических и магнитных полей, действующих на каждую частицу, вычисляются с помощью билинейной интерполяции.

2.3 Адаптивное изменение масс частиц

Поскольку на каждом шаге возможно как рождение новых частиц, так и удаление частиц, покинувших ловушку, необходимо контролировать локальное изменение плотности в каждой ячейке. Проблема заключается в том, что рекомбинирующих реальных частиц существенно меньше, чем реальных частиц, соответствующих одной модельной частице, поэтому удаление или добавление одной модельной частицы может привести к существенному нефизическому изменению плотности в ячейке сетки. Для того, чтобы избежать подобных нежелательных явлений, в настоящей работе предложена модификация метода частиц, основанная на адаптивном изменении массы.

Для каждого сорта частиц вводится константа $s = \frac{q}{m}$ - отношение заряда к массе частицы. Частица хранит заряд, пропорциональный плотности вещества в данной ячейке. Создание частиц происходит по следующему алгоритму [9]:

- вычисляется полная масса в ячейке
- вычисляется средняя скорость
- добавляются новые частицы с необходимым разбросом по скоростям (средняя скорость равна нулю)
- вычисляется масса одной частицы путём деления полной массы на полное число частиц (и старых и новых). Эта масса присваивается каждой частице в ячейке.
- к случайным скоростям новых частиц добавляется средняя скорость частиц в ячейке.

Удаляются частицы аналогичным образом.

3. Параллельная реализация алгоритма

Каждый временной шаг работы программы состоит из следующих действий:

- расчёт электрического и магнитного поля
- расчёт движения модельных частиц
- вычисления новых значений плотности тока и заряда, вычисление интеграла столкновений
- изменение масс частиц

Время работы всех этих процедур было измерено с помощью профилировщика *gprof*. Результаты профилировки представлены на рисунке 2.

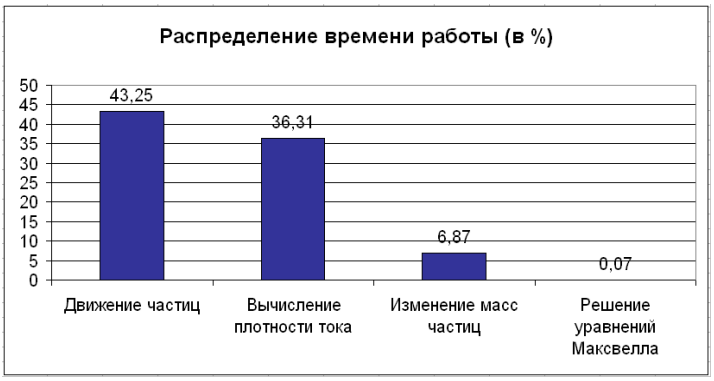


Рис. 2. Время работы основных процедур программ. Сетка 128x128, 200 частиц в ячейке.

Из рисунка 2 видно, что основное время работы программы занимают операции с частицами – расчёт движения частиц, вычисление плотности тока для каждой частицы. Исходя из этого, можно предложить несколько схем распараллеливания - равномерное распределение частиц по процессорам, эйлерова декомпозиция области и распределение частиц по процессорам в зависимости от их положения, смешанная эйлерово-лагранжевая декомпозиция, распределение частиц в зависимости от времени расчёта одного шага на каждом процессоре. В работе [4] показано преимущество эйлерово-лагранжевой декомпозиции области (рисунок 3) в случае постоянного шага по времени.

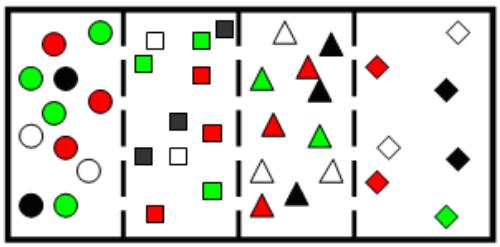


Рис. 3. Эйлерово-лагранжевая декомпозиция области. Область решения разбита вдоль координаты Y на несколько подобластей, частицы каждой подобласти равномерно распределены между процессорами отдельной группы независимо от координаты. Различные символы, обозначающие частицы: круг, квадрат, треугольник, ромб означают принадлежность частиц к разным группам процессоров, цвет фигуры выделяет принадлежность к разным процессорам в группе. В данном примере используется 16 процессоров

Однако, постоянный временной шаг будет обусловлен максимальной величиной магнитного поля в ловушке. Поскольку разница в величине магнитного поля в разных областях ловушки может составлять до 10^2, то это ведёт к существенному замедлению работы программы - приходится вычислять плавные траектории частиц в областях малой величины магнитного поля с мелким шагом.

В настоящей работе предлагается использовать динамический шаг для разных областей ловушки. Это решение накладывает следующее ограничение на распределение частиц по процессорам – на каждый процессор приходится порядка

$$\frac{\sum_i \frac{N_i}{t_i}}{N_p}, \qquad (3.1)$$

где N_i - число частиц в ячейке i, t_i - время расчёта траектории одной частицы в ячейке i (одинаковое для всех частиц данной ячейке), N_p - общее число процессоров. Такое распределение соответствует распределению «по времени» (фактически динамическая эйлерова декомпозиция области), описанному А.Н. Андриановым и К.Н. Ефимкиным (ИПМ им. М.В. Келдыша) в работе [10] , но с учётом динамического шага по времени. Представив общее число про-

цессоров N_p из формулы (3.1) в виде произведения числа групп N_g процессоров на число процессоров N_{pg} в группе, можно получить следующее число частиц на группу процессоров

$$\frac{\sum_i \frac{N_i}{t_i}}{N_g}. \qquad (3.2)$$

Распределив частицы внутри группы равномерно, можно получить эйлерово-лагранжевую декомпозицию области. Это позволяет избежать ситуации, когда на отдельный процессор приходится только несколько ячеек и начинают преобладать межпроцессорные коммуникации.

Поскольку величина магнитного поля ловушки существенно превосходит величину магнитного поля, создаваемого частицами при движении, декомпозицию области можно получить экспериментальным путём.

4. Вычислительный эксперимент

4.1 Эффективность распараллеливания

Поскольку декомпозиция области была получена исходя из вычислительных экспериментов, необходимо проверить масштабируемость полученной реализации алгоритма.

Расчёты проводились на суперкомпьютере «Ломоносов» (МГУ).

На рисунке 4 представлено ускорение при использовании различного количества процессорных ядер. В связи с большим объёмом требуемой оперативной памяти, масштабируемость рассматривается относительно 128 и 1024 процессорных ядер. Использование меньшего количества процессорных ядер не представляет собой интереса с точки зрения реальных задач.

Параметры задачи:

Число узлов 1024x1024, декомпозиция области вдоль направления Y на 64 подобласти

Число частиц в ячейке для тестирования масштабируемости относительно 128 процессорных ядер 2500 (Всего 1 310 720 000 частиц)

Число частиц в ячейке для тестирования масштабируемости относительно 1024 процессорных ядер 5000 (Всего 5 242 880 000 частиц)

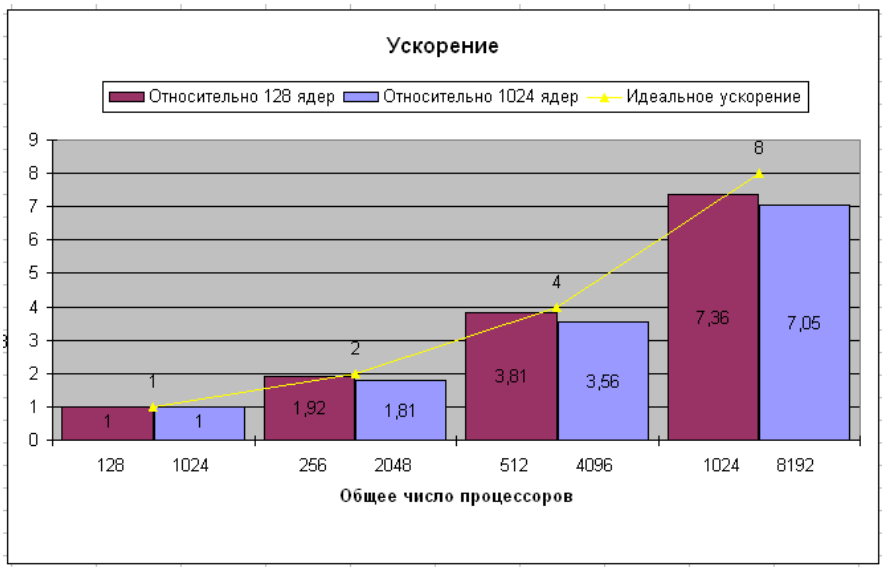


Рис.4. Полученное ускорение, относительно 128 и 1024 процессорных ядер

Как видно из рисунка 4, полученный алгоритм хорошо масштабируется до 8192 процессорных ядер. Однако, при проведении серии расчётов на таком количестве процессорных ядер критическим фактором становится отказоустойчивость – из-за специфики суперкомпьютера «Ломоносов», отказ или существенное замедление межпроцессорных коммуникаций одного

вычислительного узла, связанные с особенностями физической организации суперкомпьютера, приводят к прекращению работы всей программы. Поскольку в расчётах используется порядка 0,5 – 1,5 Тб оперативной памяти, сохранение информации на жёсткий диск в виде «контрольных точек» (как отмечено в [10]) не представляется возможным. В дальнейшем планируется проведение испытаний на других суперкомпьютерах, а также использование графических ускорителей для снятия части нагрузки с универсальных вычислительных ядер.

4.2 Результаты расчётов траекторий плазменных электронов в ловушке

Расчёт траекторий модельных частиц производился при следующих физических и модельных параметрах:

температура плазмы 5 эВ, размер области 6.1см x 1.2 см, плотность электронов (ионов) $2x10^{13}$ $см^{-3}$, Сетка 4096x128 узлов, общее число модельных частиц 5 242 880 000. Расчёты проводились на суперкомпьютере «Ломоносов» с использованием до 8192 процессорных ядер. Среднее время расчёта одного шага – 0,326 с., среднее время расчёта всей задачи – 24 часа.

На рисунке 5 траектории некоторых электронов плазмы, без учёта влияния ионизации и диссипации.

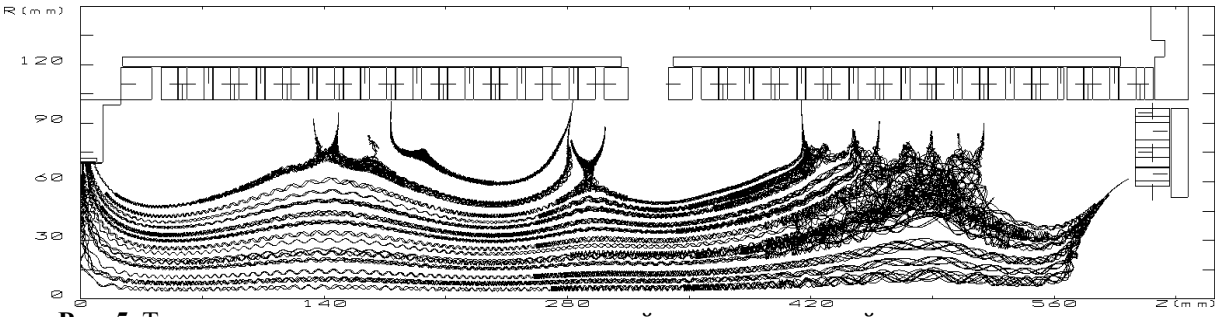


Рис. 5. Траектории движения электронов мишенной плазмы под воздействием магнитного поля.

Из рисунка 5 видно, что инверсные магнитные пробки на торцах достаточно хорошо удерживают плазму в ловушке, в то же время присутствуют потери плазмы на стенках ловушки и границе инверсных пробок. Для количественного описания потерь плазмы в дальнейшем будет учтено влияние рассеяния электронов плазмы.

4. Заключение

В работе показано, что использование современных суперЭВМ позволяет успешно решить поставленную физическую задачу. Построенный параллельный алгоритм достаточно хорошо масштабируется до десятка тысяч вычислительных ядер и учитывает баланс нагрузки на процессоры. Несмотря на то, что даже в двумерном случае необходимы огромные вычислительные затраты для расчётов траекторий миллиардов модельных частиц, удалось качественно оценить потери плазмы на стенках ловушки и границе инверсных пробок. В вычислительном эксперименте была рассмотрена уменьшенная модель экспериментальной ловушки с соотношением сторон 1:5. В дальнейшем, при увеличении числа используемых процессорных ядер до 100 000 и использовании графических ускорителей, планируется рассматривать уже всю экспериментальную ловушку.

Коллектив авторов выражает благодарность Вшивкову В.А. и Федоруку М.П.

Литература

1. Dimov G.I. Feasible scenario of startup and burnup of fusion plasma in ambipolar D-T reactor // Transactions of Fusion Science and Technology 2011. Vol. 59, No.1T, P. 208-210
2. Власов А.А. Теория многих частиц. М.-Л., ГИТТЛ, 1950, 348 с.

3. Березин Ю.А., Вшивков В.А. Метод частиц в динамике разреженной плазмы. Новосибирск., «Наука», 1980.
4. Берендеев Е.А., Ефимова А.А. Реализация эффективных параллельных вычислений при моделировании больших задач физики плазмы методом частиц в ячейках // Мат. междунар. конф. «Параллельные вычислительные технологии», Новосибирск, 2012. С. 380-385.
5. Birdsall C.K. Particle-in-Cell Charged-Particle Simulation Plus Monte Carlo Collisions With Neutral Atoms, PIC-MCC// IEEE Trans. Plasma Sci.1991. Vol. 19, No. 2. P. 65-83.
6. Boris J.P. Relativistic plasma simulation – optimization of a hybrid code // Fourth Conference on numerical Simulation of Plasmas. Washington, 1970. P. 3-67.
7. Villasenor J., Buneman O. Rigorous charge conservation for local electromagnetic field solver // Computer Phys. Comm,. 1992, Vol. 69. P. 306-316.
8. Langdon A.B, Lasinski B.F. Electromagnetic and relativistic plasma simulation models // Meth. Comput. Phys. 1976, Vol. 16. P. 327-366.
9. Вшивков В.А., Лазарева Г.Г., Снытников А.В. Адаптивное изменение массы модельных частиц при моделировании тлеющего ВЧ-разряда в Силановой плазме // Вычислительные технологии. 2008, Т. 13., № 1, С. 22-30
10. Андрианов А.Н., Ефимкин К.Н. Подход к параллельной реализации метода частиц в ячейках // Препринт ИПМ им. М.В.Келдыша №9 за 2009 г., Москва.

Библиотека параллельных алгебраических решателей Krylov*

Д.С. Бутюгин[1,2], Я.Л. Гурьева[1], В.П. Ильин[1], Д.В. Перевозкин[1], А.В. Петухов[1], И.Н. Скопин[1]

Институт вычислительной математики и математической геофизики СО РАН[1], Новосибирский государственный университет[2]

Описываются функциональные возможности и особенности программной реализации библиотеки параллельных алгоритмов Krylov, ориентированной на решение больших систем линейных алгебраических уравнений с разреженными симметричными и несимметричными матрицами (положительно определенными и знаконеопределенными), получаемых при сеточных аппроксимациях многомерных краевых задач для систем дифференциальных уравнений на неструктурированных сетках. Библиотека включает двухуровневые итерационные методы в подпространствах Крылова, предобуславливание которых осуществляется на основе сбалансированной декомпозиции расчетной области с различными размерами пересечений подобластей и краевых условий сопряжения на смежных границах. Повышение скорости сходимости итерационных крыловских процессов производится с помощью грубосеточной коррекции, алгоритмов дефляции и неполной факторизации матриц в покомпонентном и блочном вариантах. Программные реализации выполнены на типовых сжатых разреженных форматах матричных данных. Приводятся примеры численных экспериментов с демонстрацией эффективности распараллеливания для характерных плохо обусловленных задач.

1. Введение

Настоящая работа содержит описание функционального наполнения и технологических подходов библиотеки параллельных итерационных алгоритмов Krylov (см. предварительные публикации в [1, 2]), ориентированной на решение больших систем линейных алгебраических уравнений (СЛАУ с числом неизвестных до 10^{10} и выше) с разреженными матрицами, возникающими при конечно-объемных или конечно-элементных (МКО или МКЭ [3, 4]) аппроксимациях многомерных краевых задач для систем дифференциальных уравнений на неструктурированных сетках, на многопроцессорных вычислительных системах (МВС с количеством ядер в десятки и сотни тысяч). В данном случае имеется в виду отсутствие программных ограничений на проведение таких масштабных вычислительных экспериментов, а также достаточно высокая эффективность применяемых алгоритмов.

Основой применяемых в библиотеке Krylov вычислительных подходов являются двухуровневые итерационные процессы в подпространствах Крылова, предобуславливаемые с помощью аддитивного метода Шварца и декомпозиции расчетной области с пересечениями подобластей и разными типами краевых условий на смежных внутренних границах, см. [4 – 7] и цитируемые там работы.

Внешний итерационный процесс осуществляется распределенным по вычислительным узлам образом методом FGMRES [7] с динамическими (в общем случае) предобуславливателями, которые включают решение вспомогательных подсистем в расширенных подобластях с помощью или прямого решателя PARDISO из библиотеки MKL INTEL [8], или авторскими версиями алгоритмов BiCGStab и GMRES [7, 9], предобусловленных с помощью покомпонентной или “мелкоблочной” модификациями Айзенштата неполной факторизации. Для ускорения внешних итераций используются методы дефляции и грубосеточной

*Работа поддержана грантом РФФИ N 11-01-00205, а также грантами Президиума РАН N 15.9-4 и ОМН РАН N 1.3.3-4.

коррекции [10].

Коммуникации между процессорами на внешних итерациях выполняются средствами системы MPI, а "внутренние" вычисления при решении СЛАУ в подобластях реализуются параллельными потоками над общей памятью с помощью OpenMP. Библиотека Krylov включает также итерационные решатели из написанной на языке CUDA библиотеки CuSP NVIDIA [11], что позволяет решать на гетерогенном кластере внутренние подсистемы на GPU.

Особенностью программных реализаций является то, что решаемые СЛАУ представляются в стандартном сжатом строчном формате CSR [8], который является практически безальтернативным способом построения универсальных алгебраических решателей, но представляет значительные трудности для эффективного распараллеливания алгоритмов. В частности, возникает нетривиальная задача формирования CSR-форматов для вспомогательных расширенных СЛАУ в подобластях.

Библиотека Krylov является функциональным аналогом таких пакетов распределенных итерационных решателей, как например PETSc [13], Hypre [14], HIPS и pARMS [15] и др., однако в Krylov предложен ряд оригинальных методов, обсуждаемых в работе.

В п. 2 мы описываем функциональное наполнение библиотеки Krylov, в п. 3 излагаются особенности программных реализаций алгебраических решателей, а в последнем разделе приводятся примеры численных экспериментов для характерных практических задач.

2. Функциональное наполнение библиотеки Krylov

Рассматривается решение вещественной СЛАУ

$$Au = f, \;\; A = \{a_{i,j}\} \in \mathcal{R}^{N,N}, \tag{1}$$

матрица которой, возможно, предварительно разбита на блочные строки $A = \{A_p \in \mathcal{R}^{N_p,N}, \; p = 1, ..., P\}, \;\; N_1 + ... + N_p = N$, с приблизительно равным числом строк $N_p \gg 1$ в каждой. Соответствующим образом также разбиваются векторы искомого решения и правой части $u = \{u_p\}, \;\; f = \{f_p\}$, так что система уравнений (1) может быть записана в форме совокупности P подсистем

$$A_{p,p}u_p + \sum_{\substack{q=1 \\ q \neq p}}^{P} A_{p,q}u_q = f_p, \;\; p = 1, ..., P, \tag{2}$$

где $A_{p,q} \in \mathcal{R}^{N_p,N_q}$ – прямоугольные матричные блоки, получаемые при разбиении каждой блочной строки (и всей матрицы A) на блочные столбцы.

Будем считать, что СЛАУ (1) представляет собой систему сеточных уравнений, так что каждая компонента векторов u, f соответствует узлу сетки, общее число которых в расчетной сеточной области $\Omega^h = \bigcup\limits_{p=1}^{P} \Omega_p$ равно N. При этом блочное разбиение матриц и векторов соответствует разбиению (декомпозиции) Ω^h на P сеточных непересекающихся подобластей Ω_p, в каждой из которых находится N_p узлов.

Описанная декомпозиция Ω^h не использует узлов – разделителей, т.е. условные границы подобластей не проходят через узлы сетки. Формальности ради можно считать, что внешность расчетной области есть неограниченная подобласть Ω_0 с числом узлов $N_0 = 0$.

Сеточное уравнение для i-го узла сетки может быть записано в виде

$$a_{i,i}u_i + \sum_{\substack{j \in \omega_i \\ j \neq i}} a_{i,j}u_j = f_i, \;\; i \in \Omega^h, \tag{3}$$

где через ω_i обозначается сеточный шаблон i-го узла, т.е. совокупность номеров всех узлов, участвующих в i-м уравнении.

Предполагается, что сеточные узлы и соответствующие переменные пронумерованы следующим образом: сначала идут подряд все узлы 1-й подобласти Ω_1 (неважно в каком внутреннем порядке), затем – узлы второй подобласти и т.д. Отметим, что взаимнооднозначного соответствия алгебраической и геометрической (сеточной) интерпретации структуры СЛАУ может и не существовать. Типичный пример – одному узлу сетки может соответствовать $m > 1$ переменных в алгебраической системе. Если для каждого узла такие "кратные" переменные нумеруются подряд, то структура СЛАУ приобретает "мелкоблочный" ($m \ll N$) вид (в (3) u_i и f_i означают не числа, а подвекторы порядка m соответственно, $a_{i,j} \in \mathcal{R}^{m,m}$) и на таких случаях мы остановимся в последующем особо.

Пусть матрица (1) задана в сжатом разреженном CSR-формате [8] с указанием числа строк N_p в каждой из P подсистем (2) в соответствии с разбиением матрицы A на блочные строки A_p. Естественно предполагается, что строки исходной матрицы пронумерованы подряд от 1 до N, так что номера строк каждого блока A_p меняются от $1 + \sum\limits_{i=1}^{p-1} N_i$ до $\sum\limits_{i=1}^{p} N_i$ (эти номера назовем глобальными).

На основе блочного представления СЛАУ (2) стандартным образом строится аддитивный метод Шварца, на алгебраическом языке представляющий блочный итерационный метод Якоби. Однако известно, что скорость сходимости итераций возрастает, если декомпозицию расчетной области сделать с пересечением областей.

В силу этого мы используем определение расширенной сеточной подобласти $\bar{\Omega}_p \supset \Omega_p$, имеющей пересечения с соседними подобластями, величину которых мы будем определять в терминах количества околограничных сеточных слоев, или фронтов. Обозначим через $\Gamma_p^0 \in \Omega_p$ множество внутренних околограничных узлов из Ω_p, т.е. таких узлов $P_i \in \Omega_p$, у которых один из соседей не лежит в $\Omega_p (P_j \notin \Omega_p,\ j \in \omega_i,\ j \neq i))$. Обозначим далее через Γ_p^1 множество узлов, соседних с узлами из Γ_p^0, но не принадлежащих Ω_p, через Γ_p^2 – множество узлов, соседних с узлами из Γ_p^1, но не принадлежащих объединению $\Gamma_p^1 \bigcup \Omega_p$, через Γ_p^3 – множество узлов, соседних с Γ_p^2, но не принадлежащих $\Gamma_p^2 \bigcup \Gamma_p^1 \bigcup \Omega_p$, и т.д. Соответственно эти множества назовем первым внешним слоем (фронтом) узлов, вторым слоем, третьим и т.д. Получаемое объединение узлов

$$\bar{\Omega}_p \equiv \Omega_p \bigcup \Gamma_p^1 ... \bigcup \Gamma_p^{\Delta}$$

будем называть расширенной p-й сеточной подобластью, а целую величину Δ определяем как величину расширения, или пересечения (в терминах количества сеточных слоев). Случай $\Delta = 0$ фактически означает декомпозицию области Ω^h на подобласти без пересечений ($\Omega_p^0 = \Omega_p$).

На формальном алгебраическом языке каждой расширенной подобласти можно сопоставить подсистему уравнений

$$(\bar{A}_{p,p} + \theta \bar{D}_p)\bar{u}_p = \bar{f}_p - \sum_{\substack{q=1 \\ q \neq p}}^{P} \bar{A}_{p,q}\bar{u}_q + \theta \bar{D}_p = \bar{r}_p, \tag{4}$$

где $\bar{u}_p, \bar{f}_p, \bar{r}_p \in \mathcal{R}^{\bar{N}_p}$, а $\bar{D}_p$ – диагональная матрица, определяемая соотношением

$$\bar{D}_p e = \sum_{\substack{q=1 \\ q \neq p}}^{P} A_{p,q} e, \ \ e = (1, ..., 1)^T \in \mathcal{R}^{\bar{N}_p}.$$

Здесь $\theta \in [0, 1]$ – некоторый итерационный параметр, который при $\theta = 0$ соответствует итерационному краевому условию Дирихле на смежных границах подобластей, при $\theta = 1$ – условию Неймана, а при $0 < \theta < 1$ – условию 3-го рода, или Робена.

Переходя теперь к полному вектору u и вводя матрицы

$$B_p = \bar{A}_{p,p} + \theta \bar{D}_p, \tag{5}$$

из (4) получаем итерационный аддитивный процесс Шварца в виде

$$u^n = u^{n-1} + B^{-1}(f - Au^{n-1}) = u^{n-1} + B^{-1}r^n, \tag{6}$$

где B – предобуславливающая матрица, определяемая следующим образом:

$$B^{-1} = B_{AS}^{-1} = \sum_{p=1}^{P} \bar{B}_p^{-1}, \;\; \bar{B}_p^{-1} = W_p^T B_p^{-1} W_p. \tag{7}$$

Здесь $W_p : \mathcal{R}^N \to \mathcal{R}^{\bar{N}_p}$ есть матрица сужения полного вектора в подвектор из $\bar{\Omega}_p$, а W_p^T – транспонированная матрица продолжения (расширения вектора из $\bar{\Omega}_p$ в Ω).

Выполнение каждой итерации в (6) требует параллельного решения внутренних СЛАУ в подобластях $\bar{\Omega}_p$, что может производиться или прямым методом, или итерационным алгоритмом крыловского типа. В последнем случае фактически реализуются переменные (динамические) предобуславливатели B_n.

При этом реально вместо (6) используется универсальный "гибкий" метод обобщенных минимальных невязок FGMRES, оптимизирующий невязку в подпространствах Крылова. Дальнейшее ускорение этого процесса осуществляется с помощью "улучшения" крыловских подпространств, что означает добавление новых базисных векторов и может быть интерпретировано как аддивное расширение предобуславливающих матриц:

$$B_n^{-1} = B_{AS,n}^{-1} + B_c^{-1} + B_d^{-1}. \tag{8}$$

3. Особенности параллельной реализации алгоритмов

Описанные выше параллельные алгоритмы реализованы в форме библиотеки Krylov, которая ориентирована на решение сверхбольших СЛАУ на системах с распределенной памятью, включающих в себя большое количество вычислительных узлов. В связи с этим реализация итерационных решателей СЛАУ и предобуславливателей имеет определенные особенности. В частности, требуется такая организация и структура методов, которая хорошо отображается на архитектуры имеющихся вычислительных систем.

3.1. Общая организация двухуровневых итераций

Решатели в библиотеке Krylov построены на основе широко распространенного стандарта интерфейса обмена данными MPI (Message Passing Interface). Базой для итерационного решения систем уравнений в библиотеке Krylov является метод FGMRES, распределенный по различным MPI-процессам. Данный метод совместно с различными предобуславливателями типа Шварца используется для решения заданной распределенной СЛАУ.

Псевдокод алгоритма следующий (см. [7]):

- $r_0 \leftarrow f - Au_0$, $\beta = ||r_0||$, $q_0 \leftarrow r_0/||r_0||$, $\xi \leftarrow (1, 0, \ldots, 0)^T$
- $n \leftarrow 0, 1, \ldots$ while $||r_n|| > \varepsilon\beta$
  - $z_n \leftarrow B_n^{-1} q_n$
  - $\tilde{q}_{n+1} \leftarrow Az_n$
  - For $k \in [0, \ldots, n]$
    - $H_{k,n} \leftarrow (\tilde{q}_{n+1}, q_k)$
    - $\tilde{q}_{n+1} \leftarrow \tilde{q}_{n+1} - H_{k,n} q_k$
  - $H_{n+1,n} \leftarrow ||\tilde{q}_{n+1}||$, $q_{n+1} \leftarrow \tilde{q}_{n+1}/H_{n+1,n}$

- For $k \in [0, \ldots, n-1]$
  - $\begin{bmatrix} H_{k,n} \\ H_{k+1,n} \end{bmatrix} \leftarrow \begin{bmatrix} c_k & s_k \\ -\bar{s}_k & c_k \end{bmatrix} \begin{bmatrix} H_{k,n} \\ H_{k+1,n} \end{bmatrix}$
- $c_n \leftarrow |H_{n,n}| / \sqrt{|H_{n,n}|^2 + |H_{n+1,n}|^2}$
- $\bar{s}_n \leftarrow c_n H_{n+1,n} / H_{n,n}$
- $\begin{bmatrix} \xi_n \\ \xi_{n+1} \end{bmatrix} \leftarrow \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} \xi_n \\ \xi_{n+1} \end{bmatrix}$
- $H_{n,n} \leftarrow c_n H_{n,n} + s_n H_{n+1,n}$, $H_{n+1,n} \leftarrow 0$
- $||r_{n+1}|| \leftarrow \beta |\xi_{n+1}|$

- $y_n \leftarrow \beta H^{-1} \xi$
- $x_n \leftarrow x_0 + [z_0 \ldots z_{n-1}] y_n$.

Анализ вычислительной схемы алгоритма показывает, что основными операциями в методе FGMRES являются

- векторно-векторные операции (вычисление скалярных произведений, норм, и т.п.);
- матрично-векторные операции (умножение матрицы на вектор); в первых двух пунктах операции выполняются в полном N-мерном пространстве, где N – размерность решаемой СЛАУ;
- применение предобуславливателя (методы Шварца и грубосеточной коррекции, включающие решение вспомогательных СЛАУ относительно малого размера);
- QR-разложение матрицы Хессенберга, формируемой в процессе построения базисных векторов.

Все остальные операции требуют $O(1)$ времени и не вносят существенного вклада в производительность решателя. Более того, вклад QR-разложения во время работы решателей также можно игнорировать в предположении, что порядок системы много больше количества итераций. Такое разложение также не требует большого количества памяти, поэтому с целью уменьшения коммуникационных затрат оно выполняется на каждом MPI процессе.

Векторно-векторные операции в итерационных решателях распараллеливаются естественным образом за счет разбиения векторов, порождаемого декомпозицией матрицы на блочные строки. Единственная особенность реализации этих операций заключается в том, что для вычисления скалярного произведения и нормы необходима будет точка синхронизации и обмен данными, однако, к примеру, в интерфейсе MPI имеется требуемая функция **MPI_Allreduce**, реализация которой оптимизируется поставщиком библиотеки MPI.

Для проведения матрично-векторных операций решатель строит для каждой подобласти списки узлов, являющихся граничными для соседних подобластей. В дальнейшем, при умножении матрицы на вектор или при применении итерации Шварца к вектору неизвестных решатель использует полученную информацию для пересылки частей вектора между процессами. Ключевым моментом в таком подходе является уменьшение объема пересылаемых данных — необходимо пересылать в соседние подобласти только граничные коэффициенты вектора вместо того, чтобы пересылать вектор целиком. В случае, если разбиение на подобласти было построено подходящим способом, количество граничных вершин будет

существенно меньше размера подобласти (например, для двумерных задач количество граничных вершин N_p будет пропорционально квадратному корню из общего числа вершин в подобласти, а для трехмерных $N_\Gamma \sim N^{2/3}$).

Применение предобуславливателей, основанных на методе Шварца, требует на каждой итерации внешнего алгоритма решения подзадач в подобластях. Достоинством метода аддитивного Шварца является то, что решение в подобластях может проводиться независимо друг от друга и не требует коммуникаций. Действительно, как уже отмечалось выше, итерацию Шварца можно представить в виде (6), где предобуславливатель B составлен из блоков (5). Отсюда видно, что применение предобуславливателя B^{-1} в подобласти p не требует знания переменных, не принадлежащих ей.

3.2. Реализация внутренних итераций в подобластях

Для решения задач $A_{p,p}v_p = b_p$ в подобластях вида можно использовать какой-нибудь прямой решатель для разреженных систем, например решатель PARDISO из библиотеки Intel® MKL. Однако время, требуемое PARDISO для разложения матриц, в общем случае растет практически как $O(N^3/P^3)$, поэтому такой метод подходит только для решения не очень больших систем на большом числе процессоров. Однако действие обратных несимметричных матриц $A_{p,p}^{-1}$ можно вычислять приближенно при помощи предобусловленных методов в подпространствах Крылова, таких как GMRES и BiCGStab. В качестве предобуславливателя предлагается метод USOR в модификации Айзенштата, в том числе его мелкоблочный вариант. При этом пользователю предоставляется возможность выбора решателя для подобластей — либо PARDISO из Intel® MKL, либо одного из перечисленных выше итерационных решателей.

Рассмотрим более подробно предобуславливатель USOR в модификации Айзенштата. Пусть B — предобуславливающая матрица, записанная в форме

$$B = (G - L)G^{-1}(G - U) = D - L - U + LG^{-1}U, \tag{9}$$

где $A = D - L - U$, $D = block\text{-}diag\{D_k\}$, $D_k \in \mathcal{R}^{m,m}$, $k = 1, ..., M = N/m$ есть блочно-диагональная невырожденная матрица с диагональными блоками одинакового порядка m, причем порядок СЛАУ кратен m, а M — это блочный порядок матрицы A. Матрица G – некоторая пока не конкретизируемая невырожденная матрица, для которой вычислимо разложение $G = L_G U_G$ с треугольными множителями L_G и U_G.

С помощью обозначений

$$\bar{D} = L_G^{-1} D U_G^{-1},\ \bar{L} = L_G^{-1} L U_G^{-1},\ \bar{U} = L_G^{-1} U U_G^{-1} \tag{10}$$

предобусловленная матрица системы записывается в форме

$$\bar{A} = (I - I)^{-1} + (I - \bar{U})^{-1} + (I - \bar{L})^{-1}(\bar{D} - 2I)(I - \bar{U})^{-1}, \tag{11}$$

где I есть единичная матрица.

Отсюда предобусловленную СЛАУ можно представить как

$$\bar{A}\bar{u} = \bar{f} \equiv (L_G - L U_G^{-1})^{-1} f,\ \ \bar{u} = (U_G - L_G^{-1} U)u. \tag{12}$$

Важно отметить, что умножение $\bar{A}$ на некоторый вектор v можно представить в удобной для вычисления форме

$$\bar{A}v = (I - \bar{L})^{-1}[v + (\bar{D} - 2I)w] + w,\ \ w = (I - \bar{U})^{-1}v, \tag{13}$$

составляющей суть модификации Айзенштата.

Конкретизируем теперь выбор матрицы G. Мы используем

$$G = \frac{1}{\omega} D, \tag{14}$$

где ω – релаксационный числовой параметр. Очевидно, что такая матрица G сохраняет блочно-диагональную структуру D. В этом случае треугольное разложение G сводится к соответствующему разложению ее диагональных блоков:

$$G = block\text{-}diag\{G_k = L_{G,k} U_{G,k}\}, \tag{15}$$

а реализация умножения вектора на матрицу $\bar{A}$ упрощается за счет блочного представления матриц

$$\bar{L} = \{\bar{L}_{k,l} = L_{G,k}^{-1} L_{k,l} U_{G,l}^{-1}\}, \;\; \bar{U} = \{\bar{U}_{k,l} = L_{G,k}^{-1} U_{k,l} U_{G,l}^{-1}\},$$

$$k, l = 1, ..., M.$$

Более конкретно, при использовании блочного представления векторов $v = \{v_k\}, w = \{w_k\}, k = 1, ..., M$, решения фактически участвующих в (13) треугольных систем

$$(I - \bar{U})w = v, \;\; (I - \bar{L})y = z \equiv v + (\bar{D} - 2I)w$$

осуществляются фактически по следующим рекуррентным формулам:

$$\begin{aligned} &w_M = v_M, \;\; w_k = v_k + \sum_{l=k+1}^{M} \bar{U}_{k,l} v_l, \;\; k = M-1, ..., 1, \\ &y_1 = z_1, \;\; y_k = z_k + \sum_{l=1}^{k-1} \bar{L}_{k,l} z_l, \;\; k = 2, ..., M. \end{aligned} \tag{16}$$

Отметим, что используемые модификации неполной факторизации являются экономичными, но плохо распараллеливаемыми, вследствие необходимости решения вспомогательных СЛАУ с треугольными матрицами. Однако имеются различные подходы к распараллеливанию треугольных решателей, например метод планирования вычислений по уровням, см. [7]. Кроме того, умножение на "мелкоблочные" матрицы $U_{k,l}$ и $L_{k,l}$, если они вычислены заранее до итераций, для каждого фиксированного k позволяют обеспечить существенное ускорение средствами OpenMP.

4. Примеры численных экспериментов

Мы приведем результаты некоторых численных экспериментов по решению ряда практических задач с помощью библиотеки Krylov. В данных расчетах внешние итерации проводились с помощью метода GMRES с критерием окончания итераций по условию на евклидовую норму невязки

$$||r^n||_2 \le \varepsilon ||f||_2, \;\; \varepsilon = 10^{-7}.$$

Начальные приближения для искомых векторов всегда выбирались $u^0 = 0$.

Вспомогательные СЛАУ в подобластях решались либо с помощью прямого решателя PARDISO, причем наиболее трудоемкий этап – LU-разложение матрицы – выполнялся один раз до итераций, либо с помощью итерационного метода BiCGStab с предобуславливателем Айзенштата. Расчеты проводились на различном количестве P вычислительных узлах, с формированием такого же числа MPI-процессов. В нижеследующих таблицах N и NZ обозначают порядок решаемых СЛАУ, а также количество ненулевых элементов матрицы, которые характеризуют трудоемкость задачи.

Рассмотрим результаты экспериментов по решению трех несимметричных СЛАУ, возникающих из сеточных аппроксимаций практических задач гидро-газодинамики. Характеристики соответствующих плохо обусловленных матриц даны в табл. 1.

Табл. 1. Характеристики "гидро-газодинамических" матриц

Наименование	$N \cdot 10^{-6}$	$NZ \cdot 10^{-6}$
Г2	1.73	12.0
Г3	0.48	13.7
Г4	9.38	50.4

В табл. 2–4 приведены результаты расчетов с использованием PARDISO в качестве решателя в подобластях. В строках таблиц сверху вниз представлено общее время счета t в секундах и количество внешних итераций n_e.

Здесь и далее используются следующие обозначения: N_{nod} – число используемых вычислительных узлов; N_{mpi} – количество MPI-процессов на одном узле; $P = N_{nod} \cdot N_{mpi}$ – общее число MPI-процессов, равное количеству подобластей; Δ – параметр пересечения в декомпозиции (количество расширений сеточной подобласти); N_{th} – количество формируемых вычислительных потоков в одном MPI-процессе; T_1, – время решения СЛАУ без распараллеливания.

Табл. 2. Значения t и n_e для случая использования PARDISO в подобластях при $N_{th} = 4$, СЛАУ Г3 ($T_1 = 10.1$)

$P \setminus \Delta$	0	1	2	3	4	5
	9.45	7.46	7.23	6.80	6.75	6.52
5	82	41	29	22	18	15
	5.64	4.69	4.32	4.21	4.27	4.31
10	114	59	41	32	27	23
	6.03	4.09	3.72	3.76	3.77	3.59
20	164	84	58	44	37	32
	6.83	4.29	3.86	3.63	3.78	3.68
30	183	94	67	52	43	38

Табл. 3. Результаты решения СЛАУ Г4 с помощью PARDISO ($T_1 = 399.4$)

$P \setminus \Delta$	0	1	2	3	4	5
	280.4	199.4	153.9	146.5	140.6	143.5
5	235	119	85	67	55	47
	127.9	73.6	61.3	56.3	51.9	49.6
10	311	161	116	92	76	66
	147.4	71.2	51.9	47.0	41.3	40.6
20	331	175	127	101	84	72
	158.3	76.9	59.2	52.7	49.5	48.4
30	355	186	135	109	95	80

Табл. 4. Результаты экспериментов для СЛАУ Г2 с помощью PARDISO ($T_1 = 87.1$)

$P \setminus \Delta$	0	1	2	3	4	5
	60.6	39.9	33.9	30.9	28.5	27.4
5	235	116	78	59	46	37
	58.4	32.9	24.6	21.0	19.1	17.4
10	486	244	167	124	99	79
	118.5	47	28.6	22.6	18.3	16.0
20	934	479	330	246	194	157
	230.5	74.5	53.9	33.4	27.0	20.8
30	1352	690	473	355	280	228

Как видно из этих результатов, использование пересечений с ростом Δ дает стабильное уменьшение числа внешних итераций, а время счета уменьшается в два и в большее число раз (максимальный коэффициент ускорения – свыше 10). С ростом P время вычислений сначала уменьшается, но после $P > 20$ начинает увеличиваться, так как растет число внешних итераций. Для исправления ситуации, очевидно, надо использовать ускорение типа грубосеточной коррекции, которое в данных экспериментах не применялось.

Второй набор СЛАУ для численных экспериментов был представлен в блочно-строчном распределенном формате CSR, а заданное количество блочных строк указано в табл. 5. Там же указано, что первые две СЛАУ данного набора имеют “мелкоблочную” структуру с размером блоков, равным $m = 5$. В таблице также приведено количество подобластей P, на которые были разбиты соответствующие СЛАУ. Разбиение СЛАУ осуществлено без пересечений, так что, фактически, в данном наборе тестов $\Delta = 0$.

Табл. 5. Характеристики блочных распределенных СЛАУ

Наименование	$N \cdot 10^{-6}$	$NZ \cdot 10^{-6}$	P	m
Г5	9.5	330	48	5
Г6	3.7	126.8	36	5
Г7	6.5	44.4	20	1
Г8	0.35	1.74	20	1

В табл. 6 приведены результаты экспериментов для четырех СЛАУ из второго набора тестируемых задач. В столбцах таблицы указано общее время счета t, а также число внешних итераций n_e.

Для внутренних СЛАУ при использовании итерационных решателей фактически использовались ограничения числа итераций n^i_{max}. Для прямого решателя PARDISO в скобках указано число запускаемых им потоков при факторизации и решении СЛАУ. В качестве предобуславливателя для внутренних итерационных решателей использовался "мелкоблочный" предобуславливатель USOR в модификации Айзенштата. Для предобуславливателя USOR представлены результаты без использования средств распараллеливания OpenMP.

Табл. 6. Сравнение прямых и итерационных решателей для внутренних СЛАУ

Метод	Г5		Г6		Г7		Г8	
PARDISO ($N_{th} = 8$)	58.9	38	18.9	21	121.6	351	3.8	189
PARDISO ($N_{th} = 2$)	149.6	38	42.4	21	178.8	351	4.7	189
BBiCGStab ($n^i_{max} = 10$)	49.3	39	13.6	21	498.5	368	14.6	252
BBiCGStab ($n^i_{max} = 20$)	70.5	38	19.7	21	843.9	360	24.9	206

По результатам численных экспериментов можно сделать следующие выводы:

- на рассматриваемых СЛАУ с $m = 1$ прямой решатель PARDISO имеет существенное преимущество перед итерационными, причем увеличение в нем количества используемых потоков N_{th} от 2 до 8 дает коэффициент ускорения от 1.5 до 3;
- прямой решатель PARDISO проигрывает "мелкоблочному" итерационному алгоритму для СЛАУ с $m = 5$, хотя текущая реализация даже не использует средства OpenMP для распараллеливания на общей памяти;
- расчеты подтверждают ожидаемый факт, что при использовании итерационных внутренних решателей для СЛАУ в подобластях нет смысла добиваться высокой точности на различных внешних итерациях; более конкретно, при увеличении количества внутренних итераций n^i_{max} с 10 до 20 число внешних итераций несколько падает, но каждая из них делается "дороже" и общее время решения увеличивается.

Литература

1. Бутюгин Д.С., Ильин В.П., Ицкович Е.А и др. Krylov: библиотека алгоритмов и программ для решения СЛАУ // Современные проблемы математического моделирования. Математическое моделирование, численные методы и комплексы программ. Сборник трудов Всероссийских научных молодёжных школ. Ростов-на-Дону: Изд-во Южного федерального университета, 2009, 110-128.

2. Бутюгин Д.С., Ильин В.П., Перевозкин Д.В. Методы параллельного решения СЛАУ на системах с распределенной памятью в библиотеке Krylov.// Вестник ЮУрГУ. Серия “Вычислительная математика и информатика”, т. 47, N 306, 2012, 5-19.

3. Ильин В.П. Методы конечных разностей и конечных объемов для эллиптических уравнений.–Новосибирск, изд. ИВМиМГ СО РАН, 2001.

4. Ильин В.П. Методы и технологии конечных элементов.–Новосибирск, изд. ИВМиМГ СО РАН, 2007.

5. Ильин В.П. Параллельные методы и технологии декомпозиции областей.// Вестник ЮУрГУ. Серия “Вычислительная математика и информатика”, 2012, N 46(305), 31-44.

6. Ильин В.П., Кныш Д.В. Параллельные методы декомпозиции в пространствах следов.//Вычислительные методы и программирование, изд. МГУ, т. 12, N 1, 2011, 100-109.

7. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition.SIAM, 2003.

8. Intel Math Kernel Library. Reference Manual:
URL: http://software.intel.com/sites/products
/documentation/hpc/composerxe/enus/mklxe/mkl_manual_win_mac/index.htm.

9. Ильин В.П. Методы бисопряженных направлений в подпространствах Крылова.–СибЖИМ, т. 11, N 4(36), 2008, 47-60.

10. Nabben R., Vuik C. A comparison of abstract versions of deflation, balancing and additive coarse grid correction preconditioners.–Num. Lin. Alg. with Appl., v. 15, 2008, 355-372.

11. Bell N., Garland M. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations, 2012. URL: http://cusp-library.googlecode.com

12. Кластер НКС-30Т. URL: http://www2.sscc.ru/HKC-30T/HKC-30T.htm

13. PETSc: Home Page. URL: http://www.mcs.anl.gov/petsc/

14. Hypre. URL: http://acts.nersc.gov/hypre/

15. Yousef Saad – Software. URL: http://www-users.cs.umn.edu/~saad/software/

Пакет параллельных прикладных программ Helmholtz3D *

Д.С. Бутюгин[1]

Институт вычислительной математики и математической геофизики СО РАН[1]

В работе представлен пакет параллельных прикладных программ Helmholtz3D, который позволяет проводить расчеты трехмерных электромагнитных полей с гармонической зависимостью от времени, распространяющиеся в трехмерных областях со сложной геометрией. Для решения возникающих в результате аппроксимаций систем линейных алгебраических уравнений (СЛАУ) с комплексными плохообусловленными неэрмитовыми матрицами используются современные итерационные методы решения СЛАУ в подпространствах Крылова совместно с оригинальными параллельными предобуславливателями. Апробация пакета проведена на серии методических и практических задач расчета электромагнитных полей для волновых устройств и задач электромагнитного каротажа.

1. Введение

В настоящее время существует ряд коммерческих и открытых пакетов, позволяющих проводить расчеты гармонических электромагнитных полей. К ним относятся как пакеты, ориентированные на решение именно задач электромагнетизма, например ANSYS HFSS и CST Microwave Studio, так и пакеты общего назначения для решения различных краевых задач для дифференциальных уравнений в частных производных (Partial Differential Equations, PDE), к которым можно отнести, например, пакет FEniCS. Однако использование таких пакетов может быть сопряжено с определенными сложностями. Во-первых, отсутствие подходящего вычислительного инструментария в продукте может привести к невозможности адекватно описать модель и вычислить электромагнитные поля в расчетной области. Во-вторых, закрытый характер коммерческих программных продуктов может затруднить интеграцию с уже существующими у пользователей программными комплексами. В третьих, далеко не всегда имеется поддержка вычислений на удаленных вычислительных системах, имеющихся в распоряжении институтов и коммерческих компаний. И наконец, не стоит сбрасывать со счетов высокую стоимость лицензии, которая может достигать десятков тысяч долларов в год. В открытых же пакетах, зачастую, реализованы не слишком эффективные алгоритмы, а имеющаяся по ним документация ограничена. В связи с этим, актуальной является разработка пакетов, основанных на самых современных достижениях в области конечно-элементного моделирования электромагнитных полей и поддерживающих расчеты на параллельных системах с общей и распределенной памятью с тысячами и десятками тысяч вычислительных узлов.

В работе предлагается пакет параллельных прикладных програм Helmholtz3D. Данный пакет позволяет проводить расчеты трехмерных электромагнитных полей с гармонической зависимостью от времени, распространяющиеся в трехмерных областях со сложной геометрией. Цель разработки пакета — создание средства решения задач трехмерного электромагнетизма, основанного на самых современных достижения в этой области. Ключевыми особенностями данного проекта являются следующие. Решается комплексное векторное уравнение Гельмгольца (2), полученное непосредственно из уравнений Максвелла без использования приближений. Это позволяет использовать полученные алгоритмы при расчете широкого спектра моделей при различных физических параметрах, в частности, рассчитывать электромагнитные поля в широком спектре частот. Использование различных вариационных формулировок задачи позволяет выбирать оптимальный алгоритм для про-

*Работа поддержана грантами РФФИ №11-01-00205 и ОМН РАН №1.3.3-4.

ведения вычислений в зависимости от физических параметров задачи и дает возможность получить физически корректную картину распределения электромагнитных полей. Применение символьных вычислений и техник оптимизации выражений и автогенерации кода для построения конечно-элементных (МКЭ) аппроксимаций высоких порядков (вплоть до $O(h^4)$) упрощает сопровождение кода и существенно снижает вероятность появления ошибок в сложных выражениях для вычисления матриц СЛАУ.

Для решения возникающих в результате аппроксимаций систем линейных алгебраических уравнений (СЛАУ) с комплексными плохообусловленными неэрмитовыми матрицами используются современные итерационные методы решения СЛАУ в подпространствах Крылова совместно с оригинальными параллельными предобуславливателями. Предобуславливатели используют методы аддитивного Шварца и экономичной геометрической декомпозиции области, а также алгебраические мультисеточные методы на основе иерархических базисов, в том числе методы алгебраической грубосеточной коррекции. Применение таких алгоритмов позволяет добиться высокого уровня производительности решателей на системах с общей и распределенной памятью и хорошей масштабируемости на сотнях вычислительных узлов. Алгоритмы пакета были апробированы на серии методических и практических задач расчета электромагнитных полей для волновых устройств и задач электромагнитного каротажа, продемонстрировав высокую эффективность, в частности, решение получаемых СЛАУ с сотнями миллионов неизвестных может занимать около 10-15 минут. Высокая точность полученных конечно-элементных решений была показана путем сравнения с имеющимися аналитическими решениями либо результатами расчетов других групп (в том числе и полученных другими методами).

Структура данной работы следующая. В первом пункте приводится описание математической постановки решаемых пакетом Helmholtz3D задач. В П. 3 приводится описание архитектуры программного комплекса. П. 4 содержит описание особенностей используемых технологий параллельного программирования. В П. 5 представлены результаты численных экспериментов с использованием предлагаемого пакета. Наконец, в заключении обсуждаются результаты работы.

2. Математическая постановка задачи

Пусть векторы напряженности электрического и магнитного полей имеют вид $\Re(\vec{E}_a e^{-i\omega t})$, $\Re(\vec{H}_a e^{-i\omega t})$, где t — время, ω — круговая частота, i — мнимая единица, $\Re(x)$ — действительная часть x, а $\vec{E}_a$ и $\vec{H}_a$ — зависящие только от пространственных координат комплексные амплитуды. Тогда система уравнений Максвелла, описывающая электромагнитные поля с гармонической зависимостью от времени, при отсутствии сторонних электрических объемных зарядов и магнитной проводимости, записывается следующим образом:

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

Здесь $\vec{J}$ — амплитуда внешних гармонических токов, $\dot{\varepsilon} = \varepsilon_0\varepsilon_r + i\sigma_e/\omega$, $\dot{\varepsilon}_r = \dot{\varepsilon}/\varepsilon_0$, $\mu = \mu_0\mu_r$, ε_0 и μ_0 — диэлектрическая и магнитная проницаемости вакуума, ε_r и μ_r — относительная диэлектрическая и магнитная проницаемости среды, а σ_e — электрическая проводимость.

Мы полагаем, что физические параметры сред не зависят от полей $\vec{E}_a$ и $\vec{H}_a$. Тогда линейная система дифференциальных уравнений (1) после исключения вектора $\vec{H}_a$ сводится к уравнению Гельмгольца

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

где $k_0 = \omega\sqrt{\mu_0\varepsilon_0} = \omega/c > 0$, c — скорость света, $Z_0 = \sqrt{\mu_0/\varepsilon_0} = \mu_0 c$.

Решение рассматриваемых уравнений будем искать в ограниченной односвязной области Ω с липшицевой границей $\partial\Omega = \Gamma \bigcup \Sigma$, на каждой из частей которой поставлено одно из граничных условий:

$$\vec{n} \times \vec{E}_a \Big|_\Gamma = \vec{n} \times \vec{E}_0, \qquad \vec{n} \times \vec{H}_a \Big|_\Sigma = \vec{n} \times \vec{H}_0, \tag{3}$$

где $\vec{n}$ — внешняя нормаль к границе, а $\vec{E}_0$, $\vec{H}_0$ — заданные функции координат.

Будем полагать, что расчетная область Ω состоит из m непересекающихся подобластей $\bar{\Omega} = \bigcup_{k=1}^{m} \bar{\Omega}_k$, в каждой из которых физические параметры сред ε_r, μ_r и σ_e являются достаточно гладкими. Полагаем, что внутренние границы их раздела $\Gamma_{k,k'} = \bar{\Omega}_k \bigcap \bar{\Omega}_{k'}$ являются липшицевыми и, соответственно, на них почти всюду определена нормаль $n_{k,k'}$, направленная из Ω_k в $\Omega_{k'}$. Выполняются условия сопряжения

$$\begin{aligned} &\vec{n}_{k,k'} \cdot (\dot{\varepsilon}_k \vec{E}_{a,k} - \dot{\varepsilon}_{k'} \vec{E}_{a,k'}) = 0, \quad \vec{n}_{k,k'} \times (\vec{E}_{a,k} - \vec{E}_{a,k'}) = \vec{0}, \\ &\vec{n}_{k,k'} \cdot (\mu_k \vec{H}_{a,k} - \mu_{k'} \vec{H}_{a,k'}) = 0, \quad \vec{n}_{k,k'} \times (\vec{H}_{a,k} - \vec{H}_{a,k'}) = \vec{0}. \end{aligned} \tag{4}$$

Предполагаем, что k_0^2 не является максвелловским собственным числом (ω не является резонансной частотой), т.е. краевая задача (2)–(4) при $\vec{E}_0 = 0$, $\vec{H}_0 = 0$ и $\vec{J} = 0$ имеет только нулевое решение.

Вводятся стандартные соболевские пространства:

$$H^1 = \left\{ \varphi \in L^2(\Omega) : \nabla\varphi \in \left[L^2(\Omega)\right]^3 \right\}, \; H^{\mathrm{rot}} = \left\{ \vec{\psi} \in \left[L^2(\Omega)\right]^3 : \nabla \times \vec{\psi} \in \left[L^2(\Omega)\right]^3 \right\},$$
$$H_0^{\mathrm{rot}} = \left\{ \vec{\psi} \in H^{\mathrm{rot}} : \vec{n} \times \vec{\psi}|_\Gamma = 0 \right\}.$$

Мы полагаем, что существует $\vec{E}_\Gamma \in H^{\mathrm{rot}}$ такая, что $\vec{n} \times \vec{E}_\Gamma|_\Gamma = \vec{n} \times \vec{E}_0$. Введем билинейные формы $s, m : H^{\mathrm{rot}} \times H^{\mathrm{rot}} \to \mathbb{C}$:

$$s(\vec{u}, \vec{v}) = \int_\Omega \mu_r^{-1} (\nabla \times \vec{u}) \cdot (\nabla \times \vec{v}) \, \mathrm{d}\Omega, \quad m(\vec{u}, \vec{v}) = \int_\Omega \dot{\varepsilon}_r (\vec{u} \cdot \vec{v}) \, \mathrm{d}\Omega,$$

и линейный функционал $L : H^{\mathrm{rot}} \to \mathbb{C}$

$$L(\vec{v}) = k_0^2 m(\vec{E}_\Gamma, \vec{v}) - s(\vec{E}_\Gamma, \vec{v}) - ik_0 Z_0 \int_\Omega \vec{J} \cdot \vec{v} \mathrm{d}\Omega - ik_0 Z_0 \int_\Sigma \vec{H}_0 \cdot (\vec{n} \times \vec{v}) \, \mathrm{d}\Omega.$$

Вариационная формулировка уравнения (2) с краевыми условиями (3) в форме Галеркина имеет следующий вид (см. [1,2]): найти такое $\vec{E} \in H_0^{\mathrm{rot}}$, что для всех $\vec{\psi} \in H_0^{\mathrm{rot}}$ выполнено

$$s(\vec{E}, \vec{\psi}) - k_0^2 m(\vec{E}, \vec{\psi}) = L(\vec{\psi}), \tag{5}$$

при этом исходное решение $\vec{E}_a$ восстанавливается как $\vec{E}_a = \vec{E} + \vec{E}_\Gamma$.

Рассмотрим теперь дискретный аналог вариационной задачи. Для простоты мы рассматриваем только тетраэдральные неструктурированные сетки. Пусть построено соответствующее разбиение расчетной области Ω на непересекающиеся тетраэдральные элементы $\Omega = \bigcup \Omega_T$. В каждом из тетраэдров вводятся базисные функции, соответствующие его степеням свободы. Пусть $\mathcal{W}_l$ — конечномерное пространство базисных функций порядка не выше l, конформное H^{rot}. В работе рассматриваются иерархические базисные функции, предложенные в [3]:

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

где $\tilde{\mathcal{W}}_i$ — инкрементальное подпространство с базисными функциями порядка i, при этом $\tilde{\mathcal{V}}_i$ — инкрементальное подпространство со скалярными базисными функциями, конформными H^1, а $\tilde{\mathcal{A}}_i$ — инкрементальное подпространство с роторными базисными функциями. Подпространства $\mathcal{V}_{l,0}$ и $\mathcal{W}_{l,0}$ вводятся естественным образом.

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

$$\vec{E}^h = \sum_i u_i \vec{\psi}_i^0.$$

Конечно-элементая функция $\vec{E}^h_\Gamma$ строится таким образом, что коэффициенты разложения по базисным функциям с ненулевым следом на Γ принимают фиксированные значения, соответствующие первому краевому условию в (3), а остальные равны нулю.

Вводятся матрицы соответствующих билинейных форм и вектор правой части:

$$M_{i,j} = m(\vec{\psi}_j^0, \vec{\psi}_i^0), \quad S_{i,j} = s(\vec{\psi}_j^0, \vec{\psi}_i^0), \quad f_i = L(\vec{\psi}_i^0), \tag{6}$$

где базисные функции $\vec{\psi}_i^0, \vec{\psi}_j^0 \in \mathcal{W}_{l,0}$. Тогда итоговая система принимает вид

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

Для вычисления элементов матрицы и вектора правой части можно воспользоваться поэлементной технологией сборки [4], заменив интегрирование по расчетной области Ω суммой интегралов по каждому из тетраэдров и вычислением в каждом из тетраэдров локальных матриц и векторов.

3. Методология построения пакета параллельных прикладных программ Helmholtz3D

Задача моделирования гармонических электромагнитных полей может быть разбита на следующие функциональные этапы:

- геометрическое и функциональное моделирование расчетной области;
- генерация сетки в расчетной области;
- построение МКЭ аппроксимации и генерация СЛАУ;
- решение СЛАУ с разреженными матрицами высоких порядков;
- постобработка решения и его визуализация.

Пакет Helmholtz3D ориентирован на решение 2-5 пунктов. Геометрическое и функциональное моделирование могут быть осуществлены средствами CAD-систем. Для этого пакет предоставляет возможность импорта сеток из сторонних пакетов за счет использования подходящих конвертеров.

В соответствии с функциональным назначением пакета была разработана его архитектура. Схематично она представлена на рис. 1.

В качестве языка реализации был выбран C++. Использование шаблонов (templates) в сочетании с объектно-ориентированным подходом позволяет писать максимально обобщенные коды, не теряющие скорость работы при использовании современных компиляторов. В качестве компилятора использовался GCC 4.4.1. Для упрощения разработки аппроксиматора и алгебраических решателей использовалась C++ библиотека линейной алгебры Eigen версии 3.0.1 [5]. В качестве основных достоинств библиотеки можно привести следующие: универсальность, высокая скорость работы за счет явной векторизации и оптимизации выражений, простота и выразительность пользовательского программного интерфейса (API), а также распространение под лицензией MPL. Помимо Eigen, использовалась высокопроизводительная библиотека Intel® Math Kernel Library (Intel® MKL, [6]). Из данной библиотеки использовался прямой решатель PARDISO для разреженных СЛАУ.

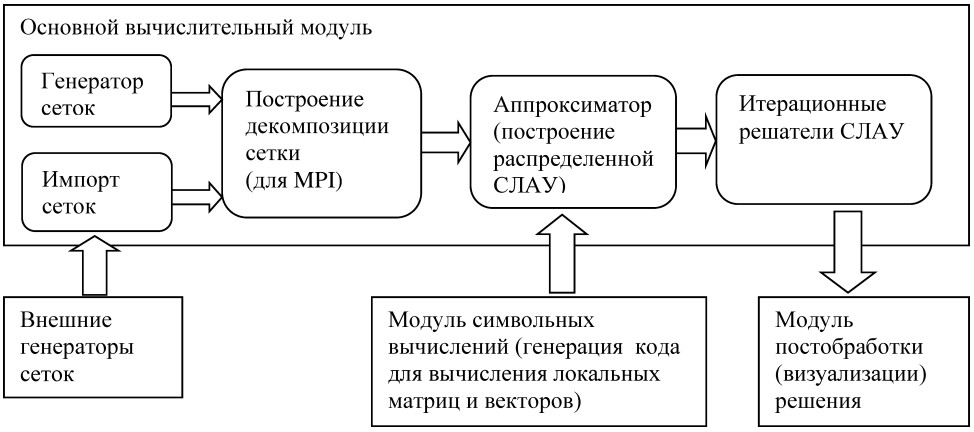


Рис. 1. Архитектура пакета Helmholtz3D

3.1. Генерация сеток

Задача генерации сеток является очень важной, так как сама сетка оказывает большое влияние на точность полученного решения. Такие сетки должны иметь адаптивные сгущения как вблизи мелких геометрических объектов расчетной области, так и вблизи особенностей решения, вызванными наличием высоко-контрастных сред. Данной проблеме посвящено большое количество работ отечественных и зарубежных авторов, однако эта проблема выходит за рамки текущей работы. Поэтому в пакете Helmholtz3D был разработан простейший генератор сеток, имеющий возможность построения квазиструктурированных сеток в областях, составленных из параллелепипедов. Однако для того, чтобы пользователи имели возможность использования качественных адаптивных сеток, был реализован их импорт из формата пакета NETGEN [7].

3.2. Декомпозиция расчетной области

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

В данной работе предлагается геометрический подход к декомпозиции области (см. подробнее [8]), заключающийся в декомпозиции сетки расчетной области при помощи построения упрощенного BSP-дерева [9]. Упрощение состоит в том, что секущие плоскости предлагается проводить ортогонально осям координат. Алгоритм рекурсивно разделяет имеющееся множество тетраэдров на две приблизительно равные части, минимизируя при этом количество тетраэдров, принадлежащих обоим подобластям. При эффективной реализации и использовании достаточно регулярных сеток можно добиться алгоритмической сложности метода $O(N_T \log N_T)$, где N_T — количество тетраэдров сетки.

3.3. Построение аппроксимаций вариационных постановок задач

Дискретизация вариационной постановки задачи приводит к задаче (6)–(7). Как уже было отмечено, построение СЛАУ такого вида может быть проведено на основе поэлементной технологии с вычислением локальных матриц и векторов. Дополнительный плюс такого подхода заключается в том, что части распределенной СЛАУ могут генерироваться непосредственно на тех узлах системы, на которых они в дальнейшем будут нужны. Это

позволяет избавиться от необходимости хранить глобальную СЛАУ на каком-либо из узлов, что снимает ограничения на размер решаемых систем.

Базисные функции пространств $\mathcal{W}_l$ могут быть выписаны с использованием кусочно-линейных функций $\xi_i(\vec{r})$ координат, равных единице в одной из вершин сетке и нулю — в остальных [3]. Далее, несложно заметить, что функции $\vec{\psi}_i^0$ и $\nabla \times \vec{\psi}_i^0$ могут быть приведены к следующему каноническому виду:

$$\sum_j c_j \xi_1^{p_{j,1}} \xi_2^{p_{j,2}} \xi_3^{p_{j,3}} \xi_4^{p_{j,4}} \vec{F}_j(\xi_1, \xi_2, \xi_3, \xi_4), \tag{8}$$

где c_j — некоторая константа, $\xi_1, \ldots, \xi_4$ — четыре не равных нулю функций ξ_i в данном тетраэдре, $p_{j,k}$ — целочисленные степени, $\vec{F}_j(\xi_1, \xi_2, \xi_3, \xi_4)$ — константная вектор-функция, не зависящая от координат. Тогда интегралы для вычисления коэффициентов матриц S и M в каждом из тетраэдров могут быть приведены к виду

$$\sum_j \sum_k c_j c_k \left(\vec{F}_j(\xi_1, \xi_2, \xi_3, \xi_4) \cdot \vec{F}_k(\xi_1, \xi_2, \xi_3, \xi_4) \right) \int\limits_{\Omega_T} \xi_{i_1}^{p_1} \xi_{i_2}^{p_2} \ldots \xi_{i_{j+k}}^{p_{j+k}} \mathrm{d}\Omega, \tag{9}$$

Такие интегралы могут быть вычислены аналитически [1]. Для автоматизации процесса генерации локальных матриц по формуле (9) был реализован модуль, по заданному описанию базисных функций генерирующий выражения для вычисления элементов матриц и проводящий символьные оптимизации получаемых выражений. Схема программы представлена на рисунке 2.

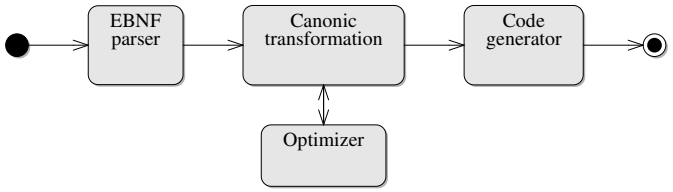


Рис. 2. Схема работы генератора выражений

Правила описания базисных функций задают некоторую формальную грамматику и могут быть записаны в расширенной форме Бэкуса–Наура (РБНФ, или EBNF). Набор базисных функций читается парсером РБНФ (EBNF parser) и преобразуется в промежуточный вид, соответствующий дереву разбора выражений. Поскольку не любое выражение, записанное в предложенной грамматике является корректным, то в парсер дополнительно встроен валидатор, который проверяет выражение на совместимость.

Проводившиеся символьные оптимизации состояли в приведении подобных слагаемых и исключении слагаемых в (9) с нулевыми коэффициентами. Основная проблема, решавшаяся в рамках процедуры приведения подобных слагаемых — идентификация одинаковых (с точностью до порядка сомножителей) векторных частей в термах (9). Для решения этой проблемы было предложено использовать рекурсивную процедуру приведения векторного выражения к виду, минимальному в заданном лексикографическом порядке.

3.4. Распределенные итерационные решатели

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

$$\bar{A}\bar{u} = \bar{f}.$$

Для решения таких СЛАУ широко используются методы Шварца. Одна итерация аддитивного метода Шварца в матричном виде может быть записана как (см. [10])

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

где R_p — оператор сужения на подобласть Ω_p, $p = 1, \ldots, P$; здесь и далее подчеркивание для u_i мы опускаем. Матрицу A_p для подобласти p можно представить в виде $A_p = R_p A R_p^T$.

Проблема состоит в том, что для уравнения Гельмгольца метод Шварца может не сходиться. Однако можно заметить, что решение системы методом Шварца есть ни что иное, как решение методом простой итерации предобусловленной системы

$$B^{-1} \bar{A} \bar{u} = B^{-1} \bar{f}, \quad B^{-1} = \sum_p R_p^T A_p^{-1} R_p. \tag{10}$$

Такую предобусловленную систему можно решать итерационными методами в подпространствах Крылова. Однако метод Шварца имеет недостаток, заключающийся в том, что количество итераций, необходимое для достижения заданной точности, растет с увеличением числа подобластей [10]. Для преодоления этого недостатка можно воспользоваться методом коррекции на грубой сетке [11] (Coarse Grid Correction, CGC).

Для этого введем операторы грубосеточного сужения R_c, определенный как оператор сужения на базис подпространства $\mathcal{W}_1$, а также оператор продолжения $P_\mathrm{c} = R_\mathrm{c}^T$. Тогда итерация метода аддитивного Шварца с грубосеточной коррекцией записывается как

$$\begin{aligned} u_{i+1/2} &= u_i + P_c A_1^{-1} R_c (\bar{f} - \bar{A} u_i), \\ u_{i+1} &= u_{i+1/2} + \textstyle\sum_p R_p^T A_p^{-1} R_p (\bar{f} - \bar{A} u_{i+1/2}), \end{aligned}$$

где $\bar{A}_1 = R_c \bar{A} P_c$ — матрица, полученная при использовании базисных функций порядка 1.

Мы будем решать предобусловленную систему вида (10), где предобуславливатель B^{-1} при наличии грубосеточной коррекции имеет вид

$$B^{-1} = \sum_p R_p^T A_p^{-1} R_p \left[I - \bar{A} P_c \bar{A}_1^{-1} R_c \right] + P_c \bar{A}_1^{-1} R_c. \tag{11}$$

Поскольку метод коррекции на грубой сетке не очень чувствителен к точности аппроксимации A_1^{-1} [12], для приближенного решения систем вида $\bar{A}_1 v = q$ можно использовать итерационные методы в подпространствах следов, см. например [13]. Однако приближенное обращение матрицы $\bar{A}_1$ приводит к тому, что предобуславливатель B^{-1} становится переменным. В этом случае хорошим выбором для итерационного решения задач с таким предобуславливателем оказывается метод Flexible GMRES [10], допускающий использование переменных предобуславливателей.

Для решения задач в подобластях $A_p x = b$ можно использовать какой-нибудь прямой решатель для разреженных систем, например решатель PARDISO из библиотеки Intel MKL [6]. Проблема состоит в том, что время, требуемое PARDISO для разложения матриц, растет практически как $O(N^3/P^3)$, поэтому такой метод подходит только для решения не очень больших систем на большом числе процессоров. Однако действие обратных матриц A_p^{-1} можно также вычислять приближенно, например методом FGMRES с предобуславливателем AMG, основанным на использовании иерархических базисов, предложенным в [13].

3.5. Методы визуализации электромагнитных полей

Визуализация электромагнитных полей является неотъемлемой частью численного эксперимента, поскольку позволяет проанализировать характерное распределение полей в расчетной области. В связи с этим, в рамках пакета Helmholtz3D был реализован модуль визуализации электромагнитных полей, основанный на технологии OpenGL.

Визуализация электромагнитных полей осуществляется следующим образом. Визуализатор предоставляет возможность визуализировать одно из полей – электрическое либо магнитное, и позволяет переключаться между этими двумя режимами. Поля отрисовываются в каждом из тетраэдров в виде стрелки, сонаправленной с полем в данной точке, и имеющей длину, пропорциональную амплитуде поля в текущий момент времени. Для большей наглядности амплитуда поля кодируется цветом – с линейным переходом от желтого (максимально возможная амплитуда для текущего поля), до синего (близкая к нулю амплитуда). Отметим, что поскольку амплитуда поля представляется в виде двух векторов – действительной и мнимой компоненты, то в заданный момент времени t реальная амплитуда поля задается формулой

$$\vec{F}(t) = \vec{F}_{re}\cos(\omega t) + \vec{F}_{im}\sin(\omega t),$$

где $\vec{F}_{re}$ и $\vec{F}_{im}$ — векторы амплитуд действительной и мнимой части для соответствующего поля (электрического либо магнитного). Для примера на рисунке 3 приведена визуализация электрического поля в волноводе с однородной средой. На рисунке хорошо видна структура электрического поля.

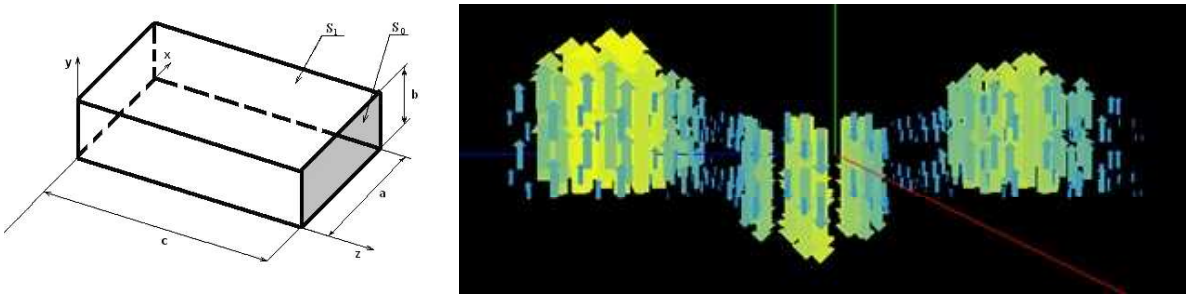


Рис. 3. Пример визуализации электрического поля в волноводе

4. Технологии параллельного программирования и оптимизации для систем с распределенной и общей памятью

Параллельные распределенные алгоритмы в пакете Helmholtz3D построены на основе широко распространенного стандарта интерфейса обмена данными MPI (Message Passing Interface). При этом используемые в пакете итерационные решатели с предобуславливателями типа аддитивного Шварца естественным образом ложатся на архитектуру современных вычислительных систем. Действительно, итерационное решение СЛАУ методом FGMRES требует 3 следующих типа операций.

К первому типу относятся векторно-векторные операции. Для имеющейся декомпозиции задачи на подобласти распараллеливание таких операций проводится естественным образом, за исключением операций вычисления скалярного произведения и нормы, для которых необходима точка синхронизации и обмен данными. Однако в интерфейсе MPI имеется требуемая функция **MPI_Allreduce**, реализация которой оптимизируется поставщиком библиотеки MPI.

Для умножения матрицы на вектор решатель строит в каждой подобласти списки узлов, являющихся граничными для соседних подобластей. В дальнейшем решатель использует собранную информацию для пересылки частей вектора между процессами. Ключевым моментом в таком подходе является уменьшение объема пересылаемых данных — в соседние подобласти требуется пересылать только граничные коэффициенты векторов вместо того, чтобы пересылать вектор целиком. В итоге, общий объем пересылаемых данных в трехмерных задачах электромагнетизма для одного умножения матрицы на вектор будет пропорционален $O(N^{2/3}\log P)$ вместо $O(N)$.

Применение предобуславливателей, основанных на методе Шварца, требует на каждой итерации внешнего алгоритма решения подзадач в подобластях, и, согласно (10), не требуют коммуникаций. Распараллеливание решателя в подобластях проводилось с использованием средств OpenMP. Были реализованы параллельные процедуры умножения матрицы на вектор, а также параллельное решение разреженных треугольных систем для оператора сглаживания SSOR (Symmetric Successive Over-Relaxation, [10]) в предобуславливателе AMG. Распараллеливание треугольного решателя проводилось при помощи метода планирования вычислений по уровням: при решении системы вида $Ux = y$ для каждой вершины вычисляется ее уровень l_i:

$$l_i = \max\{l_j + 1 : U_{i,j} \neq 0,\ j > i\}. \tag{12}$$

После этого коэффициенты вектора x с одинаковым уровнем можно вычислять независимо.

Можно отметить ряд полезных оптимизаций решателя в подобластях. Для начала необходимо отметить его особенность: такое итерационное решение СЛАУ в подобластях оказывается bandwidth-limited, то есть существенно ограничено пропускной способностью шины данных системы, поскольку за одну итерацию алгоритма, фактически, требуется несколько раз прочитать всю матрицу системы (в том числе для решения двух треугольных систем), и несколько раз прочитать и записать различные векторы. Таким образом, реализованные оптимизации можно разделить на две категории: оптимизации доступа к памяти за счет переупорядочивания матрицы и оптимизации, направленные на повышение производительности алгоритма на NUMA-системах, заключающиеся в правильной инициализации областей памяти. Подробнее об этих оптимизациях см. [14].

Отметим, что помимо матрично-векторных операций, решатель в подобластях использует прямой метод PARDISO для самой “грубой” системы в предобуславливателе AMG.

5. Численные эксперименты

Предлагаемые в работе алгоритмы были апробированы на серии методических и практических задач электромагнетизма.

Первой из задач является вычисление электромагнитного поля в параллелепипедальном волноводе $[0, a] \times [0, b] \times [0, c]$, $a = 72$ мм, $b = 34$ мм и $c = 200$ мм, см. рис. 3.

Физические параметры среды: $\mu_\text{r} = 1$, $\dot{\varepsilon}_\text{r} = 1 - 0.1i$, $\omega = 6\pi \cdot 10^9$ Гц. При $z = 200$ мм поставлено краевое условие (3) с $\vec{E}_0 = \vec{e}_y \sin(\pi x/a)$, на остальной части границы — краевое условие идеального проводника $\vec{E}_0 = \vec{0}$. Известен аналитический вид решения:

$$\vec{E} = \vec{e}_\text{y} \sin\left(\frac{\pi x}{a}\right) \frac{\sin \gamma z}{\sin \gamma c}\ . \tag{13}$$

Сетка в данном случае была сгенерирована средствами пакета Helmholtz3D.

Вторая из рассматриваемых задач — задача электромагнитного каротажа. Рассматриваемая расчетная область в виде куба с центром в начале координат и стороной 10 метров. Параметры $\mu_r = 1$, $\varepsilon_r = 1$ во всей области. Проводимость среды $\sigma = 0.1$ См/м, в среде имеется слой с $\sigma = 0.05$ См/м. Координаты слоя по оси Oz: от -0.425 м до -0.275 м. В среде пробурена вертикальная цилиндрическая скважина радиуса 0.108 м с центром в начале координат в плоскости Oxy. В скважине имеется цилиндрическая каверна внешнего радиуса 0.118 м и положением по оси Oz от -0.0725 до 0.0725 м. Проводимость скважины и каверны $\sigma = 5$ См/м. В скважину вставлен полый цилиндрический прибор радиуса 0.043 м, смещенный по оси x относительно центра скважины на -0.064, заполненный воздухом ($\sigma = 0$). В приборе имеется одна генераторная и две приемные петли радиуса 0.0365 м при $z = 0.5$ м и при $z = 0.0$ и $z = 0.1$ м соответственно. По генераторной петле течет ток 1 А с частотой 14 МГц. Искомой является разность фаз ЭДС в приемниках.

Генерация неравномерной сетки проводилась при помощи утилиты NETGEN [7]. Количество тетраэдров сетки $N_T = 490282$, порядок СЛАУ без учета пересечений $N_0 = 8946864$.

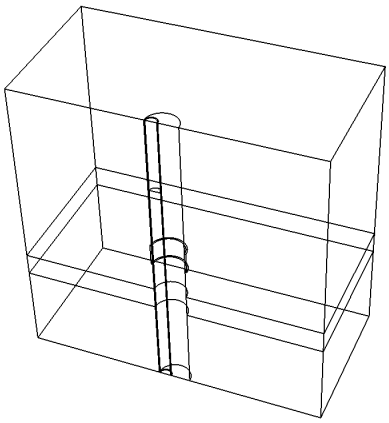


Рис. 4. Расчетная область для задачи электромагнитного каротажа

Тестирование проводились на двух системах: OpenMP решатель для систем с общей памятью (соответствует $P = 1$) — на системе Intel® Xeon X7560, 2.27 ГГц, 4×8 ядер, 64 ГБ памяти, гибридный решатель — на узлах HP BL2x220 G6 кластера НКС-30Т. Каждый узел кластера — Intel Xeon E5540, 2.53 ГГц, 2x4 ядер, 16 ГБ памяти.

Для аппроксимации использовались базисные функции $\mathcal{W}_3$ третьего порядка.

В качестве критерия останова итерационного процесса было выбрано уменьшение нормы невязки в ε раз: $|r_i| < \varepsilon |f|$. Для внутренних итераций FGMRES с предобуславливателем AMG $\varepsilon_{\mathrm{A}} = 0.2$, для грубосеточной коррекции в подпространстве следов $\varepsilon_{\mathrm{c}} = 0.1$. Решение задач в подобластях при обращении A_p проводилось с $\varepsilon_p = 0.01$. Внешние итерации проводились с точностью $\varepsilon = 10^{-7}$ за исключением сетки $116 \times 55 \times 320$ для первой задачи, для которой использовалось $\varepsilon = 10^{-9}$ для получения $\vec{E}^h$ нужной точности.

Таблица 1. Масштабируемость гибридного кластерного решателя на модельной задаче

Сетка, N_0		Количество подобластей (MPI процессов)					
и δE		16	32	64	128[1]	256[1]	512[1]
$58 \times 28 \times 160$	N_{b}	705312	1168248	1777626	2599050	3928914	5804454
	n	8	9	10	10	11	12
28519914	n_{c}	266	385	382	468	577	704
	n_{A}	24	27	29	29	32	36
$7.1 \cdot 10^{-7}$	t	1.2e3	5.2e2	2.3e2	1.1e2	8.0e1	5.7e1
$116 \times 55 \times 320$	N_{b}					13035042	18153966
	n					13	14
225335973	n_{c}	N/A	N/A	N/A	N/A	1102	1142
	n_{A}					40	42
$9.0 \cdot 10^{-8}$	t					1.2e3	9.2e2

[1]Используется более одного MPI процесса на узел (всего 64 узла).

В табл. 1–3 приведены результаты тестирования решателей на модельной задаче и задаче электромагнитного каротажа: N_0 порядок системы без учета пересечений, δE — максимальная относительная ошибка электрического поля, N_b — размерность пространства следов для A_1, n — количество внешних итераций распределенного решателя, n_3 и n_2 — общее количество внешних и внутренних (в предобуславливателе AMG) итераций OpenMP решателя, n_c — общее количество итераций грубосеточной коррекции, n_A — максимальное количество внешних итераций метода FGMRES в подобластях, t — время решения СЛАУ в секундах в экспоненциальном формате $a.b\mathsf{e}n = a.b\cdot 10^n$. Результат решения задачи электромагнитного каротажа показал хорошее соответствие с результатом группы НГТУ, которая решала эту задачу другим методом — методом выделения поля источника, при этом искомая разность фаз ЭДС в приемниках совпала с ошибкой 1.8%

Таблица 2. Масштабируемость OpenMP решателя на задаче электромагнитного каротажа

	Количество OpenMP потоков					
	1	2	4	8	16	32
n_3	14	14	14	14	14	14
n_2	33	33	33	33	33	33
t	7.5e3	4.1e3	2.2e3	1.3e3	8.8e2	6.1e2

Таблица 3. Масштабируемость MPI решателя на задаче электромагнитного каротажа

	Количество подобластей (MPI процессов)							
	4	8	16	32	64	128[1]	256[1]	512[1]
N_b	102918	236787	477672	803112	1346418	2038260	2873652	3972480
n	14	19	19	36	35	36	38	42
n_c	86	131	148	310	310	341	383	445
n_A	49	61	61	111	109	113	123	126
t	8.8e2	4.9e2	2.3e2	1.9e2	9.2e1	6.2e1	5.5e1	4.9e1

Из таблиц видно, что предобуславливатели AMG и CGC успешно уменьшают количество внешних итераций до приемлемого значения. Количество итераций грубосеточной коррекции растет, но гораздо медленнее, чем линейно, за счет трехмерной декомпозиции расчетной области. Тем не менее, при дальнейшем росте размера СЛАУ может потребоваться геометрическое огрубление задачи для A_1, например [15], в результате чего мы получим многоуровневую схему грубосеточной коррекции.

6. Заключение

В работе представлен пакет параллельных прикладных програм Helmholtz3D. Данный пакет позволяет проводить расчеты трехмерных электромагнитных полей с гармонической

[1]Используется более одного MPI процесса на узел (всего 64 узла).

зависимостью от времени. Предложенные алгоритмы позволяют рассчитывать электромагнитные поля в с областях со сложной разномасштабной геометрией и высококонтрастными средами с высокой точностью, продемонстрированной на методических и практических задачах. Параллельные алгоритмы решения СЛАУ для систем с общей и распределенной памятью позволяют решать сверхбольшие задачи — до нескольких сотен миллионов неизвестных — за время порядка 10–15 минут.

Литература

1. Соловейчик Ю.Г., Рояк М.Э., Персова М.Г. Метод конечных элементов для решения скалярных и векторных задач. Новосибирск: Изд-во НГТУ, 2007.
2. Monk P. Finite Element Methods for Maxwell's Equations. Oxford University Press, 2003.
3. Ingelstrom P. A new set of H(curl)-conforming hierarchical basis functions for tetrahedral meshes // IEEE Transactions on Microwave Theory and Techniques. 2006. Vol. 54, N. 1. P. 160–114.
4. Ильин В.П. Методы и технологии конечных элементов. Новосибирск: Изд-во ИВМиМГ СО РАН, 2007.
5. Eigen: URL: http://eigen.tuxfamily.org (дата обращения 12.12.2012).
6. Intel (R) Math Kernel Library from Intel:
URL: http://software.intel.com/en-us/articles/intel-mkl (дата обращения 12.12.2012).
7. Schöberl J. NETGEN — An advancing front 2D/3D-mesh generator based on abstract rules // Computing and Visualization in Science. 1997. Vol. 1, N. 1. P. 41–52.
8. Бутюгин Д.С. Алгоритмы решения СЛАУ на системах с распределенной памятью в применении к задачам электромагнетизма // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. Т. 46, N. 305. С. 5–18.
9. Fuchs H., Kedem Z.M., Naylor B.F. On visible surface generation by a priori tree structures // ACM Computer Graphics. 1980. Vol. 14, N. 3. P. 124–133.
10. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, 2003.
11. Bramble J., Pasciak J., Schatz A. The construction of preconditioners for elliptic problems by substructuring. I. // Math. Comput. 1986. Vol. 47, N. 175. P. 103–134.
12. Nabben R., Vuik C. A comparison of deflation and coarse grid correction applied to porous media flow // SAIM J. Numer. Anal. 2004. Vol. 42, N. 4. P. 1631–1647.
13. Butyugin D.S. Efficient iterative solvers for time-harmonic Maxwell equations using domain decomposition and algebraic multigrid // Journal of Computational Science. 2012. Vol. 3, N. 6. P. 480–485.
14. Бутюгин Д.С. Параллельный предобуславливатель SSOR для решения задач электромагнетизма в частотной области // Вычислительные методы и программирование. 2011. Т. 12, N. 1. С. 110–117.
15. Hu J., Tuminaro R., Bochev P., Garasi C., Robinson C. Toward an h-independent algebraic multigrid method for Maxwell's equations // SIAM Journ. Sci. Comp. 2005. Vol. 27, N. 5. P. 1669–1688.

Вычислительная химия на российских грид-полигонах: текущее состояние, проблемы и перспективы *

В.М. Волохов[1], Д.А. Варламов[1,2], А.В. Волохов[1],
А.В. Пивушков[1], Г.А. Покатович[1], А.И. Прохоров[1]

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

Проводится анализ текущего состояния вычислительных грид-сервисов в области квантовой химии и молекулярной динамики, реализованных в рамках существующих российских грид-полигонов. Кратко описаны принципы адаптации и функционирования основных прикладных программных пакетов (ППП) вычислительной химии. Сформулированы основные проблемы по работе с подобными сервисами в условиях российских грид-полигонов и возможные пути их решения и перспективы развития проблемно-ориентированных грид-сервисов.

1. Введение

Вычислительная химия и сопряженные с ней области знаний являются одними из наиболее заинтересованных в суперкомпьютерных и распределенных грид-вычислениях (в том числе и на входящих в состав грид-полигонов суперкомпьютерах) отраслями науки. Исследования, проводимые в области химии и смежных наук, в настоящее время, как правило, неэффективны без использования сверхмощных параллельных и распределенных вычислительных ресурсов для решения задач самых разных классов. Первыми научными задачами, использовавшими реальную петафлопсную производительность суперкомпьютеров, стали задачи вычислительной химии: расчеты электронный структуры высокотемпературных сверхпроводников и исследования эффекта магнитосопротивления в наночастицах методом Монте-Карло на суперкомпьютере “Jaguar” (Oak Ridge, USA – 1,64 Пф). В качестве ярких примеров можно привести ресурсные требования к достаточно тривиальным (на первый взгляд) химическим задачам: (а) исследования свойств воды в низкоразмерных системах – до $8 \cdot 10^6$ CPU-часов в Argonne National Laboratory; (б) исследования химических катализаторов в области нефтехимии – до $30 \cdot 10^6$ CPU-часов в год (“Jaguar”). Потенциально же задачи в области молекулярного моделирования и многоиерархического моделирования материальных объектов от квантово-механического уровня до уровня сплошных сред и конструкций могут выходить на уровень востребованности многих петафлопс. Например, некоторые задачи оптимизации крупных молекулярных структур требуют выполнения до 10^9 отдельных расчетов. Типичный докинг белковых лигандов с размерностью 200 атомов × 300 000 конфигураций × 1000 CPU-часов требует до 300 Пф вычислительных мощностей. Для построения многомерных потенциальных поверхностей, адекватно описывающих химические реакции, нужно провести $10^2 – 10^N$ независимых ресурсоемких *ab initio* расчетов. Для исследования динамики химической реакции с использованием классических траекторий зачастую нужно рассчитать до 10^7-10^9 независимых траекторий. Численное исследование многопараметрических функций $F(x_1,x_2…x_n)$ в области изменения параметров $x_1,x_2…x_n$ при разбиении диапазона изменения каждого параметра на 10 ячеек требует проведения 10^N независимых расчетов F.

Востребованность вычислительной химией все возрастающих вычислительных ресурсов подтверждается тем, что большинство суперкомпьютерных центров США (San Diego, Ohio, Illinois *etc.*) предоставляют до 40% вычислительных мощностей для нужд биохимии, молекулярного моделирования, квантовой химии, нанотехнологических расчетов. Создано достаточно много проблемно-ориентированных суперкомпьютерных центров, специализирующихся почти

* Исследовательские работы финансово поддерживаются Государственным контрактом (заявка №2012-1.1-12-000-2003-034, соглашение от 10.07.2012) в рамках ФЦП «Научные и научно-педагогические кадры инновационной России на 2009-2013 годы»

исключительно на квантово-химических и молекулярно-динамических расчетах: The UK National Service for Computational Chemistry Software (Великобритания, http://www.nsccs.ac.uk) – все виды вычислений; The National Resource for Biomedical and Chemistry Supercomputing (США, http://www.nrbsc.org) расчет молекулярных систем от 20000 до 120000 атомов, до 10^5 CPU-часов на структуру; Chemical Computing Group (http://www.chemcomp.com), Канада; Lawrence Berkeley National Laboratory (США, http://www.lbl.gov/csd), сегмент вычислительной химии до 450 Tflops; Lehrstuhl für Theoretische Chemie der Technische Universität München, Германия (http://www.lrz.de/services/software/chemie); Swiss National Supercomputing Centre (Швейцария, http://www.cscs.ch) – выделение до субпетафлопсных ресурсов для химических вычислений и т.п. К сожалению, при наличии в Российской Федерации достаточно мощных вычислительных суперкомпьютерных центров (МГУ, МСЦ, Саров и т.п.) подобные проблемно-ориентированные центры достаточной мощности в настоящее время практически отсутствуют.

Как правило, масштабные задачи химии требуют либо достаточно эксклюзивного использования суперкомпьютеров, что далеко не всегда приемлемо, или же вычислительных ресурсов, которые не может предоставить ни один из вычислительных центров. Это неизбежно приводит к необходимости использования для решения многих подобных задач мощностей крупных распределенных грид-полигонов, причем включающих суперкомпьютеры петафлопсной и выше производительности. Даже в условиях стремительного развития мощностей единичных установок (первые единицы и десятки петафлопс – в России это «Ломоносов», создаваемый суперкомпьютер МСЦ, Саровский центр и др.), значительная часть подобных задач может быть разрешена только в условиях распределенных вычислительных полигонов.

Для этого необходимы устойчиво работающие распределенные вычислительные системы, обеспечивающие как одновременный процессинг десятков и сотен тысяч разноплановых и разномасштабных задач, так и передачу данных объемом до сотен терабайт в сутки (и хранение многих петабайт данных).

Классический «грид» представляет собой географически распределенные инфраструктуры, объединяющие на основе единой технологии и установленных протоколов и правил множество ресурсов разных типов (расчетные узлы – включая суперкомпьютерные установки, хранилища и базы данных, высокоскоростные каналы связи), доступ к которым грид-пользователь может получить практически из любой точки независимо от места расположения. Грид-полигоны предлагают коллективный разделяемый доступ к ресурсам и к связанным с ними вычислительным грид-сервисам (в том числе проблемно-ориентированным) в рамках создаваемых распределенных виртуальных организаций, состоящих из ресурсных центров, сервисной грид-инфраструктуры и отдельных грид-пользователей, совместно использующих предоставляемые в общее пользование ресурсы. Каждая виртуальная организация устанавливает (в соответствии с общими правилами грид-полигона) собственную политику поведения участников, регламентирует используемое ПО (в том числе прикладное), обеспечивает контроль выполнения заданий и мониторинг и учет (аккаунтинг) использования ресурсов.

Современные грид-инфраструктуры (в том числе реализованные на российских вычислительных ресурсах) обеспечивают достаточно устойчивую интеграцию вычислительных ресурсов (и базирующихся на них сервисов), находящихся в разных организациях в единую вычислительную среду, позволяющую решать задачи по обработке большого количества задач и сверхбольших объемов данных.

2. Вычислительная химия на грид-полигонах в России

Авторы опираются на опыт использования грид-технологий в интересах вычислительной химии и смежных отраслей науки на суперкомпьютерных установках и в грид-средах, что является одним из основных направлений работы вычислительного центра Института Проблем Химической Физики в Черноголовке (ИПХФ РАН, http://www.icp.ac.ru). В настоящее время Институт располагает богатейшей в России библиотекой параллельных квантово-химических и молекулярно-динамических программ (авторских, «open source» и лицензионных), что позволяет проводить обширные эксперименты по адаптации подобных программ к грид-средам в интересах собственных пользователей. В течение года в институте проводится расчет от 3 до 4 тысяч вычислительных задач высокой сложности с публикацией более чем 400 печатных работ с

использованием результатов проведенных расчетов. Работы с использованием грид-вычислений в интересах квантовой химии и молекулярной динамики в ИПХФ ведутся с 2004 года, и в настоящее время осуществляются под эгидой нескольких государственных программ (включая Федеральные Целевые Программы, Программы Президиума РАН, гранты РФФИ). ИПХФ является инициатором по использованию средств вычислительной химии в ряде ранее созданных российских грид-полигонов разного масштаба : ГридННС (Национальная Нанотехнологическая Сеть, http://www.ngrid.ru), СКИФ-Полигон (http://skif-grid.botik.ru), EGEE-RDIG, сейчас EGI-[RU-NGI] - http://www.egee-rdig.ru), а также создаваемой в рамках ФЦП пилотной Российской грид-сети для высокопроизводительных вычислений.

2.1 Вычислительные грид-сервисы на базе проблемно-ориентированных прикладных пакетов

Одним из основных вариантов создания вычислительных грид-сервисов в рамках распределенных полигонов является адаптация прикладных программных пакетов (далее – «ППП») для их использования грид-пользователями. Адаптация прикладных пакетов проводилась авторами для различных распределенных сред, основанных на middleware gLite, Unicore, Globus Tools в условиях основных вышеперечисленных российских грид-полигонов

Как правило, реализация ППП в виде грид-сервисов помимо установки ППП на ресурсные сайты, адаптации их к распределенной среде и отладки подразумевает создание высокоуровневых дружелюбных конечному пользователю Web-интерфейсов, что значительно снижает трудоемкость работы пользователя с подобными пакетами в условиях распределенных вычислительных сред.

Для всех адаптированных прикладных пакетов были созданы подобные высокоуровневые web-интерфейсы к GRID средам с большим набором функций. Созданные интерфейсы позволяют грид-пользователю работать с распределенными средами через Интернет-браузеры и осуществлять следующие действия:

- авторизовать пользователя при входе в пространство грид-полигона на основе полученных им предварительно сертификатов полигона, далее – получать прокси-сертификаты грид-среды с учетом планируемой продолжительности заданий;
- подготовить задания (включая загрузку или создание начальных данных и конфигурационных файлов и их редакцию) в соответствие с требованиями пакета;
- запускать прикладные пакеты в инфраструктуре грид-полигона (при необходимости – на произвольном или выбранном пользователем грид-ресурсе);
- вести постоянный мониторинг выполнения задания (включая его останов и перезапуск при необходимости);
- по завершении – получить результаты счета или передавать их на указанный GridFTP сервис.

В последующей перспективе стоит построение интерактивных интерфейсов для формирования сложных конфигурационных файлов прикладных пакетов на основе систем иерархических меню и «деревьев» выбора, включая возможность работы с типовыми (для конкретных прикладных пакетов) шаблонами заданий, которые сводят определение задач к варьированию одного-двух наиболее значимых параметров, тем самым резко упрощая работу конечного грид-пользователя (например, выбор легирующих атомов металлов для фуллеренов или нанотрубок, указание фиксированных P-T условий расчета для стандартных молекул и т.п.) .

В той или иной степени для работы в указанных выше грид-средах были адаптированы следующие квантово-химические и молекулярно-динамические ППП: Gaussian 03, GAMESS US, Firefly, CPMD, Dalton 2012, NWChem, NAMD, AbInit, GROMACS, VASP, PWScf, Orca и др. Большинство ППП, их функциональность и особенности адаптации были описаны авторами ранее [1,2], в том числе в трудах данной конференции 2012 года.

Для всех указанных ППП в разной степени (в зависимости от лицензий) был организован доступ в рамках различных ВО (например, в ГридННС – «NanoChem» плюс ВО, относящихся к конкретным ППП – Gamess, AbInit, GROMACS и т.п.) через порталы ГридННС

https://ui.ngrid.ru и портал пилотной российской грид-сети для высокопроизводительных вычислений (поддерживается ФГУП «Восход»). Как уже говорилось, часть ППП была также ранее реализована авторами на пространстве грид-полигонов EGEE(EGI)-RDIG и СКИФ-Полигона, а в настоящее время большинство реализовано и в рамках созданной в 2011-2012 годах пилотной Российской грид-сети для высокопроизводительных вычислений. Как правило, адаптация пакетов для разных грид-сред различается особенностями реализации низкоуровневых интерфейсов между middleware, используемой грид-полигоном (gLite, Unicore, Globus Tools и т.п.) и собственно ППП. Большинство вышеперечисленных пакетов установлены на кластерах ресурсных центров, входящих в ГридННС (в том числе – практически все из вышеперечисленных установлены на ресурсном грид-центре ИПХФ) и доступны в качестве вычислительных грид-сервисов. Для части ППП сделаны тестовые инсталляции (в основном из-за проблем с лицензиями). Для пакетов после установки и тестирования на кластерах настроены шлюзы для приема входящих грид-заданий и работы с GridFTP ресурсами, установлены высокоуровневые интерфейсы (различной степени сложности) и проведено массовое тестирование (включая стресс-тесты) на различных задачах. ИПХФ выступал в качестве установщика ряда адаптированных пакетов (Gamess, Gaussian), предоставлял ресурсы своего грид-центра и производил тестирование созданных грид-сервисов через низкоуровневые и высокоуровневые web-интерфейсы грид-среды.

2.2 Разработка некоторых технологий для повышения эффективности работы с проблемно-ориентированными сервисами в грид-средах

Помимо адаптации ППП в качестве вычислительных грид сервисов, развитие вычислительной химии применительно к распределенным и суперкомпьютерным средам требует развития новых технологий по запуску и эксплуатации химического ПО. Для решения ряда проблем, возникших при использовании распределенных и суперкомпьютерных вычислений в интересах вычислительной химии авторами в рамках грид-сервисов разработаны (и продолжают развиваться) несколько технологий как оригинального плана, так и созданные на основе существующих разработок:

1. Проведено изучение способов применения современных методов проведения распределенных расчетов на грид-полигонах прикладных пакетов в области квантовой химии и молекулярной динамики с использованием GPU (Graphical Processor Units) устройств, что во многих случаях ведет к убыстрению расчетов от десятков процентов до первых порядков по времени, либо (как вариант) – возможности существенного повышения точности или детальности расчетов. В настоящее время резко интенсифицировался перевод программного кода прикладных пакетов на параллельные технологии с использованием технологий программирования CUDA™ и аналогичных. При этом программирование гибридного кода для квантово-химических и молекулярно-динамических пакетов занимает одно из ведущих позиций в этом направлении. Развитие GPU технологий достигло точки, когда множество существующих приложений реализуются с их использованием и работают значительно быстрее, чем на обычных многоядерных системах. Это позволяет проводить высокоинтенсивные вычисления с применением встраиваемых в расчетные узлы кластеров высокопроизводительных графических процессоров (Nvidia – Tesla и Kepler, AMD – ATI Radeon). В настоящее время авторами проводятся методические исследования на предмет широкого использования новых программных вариантов прикладных квантово-химических пакетов, тестирование и оптимизацию имеющихся вариантов с последующей адаптацией их к проведению расчетов в грид-средах. Одновременно разрабатываются методики запуска грид-заданий, ориентированных на использование ресурсов с поддержкой CUDA™ технологий, в том числе на доступных суперкомпьютерных установках. Более детально этот вопрос рассмотрен в статье авторов, представленной в данном сборнике трудов [3].
2. Проводится создание на новом технологическом уровне системы для работы с большими пулами параллельно-независимых грид-заданий с целью решения задач, требующих проведения расчетов на больших массивах равномерных данных или варьирующих входных параметров. Система предназначена как для решения большого класса многопараметрических задач квантовой химии (с многими варьирующими параметрами или на равномерных

независимых «сетках» данных), так и для проведения расчетов задач, предусматривающих использование массового запуска стандартных прикладных пакетов при значительном варьировании 1-2 входных параметров. Предусмотрено обеспечение следующих функций расчетов: разбиение первичной задачи на равномерных областях данных или входных параметров, автоматическое формирование пула независимых друг от друга грид-заданий, их «параллельный» запуск в грид-средах, мониторинг и контроль выполнения заданий в рамках пула, сборку результатов расчетов заданий с грид-ресурсов в конечный обобщающий результат. Система должна обеспечивать разные способы разбиения первичной задачи – по «сетке» области данных или с варьированием одного или нескольких входных параметров и обеспечивать работу с не менее чем 10^4 заданиями (в перспективе до 10^7). Данная технология в дальнейшем может быть также использована для проведения расчетов на суперкомпьютерных установках пета- и экзафлопсного масштаба. Идеология системы и пилотный вариант системы были ранее апробированы для промежуточного ПО gLite и Unicore [4] для числа заданий до 10^4, в настоящее время система адаптируется для работы в средах, основанных на использовании Globus Tools 4 и 5.

Все указанные технологии могут быть легко использованы для значительного масштабирования задач вычислительной химии на распределенных полигонах, и, в определенной степени, - в условиях суперкомпьютерных установок нового поколения.

3. Основные проблемы, стоящие перед работой проблемно-ориентированных грид-сервисов и возможные способы их решения

Несмотря на весьма успешное построение нескольких грид-полигонов как чисто российских, так и в качестве сегментов международных проектов, востребованность пользователями из областей науки, промышленности, бизнеса доступных грид-ресурсов, на наш взгляд, остается весьма низкой из-за ряда как объективных, так и субъективных проблем. Опыт работы авторов показал наличие достаточно масштабных проблем, возникших в процессе создания и эксплуатации грид-полигонов в интересах конечных пользователей из сфер науки, производства, бизнеса. Нижеперечисленные проблемы во многом ограничивают процессы роста и консолидации вычислительных ресурсов в рамках грид-полигонов, сокращают круг доступных вычислительных ресурсов и пользователей, желающих и могущих ими воспользоваться Охарактеризуем их кратко:

1. На уровне государства:

а) отсутствие заинтересованности со стороны науки и производства, включая общую низкую культуру проведения научных и технологических разработок в большинстве НИИ и промышленных организаций, а также неумение (и нежелание) исследователями и инженерами использовать результаты математического моделирования и высокопроизводительных расчетов. Неготовность большинства исследователей к постановке и решению масштабных задач, требующих ресурсоемких вычислений;

б) отсутствие в большинстве отраслей стратегических заказчиков в лице государства и бизнеса, особенно в рамках долгосрочных исследовательских и конструкторских программ;

в) острая нехватка квалифицированных специалистов как со стороны вычислителей (системщики, программисты), так и со стороны пользователей (химики, технологи);

г) практическое отсутствие отечественных прикладных пакетов и малая доступность зарубежных коммерческих высокоэффективных ППП --> высокая стоимость ПО, лицензионные ограничения, сокращенная область использования, трудности освоения;

д) высокие экономические затраты на создание и эксплуатацию вычислительных ресурсов и инфраструктуры, при этом – отсутствие устойчивой финансовой поддержки российских суперкомпьютеров и грид-полигонов после окончания проектов (обычно успешных) по их созданию и вводу в эксплуатацию;

е) неурегулированность правовых и денежных отношений между собственниками вычислительных сервисов (как грид-ресурсов, так и суперкомпьютеров) и потребителями.

2. На уровне собственно вычислительной химии:

а) высокие требования к аппаратно-программному оснащению расчетных узлов суперкомпьютеров и ресурсных грид-сайтов: RAM > 4 Gb на ядро, локальный дисковый массив от 1 Tb, наличие специализированных библиотек, организация доступа к внешним хранилищам данных и т.п., что зачастую не реализуется в составе суперкомпьютерных центров;

б) как правило, персонал непрофильных суперкомпьютерных центров и грид-ресурсов некомпетентен в проблемных областях и мало заинтересован в области организации квантово-химических и молекулярно-динамических расчетов;

в) отсутствуют стабильные, не зависящие от «капризов» разработчиков, площадки в рамках существующих грид-полигонов для отладки прикладного ПО и новых технологий вычислений;

г) только устойчиво и долговременно работающие виртуальные организации в области прикладных вычислений могут привлечь пользователей, «погрязших» в выполнении краткосрочных грантов.

Каковы же возможные пути решения возникших проблем и возможные политики государства в рамках развития суперкомпьютерных и грид-ориентированных вычислительных ресурсов:

1. создание государством (или долевое участие) вычислительной инфраструктуры (суперкомпьютеры, кластеры средних масштабов, высокоскоростные каналы связи и т.п.), последующая компенсация государством на долговременной основе части эксплуатационных расходов суперкомпьютерных центров и ресурсных грид-центров;
2. налоговые льготы на разработку и приобретение отечественного прикладного ПО, создание ориентированных на это инновационных парков;
3. создание в рамках проектов достаточного количества дружелюбных, с большим количеством базовых шаблонов, WWW-ориентированных интерфейсов ППП к вычислительным суперкомпьютерным и грид-сервисам;
4. стимулирование пользователя (промышленного в первую очередь) на использование суперкомпьютерных и грид-сервисов, например, отсутствие или минимальная оплата пользования государственными вычислительными ресурсами на первых стадиях инновационных разработок (при компенсации расходов на их эксплуатацию из бюджета);
5. введение государством в качестве обязательного условия сертификации применения средств математического моделирования и тестирования для разработки ряда товарных продуктов и наукоемкой продукции.
6. в области государственного образования: (а) расширение количества бюджетных мест в вузах для специальностей «системное администрирование/программирование», «прикладное программирование» и т.п.; (б) использование в процессе обучения широкого круга студентов и аспирантов вузов для разработки необходимого системного и прикладного ПО; (в) обучение пользователей – как очное в вузах и на курсах повышения квалификации, так и в режиме интерактивного on-line: создание информационных web-ресурсов по обучению работе с суперкомьютерами и грид-ресурсами;
7. введение в планы обучения прикладных специалистов (инженеров, химиков и т.п.) дополнительных курсов обучения работы с прикладными пакетами ПО и проведение студентами обязательных расчетов реальных научно-технических и инженерных задач с использованием суперкомпьютеров и грид-структур;
8. совершенствование нормативно-правового регулирования в области стратегических информационных технологий, в первую очередь – упорядочение денежно-правовых отношений между собственниками и потребителями вычислительных ресурсов..

Хотя бы частичное решение поставленных проблем позволит во многом решить проблемы существующих грид-полигонов и повысить эффективность их использования в проблемно-ориентированных ресурсоемких вычислениях, в том числе в области вычислительной химии.

3. Заключение

Всемерное развитие проблемно-ориентированных грид-сервисов (в том числе в области вычислительной химии, молекулярного моделирования и дизайна и большинства смежных отраслей науки), применение новых технологий применительно к грид-вычислениям позволяет существенно повысить эффективность научных исследований в данных отраслях и решать ранее недоступные из-за вычислительных проблем ресурсоемкие задачи. На это ориентировано как создание на базе грид-ресурсов проблемно-ориентированных ВО и специализированных грид-центров, так и вовлечение максимального количества исследователей в круг заинтересованных в супервычислениях. Однако, проблемы, стоящие перед продвижением грид-технологий в повседневную практику российских исследователей, во многом тормозят как собственно научно-исследовательские работы, так и развитие самих грид-полигонов в российской Федерации ввиду низкой заинтересованности научных исследователей и технических разработчиков. Авторами предложены некоторые пути возможных решений данных проблем.

Литература

1. Волохов В.М., Варламов Д.А., Волохов А.В., Пивушков А.В., Покатович Г.А., Сурков Н.Ф. Квантово-химические прикладные пакеты на Российских грид-полигонах // «Параллельные вычислительные технологии ПАВТ-2012»: труды международной научной конференции, (Новосибирск, 26-30 марта 2012 г.), Челябинск: Изд-во ЮУрГУ, 2012. с.394–399
2. Волохов В.М., Варламов Д.А., Пивушков А.В., Волохов А.В. Применение GRID технологий в области вычислительной химии // «Известия Академии наук. Серия химическая», 2011, № 7, с.1483-1490
3. Волохов А.В., Волохов В.М., Варламов Д.А., Пивушков А.В., Покатович Г.А., Прохоров А.И. Использование гибридных вычислительных узлов на базе GPU TESLA C2075 при проведении расчетов в области вычислительной химии и молекулярной динамики // ПАВТ-2013 (настоящий сборник трудов)
4. 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

GPU версия CFD пакета SigmaFlow: портирование и оптимизация с использованием инструментария TTG Apptimizer

А.А. Гаврилов[1], М.А. Кривов[2], С.А. Гризан[2], А.А. Дектерёв[1]

Институт теплофизики СО РАН[1], ООО «ТТГ Лабс»[2]

В данной статье представлены результаты работ по переносу на GPU программы для пространственных тепло-гидродинамических расчетов SigmaFlow. Рассмотрены возникшие проблемы портирования и предложенные варианты их решения, в том числе описаны произведённые модификации форматов данных и самих алгоритмов, подходы к минимизации пересылок данных между вычислителями и узлами кластера. Отдельно приведены результаты оптимизации разработанных версий алгоритмов с целью повышения степени использования блоков GPU. Продемонстрировано итоговое ускорение от переноса вычислений на GPU порядка 7.4 - 11.3 раз относительно последовательной реализации и более 3 раз относительно параллельной версии, выполняющейся на четырёхядерном центральном процессоре.

1. Введение

В связи с массовым переходом к гетерогенным вычислительным системам, которые оснащены вспомогательными ускорителями, крайне актуальной становится задача адаптации существующих пакетов программ к новым подобным архитектурам. В настоящий момент данные гетерогенные системы в основном представлены кластерами, узлы которых содержат от двух до трёх графических ускорителей серии NVidia Tesla. Однако на смену (или же в дополнение) к ним приходят кластера, оснащённые сопроцессорами Intel Xeon Phi, а также начинают появляться прототипы суперкомпьютеров на базе гибридных процессоров, содержащих вычислительные ядра нескольких типов, каждый из которых лучше подходит для решения соответствующего класса задач.

Какой бы из трёх перечисленных выше типов гетерогенных вычислительных систем не стал доминирующим, проблема переноса большинства существующих программ на новые архитектуры по-прежнему остаётся актуальной. Во-первых, во всех концепциях гетерогенных вычислительных архитектур предполагается многократное повышение степени параллелизма. К примеру, если раньше количество независимых потоков вычислений, необходимых (но не достаточных!) для эффективной загрузки всех блоков центрального процессора равнялось количеству ядер, помноженному на количество выполняемых за такт одним ядром инструкций, и измерялось несколькими десятками, то для обеспечения аналогичной загрузки лишь одного графического ускорителя требуется не менее 512 потоков вычислений. Понятно, что далеко не все программные реализации и, тем более, алгоритмы обладают необходимым потенциальным параллелизмом, и поэтому при осуществлении их портирования на гетерогенные системы редко удаётся обойтись лишь «косметическими» изменениями.

Во-вторых, с появлением гетерогенных архитектур неизменно становится актуальной проблема разнородности памяти, причём сразу для всех уровней иерархии. Каждый ускоритель обладает своей оперативной памятью и своими кэшами, прямая адресация которых в общем случае невозможна ни для центрального процессора, ни других ускорителей. Поэтому достичь производительности, составляющей хотя бы десятки процентов от пиковой, удаётся лишь для тех задач, алгоритмы решения которых кроме необходимого уровня параллелизма также обладают свойством локальности данных. А точнее, позволяют «нарезать» исходные данные на необходимое количество независимых блоков, время обработки которых будет много больше времени их копирования. В противном же случае программисту часто приходится явно или неявно осуществлять барьерную синхронизацию, в результате чего вычислители большую часть времени простаивают, а достигнутая производительность начинает составлять лишь доли

процентов от пиковой.

Для рассматриваемой предметной области, а именно задач моделирования различных аэрогидродинамических процессов, обе вышеприведённые проблемы являются достаточно актуальными. Однако если необходимый потенциальный параллелизм обычно обеспечивается за счёт постоянного увеличения размера сеток, дискретизирующих расчётную область, то при попытке разбиения данных на необходимое количество блоков возникает ряд трудностей.

Практически все промышленные расчёты производятся на неструктурированных, а иногда даже и адаптивных сетках, поэтому возможности по их «разрезанию» непосредственно в процессе расчётов на необходимое количество блоков крайне ограничены. И если в случае обычных однородных кластеров подобное разбиение области на домены удаётся выполнить за счёт сторонних внешних программ типа MeTiS, то для осуществления аналогичных действий для гетерогенных систем данный подход оказывается малоприменимым — количество необходимых блоков увеличивается на порядок (в идеальном случае, требуется один блок на один мультипроцессор графического ускорителя, или же порядка 35-50 блоков в пересчёте на узел), а на их размеры накладывается ряд достаточно жёстких ограничений. И в результате процесс разбиение всей области на требуемые блоки с целью обеспечения эффективной загрузки всех вычислителей может оказаться более долгим, чем непосредственно сами расчёты. И это даже без учёта необходимости пересылок значений искомых величин на границах блоков, что является отдельной проблемой.

В частности, из-за озвученных проблем GPU версии большинства CFD-пакетов демонстрируют относительно небольшое ускорение, особенно по сравнению с пакетами из других предметных областей типа молекулярной динамики, где выигрыш от перехода на гетерогенные системы может доходить до нескольких сотен раз. Одним из ярких примеров может служить открытый программный комплекс OpenFOAM для проведения моделирования различных гидродинамических процессов, часть решателей которого в рамках проекта SpeedIT была адаптирована для работы на GPU. В соответствии с опубликованными результатами [1] тестирования двух предобуславливателей, полученное ускорение составило от 2.4 до 3.8 раз по сравнению с последовательной версией пакета. Если же учесть, что пиковые производительности ускорителя и одного ядра тестовой системы равнялись 450 и 12 гигафлопсам соответственно (т. е. соотносились как 37.5 к 1), то достигнутое ускорение демонстрирует явно неэффективное использование вычислительных блоков GPU. В частности, опять же из-за озвученных выше проблем.

Если же рассматривать закрытые платные CFD-пакеты, то ситуация оказывается примерно аналогичной. В случае пакета ANSYS конкретное ускорение от переноса вычислений на GPU сильно зависит как от решателя, так и от размера и формата данных. Если ориентироваться на опубликованные результаты от самой компании ANSYS [2], то ускорение составило примерно от 2 до 9 раз относительно последовательной версии при рассмотрении только решателей СЛАУ. В случае оценки ускорения для всего пакета оно становится более скромным — примерно от 1.3 до 4 раз относительно последовательной версии. В другой статье [3], выполненной уже независимыми авторами, проводилось тестирование GPU версии пакета ANSYS при решении собственной задачи, результаты которого оказались ещё менее радужными — ускорение от добавления GPU составило порядка 1.2 — 1.7 раз.

В качестве ещё одного примера можно привести опыт российской компании ООО «Тесис», разрабатывающей собственный CFD-пакет FlowVision. В статье от разработчиков данного решения [4], посвящённой реализации GPU-версии одного алгоритма для решения СЛАУ, описывается поход, позволивший получить ускорение для отдельных этапов более чем в 30 раз относительно одного ядра центрального процессора. К сожалению, в статье не приведены оценки для итогового ускорения, поэтому данные результаты сложно сравнивать с предыдущими.

В следующих разделах данной работы будут приведены результаты частичного портирования на GPU CFD пакета SigmaFlow, разрабатываемого институтом теплофизики СО РАН, которое было осуществлено совместно с компанией ООО «ТТГ Лабс». Предварительные результаты данных работ по портированию отдельных решателей СЛАУ были приведены в статье [5], в соответствии с которыми авторами было достигнуто 8-12 кратное ускорение относительного одного ядра центрального процессора.

2. Описание пакета SigmaFlow

Стоит сразу отметить, что в оптимизируемом пакете SigmaFlow реализованы решатели для различных видов уравнений и систем уравнений, однако GPU версия была разработана лишь для двух модулей, отвечающих за решение эллиптических уравнений и системы уравнений Навье-Стокса, описывающей движение ламинарных безвихревых ньютоновских жидкостей. Однако благодаря модульной структуре пакета, даже к текущей GPU версии могут быть подключены некоторые RANS-модели турбулентности — в определённые моменты (а точнее, после каждой итерации) все необходимые данные будут автоматически перемещены в память центрального процессора, обработаны, а после чего загружены обратно в память GPU. В случае же необходимости, добавление GPU поддержки для соответствующих моделей турбулентности может быть легко осуществлено в рамках дальнейших работ.

Все расчётные области задаются с помощью трёхмерных неструктурированных сеток, в случае GPU версии на которые накладывается дополнительное, но опциональное ограничение — каждый узел сетки имеет не более 6 соседей. Для хранения дискретных величин, определённых на вершинах и дугах построенного таким образом графа, независимо используются два внутренних формата данных. В обоих случаях все вершины и дуги перенумеровываются, после чего дискретизируемые на них величины сохраняется в массивы под соответствующими номерами. Далее, в соответствии с первым форматом данных, вводится вспомогательный массив для описания структуры графа, содержащий для каждой дуги индексы двух соединяемых ею вершин. Таким образом, для получения значения величины, заданной на дуге графа, достаточно найти её индекс в данном вспомогательном массиве, после чего с его помощью адресовать массив, содержащий данную величину. В соответствии со вторым форматом вводятся вспомогательные массивы индексов вершин-соседей, позволяющие для заданного узла сетки найти все другие узлы, с ним соединённые, а также индексы соединяющих дуг.

Подробное описание реализации решателя для уравнений Навье-Стокса, на примере которого далее будет описан процесс портирования, может быть найдено в статье [6], однако для краткого описания схемы решения следует отметить ряд моментов. Так, для аппроксимации конвективных членов уравнений переноса применяется противопоточная TVD схема второго порядка точности. Связь между полями скорости и давления, обеспечивающая выполнение уравнения неразрывности, реализуется при помощи SIMPLE-C процедуры на совмещенных сетках. Для устранения осцилляций поля давления используется подход Рхи-Чоу, заключающийся в специальной интерполяции вектора скорости на грани контрольных объемов. Полученные в результате дискретизации исходной системы дифференциальных уравнений разностные уравнения решаются итерационным способом с применением предобусловленного метода сопряженных невязок.

В исходной версии пакета была реализована возможность запуска расчётов на кластерных системах в соответствии с моделью «один MPI процесс на одно физическое ядро CPU», однако в рамках данных работ по разработке GPU версии осуществляется частичный переход к модели «один MPI процесс на один физический узел». При этом в качестве реализации MPI используется библиотека MPICH2, а для распараллеливания вычислений по узлам кластера производится разбиение всей расчётной области на домены с применением программы MeTiS.

3. Портирование пакета на GPU

3.1. Подготовительные работы

Так как оптимизируемый пакет содержит достаточно много модулей, суммарный размер которых составляет сотни тысяч строк кода, то перед началом непосредственного его переноса на GPU был выполнен ряд подготовительных работ. Во-первых, была встроена разработанная компанией ООО «ТТГ Лабс» небольшая вспомогательная утилита ttgUtils, опционально делающая замеры времени работы отдельных частей пакета и по завершению работы подготавливающая мини-отчёт, и также опционально сохраняющая на диск промежуточные

данные с целью дальнейшей проверки корректности вычислений между различными реализациями одного и того же алгоритма. Таким образом, в зависимости от опций компиляции, получается три сборки пакета — первая по завершению работы подготавливает мини-отчёт, сколько времени ушло на каждый этап вычислений, вторая проверяет, как сильно отличаются на каждой итерации промежуточные данные в CPU и GPU версиях (для чего используется L2 и C нормы), а третья предназначена для проведения расчётов в обычном режиме.

Использование данной утилиты, а не специализированных инструментов типа профайлера Intel Parallel Studio, было обусловлено необходимостью автоматизации тестирования создаваемой GPU версии пакета. К примеру, благодаря встроенной системе замеров времени и небольшого вспомогательного скрипта удалось полностью автоматизировать процесс тестирования сразу на наборе тестовых сеток разного размера, результаты которого группировались в отдельный файл с отчётом. Аналогично, с помощью другого скрипта была реализована функциональность сравнения результатов работы оптимизированной версии с эталонной, причём сравнивались именно промежуточные результаты (точнее, нормы массивов) на выбранных итерациях, а само тестирование, опять же, проводилось сразу на наборе сеток. Так как полный набор тестов занимал порядка нескольких часов, то данная схема в дальнейшем существенно упростила процесс портирования.

Во-вторых, опять же перед началом непосредственных работ по портированию, были изменены контейнеры, используемые для хранения массивов. Весь пакет SigmaFlow разработан с использованием языка C++ и принципов ООП, поэтому все контейнеры были реализованы с помощью шаблонных классов. Так как для работы GPU версии необходимо иметь возможность загружать нужные данные в память графического ускорителя, то было рассмотрено два сценария — модификация исходных контейнеров с целью авто-копирования хранящихся в них данных на GPU и «ручное» копирование массивов в память графического ускорителя непосредственно перед вызовом соответствующих CUDA-ядер. От второго варианта пришлось сразу же отказаться, так как в пакете не было единственного «узкого места», а сами вычисления были равномерно «размазаны» по многим модулям, между которыми вызывались легковесные CPU методы для корректировки данных. Как следствие, данный подход потребовал бы постоянного копирования массивов между CPU и GPU, что вместо ускорения привело бы лишь к замедлению вычислений.

Таким образом, было решено внести изменения в исходные контейнеры, позволившие не только адресовать хранящийся в них массив, но и получать ссылку на актуальную копию данных в памяти GPU. При этом в самом контейнере автоматически выделялся буфер в памяти GPU, запоминалось текущее местоположение данных и при соответствующем вызове незаметно осуществлялось копирование. К сожалению, данная схема, которая является вполне логичной и естественной при разработке программы «с нуля», оказалась крайне неудачной для внедрения в существующий пакет подобного размера, большую часть которого не планировалось модифицировать. Ниже перечислены основные трудности, с которыми пришлось столкнуться при встраивании данной схемы и дальнейшем портировании:

– Переход от контейнеров к указателям и обратно. В некоторых более старых частях пакета (например, отвечающих за подготовку данных и вызываемых лишь один раз при старте программы) работа с массивами осуществлялась через указатели, в результате чего контейнеры теряли информацию о местоположении актуальной версии данных. Как следствие, обновлённые массивы иногда затирались более старыми версиями, сохранёнными в памяти GPU.

– Работа с подмассивами. В одном из решателей обнаружилась не совсем стандартная схема представления данных, в соответствии с которой выделялся блок памяти размера 2*N, который далее «оборачивался» в два независимых контейнера размерностью N, что позволяло из первого массива адресовать элементы второго. Изначально такая возможность предусмотрена не была, в результате чего соответствующие им участки GPU памяти не были размещены последовательно.

– Создание контейнеров, ссылающихся на общий участок памяти. В отличие от предыдущего сценария, здесь используется общий блок памяти, который адресуется из-под разных контейнеров. Соответственно, в начальной версии для каждого из них создавался собственный GPU буфер, что в дальнейшем приводило к ошибкам.

В связи с большим размером оптимизируемого пакета подобные ошибки не удалось обнаружить на этапе проектирования, а что более страшно — они приводили не к аварийной остановке программы, а лишь к более высокой погрешности. В результате их поиск оказался крайне трудоёмким и был успешно осуществлён лишь благодаря подсистеме сравнения промежуточных результатов между CPU и GPU версиям.

3.2. Работы по портированию

В рамках работ по портированию пакета на GPU был осуществлён переход к ядро-ориентированной архитектуре. Соответственно, для каждой функции, в которой осуществляются непосредственные вычисления, было создано одно или несколько тривиальных вычислительных ядер и функция-обёртка, выполняющая базовые проверки, получение указателей на массивы и прочую подготовку, после чего вызывающая соответствующие вычислительные ядра. Причём переключение между реализациями ядер (в данном случае CUDA или исходный CPU код) было унифицировано и осуществлялось при компиляции с помощью макроса. Благодаря подобной структуре (вычислительный код полностью отделён от управляющего) удаётся минимизировать количество дублируемых участков пакета, а также появляется возможность в дальнейшем без дополнительных затрат добавить поддержку технологии OpenCL (и, как следствие, ускорителей от компании AMD), или же альтернативных версий ядер для CPU, написанных с использованием расширений SSE/AVX. Более того, в рамках текущей реализации для некоторых исходных функций были разработаны различные версии CUDA-ядер, отдельно оптимизированные под ускорители с разной архитектурой (в данном случае — CC 1.3 и CC 2.0). Стоит также отметить, что в результате данных модификаций было создано порядка 40 тривиальных CUDA-ядер.

В большинстве случаев разработка CUDA-аналогов для исходных CPU функций являлась сугубо технической работой, однако в некоторых из них при отображении на архитектуру графического ускорителя были обнаружены некоторые трудности. Так, в одной функции требовалось одновременно хранить множество значений вспомогательных переменных, в результате чего созданная компилятором NVCC GPU-подпрограмма завершалась с ошибкой из-за нехватки количества регистров. Наиболее простым вариантом решения данной проблемы является размещение части переменных в разделяемой памяти, однако в общем случае её также оказалось недостаточно. В результате пришлось сделать две версии данного ядра, одна из которых запускается с уменьшенным до 32 нитей размером блока (и, как следствие, с большим объёмом разделяемой памяти на одну нить), а вторая использует возможность архитектуры Fermi увеличить размер разделяемой памяти с 16 до 48 килобайт за счёт уменьшения размера кэша L1.

Другим проблемным моментом, потребовавшим внесения существенных изменений в ряд исходных CPU функций, оказалось активное использование операции +=, которая в случае CUDA-версии приводит к гонке данных, когда несколько разных нитей пытаются обновить один и тот же элемент массива, затирая изменения друг друга. Рекомендуемым решением данной проблемы является переход к аппаратно поддерживаемым атомарным операциям, однако в случае, если ускоритель имеет архитектуру CC 1.1, CC 1.2 или CC 1.3, они являются крайне неэффективными, в результате чего «ускоренная» версия пакета может оказаться намного медленнее исходной последовательной. В случае же более новых архитектур CC 2.0 и CC 2.1 атомарные операции хоть и являются более быстрыми, их использование всё равно на порядок замедляет соответствующее CUDA-ядро. Для решения данной проблемы потребовалось перейти к другому формату данных, благодаря которому удалось полностью избежать проблемной операции +=, и даже на архитектуре поколения Fermi получить 3-кратное ускорение для данного ядра.

Ещё одной проблемой, которая возникла при портировании двух ядер, используемых в предобуславливателе D-ILU, оказался специфический доступ к памяти, не позволяющий распараллелить исходную функцию. Фактически, для обработки i-того элемента массива требовалось обработать все элементы с индексами от 0 до i-1, что легко осуществимо в последовательной версии, но никак не реализуемо в рамках SIMT модели, используемой в программах для графических ускорителей. В результате работ был разработан аналог данного

предобуславливателя, который позволяет достичь схожих результатов, но не прямым, а итерационным путём, и для которого имеется возможность эффективного отображения на архитектуру графического ускорителя. В результате в GPU версии пакета SigmaFlow осуществляется несколько типов итераций — внешние для решения исходного СЛАУ и внутренние для применения предобуславливателя, выполняемые в рамках каждой внешней итерации. Соответственно, возникает вопрос — сколько требуется внутренних итераций, и как их количество влияет на скорость решения исходного СЛАУ? Частично ответ на данный вопрос был дан в рамках предыдущей статьи [5], в соответствии с которой достаточно трёх внутренних итераций. Однако в результате тестирования на дополнительных сетках выяснилось, что в ряде случаев этого недостаточно для обеспечения аналогичной скорости сходимости, поэтому в текущей версии их количество было повышено с 3 до 7-15 в зависимости от модификации исходного предобуславливателя. Как следствие, в разы возрос объём вычислений, однако при этом промежуточные данные на внешних итерациях в CPU и GPU версии пакета отличаются не более чем на 0.5% по норме C.

Резюмируя, стоит отдельно ещё раз отметить, что благодаря всем перечисленным модификациям удалось сохранить полную совместимость со всеми модулями пакета на уровне структур данных и сигнатур функций, при этом избежав необходимости копирования данных между CPU и GPU в основном цикле решения уравнения, время выполнения которого составляет более 99% от времени работы всей программы.

4. Оптимизация созданной GPU версии пакета

После завершения базового портирования пакета SigmaFlow на GPU итоговое ускорение на системе с центральным процессором Intel Xeon E3-1230 и ускорителем NVidia GeForce 580GTX при вычислениях на сетке из 512 тыс. узлов составило порядка 5 раз относительно исходной версии, выполняемой на одном ядре. В абсолютных значениях время работы соответствовало примерно 4200 и 840 секундам. Так как для решателей СЛАУ, являющихся в CPU версии наиболее вычислительноёмким этапом, было достигнуто 12-кратное ускорение, то было решено проводить дополнительную оптимизацию, основные направления которой рассмотрены ниже.

При более детальном анализе программы с помощью профайлера NVidia Nsight было установлено, что время работы CUDA-ядер составляет порядка 480 секунд, а в течение же остальных 360 секунд программа выполняет различные вспомогательные операции, никак не связанные с непосредственным вычислениями. Одним из таких мест оказалось выделение и освобождение памяти для временных GPU буферов, выполняемое на каждой итерации. Так как, фактически, осуществлялось 20 000 итераций, то суммарно на вызовы функции cudaMalloc() тратилось 160 секунд. В пересчёте же на одну итерацию затрачиваемое время получалось равным всего 8 микросекундам, в связи с чем при портировании данная операция казалась крайне незначительной. После введения механизма переиспользования всех выделяемых буферов от этих 160 секунд удалось полностью избавиться.

Другой достаточно «долгой» операцией было обнуление данных контейнеров, на которое тратилось порядка 80 секунд. Так как после внесённых изменений контейнеры стали содержать две копии данных (в памяти CPU и GPU), то их обнуление проводилось сразу во всех буферах. Как следствие, затраты на относительно меленный библиотечный вызов memset(), результаты которого фактически игнорировались, и составили те самые 80 секунд. Другой библиотечной операцией, внёсший свой существенный вклад в общее время работы программы, оказался текстовый вывод в консоль, осуществляемый с помощью оператора перенаправления потока языка C++. На весь вывод тратилось порядка 30 секунд, что составляло менее процента в исходной CPU версии пакета, но уже порядка 5% в оптимизированной. После замены вывода через потоки на вызов printf() и отказа от операции flush() после каждой записи затрачиваемое время удалось сократить в 3 раза (примерно до 10 секунд).

Наконец, последними функциями, непосредственно не связанными с вычислениями, но суммарное время выполнения которых было достаточно большим, оказались реализации скалярного произведения и редукции. В начальном варианте в них производилось несколько шагов редукции на графическом ускорители, после чего существенно уменьшенный массив копировался в память центрального процессора и окончательно редуцировался до числа.

Собственно, данный процесс копирования даже уменьшенного в несколько сотен раз массива суммарно занимал порядка 70-80 секунд. Поэтому потребовалось более точно определять необходимое число шагов редукции на GPU, в результате чего в обновлённой версии на центральный процессор копировался уже массив не более чем из 256 элементов, время на обработку которого стало действительно незначительным.

Следующим этапом была оптимизация непосредственно CUDA-ядер, которая свелась к трём действиям. Во-первых, в достаточно редко вызываемом ядре для вычисления градиента используемые атомарные операции были заменены на индексацию через вспомогательные массивы, что позволило сэкономить порядка 30 секунд. Далее, в ядре, отвечающем за проведение внутренних итераций при применении предобуславливателя, часть косвенно адресуемых коэффициентов была заранее сохранена во вспомогательный буфер, благодаря чему удалось отказаться от одного обращения с невыравненным паттерном доступа в память графического ускорителя и тем самым уменьшить время расчётов ещё на 20 секунд.

Таким образом, благодаря проведённым модификациям время работы GPU версии пакета на тестовой сетке сократилось с 840 до 440 секунд. Для проведения дальнейшей оптимизации были начаты работы по встраиванию разработанного компанией ООО «ТТГ Лабс» инструментария TTG Apptimizer, позволяющего динамически подстраивать параметры вычислительных ядер под связку «целевая система + обрабатываемые данные». В случае пакета SigmaFlow данными параметрами являлись размер блока нитей, которые будут выполняться на одном мультипроцессоре, а также, при возможности, количество узлов сетки, обрабатываемых одной нитью.

В связи с достаточно сложной архитектурой графических ускорителей задача обеспечения максимально эффективной загрузки все вычислительных блоков, в данном случае сводящаяся к подбору двух перечисленных параметров, является крайне нетривиальной и требующей учёта множество факторов, таких как размер входных данных, необходимые CUDA-ядру ресурсы, а также особенности модели ускорителя. С помощью же инструментария TTG Apptimizer данный процесс удалось полностью автоматизировать — на первых 100 итерациях данный инструмент для каждого оптимизируемого CUDA-ядра производит обучение, что выражается во временном понижении производительности, после чего подбирает параметры, оптимальные для данного конкретного случая, тем самым повышая итоговую скорость расчётов.

5. Оценка полученного ускорения

В качестве тестовой задачи, на которой производилась оценка достигнутого ускорения, была выбрана традиционная задача о стационарном ламинарном течении в пространственной каверне с подвижной верхней крышкой. Для расчетов использовалась неравномерная структурированная сетка, представляющая собой куб, количество узлов которого зависело от типа теста. При этом координатные линии были сгущены к поверхностях, а число Рейнольдса, посчитанное по скорости движения крышки и линейному размеру каверны, составляло 1000. Характерная картина течения в каверне приведена на рис.1.

Для оценки ускорения все тесты были разделены на две независимые группы. В первой производилось сравнение GPU версии пакета относительно MPI версии, запускаемой на одном ядре, на всех ядрах одного процессора и на кластере при выполнении расчётов на сетке из 4 миллионов узлов. Вторая группа содержит сравнение достигнутого ускорения относительно одного ядра центрального процессора, но на сетках различного размера.

Первая группа тестов выполнялась на обычном персональном компьютере, оснащённом графическим укорителем NVidia GeForce 580 GTX (пиковая производительность которого составляет порядка 1.5 терафлопс) и четырёхядерным центральным процессором Intel Core i7 2600k, частота которого была повышена до 4.4 гигагерц (в результате чего его пиковая производительность равнялась 140 гигафлопсам). В качестве кластера использовалась система из 12 узлов IBM Blade HS21, каждый узел которой включает в себя два 4-х ядерных процессора Intel Xeon, работающих на частоте 2.33 ГГц (пиковая производительность составляла 895 гигафлопс).

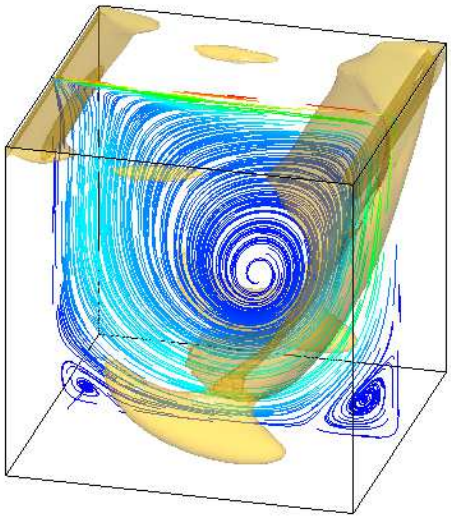


Рис. 1. Траектории частиц жидкости и изоповерхность Q–критерия для течения в каверне.

Результаты тестирования описанной ранее задачи на всех целевых система приведены в следующей диаграмме, демонстрирующей ускорение относительно одного ядра процессора Intel Core i7.

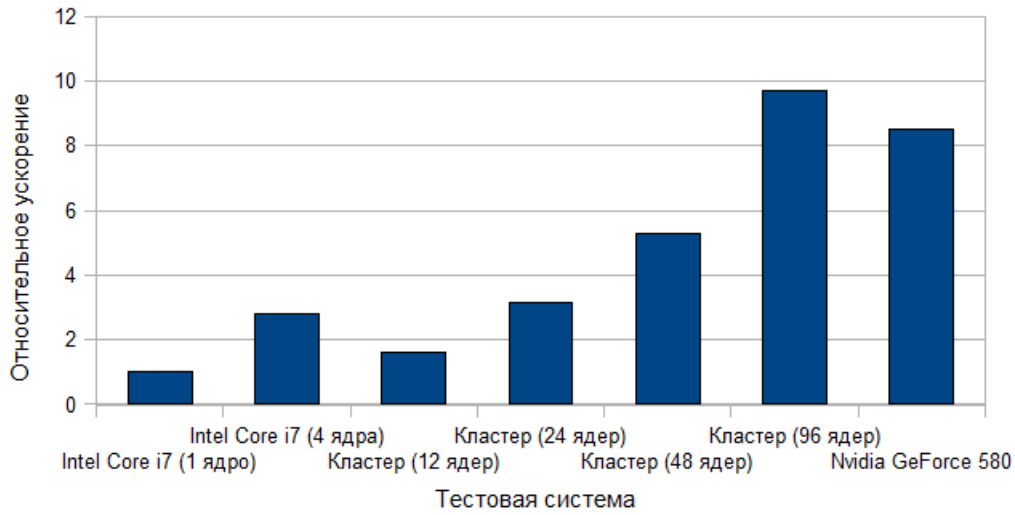


Рис. 2. Ускорение относительно одного ядра процессора Intel Core i7 на сетке из 4 млн. узлов.

Полученные результаты оказались достаточно неоднозначными. С одной стороны, GPU версия пакета обеспечивает примерно такое же ускорение, как и кластер из 12 узлов (8.5 раз против 9.7). Однако если взять всего один современный центральный процессор с повышенной тактовой частотой и задействовать все его четыре ядра, то по сравнению с ним выигрыш от переноса вычислений на GPU становится более скромным и уже составляет ровно 3 раза. Таким образом, сравнение полученных ускорений с известными результатами портирования на графические ускорители CFD пакетов оказывается достаточно проблематичным, так как в зависимости от выбора системы заявляемое ускорение может изменяться в разы.

Для проведения тестов из второй группы использовался сервер с одним четырёхядерным центральным процессором Intel Xeon E3-1230 (порядка 100 пиковых гигафлопс) и, опять же, графическим ускорителем NVidia GeForce 580 GTX (обладающего, напомним, пиковой производительностью в 1.5 терафлопс). Приведённые ниже результаты первого теста из данной группы демонстрируют относительную производительность GPU версии пакета в сравнении с исходной версией, запущенной на одном ядре центрального процессора.

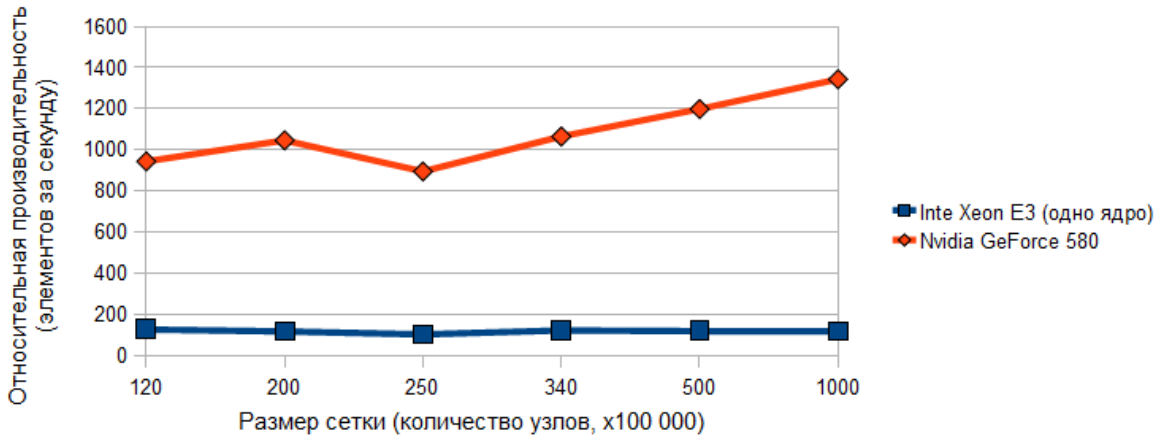


Рис. 3. Производительность версий пакета в зависимости от размера сеток.

Согласно данным результатам, с ростом размера сетки производительность GPU также повышается, что можно объяснить большим количеством вычислений, благодаря которым уменьшается влияние достаточно медленных операций доступа к глобальной памяти. При этом ускорение относительно одного ядра центрального процессора в зависимости от размера сетки составило от 7.4 до 11.3 раз.

Другим достаточно интересным тестом было сравнение вклада в суммарное время вычислений различных этапов расчётов в зависимости от размера сетки. В данном случае для GPU версии пакета сравнивались суммарные времена трёх относительно независимых операций — (1) вычисления скалярного произведения, (2) вычисления градиента и (3) применения предобуславливателя. Результаты данного теста приведены на следующем графике.

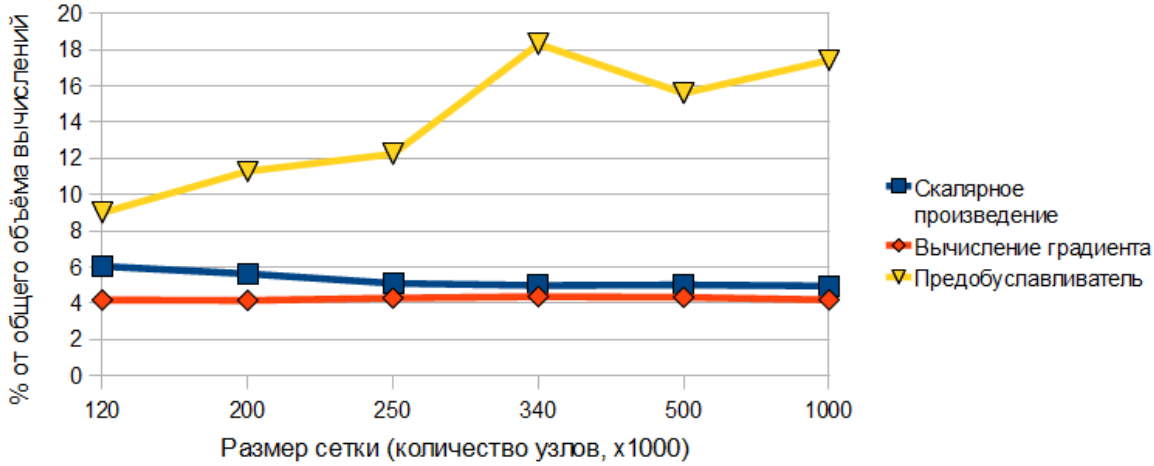


Рис. 4. Вклад некоторых этапов в общее время вычислений в зависимости от размера сеток.

Как оказалось, доля времени выполнения некоторых операций типа вычисления градиента практически инвариантна и не зависит от размера сеток. С другой стороны, более сложные ядра типа предобуславливателя D-ILU относительно быстрее выполняются на небольших сетках.

Наконец, последним тестом была оценка разности промежуточных результатов из CPU и GPU версий пакетов в зависимости от номера итерации при решении СЛАУ. Так как из-за ряда особенностей, а именно таких как накопление погрешности при выполнении атомарных операций, изменение порядка суммирования при редукции массивов, различия в алгоритмах вычисления ряда функций тип синуса, результаты вычислений на CPU и GPU всегда немного отличаются. В данном же случае для предобуславливателя была разработана альтернативная реализация, что ещё больше усилило данные отличия. Результаты, представленные в следующем графике, были получены путём сравнения L2-норм для соответствующих массивов с промежуточными результатами.

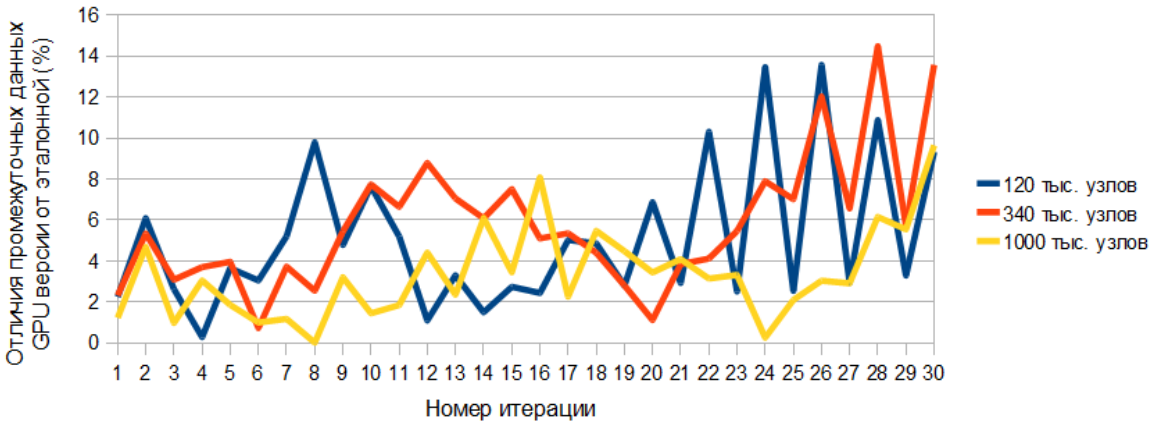


Рис. 5. Отличия между промежуточными результатами в CPU и GPU версиях.

Стоит отметить, что в результате работы CPU и GPU версий пакета были получены практически идентичные решения, поэтому данный график больше иллюстрирует скорость сходимости, которая фактически оказывается также идентичной — на всех итерациях вплоть до конца вычислений данные отличия составляют не более 30%.

6. Заключение

В результате данных работ была создана GPU версия пакета SigmaFlow, предназначенная для моделирования ламинарных течений ньютоновских жидкостей. Были описаны проблемы, с которыми авторы столкнулись при осуществлении портирования, а также рассмотрены подходы для повышения производительности созданной GPU версии пакета. Согласно проведённому тестированию, было зафиксировано ускорение от 7.4 до 11.3 раз в зависимости от размера сетки относительно исходной версии, выполняемой одним ядром центрального процессора.

В рамках дальнейших работ планируется добавление в разработанную версию пакета поддержки систем с несколькими ускорителями, осуществление портирования ряда модулей, реализующих требуемые RANS-модели турбулентности, а также проведение дополнительных работ по оптимизации пакета с целью доведения итогового ускорения до 15 раз.

Литература

1. Acceleration of OpenFOAM with SpeedIT 2.1, Comparision of GAMG and DIC preconditioners URL: http://vratis.com/blog/?p=119 (дата обращения 01.12.2012)
2. Beisheim J., Speed Up Simulations with a GPU // Ansys Advantage, Vol. 4, Issue 2, 2010
3. Газизов Р.К., Касаткин А.А., Юлдашев А.В., Исследование ускорения решения термо-структурной задачи в пакете ANSYS средствами GPU // Труды конференции «Применение гибридных высокопроизводительных вычислительных систем для решения научных и инженерных задач», Нижний Новгород, 2011, с. 38 — 41.
4. Коньшин И.Н., Сушко Г.Б., Харченко С.А., Трёхуровневая MPI+TBB+CUDA параллельная реализация блочного итерационного алгоритма решения СЛАУ для мелкоблочных неструктурированных разреженных матриц // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции, Москва, Издательство МГУ, 2012, с. 522 — 528.
5. Кривов М.А., Гризан С.А., Опыт разработки гибридных версий версий решателей разреженных СЛАУ // Сборник трудов конференции «Параллельные вычислительные технологии 2012», Новосибирск, 2012.
6. Гаврилов А.А., Минаков А.В., Дектерев А.А., Рудяк В.Я. Численный алгоритм для моделирования ламинарных течений в кольцевом канале с эксцентриситетом // СибЖИМ. – 2010. – Т. 13, №4(44). – С. 46–61.

Параллельный алгоритм RKDG метода второго порядка для решения двумерных уравнений идеальной магнитной гидродинамики* †

М.П. Галанин, В.В. Лукин, К.Л. Шаповалов

Институт прикладной математики им. М.В. Келдыша РАН

Рассмотрена параллельная реализация RKDG метода решения двумерных уравнений идеальной магнитной гидродинамики на треугольных неструктурированных сетках. Описан программный комплекс, построенный с применением технологий MPI и OpenMP. Приведены алгоритмы монотонизации решения и бездивергентной реконструкции магнитного поля, позволяющие получать физически адекватные результаты расчетов с высоким порядком точности. Обсуждены результаты тестовых расчетов и полученные значения ускорения вычислений за счет использования параллельных технологий.

1. Введение

Уравнения идеальной магнитной гидродинамики (МГД) описывают множество явлений, представляющих как чисто научный, так и прикладной технический интерес. В частности, в рамках моделей, основанных на системе уравнений МГД, рассматриваются процессы электромагнитного ускорения вещества в различных условиях (плазменные ускорители, астрофизические джеты). Особый интерес представляет моделирование развития неустойчивости течения замагниченной плазмы в окрестности поверхности разрыва (ударной волны), для адекватного описания которого целесообразно использовать численные методы второго и более высоких порядков точности.

Одним из наиболее современных и популярных при решении задач газовой динамики и МГД является RKDG метод [1,2] (Runge — Kutta Discontinuous Galerkin). Этот метод обеспечивает высокий уровень разрешения разрывов и позволяет повышать порядок точности метода без увеличения шаблона аппроксимации, что особенно важно при использовании неструктурированных сеток. В то же время методы высокого порядка являются немонотонными и могут приводить к накоплению численного магнитного заряда, что создает существенные сложности при моделировании замагниченных течений.

В работе рассмотрена модификация разрывного метода Галеркина второго порядка для решения системы уравнений двумерной магнитной гидродинамики на треугольных сетках. Построенный метод включает процедуры монотонизации численного решения, а также позволяет получать решение, удовлетворяющее условию бездивергентности магнитного поля.

Анализ процесса формирования неустойчивости течения замагниченной плазмы требует использования подробных сеток. В то же время применение RKDG метода само по себе является вычислительно затратным и приводит к необходимости использования параллельных вычислительных технологий для систем с общей и распределенной памятью [3, 4]. В работе описана параллельная реализация данного метода. Показана эффективность распараллеливания на тестовых задачах МГД.

*Статья рекомендована к публикации программным комитетом международной научной конференции "Параллельные вычислительные технологии 2013".

†Работа выполнена при частичной финансовой поддержке РФФИ (проекты №№ 12-01-00109, 12-01-31193, 12-02-12096, 12-02-00687), гранта Президента Российской Федерации для государственной поддержки ведущих научных школ Российской Федерации (проект НШ-1434.2012.2).

2. Система нестационарных уравнений идеальной магнитной гидродинамики

Система нестационарных уравнений идеальной магнитной гидродинамики, записанная в консервативных переменных, включает в себя законы сохранения массы, импульса, энергии, а также уравнение Фарадея для магнитной индукции [5]:

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial \vec{F_1}}{\partial x_1} + \frac{\partial \vec{F_2}}{\partial x_2} = 0, \tag{1}$$

где

$$\vec{u} = (\rho, \rho v_1, \rho v_2, \rho v_3, e, B_1, B_2, B_3)^T,$$
$$\vec{F_1} = \left(\rho v_1,\ \rho v_1^2 + p + \frac{\boldsymbol{B}^2}{8\pi} - \frac{B_1^2}{4\pi},\ \rho v_1 v_2 - \frac{B_1 B_2}{4\pi},\ \rho v_1 v_3 - \frac{B_1 B_3}{4\pi},\right.$$
$$\left.\left(e + p + \frac{\boldsymbol{B}^2}{8\pi}\right) v_1 - B_1 \frac{(\boldsymbol{B}\cdot\boldsymbol{v})}{4\pi},\ 0,\ v_1 B_2 - v_2 B_1,\ v_1 B_3 - v_3 B_1\right)^T,$$
$$\vec{F_2} = \left(\rho v_2,\ \rho v_2 v_1 - \frac{B_2 B_1}{4\pi},\ \rho v_2^2 + p + \frac{\boldsymbol{B}^2}{8\pi} - \frac{B_2^2}{4\pi},\ \rho v_2 v_3 - \frac{B_2 B_3}{4\pi},\right.$$
$$\left.\left(e + p + \frac{\boldsymbol{B}^2}{8\pi}\right) v_2 - B_2 \frac{(\boldsymbol{B}\cdot\boldsymbol{v})}{4\pi},\ v_2 B_1 - v_1 B_2,\ 0,\ v_2 B_3 - v_3 B_2\right)^T.$$

Здесь ρ — плотность газа, p — давление, e — полная энергия единицы объема газа, $\boldsymbol{v} = (v_1, v_2, v_3)^T$ — вектор скорости, $\boldsymbol{B} = (B_1, B_2, B_3)^T$ — вектор магнитной индукции. Из закона Фарадея следует условие бездивергентности магнитного поля $\operatorname{div} \boldsymbol{B} = 0$. Будем считать, что плазма является сильно ионизированным, неполяризованным совершенным газом с уравнением состояния $p = (\gamma - 1)\rho\varepsilon$, где γ — показатель адиабаты, ε — удельная внутренняя энергия. При этом $e = \frac{p}{\gamma - 1} + \frac{\rho \mathbf{v}^2}{2} + \frac{\boldsymbol{B}^2}{8\pi}$.

Система (1) с уравнением состояния газа является замкнутой. Можно показать, что эта система является гиперболической [5] и, следовательно, существует разложение

$$\frac{\partial \vec{F_i}}{\partial \vec{u}} = A_i = L_i \Lambda_i R_i,\ i = 1, 2, \tag{2}$$

где L_i и R_i — ортогональные матрицы, составленные из левых и правых собственных векторов, а Λ_i — действительная диагональная матрица, составленная из собственных чисел матрицы A_i.

3. Численный метод

3.1. Разрывный метод Галеркина

Для решения системы (1), или в индексной форме записи (здесь и далее в этом параграфе принято правило суммирования по повторяющимся индексам)

$$\frac{\partial u_i}{\partial t} + \frac{\partial F_{j,i}}{\partial x_j} = 0, \tag{3}$$

на треугольной сетке используем разрывный метод Галеркина [1,2]. Применим стандартную схему методов галеркинского типа — умножим правую и левую часть (1) на произвольную скалярную функцию $\omega(\boldsymbol{x})$, проинтегрируем по объему V_Δ, применим правило интегрирования по частям и получим:

$$\int\limits_{V_\Delta} \frac{\partial u_i}{\partial t} \omega \, dV + \int\limits_{S_\Delta} F_{j,i} n_j \omega \, dS - \int\limits_{V_\Delta} F_{j,i} \frac{\partial \omega}{\partial x_j} \, dV = 0, \tag{4}$$

где $\boldsymbol{n}$ — единичный вектор внешней нормали к границе области V_Δ — поверхности $S_\Delta = \partial V_\Delta$.

Рассмотрим двумерную область D и соответствующую ей трехмерную область $\mathcal{D} = D \times \mathbb{R}$. Введем в области D сетку Ω_h, состоящую из системы N_T треугольников $\{K^i\}_{i=1}^{N_T}$. В качестве области V_Δ рассмотрим единичную треугольную призму с сечением K^l. Тогда второй интеграл в (4) в силу свойств системы (1) можно преобразовать к виду [6]

$$\int\limits_{S_\Delta} n_j F_{j,i}(\dots, u_q, \dots)\omega \, dS = \sum_{k=1}^{3} \int\limits_{S_k^l} T^{-1}_{k,ij} F^*_j(\dots, T_{k,qj} u_j, \dots)\omega \, dS,$$

где S_k^l — поверхность k-ой боковой грани призмы V_Δ, $T_k = T(\boldsymbol{n}_k)$ — матрица, осуществляющая поворот векторов $\boldsymbol{v}$ и $\boldsymbol{B}$ на угол, равный углу между нормалью $\boldsymbol{n}_k$ и положительным направлением оси x_1, а $\vec{F}^*$ — вектор, описывающий поток консервативных переменных через поверхность S_k^l.

Будем искать решение в виде

$$\vec{u} = \sum_{\alpha=1}^{N_T} \sum_{\beta=1}^{N_b} \vec{y}^{\alpha,\beta}(t)\varphi_{\alpha,\beta}(x_1, x_2),$$

где N_b — зависящее от порядка аппроксимации число базисных функций, определенных в ячейке K^α; $\{\varphi_{\alpha,\beta}(x_1, x_2)\}_{\beta=1}^{N_b}$ — ортонормированная система базисных функций (полиномов степени не выше $m-1$), определенных в α-й ячейке; $\vec{y}^{\alpha,\beta}(t)$ — вектор коэффициентов при базисных функциях. В качестве пробных функций ω выберем те же базисные функции $\varphi_{\alpha,\beta}$. Тогда приходим к системе обыкновенных дифференциальных уравнений разрывного метода Галеркина порядка m

$$\frac{dy_i^{\alpha,\beta}}{dt} + \sum_{\psi=1}^{3} \int\limits_{S_\psi^\alpha} F_i^*(\vec{y}^\alpha, \vec{y}^{\eta(\alpha,\psi)})\varphi_{\alpha,\beta} dS + \int\limits_{V^\alpha} F_{j,i} \frac{\partial \varphi_{\alpha,\beta}}{\partial x_j} dV = 0,$$

$$\alpha = \overline{1, N_T}, \ \beta = \overline{1, N_b}, \ i = \overline{1, 8}, \quad (5)$$

где ψ — номер ребра α-й ячейки, $\eta(\alpha, \psi)$ — номер соседней с ней ячейки, $F^*(\vec{y}^\alpha, \vec{y}^{\eta(\alpha,\psi)})$ — численный поток, зависящий от значений решения в ячейках, примыкающих к ребру ψ, и определяемый из решения соответствующей задачи Римана о распаде разрыва [5].

Система (5) дополняется начальным условием вида

$$\vec{y}^{\alpha,\beta}(0) = \int\limits_{V^\alpha} \vec{u}_0 \varphi_{\alpha,\beta} \, dV, \quad (6)$$

где $\vec{u}_0(x_1, x_2)$ — начальное распределение консервативных переменных.

Интегралы по поверхностям S_ψ^α и по областям V^α в рассматриваемом двумерном случае сводятся к интегралам по ребрам P_ψ^α и треугольнику K^α соответственно.

Для приближенного решения задачи Римана для системы уравнений МГД используется ряд численных методов, таких как HLL, HLLC, HLLD [5, 7]. Метод HLLD позволяет получить наилучшее разрешение разрывов, но во многих случаях приводит к возникновению численной немонотонности в решении. Чтобы этого избежать, может быть использован численный поток HLL, обладающий большей численной вязкостью.

При расчете интегралы заменяются квадратурными формулами, точность которых согласована с порядком метода m. Решение задачи Коши (5)–(6) производится численным интегрированием явным методом Рунге — Кутты, шаг по времени τ определяется динамически из условия устойчивости Куранта — Фридрихса — Леви:

$$\max_i |\lambda_i| \frac{\tau}{h_{\min}} \le 1, \quad (7)$$

где $h_{\min}$ — длина наименьшего ребра ячейки в сетке, λ_i — собственные числа системы (1) на предыдущем шаге по времени. Разрывный метод Галеркина первого порядка совпадает с соответствующими методами типа Годунова [5].

Для аппроксимации граничных условий используется метод фиктивной ячейки.

3.2. Монотонизация решения

Решение уравнений МГД разрывным методом Галеркина второго порядка точности требует использования функций-ограничителей наклона решения в ячейке — лимитеров [6] — как для поддержания монотонности решения, так и для предотвращения появления на очередном временном слое нефизичных отрицательных значений плотности и давления.

В данной работе использован "TVB minmod" лимитер для кусочно-линейных функций [2,5], значение которого зависит от исходного наклона решения в данной ячейке и от средних значений решения в соседних ячейках. Процедура лимитирования магнитного поля может привести к возникновению численного магнитного заряда и поэтому применяется только при вычислении потоков. Лимитирование проводится для локальных характеристических переменных, получаемых из консервативных умножением на матрицу $L = R^{-1}$ левых собственных векторов, вычисляемую в каждой ячейке вместе с разложением (2).

Рассмотрим алгоритм применения лимитера к вектору консервативных переменных, заданному линейной функцией $\vec{u}_{K^0}(x_1, x_2)$ на треугольнике K^0 с соседями K^1, K^2, K^3 (рис. 1).

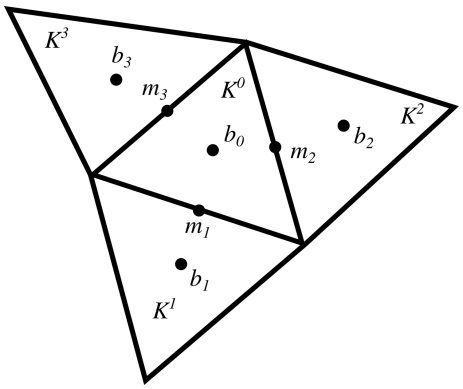


Рис. 1. Построение лимитера

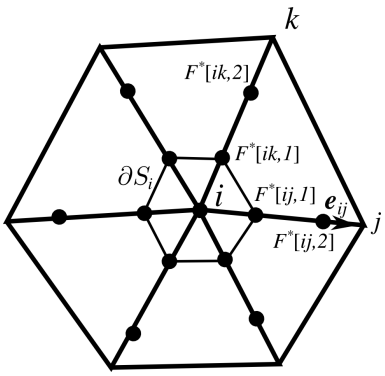


Рис. 2. Схема перераспределения магнитных потоков

Для каждой пары соседних треугольников K^i, K^j (см. рис. 1) из системы уравнений

$$\boldsymbol{m}_i - \boldsymbol{b}_0 = \alpha_{ij,1}(\boldsymbol{b}_i - \boldsymbol{b}_0) + \alpha_{ij,2}(\boldsymbol{b}_j - \boldsymbol{b}_0),$$

где $\boldsymbol{b}_i$ — барицентр i-ого треугольника, $\boldsymbol{m}_i$ — центр i-ого ребра, общего между K^0 и K^i, определим скалярные коэффициенты $\alpha_{ij,1}$ и $\alpha_{ij,2}$, характеризующие геометрические свойства данной группы треугольников K^0, K^i, K^j.

На основании полученных коэффициентов определим средние значения скачка функции $\vec{u}_{K^0}$ в m_i:

$$\vec{u}^*_{K^0}(\boldsymbol{m}_i) = \alpha_{ij,1}(\vec{u}_{K^i}(\boldsymbol{b}_i) - \vec{u}_{K^0}(\boldsymbol{b}_0)) + \alpha_{ij,2}(\vec{u}_{K^j}(\boldsymbol{b}_j) - \vec{u}_{K^0}(\boldsymbol{b}_0)).$$

Исходя из полученных значений, проведем ограничение скачков $\vec{u}_{K^0}$ в центрах ребер:

$$\vec{\Delta}_i = R\bar{m}(L(\vec{u}_{K^0}(\boldsymbol{m}_i) - \vec{u}_{K^0}(\boldsymbol{b}_0)), \nu L\vec{u}^*_{K^0}(\boldsymbol{m}_i)),\ i = \overline{1,3},$$

где $\nu > 1$ — константа, определяющая степень сглаживания решения, L и R — матрицы левых и правых собственных векторов из разложения (2), $\bar{m}$ — "TVB minmod" функция,

применяемая покомпонентно для каждой характеристической переменной.

$$\bar{m}_j\big(\vec{a},\vec{b}\big) = \begin{cases} a_j, \text{ если } |a_j| \le M_j(h_{\min})^2, \\ 0.5(\operatorname{sign} a_j + \operatorname{sign} b_j)\min(|a_j|,|b_j|), \text{ иначе,} \end{cases} \qquad j = \overline{1,8},$$

где M_j — константа, зависящая от задачи, $\operatorname{sign} a$ — функция знака. После определения скачков Δ_i, задающих новый наклон решения в K^0, их значения используем для обновления коэффициентов при базисных функциях.

3.3. Условие отсутствия магнитного заряда

Выполнение условия бездивергентности магнитного поля $\operatorname{div} \boldsymbol{B} = 0$ является одной из основных проблем при создании численного метода решения уравнений магнитной гидродинамики [8]. Несмотря на то, что дивергенция начального распределения магнитного поля равна нулю, при использовании консервативной формы записи системы уравнений МГД (1) в ходе расчетов может накапливаться численный магнитный заряд. Такая ситуация приводит к нефизичным результатам или даже преждевременному прерыванию расчета из-за получения отрицательных значений плотности или давления.

Для поддержания бездивергентности магнитного поля нами реализован метод, основанный на использовании z-компоненты электрического поля и её градиента в узлах сетки (рис. 2) для определения бездивергентного потока магнитного поля. Для случая метода первого порядка (без использования градиентов) процедура описана в [9].

После определения известными методами (HLL, HLLC, HLLD) значений численного потока $\vec{F}^*[ij,k]$ в точках интегрирования (для метода второго порядка две точки — $k = 1,2$) на ребре, содержащем i и j узлы, в каждом из узлов сетки вычислим средние значения z-компоненты электрического поля E_z и $\nabla E_z = \left(\dfrac{\partial E_z}{\partial x_1}, \dfrac{\partial E_z}{\partial x_2}\right)$:

$$E_{zi} = \frac{1}{2}\left(\sum_{k=1}^{\sigma_i} \frac{(F_6^*[ik,1], F_7^*[ik,1])\cdot \boldsymbol{e}_{ik}}{\sigma_i} + \sum_{k=1}^{\sigma_i} \frac{(F_6^*[ik,2], F_7^*[ik,2])\cdot \boldsymbol{e}_{ik}}{\sigma_i}\right). \tag{8}$$

Здесь и далее $\boldsymbol{e}_{ik}$ — единичный вектор, касательный к ребру ik и направленный из i-й вершины в k-ю, h_{ik} — длина ребра ik, σ_i — число ребер, выходящих из вершины i. Восстановление градиента ∇E_{zi} производится по формуле $\nabla E_z \approx \frac{1}{S}\int\limits_S \nabla E_z\, dS = \frac{1}{S}\int\limits_{\partial S} E_z \boldsymbol{n}\, dl$ (рис. 2). Этот интеграл берется численно с использованием вычисленных значений потоков $\vec{F}^*[ij,k]$.

Для монотонизации решения необходимо провести ограничение градиента электрического поля аналогично процедуре, рассмотренной выше. Для этого на основании значений E_{zi} в узлах расчитываем средний градиент ∇E_{zi}^{Lim}, результирующее значение градиента, используемое далее в расчетах, определяем путем применения minmod функции $\nabla E_{zi} = \bar{m}(\nabla E_{zi}, \nu \nabla E_{zi}^{Lim})$.

Используя значения электрического поля и его градиента, можно построить бездивергентные потоки $\vec{\mathcal{F}}^*[ij,1]$ и $\vec{\mathcal{F}}^*[ij,2]$, отличающиеся от исходного только компонентами потока магнитного поля:

$$\vec{\mathcal{F}}_l^*[ij,k] = \vec{F}_l^*[ij,k],\ k = 1,2,\ l = \overline{1,5},8,$$

$$\big(\vec{\mathcal{F}}_6^*[ij,1], \vec{\mathcal{F}}_7^*[ij,1]\big)^T = \frac{1}{6}\Big(\big(3+\sqrt{3}\big)E_{zi} + \big(3-\sqrt{3}\big)E_{zj} + \frac{1}{2}\big(\nabla E_{zi} - \nabla E_{zj}\big)\cdot \boldsymbol{e}_{ij}h_{ij}\Big)\, h_{ij}\boldsymbol{e}_{ij},$$

$$\big(\vec{\mathcal{F}}_6^*[ij,2], \vec{\mathcal{F}}_7^*[ij,2]\big)^T = \frac{1}{6}\Big(\big(3-\sqrt{3}\big)E_{zi} + \big(3+\sqrt{3}\big)E_{zj} + \frac{1}{2}\big(\nabla E_{zi} - \nabla E_{zj}\big)\cdot \boldsymbol{e}_{ij}h_{ij}\Big)\, h_{ij}\boldsymbol{e}_{ij},$$

Такое перераспределение потоков сохраняет консервативность схемы и гарантирует бездивергентность решения. Это подтверждается численными экспериментами: в случае метода первого порядка этот факт показан в [9]. В то же время численные эксперименты показывают, что такой подход приводит к сглаживанию разрывов (вносит в схему дополнительную численную взякость): на 4-5 ячеек для разрывного метода Галеркина первого порядка, и 2–3 ячейки — для метода второго порядка.

4. Параллельный алгоритм

Описанный численный метод реализован в параллельном программном комплексе для кластерных систем с разделенной памятью. Комплекс написан на языке Fortran с использованием технологий OpenMP и MPI. Параллельный алгоритм комплекса основан на идее разбиения исходной расчетной области на подобласти, решение в каждой из которых рассчитывается отдельным вычислительным модулем [10]. Поскольку численный метод — явный, подобный подход позволяет добиться высокой эффективности распараллеливания. Целостность решения обеспечена наличием обменов смежными граничными данными между модулями, обмены реализованы средствами технологии MPI. Расчет внутри каждой из подобластей может быть ускорен с помощью применения технологии OpenMP в том случае, если задействованные для решения задачи модули состоят из нескольких вычислительных ядер [3].

Опишем основные этапы работы параллельного программного комплекса.

4.1. Подготовка к запуску

Расчет производится на треугольных неструктурированных сетках. Для построения сеток использован программный комплекс Gridder2D [11], позволяющий производить триангуляцию областей сложной формы. На подготовительном этапе производится разбиение сетки на подобласти по числу используемых вычислительных модулей. При этом необходимо с одной стороны минимизировать количество смежных данных, подлежащих обмену между модулями, а с другой — сбалансировать число ячеек сетки в каждой подобласти для обеспечения равномерной загрузки ядер. Эту задачу можно представить как построение разбиения графа, каждая вершина которого соответствует ячейке, а ребро между вершинами — общему ребру между ячейками. Для приближенного решения данной задачи используется программа Metis [12], устанавливающая соответствие между номером ячейки в сетке и номером подобласти.

4.2. Инициализация данных

Инициализация расчета производится процессом, исполняемым на нулевом модуле (“корневой процесс”). На основании информации о расчетной сетке и ее разбиении на подобласти, а также о сеточном шаблоне, используемом как для расчета решения на очередном временном слое, так и для его лимитирования, корневой процесс формирует для каждого расчетного модуля ряд вспомогательных массивов.

- Массив перенумерации узлов, устанавливающий соответствие между номерами узлов в исходной сетке и номерами узлов, используемых в модуле — на каждом вычислительном модуле хранится и участвует в расчетах лишь минимально необходимая часть сетки и определенные на ней распределения физических переменных задачи.

- Массив перенумерации ячеек сетки — аналогично массиву перенумерации узлов данный массив позволяет хранить в памяти информацию только о тех ячейках, которые необходимы для расчета на данном модуле. Этот массив специальным образом упорядочен. Самые маленькие номера имеют ячейки, которые используются только на

данном модуле. Далее размещены блоки ячеек, значение решения в которых рассчитывается данным модулем и отсылается на другой модуль путем групповой пересылки. Затем располагаются ячейки, значение решения в которых по отдельности рассылается нескольким модулям (угловые ячейки). В конце массива расположены блоки ячеек, значения решения в которых принимаются от других модулей блочным способом, ячейки, решение в которых принимается по отдельности, а также граничные ячейки. Такое упорядочивание позволяет достичь большей эффективности пересылки данных между модулями.

- Массивы, описывающие начало и конец рассылаемых и принимаемых блоков и одиночных данных о решении.

После рассылки данных корневым модулем все вычислительные модули (включая корневой) начинают независимый расчет решения в своей подобласти, инициализируя массивы, описывающие базисные функции, содержащие скалярные коэффициенты лимитера и ряд других.

Для обмена значениями решения в смежных ячейках и электрического поля в смежных узлах на границах подобластей каждым модулем инициализируются отложенные передача и прием данных. При этом благодаря непрерывной блочной структуре хранения данных в памяти пересылка значений в смежных ячейках и узлах осуществляется без формирования дополнительных буферных массивов (и без дополнительных накладных расходов) прямым обращением к соответствующим указателям.

Так как z-компонента электрического поля, ее производные и значения лимитера для этих производных вычисляются в узлах сетки по формуле (8) как сумма значений на ребрах, прилегающих к данному узлу, то при параллельном счете требуется учитывать значения этих величин на ребрах, обрабатываемых разными модулями. В этих случаях каждый из модулей вычисляет сумму только для своей части ребер и рассылает отложенным способом это значение на другие модули, обрабатывающие данный узел. Затем каждый из модулей суммирует принятые данные и получает правильные значения в узле. Такая методика оказывается эффективнее централизованого расчета соответствующих значений.

4.3. Расчет

Интегрирование по времени производится явным методом Рунге — Кутты второго порядка: каждый шаг по времени состоит из двух этапов вычисления решения в моменты времени, соответствующие выбранному методу Рунге-Кутты. Величина очередного шага по времени для метода Рунге — Кутты в каждой подобласти определяется на основании условия (7), а из этих шагов при помощи средств библиотеки MPI выбирается минимальный, при котором условие устойчивости численной схемы оказывается выполнено во всей расчетной области. Процедура расчета решения в очередной временной точке включает в себя как пересчет решения внутри ячеек подобласти, так и вычисление лимитеров для всех консервативных переменных, а также операции по вычислению в узлах сетки электрического поля, его градиента и соответствующих ограничителей наклона. Все указанные действия требуют обмена данными между смежными вычислительными модулями, которые выполняются в режиме отложенных запросов, что позволяет избежать простоя вычислителей. Запись результатов расчета на диск производится буферизованным способом. В случае, когда вычислительный модуль включает несколько вычислительных ядер, последние два приема позволяют существенно уменьшить реальную долю последовательного кода во времени исполнения алгоритма. Кроме того, расчеты, выполняемые внутри одного вычислительного модуля, в этом случае легко распараллелены при помощи технологии OpenMP.

5. Результаты расчетов

Созданный программный комплекс протестирован на ряде стандартных тестовых задач двумерной идеальной МГД. Результаты решения задачи о вихре Орзага-Танга, демонстрирующие разрешающую способность метода и эффективность распараллеливания алгоритма, приведены ниже. Результаты получены RKDG методом второго порядка точности по пространству и времени с HLLD разностными потоками.

5.1. Вихрь Орзага-Танга

Данная задача иллюстрирует процесс формирования и взаимодействия ударных волн [13]. Начальные распределения физических величин — периодические по пространству гладкие функции. Тем не менее достаточно быстро в решении появляются разрывы плотности, давления и скорости. Благодаря периодичности решения по пространству в качестве расчетной области достаточно взять единичный квадрат с периодическими граничными условиями. Использованы следующие начальные данные

$$(\rho,\ v_1,\ v_2,\ v_3,\ p,\ B_1,\ B_2,\ B_3) = \\ = (\frac{25}{36\pi},\ -\sin 2\pi x_2,\ \sin 2\pi x_1,\ 0,\ \frac{5}{12\pi},\ -\sin 2\pi x_2,\ \sin 4\pi x_1,\ 0).$$

Показатель адиабаты $\gamma = 5/3$, что соответствует идеальному одноатомному газу. Число Куранта выбрано равным $C = 0.4$, в расчетах использовались треугольные сетки различной подробности. На рис. 3 представлены распределения плотности и модуля вектора магнитной индукции в момент времени $t = 0.5$, полученные на сетке из 903222 треугольников. Метод показывает высокий уровень разрешения структуры решения, не сравнимый с методами первого порядка (см. аналогичные результаты в [13]), причем это касается не только обострения разрывов (в среднем разрывы размазаны на 2–3 ячейки), но и выявления мелкомасштабной структуры разрывов в решении.

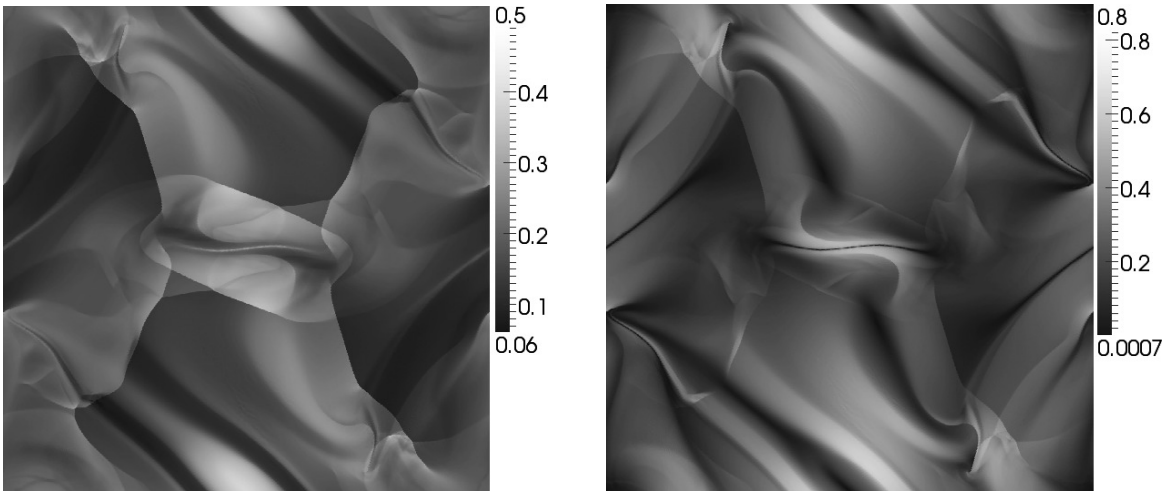


Рис. 3. Распределение плотности (слева) и модуля вектора магнитной индукции (справа) в задаче о вихре Орзага-Танга.

5.2. Параллельные расчеты

Проведено тестирование параллельного алгоритма на гибридном вычислительном кластере К-100 Института прикладной математики им. М.В. Келдыша РАН. Оценка ускорения проводилась для решения на сетках из 585508 и 903222 элементов. Доля параллельного кода

в описанном алгоритме при решении задачи о вихре Орзага-Танга составляет 99.98% для сетки из 585508 элементов и 99.99% — для 903222 элементов. Согласно закону Амдала [14] такие доли ограничивают максимальное ускорение расчетов на 256 ядрах относительно последовательной версии величинами 243 и 249 раз соответственно. Решение задачи на сетке из 903222 элементов на одном ядре заняло 1843,8 минуты, на 256 ядрах — 8,17 минуты (ускорение в 225.7 раза). На сетке из 585508 элементов на одном ядре — 720 минут, на 256 ядрах — 3,68 минуты (ускорение в 195.7 раза).

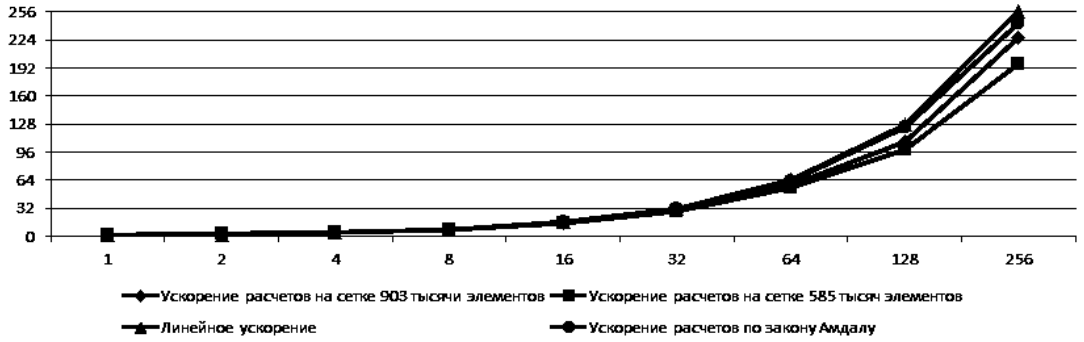


Рис. 4. Ускорение расчетов в задаче о вихре Орзага-Танга на различных сетках

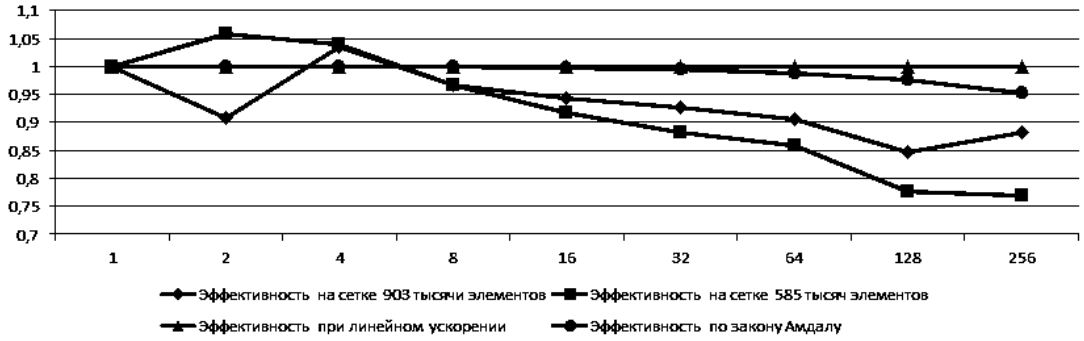


Рис. 5. Эффективность параллельного алгоритма в задаче о вихре Орзага-Танга на различных сетках

Расхождение реального ускорения с оценкой по закону Амдала объясняется главным образом не влиянием межмодульных обменов, а спецификой аппаратной структуры кластера. Вычислительные модули кластера К-100 сгруппированы по 11 на вычислительный узел (в одноядерной конфигурации модуля), каждый из которых представляет собой сервер с двумя шестиядерными процессорами Intel Xeon 5670. Подобная плотность вычислительных модулей в расчете на узел приводит к конкуренции запросов модулей одного узла к оперативной памяти, а следовательно — к большому числу кэш-промахов. Это особенно критично для текущей реализации метода решения уравнений МГД второго порядка точности, поскольку каждой ячейке сетки соответствует как минимум 32 числа с плавающей точкой. Расчеты показали, что процедура вычисления потоков через ребра на четырех ядрах ускоряется в 3,8 раза, в то время как процедура копирования решения с предыдущего слоя на новый — только в 2,1 раза. Специфика аппаратной структуры кластера К-100 объясняет и наблюдаемое на рис. 4 и рис. 5 сверхлинейное ускорение расчетов, возникающие на 2 и 4 ядрах. В этом случае не возникает конкуренции за ресурсы между вычислительными модулями одного узла, причем часть обменов может выполняться незанятыми в расчете ресурсами узла.

На рис. 4 представлены графики ускорения расчетов, полученные при решении задачи

о вихре Орзага-Танга на сетках из 585508 и 903222 элементов. Также представлен график ускорения согласно закону Амдала для 99.98% параллельного кода и график линейного ускорения. На рис. 5 представлены графики эффективности (отношение ускорения к числу задействованных ядер) распараллеливания расчетов на различных сетках, эффективности согласно закону Амдала для 99.98% параллельного кода и эффективности при линейном ускорении. Видно, что эффективность распараллеливания при большом числе используемых вычислительных модулей существенно зависит от трудоемкости задачи. В целом метод показывает высокую эффективность распараллеливания и хорошую масштабируемость вплоть до сотен задействованных вычислительных модулей.

6. Заключение

Создан алгоритм для решения двумерных уравнений магнитной гидродинамики разрывным методом Галеркинна, основывающийся на методах приближенного решения задачи Римана о распаде разрыва (HLL, HLLC, HLLD). Рассмотрены способы монотонизации схемы, введена процедура реконструкции магнитных потоков, позволяющая в случае методов первого и второго порядков точности получать бездивергентные распределения магнитного поля. Протестирован метод второго порядка точности по пространству и времени. Тестовая задача о вихре Орзага-Танга показывает высокий уровень разрешения разрывов и монотонность получаемого численного решения. Создана и протестирована параллельная версия программы с использованием технологии параллельных вычислений для систем с распределенной памятью MPI. Представлены графики ускорения и эффективности распараллеливания расчетов для тестовой задачи, показывающие высокую эффективность и хорошую масштабируемость параллельного алгоритма.

Литература

1. Cockburn B., Shu C.-W. Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. // Journal of scientific computing. 2001. № 3 (16). Pp. 173–261.

2. Галанин М. П., Савенков Е. Б., Токарева С. А. Решение задач газовой динамики с ударными волнами RKDG-методом // Математическое моделирование. 2008. Т. 20, № 11. С. 55–66.

3. Лукин В.В., Марчевский И.К. Учебно-экспериментальный вычислительный кластер. Часть 1. Инструментарий и возможности // Вестник МГТУ им. Н.Э. Баумана. Сер. Естественные науки. 2011. № 4. С. 28–44.

4. Лукин В.В., Марчевский И.К., Морева В.С., Попов А.Ю., Шаповалов К.Л., Щеглов Г.А. Учебно-экспериментальный вычислительный кластер. Часть 2. Примеры решения задач // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. 2012. № 4. С. 82–102.

5. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. М.: Физматлит, 2001. 608 с.

6. Toro E.F. Riemann solvers and numerical methods for fluid dynamics. A practical introduction. Berlin: Springer, 2009. 724 pp.

7. Mignone A., Bodo G. Shock-Capturing Schemes in Computational MHD // Lect. Notes Phys. 2008. V. 754. P. 71-101.

8. Toth G. The div $\boldsymbol{B} = 0$ constraint in shock-capturing magnetohydrodynamics codes // J. of Comp. Physics. 161, 2000, p. 605-652.

9. Torrilhon M. Locally Divergence-preserving Upwind Finite Volume Schemes for Magnetohydrodynamic Equations // J. Sci. Comp. 2005. V. 26. P. 1166–1191.

10. Марчевский И.К., Токарева С.А. Сравнение эффективности параллельных алгоритмов решения задач газовой динамики на разных вычислительных комплексах // Вестник МГТУ им. Н.Э. Баумана. Сер. “Естественные науки”. 2009. № 1. С. 90–97.

11. Щеглов И.А. Программа для триангуляции сложных двумерных областей Gridder2D. 2008. 32 с. Препр. Инст. прикл. матем. им. М.В.Келдыша РАН №60.

12. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. V. 20. № 1. Pp. 359 – 392.

13. Галанин М.П., Лукин. В.В. Разностная схема для решения двумерных задач идеальной МГД на неструктурированных сетках. 2007. 29 с. Препр. Инст. прикл. матем. им. М.В.Келдыша РАН №50.

14. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2004. 608 с.

Неманипулируемый для пользователей механизм распределения процессорного времени между неделимыми заданиями

А.А. Замятин

Новосибирский государственный университет

Задачу распределения ресурсов между пользователями в многопроцессорной системе можно решать с помощью экономических механизмов. J. Stößer, D. Neumann и C. Weinhardt в [1] предложили механизм, который является слабо бюджетно-сбалансированным, индивидуально рациональным, обеспечивает неманипулируемость для пользователей, но не является эффективным. Цель работы: модифицировать правило ценообразования так, чтобы новый механизм при той же оценке трудоемкости был в любой исходной ситуации не менее эффективен, а в некоторых случаях — более эффективен, чем исходный механизм.

1. Введение

ИТ-отделам крупных компаний требуется поддерживать работоспособность трудоемких приложений, которые становятся все более требовательными к вычислительным ресурсам. Чтобы система выдерживала пиковые нагрузки, приходится содержать значительное количество вычислительных ресурсов, которые большую часть времени бездействуют**.** Результаты исследований показали, что корпоративные центры обработки данных используют от 10% до 35% вычислительной мощности [2].

Эффективный путь снижения затрат на вычислительные ресурсы — использование распределенных вычислений, позволяющее организациям своевременно реагировать на пиковые нагрузки и арендовать требующиеся ресурсы. Рынок вычислительных услуг позволяет предприятию содержать оборудование, рассчитанное на базовую нагрузку, а пиковые нагрузки обслуживать с помощью внешних сетевых ресурсов.

Для вычислительной системы, оказывающей платные услуги, главной проблемой является распределение ресурсов и назначение цен, то есть определение того, какой ресурс в какое время и по какой цене предоставляется каждому пользователю.

Решать эту проблему можно с помощью экономического механизма распределения ресурсов. Дизайн экономических механизмов – это конструктивный подход, позволяющий создать такой механизм взаимодействия, при котором эгоистические действия каждого из агентов в сумме приведут к решению, оптимальному с точки зрения общей целевой функции [3]. Экономический механизм распределения вычислительных ресурсов есть набор правил взаимодействия между пользователями, арендующими вычислительные ресурсы, и провайдерами — арендодателями.

Для успешного применения в сетях желательно, чтобы экономический механизм обладал рядом свойств. В [1] описаны следующие свойства.

Двусторонний рынок. Двусторонние рынки (двусторонние сети) — сетевые рынки, которые имеют две группы участников (в нашем случае это пользователи и провайдеры) с возникновением сетевых эффектов между ними. Механизм должен обеспечивать заключение сделок между множеством пользователей и множеством провайдеров. Это свойство является обязательным для сетей, которые по определению состоят из ресурсов, не подчиненных централизованному контролю.

Вычислительная приемлемость. Механизм должен распределять задания по вычислительным узлам и устанавливать цены за полиномиальное время относительно размера входных данных, т.е. суммарного количества заявок спроса и предложения.

Комплектация. Пользователи должны иметь возможность описывать зависимость между несколькими заданиями и вычислительными узлами и указывать технические ограничения сво-

их заданий, что сужает множество вычислительных узлов, на которые эти задания могут быть распределены [4].

Временные ограничения. Предполагается, что пользователи знают желательные сроки выполнения своих заданий, а провайдеры время доступности своих вычислительных узлов. Механизм должен учитывать сформулированные участниками временные ограничения при распределении ресурсов.

Эффективность. Механизм является эффективным, если при любых входных данных он всегда генерирует распределение, максимизирующее суммарный излишек (общее благосостояние) участников взаимодействия.

Бюджетная сбалансированность. Механизм называется слабо бюджетно сбалансированным, если его платежная схема не нуждается в субсидировании извне. Платежи пользователей полностью покрывают платежи, которые получают провайдеры.

Индивидуальная рациональность (добровольность). Механизм обладает свойством индивидуальной рациональности, если участники не могут понести убытки от участия в механизме. В частности, пользователи не должны вносить плату за участие, не получая при этом ресурсов.

Неманипулируемость. Механизм называется неманипулируемым, если слабо доминирующей стратегией каждого участника является сообщение истинных параметров своего задания или вычислительного узла при произвольных параметрах других участников. Манипулируемый механизм не может гарантировать эффективность, так как распределение, эффективное при значениях параметров, сообщенных участниками, скорее всего не будет эффективным для истинных данных. Поэтому неманипулируемость является желательным свойством.

Экономический механизм состоит из трех основных компонентов [5]: правил описания заявок, алгоритма распределения ресурсов и схемы платежей. Правила описания заявок задают множества стратегий участников. Алгоритм распределения определяет, какое задание, на каком узле и в какое время выполняться, а схема платежей по результатам распределения назначает платежи пользователей провайдерам.

Существуют теоретические результаты, показывающие недостижимость некоторых комбинаций свойств экономического механизма [6]. В частности, из теоремы Майерсона-Сэттерсвэйта следует, что при естественных предположениях неманипулируемый механизм не может быть одновременно эффективным, бюджетно-сбалансированным и индивидуально рациональным. Так как последние два свойства необходимы для устойчивости механизма, жертвовать приходится или неманипулируемостью, или эффективностью. При большом количестве входных данных конфликтуют вычислительная приемлемость и эффективность, так как для вычисления оптимального распределения требуется много времени.

J. Stößer, D. Neumann, C. Weinhardt в [1] предложили механизм (назовем его SNW-механизм), который распределяет ресурсы между заданиями (этап 1) и определяет платежи пользователей (этап 2). Этот механизм является слабо бюджетно-сбалансированным, индивидуально рациональным и обеспечивает неманипулируемость для пользователей. Однако SNW-механизм не является эффективным, то есть не максимизирует суммарную полезность, полученную всеми участниками взаимодействия (так называемый, суммарный излишек).

Цель работы: в SNW-механизме модифицировать правило ценообразования так, чтобы новый механизм при той же оценке трудоемкости в любой исходной ситуации обеспечивал суммарный излишек не меньший, а в некоторых случаях больший, чем SNW-механизм.

2. SNW-механизм

2.1 Описание модели

Задан горизонт планирования, промежуток времени T_0, на котором решается задача распределения ресурсов. Он разделен на промежутки равной длины (назовем их единичными промежутками, е.п.) с номерами $1,\ldots,T$.

N — множество номеров заявок от вычислительных узлов с предложениями ресурсов на период $[0,T_0]$. Чтобы сделать предложение провайдер узла $n \in N$ должен отправить системе заявку вида $\eta_n = (V_n, C_n, M_n, R_n, E_n)$. Здесь: $V_n \in \mathrm{R}_+$ — себестоимость использования единицы компьютерной мощности в течение одного е.п. (единицы мощности-времени в терминологии авто-

ров) в денежном эквиваленте; $C_n \in \mathrm{N}$ и $M_n \in \mathrm{N}$ — максимально доступные мощность и память вычислительного узла соответственно; R_n и E_n, $1 \le R_n \le E_n \le T$, — такие е.п., что узел n доступен в единичные промежутки с номерами $R_n,\ldots, E_n$; V_n — минимальная цена, по которой провайдер согласится сдать в аренду узел (отправная цена). Вычислительная мощность C_n может измеряться, например, числом циклов ЦПУ (тактов) за е.п, а память M_n — в мегабайтах, гигабайтах и т.п.

J — множество номеров заявок на выполнение заданий. Пользователь, желающий выполнить задание $j \in J$, отправляет заявку вида $\theta_n = (v_n, c_n, m_n, r_n, e_n)$, где: $v_j \in \mathrm{R}_+$ — максимальная цена, которую пользователь готов заплатить за единицу мощности-времени; $c_j \in \mathrm{N}$ и $m_j \in \mathrm{N}$ — минимально требующиеся заданию за один е.п. количества мощности и памяти соответственно; r_j и e_j, $1 \le r_j \le e_j \le T$, — такие номера е.п, что задание j должно выполняться в единичные периоды $r_j,\ldots, e_j$. Время выполнения задания j, выраженное числом единичных промежутков, обозначим $d_j = e_j - r_j + 1$. Тогда денежную оценку полезности (бюджет) задания можно посчитать по формуле $b_j = v_j c_j d_j$.

Множества заявок провайдеров и потребителей обозначим соответственно $\eta = \{\eta_n \mid n \in N\}$ и $\theta = \{\theta_j \mid j \in J\}$.

Предполагается, что вычислительный узел может выполнять несколько заданий одновременно, а задание не может выполняться одновременно на разных вычислительных узлах, но может мигрировать между ними без необходимости перезапуска, то есть в различные промежутки времени может выполняться на различных узлах.

Требуется распределить задания по вычислительным узлам и определить платежи пользователей провайдерам. Решение задачи включает два этапа: распределительный и оценочный.

Положим $X = (x_{jnt} \mid j \in J, n \in N, t \in [r_j, e_j] \cap [R_j, E_j])$, где

$$x_{jnt} = \begin{cases} 1, & \text{если задача } j \text{ выполняется на узле } i \text{ в е.п. } t, \\ 0 & \text{иначе.} \end{cases}$$

Пусть W — общее благосостояние, которое определяется как разность суммы денежных оценок полезности, полученной пользователями, и суммарных затрат провайдеров. W есть суммарный излишек участников механизма:

$$W = \sum_j c_j \sum_n \sum_t x_{jnt}(v_j - V_n). \qquad (*)$$

Проблему распределения заданий по узлам можно представить в виде задачи целочисленного линейного программирования:

$$W_{\max} = \max_X \sum_j c_j \sum_n \sum_t x_{jnt}(v_j - V_n), \qquad (1)$$

$$\sum_n x_{jnt} \le 1 \text{ для всех } j \text{ и } t \in [r_j, e_j], \qquad (2)$$

$$\sum_j x_{jnt} c_j \le C_n \text{ для всех } n \text{ и } t \in [R_n, E_n], \qquad (3)$$

$$\sum_j x_{jnt} m_j \le M_n \text{ для всех } n \text{ и } t \in [R_n, E_n], \qquad (4)$$

$$\sum_{u=r_j}^{e_j} \sum_n x_{jnu} = d_j \sum_n x_{jnt} \text{ для всех } j \text{ и } t \in [r_j, e_j], \qquad (5)$$

$$x_{jnt} \in \{0,1\} \text{ для всех } j, n, t \in [r_j, e_j] \cap [R_n, E_n],\ V_n \le v_j. \qquad (6)$$

Ограничения (2) - (6) имеют следующую интерпретацию. Неравенство (2) гарантирует, что задание не будет запланировано на разных узлах одновременно. Неравенства (3) и (4) требуют, чтобы распределенные на вычислительный узел задания не запрашивали больше ресурсов, чем этот узел может предоставить. Равенство (5) обеспечивает выполнение условия неделимости заданий: задание либо выполнится полностью, либо не будет выполняться вообще. Наконец, ограничение (6) сокращает количество возможных распределений в соответствии с указанными участниками временными интервалами, а также обеспечивает, что задание не будет размещено на узле, отправная цена которого больше, чем пользователь готов заплатить.

Описанная задача является гибридом задачи об упаковке в контейнеры и обобщенной задачи о назначениях [7]. Нетрудно заметить, что задача (1) – (6) не менее трудоемка, чем задача об упаковке в контейнеры, так как функция благосостояния зависит от отправных цен вычислительных узлов, а в задаче об упаковке в контейнеры стоимость использования всех контейнеров равна нулю. Кроме того, постановка задачи осложнена наличием двух различных ресурсов, а главное, ограничение (5) связывает между собой распределительные задачи всех временных промежутков, каждая из которых по отдельности сложнее задачи об упаковке в контейнеры. Так как задача об упаковке в контейнеры является NP-полной [8], задача (1) – (6) также NP-полна. Следовательно, вычисление оптимального распределения может оказаться очень трудоемким. Но для нормального функционирования системы распределение должно происходить как можно быстрее.

2.2 Алгоритм распределения ресурсов

Чтобы обойти вычислительную сложность, в [1] предложено на этапе 1 работы SNW-механизма использовать жадный эвристический алгоритм. Идея алгоритма состоит в том, чтобы на каждом шаге максимизировать разность $x_{jnt}(v_j - V_n)$, содержащуюся в функции общего благосостояния. Приведем этот алгоритм.

Алгоритм 1. Жадный эвристический алгоритм распределения ресурсов.

Шаг 0. Сортируем задания $j \in J$ в порядке невозрастания v_j. Сортируем вычислительные узлы $n \in N$ в порядке неубывания V_n. Упорядоченные множества J и N будем в дальнейшем называть очередями. Пусть $j(k)$ — номер задания, стоящего на месте k в очереди J.

Шаг k. Последовательно пытаемся разместить задание $j(k)$ на вычислительных узлах очереди N, начиная с первого. То есть, для задания $j(k)$ на каждом временном промежутке $t \in \{r_{j(k)},\ldots, e_{j(k)}\}$ подбираем первый в очереди N вычислительный узел, удовлетворяющий техническим характеристикам задания. Если хотя бы на одном промежутке не удалось распределить задание, то, в соответствии с ограничением (5), задание не попадет в распределение и не будет выполнено. Если же для каждого временного промежутка найдется подходящий вычислительный узел, то задание будет распределено. Тогда на соответствующих узлах резервируются указанные в заявке задания количества ресурсов.

Конец алгоритма.

Заметим, что в худшем случае общее благосостояние, полученное данным алгоритмом, может быть в сколько угодно раз меньше благосостояния при оптимальном распределении [1]. И все же в среднем жадный эвристический алгоритм обеспечивает близкие к оптимальным распределения. На смоделированных случайных начальных данных алгоритм показал эффективность около 95%, то есть $W / W_{opt} \approx 0{,}95$ [1]. При этом алгоритм имеет полиномиальную трудоемкость, делая механизм вычислительно приемлемым.

Кроме того, в [1] утверждается, что при некоторых технических правилах функционирования рынка жадный эвристический алгоритм распределения ресурсов существенно ограничивает стратегическое поведение как пользователей, так и провайдеров. А именно, для участников рынка доминирующей стратегией является сообщение истинных технических характеристик задания (c_j, m_j для пользователя или C_n, M_n для провайдера). Пользователю невыгодно занижать технические характеристики задания, так как в сетевых системах обычной политикой является прерывание приложений, запрашивающих больше ресурсов, чем заявлено. Очевидно, и завышать характеристики не имеет смысла: в этом случае пользователь не только заплатит больше за выполнение задания, но и уменьшит число узлов, подходящих заданию по техническим характеристикам. Для провайдера завышение характеристик вычислительного узла может привести к тому, что назначенным на этот узел заданиям не хватит ресурсов для корректной работы (что грозит штрафом провайдеру), а занижение характеристик приведет к тому, что незаявленные мощности вычислительного узла будут простаивать.

2.3 Платежная схема SNW-механизма

Рассмотрим этап 2 работы SNW-механизма: назначение платежей. Предложенная авторами платежная схема — оценивание по критическому значению. Идея алгоритма состоит в том, что

каждое задание должно заплатить за единицу мощности-времени минимальную цену, при которой механизм включил бы его в распределение. Обозначим эту цену $\varphi_j(\eta, \theta)$. Тогда задание j должно заплатить $p_j = \varphi_j c_j d_j$, если будет выполнено, и $p_j = 0$ иначе. Главное преимущество оценивания по критическому значению состоит в том, что оно, как доказано в [1], обеспечивает неманипулируемость по параметрам v_j.

Для $j \in J$ положим $\theta_{-j} = \theta \setminus \{j\}$. Неманипулируемость механизма по v_j означает, что

$$u_j(\theta_j, \tilde{\theta}_{-j}, \tilde{\eta} \mid \theta_j) \geq u_j(\tilde{\theta}_j, \tilde{\theta}_{-j}, \tilde{\eta} \mid \theta_j) \text{ для всех } j \in J .$$

Приведем алгоритм вычисления критических значений для заданий.

Алгоритм 2. Вычисление критических значений.

Шаг $k \geq 1$. Если задание k не было распределено, переходим к шагу $k + 1$, так как для нераспределенного задания искать критическое значение не нужно, его платеж равен нулю. Иначе начинаем распределять задания по жадному эвристическому алгоритму, исключив из очереди задание k. После распределения каждого задания проверяем, осталось ли достаточно ресурсов для распределения задания k. Возможны два исхода.

Если осталось достаточно ресурсов для распределения задания k, переходим к следующему заданию в очереди и продолжаем распределение. Если очередь закончилась, задание k должно платить по отправным ценам вычислительных узлов, на которые оно назначено,

$$p_k = c_k \sum_n \sum_{t=r_k}^{e_k} x_{kmt} V_n .$$

Если же для задания k недостаточно ресурсов, то $\varphi_k = v_j$, где j – последнее распределенное задание из очереди, и $p_k = v_j c_k d_k$.

Конец алгоритма.

Фактически, чтобы определить критическое значение задания j, требуется частично построить распределение заданий без j. Нахождение критических значений является наиболее трудоемкой процедурой при реализации механизма, и все же выполняется за полиномиальное время благодаря эвристическому алгоритму распределения. Распределение выполняется за полиномиальное время относительно размера входных данных: сортировочный этап выполняется за время $O(|J|\ln|J|)$ и $O(|N|\ln|N|)$, а непосредственно распределение — за $O(|J||N|)$. Процесс нахождения критических значений всех заданий имеет трудоемкость $O(|J|^2|N|)$.

В [1] доказано, что при использовании жадного эвристического алгоритма распределения не существует такой схемы платежей, которая обеспечивала бы неманипулируемость в отношении отправных цен вычислительных узлов. Предложено распределять платежи между провайдерами пропорционально количеству предоставленной вычислительной мощности (основной ресурс). Авторы считают, что этот способ ограничивает стратегическое поведение провайдеров. Общая прибыль провайдеров равна

$$S(\theta, \eta) = \sum_j p_j(\theta, \eta) - \sum_j \sum_n \sum_t x_{jmt} c_j V_n .$$

Если распределять прибыль пропорционально количеству арендованной пользователями вычислительной мощности, то платежи провайдерам будут равны

$$P_n = \sum_j \sum_t x_{jmt} c_j V_n + S(\theta) \frac{\sum_j \sum_t x_{jmt} c_j}{\sum_j \sum_{m \in N} \sum_t x_{jmt} c_j} .$$

Подведем итог. SNW-механизм обладает свойствами двойного рынка, вычислительной приемлемости, комплектации, временных ограничений, бюджетной сбалансированности, индивидуальной рациональности, является частично неманипулируемым (для пользователей), но не является эффективным. Поиск эффективного распределения является NP-полной задачей, а так как для использования механизма на практике вычислительная приемлемость обязательна, приходится довольствоваться эвристическим приближением к эффективному распределению.

3. Ценообразование в соответствии с обобщенным механизмом Викри

Как отмечено выше, для нахождения критического значения задания требуется частично построить распределение без этого задания. При этом может оказаться, что $W_{-j} > W$, где W_{-j} — общее благосостояние при распределении без задания j. То есть, на оценочном этапе может быть обнаружено лучшее распределение, чем найденное на распределительном этапе.

Последовательное выполнение распределительного и оценочного этапов в описанном выше механизме не позволяет корректировать уже полученное распределение. Поэтому мы предлагаем динамически корректировать распределение, используя цены, порождаемые обобщенным механизмом Викри [9]. Для распределения будем использовать изложенный выше жадный алгоритм из [1]. Принцип ценообразования состоит в том, что платеж назначенного задания j равен денежной оценке полезности, которую другие задания «теряют» из-за присутствия задания j. Другими словами, платеж равен разности суммарного излишка при распределении без задания j и суммарного излишка при распределении с заданием j за вычетом полезности, полученной заданием j, то есть $p_j = W_{-j} - (W - v_j c_j d_j)$. При этом задание j, для которого $W_{-j} > W$, окажется неплатежеспособным, будет превышен его бюджет. В таком случае механизм корректирует распределение.

Приведем алгоритм назначения цен с динамическим изменением распределения.

Алгоритм 3. Ценообразование по Викри с корректировкой распределения.

Шаг 0. Имеется X_0 — начальное распределение, полученное жадным эвристическим алгоритмом, а также отсортированные списки заданий J (в порядке невозрастания v_j) и вычислительных узлов N (в порядке неубывания V_n). Пусть W_0 — суммарный излишек при распределении X_0. Положим $X := X_0$, $W := W_0$. Будем говорить, что X — текущее распределение. Введем пустую очередь F, включающую задания, платежи которых уже назначены, а также пустую очередь D, в которую будем помещать «вытесненные» задания.

Шаг $m \geq 1$. Если очередь J пуста, то присваиваем $J := D$, $D := \{\varnothing\}$. Выбираем первое в очереди J задание (пусть его номер равен k). Если это задание не входит в текущее распределение, перемещаем его в конец очереди F, полагаем $p_k := 0$ и переходим к следующему шагу. Иначе производим распределение по жадному эвристическому алгоритму для очереди $F+J+D$ (пропуская этап сортировки, и распределяя сначала задания из F, затем из J, и затем из D) без задания k и находим общее благосостояние W_{-k} при таком распределении. Возможны два исхода.

Если $W_{-k} \leq W$, то назначаем платеж $p_k = W_{-k} - W + v_k c_k d_k$ и перемещаем задание j в конец очереди F.

Если $W_{-k} > W$, то задание k неплатежеспособно, т.к. $p_k > b_k = v_k c_k d_k$, платеж по обобщенному механизму Викри превышает бюджет задания. Назначаем построенное распределение X_{-k} (без задания k) текущим, $X := X_{-k}$, $W := W_{-k}$. Задание k «вытесняется», перемещаем его в конец очереди D.

Алгоритм завершается, когда все задания перейдут в очередь F (при этом очереди J и D будут пусты).

Конец алгоритма.

Результатом выполнения алгоритма будет распределение X, которое полностью задается очередью F (если произвести распределение по жадному эвристическому алгоритму без шага сортировки для очереди F, получим распределение X), и вектор платежей $(p_j \mid j \in J)$. Заметим, что после того, как заданию назначен платеж и оно перешло в очередь F, его позиция в распределении не изменяется. То есть на каждом шаге имеется частичное распределение (заданий из очереди F). Поэтому нет необходимости на каждом шаге заново вычислять распределение, достаточно хранить в памяти частичное распределение заданий из F.

Описанный механизм распределения ресурсов и назначения платежей обозначим GA+GVA (Greedy Algorithm + Generalized Vickrey Auction).

Теорема 1. GA+GVA выполняется за конечное число шагов при любых начальных данных.

Доказательство. Проведем доказательство от противного. Предположим, что при некоторых данных алгоритм не выполняется за конечное количество шагов (зацикливается). Условие завершения алгоритма — переход всех заданий в очередь F. Заметим, что задание, попав в очередь F, остается там и не переходит в другие очереди в процессе выполнения алгоритма. По-

этому бесконечная работа алгоритма означает, что после какого-то шага задания перестанут попадать в очередь F.

Следовательно, все задания очереди J с некоторого момента переходят только в очередь D. Когда кончается очередь J, выполняются присвоения $J := D$, $D := \{\varnothing\}$ (что не меняет общую очередь $F+J+D$), и далее задания снова переходят в очередь D. Задание k переходит из J в D при условии, что $W_{-k} > W$. То есть на каждом следующем шаге общее благосостояние возрастает. А так как количество распределений, достижимых жадным эвристическим алгоритмом при произвольно заданной очереди заданий, конечно (и оценивается числом $J!$), то конечно и множество возможных значений функции общего благосостояния. Следовательно, переход заданий в очередь D может произойти только конечное число раз. Противоречие. Теорема доказана.

Теорема 2. Распределение заданий, порожденное механизмом GA+GVA, обеспечивает суммарный излишек, не меньший, а в некоторых случаях — больший, чем распределение, построенное SNW-механизмом.

Доказательство. Механизм GA+VGA начинает работу с распределения X_0, построенного SNW-механизмом и дающего общее благосостояние W_0. По теореме 1 GA+GVA за конечное число шагов построит финальное распределение X с общим благосостоянием W. В этом алгоритме переход от одного распределения к другому осуществляется, только если общее благосостояние в новом распределении больше общего благосостояния в текущем распределении. Поэтому либо $X = X_0$ (и тогда $W = W_0$), либо $W > W_0$.

Теорема 3. GA+GVA имеет трудоемкость $O(|J|^2|N|)$.

Доказательство. Для времени выполнения одного шага алгоритма справедлива оценка $O(|J||N|)$., это трудоемкость жадного эвристического алгоритма без этапа сортировки. Построим верхнюю оценку числа шагов, нужных для того, чтобы по крайней мере одно задание перешло в очередь F.

Как в доказательстве теоремы 1, предположим, что с некоторого шага m задания не попадают в F. Когда все задания перейдут из J в D и будут выполнены присвоения $J := D$, $D := \{\varnothing\}$, новая очередь J станет такой же, какой была изначально (на шаге m). Следовательно, если очередь J будет очищена без перехода заданий в F, то получим ту же общую очередь ($F+J+D$), и распределение X, порожденное этой очередью, будет таким же. Но тогда $W(X) > W(X)$, так как задание переходит в D только при условии $W_{-k} > W$. Противоречие.

Отсюда следует, что не более чем за $|J|$ шагов хотя бы одно задание перейдет в очередь F. То есть GA+GVA имеет трудоемкость $O(|J|^2|N|)$. Теорема доказана.

Очевидно, механизм GA+GVA является индивидуально рациональным, а благодаря использованию GVA для назначения платежей — неманипулируемым для пользователей. Покажем, что GA+GVA будет слабо бюджетно-сбалансированным.

Теорема 4. Механизм GA+GVA является слабо бюджетно-сбалансированным.

Доказательство. Достаточно показать, что платеж каждого задания покрывает отправную цену занятых этим заданием вычислительных ресурсов.

Платеж задания k, попавшего в распределение, равен $p_k = W_{-k} - W + v_k c_k d_k$. Если при распределении без задания k ресурсы, которые использовало k, остаются незадействованными (то есть у задания k нет конкурентов), то

$$W_{-k} = W - v_k c_k d_k + \sum_{t=r_k}^{e_k} \sum_{n \in N} V_n C_n x_{knt} .$$

Подставляя в выражение для платежа задания, получаем

$$p_k = \sum_{t=r_k}^{e_k} \sum_{n \in N} V_n C_n x_{knt} ,$$

то есть платеж задания k в точности равен сумме отправных цен арендованных им вычислительных ресурсов.

Если у задания k есть конкуренты, то его платеж будет не меньше полученного выше [6]. Таким образом, платежи пользователей всегда будут покрывать затраты провайдеров, механизм GA+GVA является слабо бюджетно-сбалансированным. Теорема доказана.

Можем заключить, что GA+GVA обладает всеми полезными свойствами SNW-механизма, работает с такой же скоростью, как и SNW-механизм, но в некоторых случаях обеспечивает более эффективные распределения.

Приведем пример, который демонстрирует преимущество механизма GA+VGA.

Пусть η_1=(1, 10, 2, 1, 1), η_2=(2, 6, 1, 1, 1), θ_1=(5, 6, 1, 1, 1), $\theta_2 = \theta_3 = \theta_4 = (4, 5, 1, 1, 1)$.

В систему поступают заявки двух вычислительных узлов, первый производит 10 единиц мощности, имеет 2 единицы памяти и отправную цену 1; второй производит 6 единиц мощности, имеет 1 единицу памяти и отправную цену 2. Также поступают заявки от четырех заданий, первое запрашивает 6 единиц мощности и 1 единицу памяти за е.п. и оценивает полезность единицы мощности-времени 5. Остальные три задания имеют идентичные характеристики: запрашивают 5 единиц мощности и 1 единицу памяти за е.п. и оценивают полезность единицы мощности-времени 4. Для простоты рассматривается ситуация на одном промежутке.

В начале очереди заданий имеют вид $F = \{\varnothing\}$, $J = \{1, 2, 3, 4\}$, $D = \{\varnothing\}$. При распределении по жадному эвристическому алгоритму задание 1 выполняется на узле 1, задание 2 выполняется на узле 2, задания 3 и 4 в распределение не попадают. $W_0 = (5 - 1) \cdot 6 + (4 - 2) \cdot 5 = 34$.

При ценообразовании в соответствии с GVA на первом шаге алгоритма GA+GVA первое в очереди задание 1 оказывается неплатежеспособным и перемещается в D:

$$W_0 < W_{-1} = (4 - 1) \cdot 5 + (4 - 1) \cdot 5 + (4 - 2) \cdot 5 = 40,\ F = \{\varnothing\},\ J = \{2, 3, 4\},\ D = \{1\}.$$

Согласно алгоритму, распределение изменяется на $X := X_{-1}$, $W := W_{-1}$. Задания 2 и 3 выполняются на узле 1, задание 4 — на узле 2. Задание 1 не попадает в распределение. Переходим ко второму шагу алгоритма. Первое в очереди J задание 2 оказывается неплатежеспособным и перемещается в D:

$$W < W_{-2} = (4 - 1) \cdot 5 + (4 - 1) \cdot 5 + (5 - 2) \cdot 6 = 48,\ F = \{\varnothing\},\ J = \{3, 4\},\ D = \{2, 1\}.$$

Полагаем $X := X_{-2}$, $W := W_{-2}$. В новом распределении задания 3 и 4 выполняются на узле 1, задание 1 выполняется на узле 2, а задание 2 не попадает в распределение.

На следующих двух шагах GA+GVA рассматриваются задания 3 и 4. Они платежеспособны и платят суммы, равные их бюджетам:

$$p_3 = p_4 = 48 - 48 + 20 = 20,\ F = \{3, 4\},\ J = \{\varnothing\},\ D = \{1, 2\}.$$

Вполне логично, что заданиям 3 и 4 приходится платить по максимуму, ведь отсутствие одного из них не уменьшает суммарный излишек, так как есть задание 2 с такими же характеристиками, не попавшее в распределение.

Очередь J закончилась, присваиваем $J := \{1, 2\}$, $D := \{\varnothing\}$.

Следующее в очереди задание 1 платежеспособно, его платеж равен: $p_1 = 40 - 48 + 30 = 22$. После этого шага $F = \{3, 4, 1\}$, $J = \{2\}$, $D = \{\varnothing\}$.

И последнее в очереди задание 2 не входит в текущее распределение, его платеж $p_2 = 0$. После этого шага $F = \{3, 4, 1, 2\}$, $J = \{\varnothing\}$, $D = \{\varnothing\}$.

Все задания перешли в очередь F, работа алгоритма завершена.

В результате получили новое распределение, в котором задания 3 и 4 выполняются на узле 1, а задание 1 — на узле 2. При этом

$$W = 48,\ p_1 = 22,\ p_2 = 0,\ p_3 = p_4 = 20.$$

При использовании SNW-механизма получается следующий результат:

$$W = 34,\ p_1 = 24,\ p_2 = 20,\ p_3 = p_4 = 0.$$

4. Заключение

В работе рассматривается проблема построения экономического механизма для распределения вычислительных ресурсов в многопроцессорной системе. За основу взят SNW-механизм из [1], в котором вычислительная эффективность достигается в ущерб экономической эффективности. Цель работы — модифицировать SNW-механизм так, чтобы модифицированный механизм в любой исходной ситуации обеспечивал суммарный излишек не меньший, а в некоторых ситуациях — больший, чем при SNW-механизме.

Построен механизм распределения, сочетающий жадный эвристический алгоритм из [1] с ценообразованием в соответствии с обобщенным механизмом Викри и динамически изменяющий распределение при выявлении неплатежеспособного задания. Доказана конечность алго-

ритма. Показано, что этот алгоритм обеспечивает распределение с не меньшим, а в некоторых случаях — большим суммарным излишком, чем при SNW-механизме. Построена оценка трудоемкости алгоритма, которая совпадает с оценкой трудоемкости алгоритма поиска критического значения [1] и равна $O(|J|^2|N|)$.

Литература

1. Stößer J., Neumann D., Weinhardt C., et al. Market-based pricing in grids: On strategic manipulation and computational cost. European Journal of Operational Research (EJOR) 203(2). 464–475. 2009.
2. Carr N. The end of corporate computing. MIT Sloan Management Review 46 (3), 67. 2005.
3. Николенко С.И., Теория экономических механизмов. 2009.
4. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V. Algorithmic Game Theory. Cambridge Univercity Press, New York, NY, USA. 2007.
5. de Vries, S., Vohra, R., 2003. Combinatorial auctions: A survey. INFORMS Journal on Computing 15 (3), 284.
6. Parkes, D. Iterative combinatorial auctions: Achieving budget-balance with Vickrey-based payment schemes in combinatorial exchanges. 2001.
7. Hans Kellerer, Ulrich Pferschy, David Pisinger. Knapsack Problems. Springer Verlag. 2005.
8. Chekuri, C., Khanna, S. A PTAS for the multiple knapsack problem. SIAM Journal on Computing 35 (3), 713-728. 2006.
9. Varian, R. Economic Mechanism Design for Computerized Agents. 1995.

Математические и технологические проблемы распараллеливания крыловских итерационных процессов*

В.П. Ильин[1,2]

Институт вычислительной математики и математической геофизики СО РАН[1],
Новосибирский государственный университет[2]

Рассматриваются математические аспекты многообразных вычислительных технологий повышения эффективности методов распараллеливания итерационных процессов крыловского типа для решения больших разреженных несимметричных СЛАУ, возникающих при сеточных аппроксимациях многомерных краевых задач для систем дифференциальных уравнений. Характерным примером являются конечно-элементные приближения в газо-гидродинамических приложениях, где в каждом узле определены пять неизвестных функций, в силу чего СЛАУ имеет мелкоблочную структуру. Основой применяемых алгоритмов является гибкий метод обобщенных минимальных невязок FGMRES с динамическими предобуславливателями аддитивного типа, представляющий собой верхний уровень двухступенчатого итерационного алгоритма Шварца.
Повышение производительности алгебраических решателей достигается за счет применения различных подходов: декомпозиция расчетной области с различными топологиями, типами краевых условий на смежных границах и размерами пересечений подобластей, методы грубосеточной коррекции и агрегации, дефляции и неполной факторизации матриц. Описываются унифицированные формулировки используемых алгоритмов, а также вопросы их вычислительной эффективности и масштабируемого распараллеливания на суперкомпьютерах гетерогенной архитектуры. Приводятся примеры технологических требований к особенностям программных реализаций библиотек параллельных алгоритмов для решения СЛАУ.

1. Введение

В последние годы сформировалось бурно развивающееся направление вычислительной алгебры, связанное с решением больших разреженных систем линейных алгебраических уравнений (СЛАУ), возникающих из сеточных аппроксимаций (конечно-элементных или конечно-объемных, см. [1] и цитируемые там работы) многомерных краевых задач для систем дифференциальных уравнений в частных производных. Естественно, это связано с актуальными проблемами математического моделирования для сложных междисциплинарных приложений, включающих гидро-газодинамические процессы, динамику напряженно-деформированных состояний и др., на современных архитектурах суперкомпьютеров петафлопных масштабов. Само понятие большой задачи быстро эволюционирует, и сейчас необходимо говорить о решениях СЛАУ с порядками 10^{10} на многопроцессорных вычислительных системах (МВС) с числом ядер, или потоков, в десятки и сотни тысяч.

Повышение производительности вычислений, или быстродействия, в рассматриваемой прикладной сфере обуславливается двумя основными факторами. Первый – это конструирование итерационных процессов с высокой скоростью сходимости (прямые алгоритмы применимы только для относительно небольших размерностей задач), а второй – технологическое обеспечение масштабируемости распараллеливания, что означает сохранение эффективности с ростом порядков СЛАУ и/или количества вычислительных устройств. Ар-

*Работа поддержана грантом РФФИ №11-01-00205, а также грантами Президиума РАН №15.9-4 и ОМН РАН №1.3.3-4.

хитектура последних развивается главным образом количественно, а не качественно, и типовая структура гетерогенной МВС — это кластер с одинаковыми или почти одинаковыми вычислительными узлами, каждый из которых содержит несколько центральных процессорных многоядерных элементов (CPU) с общей памятью, а также несколько специализированных ускорителей, среди которых наибольшее распространение получили "графические процессорные элементы общего назначения" (GPGPU, или просто GPU) с очень большим числом ядер и сложной организацией внутренней иерархической оперативной памяти. Реализация параллельных вычислений на такого типа МВС осуществляется средствами систем MPI, OpenMP и CUDA, причем оптимизация программного кода фактически производится разработчиком в основном вручную, поскольку соответствующие средства автоматизации недостаточно функциональны.

Главными инструментами для решения возникающих здесь алгоритмических проблем являются методы декомпозиции областей [2–4] и итерационные процессы в подпространствах Крылова [1, 5]. Хотя эти направления уже стали классическими в вычислительной математике, они сейчас переживают период бурного развития, и буквально каждый год появляются новые интересные идеи. Рост эффективности алгебраических решателей достигается с помощью применения многообразных подходов: оптимизация размеров и топологий пересечений соседних подобластей, а также типов краевых условий на смежных границах, методы грубосеточной коррекции и агрегации, дефляции и неполной факторизации.

В п. 2 мы даем постановку и особенности рассматриваемых задач, а также сравнительный анализ методов их решения. Пункт 3 посвящен как общим, так и специальным технологическим аспектам распараллеливания алгоритмов, а в последнем пункте описываются примеры технологических требований к программным реализациям алгебраических решателей и, в частности, к библиотеке KRYLOV, предварительные сообщения о которой опубликованы в [6, 7].

2. Математические аспекты двухуровневых методов декомпозиции

Рассматривается следующая задача: пусть вещественная матрица СЛАУ

$$Au = f, \;\; A = \{a_{i,j}\} \in \mathcal{R}^{N,N}, \tag{1}$$

разбита на блочные строки $A = \{A_p \in \mathcal{R}^{N_p,N}, \;\; p = 1, ..., P\}, \;\; N_1 + ... + N_p = N$, с приблизительно равным числом строк $N_p \gg 1$ в каждой. Соответствующим образом также разбиваются векторы искомого решения и правой части $u = \{u_p\}, \;\; f = \{f_p\}$, так что система уравнений (1) может быть записана в форме совокупности P подсистем

$$A_{p,p}u_p + \sum_{\substack{q=1 \\ q \neq p}}^{P} A_{p,q}u_q = f_p, \;\; p = 1, ..., P, \tag{2}$$

где $A_{p,q} \in \mathcal{R}^{N_p,N_q}$ – прямоугольные матричные блоки, получаемые при разбиении каждой блочной строки (и всей матрицы A) на блочные столбцы.

Пусть матрица (1) задана в сжатом разреженном CSR-формате [8] с указанием числа строк N_p в каждой из P подсистем (2), в соответствии с разбиением матрицы A на блочные строки A_p. Естественно предполагается, что строки исходной матрицы пронумерованы подряд от 1 до N, так что номера строк каждого блока A_p меняются от $1 + \sum\limits_{i=1}^{p-1} N_i$ до $\sum\limits_{i=1}^{p} N_i$ (эти номера назовем глобальными).

Предполагается, что СЛАУ (1) представляет собой систему сеточных уравнений, так что каждая компонента векторов u, f соответствует узлу сетки, общее число которых в

расчетной сеточной области $\Omega^h = \bigcup\limits_{p=1}^{P} \Omega_p$ равно N. При этом блочное разбиение матриц и векторов соответствует разбиению (декомпозиции) Ω^h на P сеточных непересекающихся подобластей Ω_p, в каждой из которых находится N_p узлов.

Описанная декомпозиция Ω^h не использует узлов – разделителей, т.е. условные границы подобластей не проходят через узлы сетки. Формальности ради можно считать, что внешность расчетной области есть неограниченная подобласть Ω_0 с числом узлов $N_0 = 0$.

Сеточное уравнение для i-го узла сетки может быть записано в виде

$$a_{i,i}u_i + \sum_{\substack{j\in\omega_i \\ j\neq i}} a_{i,j}u_j = f_i, \;\; i \in \Omega^h, \tag{3}$$

где через ω_i обозначается сеточный шаблон i-го узла, т.е. совокупность номеров всех узлов, участвующих в i-м уравнении.

Будем считать, что сеточные узлы и соответствующие переменные пронумерованы следующим образом: сначала идут подряд все узлы 1-й подобласти Ω_1 (неважно в каком внутреннем порядке), затем – узлы второй подобласти и т.д. Отметим, что взаимнооднозначного соответствия алгебраической и геометрической (сеточной) интерпретации структуры СЛАУ может и не существовать. Типичный пример – одному узлу сетки может соответствовать $m > 1$ переменных в алгебраической системе. Если для каждого узла такие кратные переменные нумеруются подряд, то структура СЛАУ приобретает мелкоблочный ($m \ll N$) вид (в (3) u_i и f_i означают не числа, а подвекторы порядка m соответственно, $a_{i,j} \in \mathcal{R}^{m,m}$) и на таких случаях мы остановимся в последующем особо.

Для заданного блочного разбиения СЛАУ, или декомпозиции сеточной расчетной области без пересечения подобластей, можно построить хорошо распараллеливаемый аддитивный метод Шварца, представимый итерационным процессом “по подобластям”:

$$A_{p,p}u_p^n = f_p - \sum_{\substack{q=1 \\ q\neq p}}^{P} A_{p,q}u_q^{n-1} \equiv g_p^{n-1}. \tag{4}$$

Здесь n будем называть номером внешней итерации, которая в действительности реализуется более утонченным образом, чему будет посвящен специальный раздел.

В целом алгоритм решения крупноблочной СЛАУ является двухуровневым и нижний, или внутренний, уровень заключается в решении (прямым или итерационным методом) для каждого n – независимо и параллельно – подсистем вида (4) с матрицами $A_{p,p}$.

При вычисленных правых частях g_p^{n-1} нахождение всех u_p^n требует для каждой подобласти решения подсистем с порядками N_p, которые на каждой n-й итерации могут выполняться независимо и одновременно на разных процессорах. В данном случае перевычисление g_p^{n-1} фактически означает использование новых значений u_p^{n-1} для соседних подобластей в качестве граничного условия 1-го рода (Дирихле) для околограничных внутренних узлов из Ω_p.

Известно, что скорость сходимости итераций метода Шварца (4) возрастет, если декомпозицию расчетной области сделать с пересечением подобластей.

В силу этого мы дадим определение расширенной сеточной подобласти $\bar{\Omega}_p \supset \Omega_p$, имеющей пересечения с соседними подобластями, величину которых мы будем определять в терминах количества околограничных сеточных слоев, или фронтов. Обозначим через $\Gamma_p^0 \in \Omega_p$ множество внутренних околограничных узлов из Ω_p, т.е. таких узлов $P_i \in \Omega_p$, у которых один из соседей не лежит в $\Omega_p (P_j \notin \Omega_p, \;\; j \in \omega_i, \;\; j \neq i))$. Обозначим далее через Γ_p^1 множество узлов, соседних с узлами из Γ_p^0, но не принадлежащих Ω_p, через Γ_p^2 – множество узлов, соседних с узлами из Γ_p^1, но не принадлежащих объединению $\Gamma_p^1 \bigcup \Omega_p$, через Γ_p^3 – множество узлов, соседних с Γ_p^2, но не принадлежащих $\Gamma_p^2 \bigcup \Gamma_p^1 \bigcup \Omega_p$, и т.д. Соответственно

эти множества назовем первым внешним слоем (фронтом) узлов, вторым слоем, третьим и т.д. Получаемое объединение узлов

$$\bar{\Omega}_p \equiv \Omega_p \bigcup \Gamma_p^1 ... \bigcup \Gamma_p^\Delta$$

будем называть расширенной p-й сеточной подобластью, а целую величину Δ определяем как величину расширения, или пересечения (в терминах количества сеточных слоев). Случай $\Delta = 0$ фактически означает декомпозицию области Ω^h на подобласти без пересечений ($\Omega_p^0 = \Omega_p$).

На формальном алгебраическом языке каждой расширенной подобласти можно сопоставить подсистему уравнений

$$(\bar{A}_{p,p} + \theta \bar{D}_p)\bar{u}_p = \bar{f}_p - \sum_{\substack{q=1 \\ q \neq p}}^{P} \bar{A}_{p,q}\bar{u}_q + \theta \bar{D}_p \bar{u}_p \tag{5}$$

где $\bar{D}_p$ – диагональная матрица, определяемая соотношением

$$\bar{D}_p e = \sum_{\substack{q=1 \\ q \neq p}}^{P} \bar{A}_{p,q} e, \;\; e = (1, ..., 1)^T \in \mathcal{R}^{\bar{N}_p}.$$

Здесь размерность векторов $\bar{u}_p^n$ и $\bar{f}_p$ равна числу узлов $\bar{N}_p$ в расширенной подобласти $\bar{A}_{p,p}, \bar{D}_p \in \mathcal{R}^{\bar{N}_p, \bar{N}_p}$, а введенный параметр θ формально позволяет рассматривать зависящие от $\bar{u}_p$ и $\bar{u}_q$ правые части (5) как аппроксимации краевых условий на смежных границах подобластей: $\theta = 0$ соответствует условию 1-го рода (Дирихле), $\theta = 1$ – условию 2-го рода (Неймана), а $\theta \in (0, 1)$ – условию 3-го рода (Робена), [9].

Переходя теперь к полному вектору u и вводя блочно-диагональные матрицы

$$B_p = block\text{-}diag\{\bar{A}_{p,p} + \theta \bar{D}_p\}, \tag{6}$$

из (5) получаем итерационный процесс вида

$$u^n = u^{n-1} + B^{-1}(f - Au^{n-1}) = u^{n-1} + B^{-1}r^{n-1}, \tag{7}$$

где B – предобуславливающая матрица, определяемая следующим образом.

Введем матрицу $W_p = (w_1, ..., w_{\bar{N}_p})^T \in \mathcal{R}^{\bar{N}_p, N}$ "сужения" вектора $u \in \mathcal{R}^N$ в пространство $\mathcal{R}^{\bar{N}_p}$ соответствующей подобласти, для чего компоненты каждой вектор-строки $w_k = \{w_{k,i}\}$ полагаются равными единице, если $i \in \bar{\Omega}_p$, и нулю в противном случае. При этом W_p^T является матрицей оператора продолжения пространства $\bar{\Omega}_p$ в Ω, и в итоге предобуславливатель аддитивного метода Шварца (additive Swartz) принимает вид

$$B^{-1} = B_{AS}^{-1} = \sum_{p=1}^{P} \bar{B}_p^{-1}, \;\; \bar{B}_p^{-1} = W_p^T B_p^{-1} W_p. \tag{8}$$

На алгебраическом языке формулы (7) при $B = B_{AS}$ определяют стационарный блочный метод Якоби (BJ), на каждом шаге которого надо обращать матрицу B_p, что фактически означает одновременное (параллельное) решение расширенных СЛАУ в подобластях. Если эти процедуры осуществляются с помощью "внутренних" итерационных процессов, то это неизбежно приводит к тому, что в (7) на каждом n-м шаге реально используется переменное (динамическое) предобуславливание с матрицами B_n.

Кардинальный подход к ускорению итераций заключается в переходе от формул (7) для BJ к алгоритмам в предобусловленных подпространствах Крылова $\mathcal{K}_n(v^0, v^1, ..., v^{n-1})$, где

введены обозначения $v^0 = B_0^{-1} r^0, v^k = B_k^{-1} A v^{k-1}$. Один из возможных путей здесь заключается в применении гибкого (flexible) метода обобщенных минимальных невязок FGMRES [5], который на каждом n-м шаге минимизирует норму $||r^n||$, но требует для своей реализации оперативную память объемом $2nN$. Использование FGMRES с рестартами через каждые m итераций при больших n позволяет значительно сокращать объем требуемой памяти до $2mN$, но существенно замедляет скорость сходимости.

Главный, и практически единственный, путь повышения эффективности алгоритмов — это развитие крыловских методов или за счет совершенствования предобуславливателей, или путем улучшения итерационных подпространств (добавление новых базисных векторов, в частности, что тоже может интерпретироваться как введение дополнительного предобуславливания).

Одним из резервов ускорения является грубосеточная коррекция (CGC, от coarse grid correction), или агрегирование, а также идейно примыкающие сюда алгебраические многосеточные методы (AMG). Суть здесь заключается в том, что на каждом шаге стандартного BJ эволюция последовательных приближений от каждой области передается только ее непосредственным соседям. Идея исправления данной ситуации — формирование и решение дополнительных относительно небольших СЛАУ, которые соединяли бы все подобласти и передавали бы, пусть грубо, глобальные итерационные возмущения.

Реализация данного подхода заключается в конструировании дополнительного предобуславливателя и внешне аналогична (8). Мы формируем новый оператор сужения с матрицей $W_c \in \mathcal{R}^{N_c,N}$, $N_c \ll N$, в которой каждая строка содержит только по несколько ненулевых элементов (обычно равных единице), соответствующих одной из подобластей. Транспонированная матрица W_c^T будет представлять тогда оператор продолжения, грубосеточный предобуславливатель принимает вид

$$B_c^{-1} = W_c^T A_c^{-1} W_c, \;\; A_c = W_c A W_c^T \in \mathcal{R}^{N_c,N_c}, \tag{9}$$

и полный предобуславливатель определяется аддитивным образом:

$$B^{-1} = B_{AS}^{-1} + B_c^{-1}. \tag{10}$$

Отметим, что реализация CGC на каждой n-й итерации требует дополнительного решения СЛАУ с матрицей невысокого порядка A_c, что можно сделать с помощью прямого решателя, например, PARDISO из библиотеки MKL INTEL [8].

Развитие идей грубосеточной коррекции активно продолжается в различных направлениях, см. [10, 11] и цитируемые там работы. Многосеточные методы используют технологии интерполяции на каждом из последовательных этапов дискретизации. Адаптивные алгоритмы сглаженного агрегирования основаны на взвешенных усреднениях различных приближений. Методы типа FETI применяют декомпозицию области с явным выделением разделителей сеточных подмножеств (макрограни, макроребра, вершины) и построение дополнений Шура на иерархическом принципе.

Опишем еще один подход к ускорению крыловских процессов — метод дефляции, который имеет широкое распространение в разных версиях. Мы его представим в применении к выбору начального приближения, если СЛАУ с одинаковой матрицей решается многократно с разными правыми частями. Именно такая ситуация возникает в двухуровневых методах декомпозиции областей. Пусть в результате предыдущего решения СЛАУ с помощью какого-то крыловского метода вычислены A-ортогональные векторы $(w_1, ..., w_m) = W_d$, составляющие базис пространства, которое будем называть дефляционным. Для решения новой системы выбираем начальное приближение u^0 таким, чтобы соответствующая начальная невязка r^0 и начальный направляющий вектор p^0 удовлетворяли условиям ортогональности

$$W_d^T r^0 = 0, \;\; W_d^T A p^0 = 0. \tag{11}$$

Допустим, что u^{-1} есть "предварительный" начальный вектор, а $r^{-1} = f - Au^{-1}$ – соответствующая невязка. Тогда условия (11) будут выполняться, если положить

$$\begin{aligned} &u^0 = u^{-1} + W_d A_d^{-1} W_d^T r^{-1}, \;\; r^0 = f - Au^0, \\ &p^0 = [I - W_d A_d^{-1}(AW_d^T)]r^0, \;\; A_d = W_d^T A W_d. \end{aligned} \tag{12}$$

В рамках одной статьи невозможно дать содержательный обзор современных тенденций в развитии предобусловленных итерационных процессов крыловского типа. Можно только констатировать, что оригинальные подходы появляются практически непрерывно и данный момент следует рассматривать как важный технологический фактор при создании математического и программного обеспечения, ориентированного на длительный жизненный цикл.

3. Технологические вопросы реализации параллельных итерационных алгоритмов

Рассматриваемые в данном пункте вопросы — это конкретизация общей проблемы "отображение алгоритмов на архитектуру МВС". В качестве типовой модели вычислительной системы может служить НКС-30Т – гетерогенный кластер ИВМиМГ СО РАН [12]. Такая МВС формально представляет набор вычислительных узлов, которые с помощью коммуникационной сети обеспечиваются связями "каждый с каждым", управляемыми с помощью программных средств системы передачи сообщений MPI [13]. Каждый узел имеет свою многоуровневую память (большую оперативную и поменьше – сверхбыстрый кэш, тоже имеющий внутренние уровни), общую для расположенных в нем многоядерных центральных процессорных устройств (CPU, в НКС-30Т их два, каждый с четырьмя или шестью ядрами), а также графические процессорные устройства с очень большим числом ядер (GPU, в НКС-30Т их три, с 512 ядрами в каждом). На каждом CPU (и даже на их отдельных ядрах) можно формировать вычислительный MPI-процесс, внутри которого распараллеливание реализуется средствами системы OpenMP [14] над общей памятью (более конкретно – организуется несколько вычислительных потоков). На одном GPU допускается запуск только одного MPI-процесса, причем внутри него программирование осуществляется на языке CUDA [15] с достаточно сложными средствами управления внутренней иерархической памятью. Особенностью (и недостатком) GPU является относительно медленные коммуникации с памятью CPU. В целом средства MPI обеспечивают организацию синхронных или асинхронных вычислений, в том числе при совмещении их во времени с передачей данных от процессора к процессору.

Как видно из приведенного описания, построение даже грубой математической модели вычислительного процесса на такой архитектуре, с целью его оптимизации и оценок коммуникационных потерь, не представляется возможным. Поэтому соответствующие исследования являются сугубо экспериментальными, а основной инструмент в данном случае — это метод проб и ошибок.

Справедливости ради следует сказать, что математик-программист работает не с пустыми руками, а при наличии достаточного богатого вычислительного инструментария (главным образом библиотеки BLAS и SPARSE BLAS [8], содержащие основные алгебраические операции с векторами и матрицами, в том числе разреженными), созданного профессионалами с помощью экономичных языковых средств низшего уровня. В частности, можно упомянуть библиотеку CUSP [16] для решения на GPU разреженных СЛАУ с использованием средств BLAS.

По поводу вопросов реализации параллельных итерационных алгоритмов мы отметим две проблемы, относящиеся к области вычислительных технологий. Первая из них касается формирования вспомогательных СЛАУ для подобластей, которые необходимо решать на нижнем уровне двухуровневых итерационных методов декомпозиции областей. В силу

существующего для большинства случаев изоморфизма сеточных шаблонов и портретов матричных строк анализ декомпозиции может проводиться одинаковым образом как в терминах сеточных графов, так и на языке матричных графов, в силу чего употребляются названия “алгебраическая декомпозиция” и “геометрическая декомпозиция”. Более целесообразным и естественным выглядит реализация декомпозиции на этапе построения сетки, когда можно оперировать информацией о геометрических объектах расчетной области, топологическими связями и расстояниями, и т.д. Однако на практике зачастую при использовании библиотеки алгебраических решателей декомпозиция не задается, и в таких случаях фактически требуется построить по каким-то критериям блочную структуру СЛАУ, пользуясь только матричным CSR-форматом. Эта задача имеет достаточно высокую информационную и логическую сложность, особенно если ее требуется решить в параллельном, т.е. распределенном по различным процессорам, режиме.

Формирование пересекающихся сеточных подобластей и соответствующих расширенных СЛАУ осуществляется в два этапа. На первом определяются сбалансированные (с примерно равным числом узлов) подобласти без перехлеста, для чего могут использоваться инструментарии типа популярной библиотеки METIS ([17], в том числе существующей в параллельном варианте) осуществляющей операции над графами. При этом подобласти каким-либо образом упорядочиваются (это соответствует разбиению матрицы на блочные строки), и в соответствии с этим нумеруются узлы сетки (или матричные строки): сначала идут все узлы (строки) из первой подобласти (блочной строки), затем – из второй, и т.д. На втором этапе производится постепенное расширение подобластей по слоям соседних узлов, или фронтов: сначала к внутренним узлам подобласти присоединяются ближайшие внешние узлы, являющиеся непосредственными соседями к внутренним и которые образуют 1-й слой расширения, затем к ним аналогично присоединяются узлы 2-го слоя, и т.д. до заданного априори количества фронтов расширения.

В матричной технологии это осуществляется только на основе глобального CSR-формата, а результатом являются локальные матричные CSR-форматы для каждой из расширенной подобластей (при этом, естественно, требуется перенумерация матричных строк и столбцов соответствующих ненулевых элементов из глобальной упорядоченности в локальную). Если матрицы расширенных СЛАУ в подобластях формируются только один раз, то их правые части необходимо пересчитывать в околограничных узлах на каждой внешней итерации. Для этого требуется определять совокупности множеств узлов-доноров и узлов-акцепторов, которые передают информацию от своей подобласти к соседним и, наоборот, принимают данные.

Второй требующий решения технологический вопрос — как и в каких пространствах осуществлять внешний итерационный процесс? Наиболее прямой и естественный путь заключается в реализации крыловского метода на основе формулы (7), где u^n и f полные векторы с размерностью, равной порядку исходной СЛАУ (1), т.е. $O(h^{-3})$ в трехмерном случае, где h есть характерный шаг сетки. В этом случае метод FGMRES исполняется в кластерном варианте с помощью P MPI-процессов, что требует, в частности, дополнительных обменов при вычислении скалярных произведений векторов.

Однако существует и альтернативный подход, состоящий в проведении внешних итераций в пространстве следов, т.е. для векторов, определенных только на смежных границах пересекающихся подобластей [9]. В этом случае исходная СЛАУ формально редуцируется путем исключения неизвестных, соответствующих внутренним узлам подобластей, и итоговая размерность системы понижается на порядок, т.е. до $O(h^{-2})$. При этом реализацию внешнего FGMRES можно осуществлять только на одном процессоре, используя распараллеливание ограниченными средствами OpenMP, но полностью избегая при этом коммуникационных потерь. Однако такая вычислительная технология имеет существенный недостаток — неизбежный дефицит оперативной памяти одного процессора при решении больших СЛАУ с масштабируемым параллелизмом, требующим формирования очень большого ко-

личества подобластей.

Что касается не вычислительных, а программных и информационных технологий ускорения параллельных процессов для рассматриваемых классов задач и алгоритмов, то здесь также имеются значительные резервы за счет оптимизации кода, организации развертки циклов, выбора режимов компиляции, профессионального использования командных возможностей систем MPI и OpenMP, и т.д. Конечный результат здесь в существенной степени зависит от искусства математика-программиста в синхронизации массовых расчетных и обменных операций, имея целью добиться максимум возможного в условиях ограниченных рамок маневрирования вычислительными потоками на фиксированных функциональных характеристиках аппаратных устройств МВС. Но так или иначе, "человеческий фактор" в существующих условиях еще играет далеко не последнюю роль.

4. Примеры технических требований к библиотекам алгебраических решателей

Программного обеспечения по вычислительной алгебре в мире существует достаточно много и давно, как самостоятельного, так и в составе пакетов прикладных программ, как коммерческого, так и общедоступного. Однако библиотек по итерационным параллельным алгоритмам решения больших разреженных СЛАУ имеется не так уж много. Помимо уже упоминавшейся библиотеки CUSP для GPU, можно назвать такие разработки, как Hypre [18], PETSc [19] и Saad Software [20]. Высокопроизводительные продукты для решения сверхбольших СЛАУ на суперкомпьютерах петафлопного масштаба еще ждут своего часа, так что мы рассмотрим некоторые технические требования к библиотеке алгебраических решателей, руководствуясь в первую очередь практическими требованиями, возникающими из актуальных проблем математического моделирования. Более того, мы будем исходить из целевой постановки о создании программного обеспечения, ориентированного на широкого круга пользователей и рассчитанного на длительный жизненный цикл с регулярным развитием функционального наполнения, а также адаптирующегося к эволюции компьютерных архитектур.

а. *Состав решаемых задач и алгоритмов.* Обладая определенным багажом знаний и технологий, или ноу-хау, заманчиво на единых принципах охватить СЛАУ различных типов: вещественных и комплексных, эрмитовых и неэрмитовых, положительно определенных и знаконеопределенных и т.д. Такую систематизацию можно продолжить до выделения специальных систем, для которых применимы сверхбыстрые алгоритмы.

   Основа современных итерационных процессов — это предобусловленные методы в крыловских подпространствах. Однако в этих обоих направлениях имеется огромное разнообразие вариантов, и выше мы только коротко остановились на двухуровневых методах декомпозиции областей.

б. *Принципы организации интерфейса.* Традиционный вопрос в данном случае такой — делать ли алгебраический решатель в форме "черного ящика" или, наоборот, в качестве полностью открытого пользователю и настраиваемого на конкретные задачи инструмента? Возможны, конечно, и различные компромиссные "серые" варианты. Целесообразные решения, естественно, зависят от того, на какого типа пользователя рассчитан программный продукт. Наиболее реальной является ситуация, когда решатель нужен не в автономном варианте, а встраиваемый в уже существующий ППП.

   Традиционная форма универсальной программной реализации итерационного метода крыловского типа такова: широкое использование BLAS-овских функций для векторных операций, а также открытость для внешних процедур умножения вектора на матрицу исходной СЛАУ и на предобуславливатель. Такое абстрагирование изначально

закладывается в концепцию объектно-ориентированного программирования, например, в языке C++. Однако хорошо известно, что чрезмерный универсализм — это враг эффективности и экономичности.

Отсюда возникает немаловажный вопрос — как встраивать в широко-форматную библиотеку специальные сверхбыстрые алгоритмы для частного вида СЛАУ, которые не укладываются в общую вышеприведенную схему? Например, расчетная область или даже одна из подобластей могут допускать разделение переменных и, как следствие, применение быстрого преобразования Фурье (БПФ), которого очень жалко было бы запрещать к использованию.

в. *Проблема переиспользования программ.* Разработанное компьютерным сообществом прикладное программное обеспечение представляет огромный интеллектуальный потенциал, и умение его использовать дает заведомое преимущество разработчикам программного продукта. В рассматриваемых нами обстоятельствах это означает, в первую очередь, наличие первоклассных программ, реализующих прямые методы, которые можно и нужно использовать для решения СЛАУ в подобластях.

г. *Внутренняя структура и организация библиотеки.* Один из возможных способов построения библиотеки алгебраических решателей — это создание каталогизированного и систематизированного хранилища большого количества алгоритмов, из которого специальными конфигурационными средствами может формироваться конкретная версия продукта для определенных условий эксплуатации. Более примитивный подход — конкретная программа просто вынимается из хранилища (переписывается в файл) и отторгается. Третий, и наиболее продвинутый, способ существования прикладной библиотеки состоит в создании баз данных для типовых СЛАУ, для результатов их решения различными алгоритмами, а также в организации тренинга и обучения потенциальных пользователей.

Список технологических и примыкающих организационных вопросов по созданию библиотеки параллельных алгоритмов решения больших разреженных СЛАУ можно было бы значительно продолжить (многоязычность и многоверсионность, платформонезависимость, внутренние средства развития и адаптируемости, сопровождение, документируемость, лицензионность и т.п.). Частично эти вопросы были решены разработчиками библиотеки KRYLOV, которая находится на стадии опытной эксплуатации в ИВМиМГ СО РАН в авторском сопровождении.

Режим исполнения в библиотеке KRYLOV можно назвать полуавтоматическим, или в стиле “серого ящика”. Внешний итерационный процесс по входному заданию пользователя может выполняться или на корневом процессоре в пространстве следов, или в распределенном варианте. Для решения внутренних СЛАУ в подобластях допустимо использование как авторских реализаций различных крыловских процессов, так и прямой алгоритм PARDISO, а также ряд решателей из библиотеки CUSP NVIDIA. В распоряжении пользователя – различные методы декомпозиции с параметризованными размерами пересечений подобластей, типами итерируемых краевых условий на смежных границах, количество задействованных подобластей и соответствующих MPI-процессов, число вычислительных потоков на CPU, а также различные счетные параметры для возможного управления скоростью сходимости итераций. Предусмотрены также режимы выборы параметров по умолчанию без вмешательства пользователя.

Список литературы

1. Ильин В.П. Методы и технологии конечных элементов. Новосибирск: изд. ИВМиМГ СО РАН, 2007.

2. Лебедев В.И., Агошков В.И. Вариационные алгоритмы метода разделения области. М., 1983 (Препр. ОВМ РАН; N 54).

3. Bramble J.H., Pasciak J.E., Wang J., Xu J. Convergence estimates for product iterative methods with applications to domain decomposition // Math. Comp. 1991. Vol. 57, N. 195. P. 1–21.

4. Ильин В.П. Параллельные методы и технологии декомпозиции областей // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. Т. 46, N. 305. С. 31–44.

5. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, 2003.

6. Бутюгин Д.С., Ильин В.П., Ицкович Е.А и др. Krylov: библиотека алгоритмов и программ для решения СЛАУ // Современные проблемы математического моделирования. Математическое моделирование, численные методы и комплексы программ. Сборник трудов Всероссийских научных молодёжных школ. Ростов-на-Дону: Изд-во Южного федерального университета. 2009. С. 110–128.

7. Бутюгин Д.С., Ильин В.П., Перевозкин Д.В. Методы параллельного решения СЛАУ на системах с распределенной памятью в библиотеке Krylov // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. Т. 47, N. 306. С. 5–19.

8. Intel Math Kernel Library. Reference Manual: URL: http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mklman/index.htm (дата обращения: 15.02.2013).

9. Ильин В.П., Кныш Д.В. Параллельные методы декомпозиции в пространствах следов // Вычислительные методы и программирование. 2011. Т. 12, N. 1. С. 100–109.

10. Brezina M., Vanek P., Vassilevsky P.S. An improved convergence analysis of smoothed aggregation algebraic multigrid // Numer. Linear Algebra Appl. 2012. Vol. 19. P. 441–469.

11. Farhat C., Lesoinne M., LeTollei P., Pierson K., Rixen D. FETI-DP: A dual-primal unified FETI method. Part I: A faster alternative to the two-level FETI method // Int. J. Numer. Math. Engrg. 2001. Vol. 50. P. 1523–1544.

12. Кластер НКС-30Т: URL: http://www2.sscc.ru/HKC-30T/HKC-30T.htm (дата обращения: 15.02.2013).

13. Message Passing Interface at Open Directory Project: URL: http://www.dmoz.org/Computers/Parallel_Computing/Programming/Libraries/MPI/ (дата обращения: 15.02.2013).

14. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. Новосибирск: НГТУ, 2006.

15. CUDA Tools & Ecosystem: URL: https://developer.nvidia.com/cuda-tools-ecosystem (дата обращения: 15.02.2013).

16. Bell N., Garland M. CUSP: Generic parallel algorithms for sparse matrix and graph computations: URL: http://cusp-library.googlecode.com (дата обращения: 15.11.2012).

17. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM J. Sci. Comp. 1999. Vol. 20, N. 1. P. 359–392.

18. Hypre: URL: http://acts.nersc.gov/hypre/ (дата обращения: 15.02.2013).

19. PETSc: Home Page: URL: http://www.mcs.anl.gov/petsc/ (дата обращения: 15.02.2013).

20. Yousef Saad – SOFTWARE: URL: http://www-users.cs.umn.edu/ saad/software/ (дата обращения: 15.02.2013).

Клеточно-автоматное моделирование физико-химических процессов нано уровня на графических ускорителях*

К.В. Калгин

Институт вычислительной математики и математической геофизики СО РАН

Моделирование каталитических физико-химических процессов на нано уровне кинетическим методом Монте-Карло требует достаточно больших вычислительных ресурсов. Создание "точных" параллельных алгоритмов моделирования таких процессов на графических ускорителях проблематично. В работе исследуются возможности применения блочно-синхронных режимов теории клеточных автоматов для моделирования кинетическим методом Монте-Карло. Предлагаются новые блочно-синхронные режимы, реализуемые на графическом ускорителе с достаточно высокой эффективностью, но имеющие бóльшую точность воспроизведения динамики оригинальных моделей.

Введение

Большое число уже существующих моделей физико-химических процессов нано уровня, в основе которых лежит кинетический метод Монте-Карло (см., например, [1, 2]), в которых обычно используются регулярные решетки и локальные правила изменения характеристик частиц материала, представляются в форме асинхронного клеточного автомата.

Асинхронное клеточно-автоматное (АКА) моделирование реальных процессов требует больших вычислительных мощностей, поскольку для выявления каких-либо свойств моделируемых процессов необходимо оперировать большими количествами частиц (10^{10}-10^{12}) в течение длительного времени (10^3-10^5 итераций). Естественно возникает потребность использовать параллельные вычислители для ускорения вычислений, к этому также располагают основные свойства клеточного автомата – естественная мелкозернистая параллельность и локальность взаимодействий при функционировании. Однако, асинхронные клеточные автоматы не поддаются столь же легкому распараллеливанию, как и обычные, синхронные, клеточные автоматы.

Это связано с тем, что при распараллеливании необходимо соблюдать следующие свойства: корректность и эффективность распараллеливания, а также равноправие в выборе клеток. Первые два свойства типичны для большинства параллельных программ. Последнее свойство возникает из определения асинхронного режима работы клеточного автомата и заключается в следующем: вероятность выбора некоторой клетки на некотором шаге должна быть равна вероятности выбора любой другой клетки на любом другом шаге.

Существующие параллельные алгоритмы можно разделить на два класса: не нарушающие свойства равноправия в выборе клеток [2, 3] (сохраняющие асинхронизм), и нарушающие его [4, 5] (нарушающие асинхронизм). Алгоритмы первого класса можно назвать "точными", они пригодны для исполнения только на мультипроцессорах и мультикомпьютерах, но не на графических ускорителях.

В работах [4, 5, 6] на трёх моделях показано, что применение алгоритмов, нарушающих асинхронизм, не вносит существенных изменений в моделируемый процесс. В основе этих алгоритмов лежат блочно-синхронные режимы работы клеточного автомата. В данной работе на примере модели ZGB (Ziff-Gulari-Barshad) [7] физико-химического процесса

*Исследование выполнено по Программе фундаментальных исследований Президиума РАН, проект 15-9; поддержано Междисциплинарным Интеграционным проектом СО РАН 47 и РФФИ в рамках научных проектов 11-01-00567а и 12-01-31455;

окисления углекислого газа на каталитической поверхности показывается, что использование блочно-синхронных режимов может существенно изменить моделируемый процесс. Вопрос о том, для каких моделей использование алгоритмов, нарушающих асинхронизм, допустимо, то есть не вносит изменения в моделируемый процесс, остаётся открытым.

Далее в разделе 1 даётся формальное определение клеточного автомата; в разделе 2 даются определения различных режимов, вводится новый расширенный блочно-синхронный режим; в разделе 3 описывается классическая модель ZGB [7] протекания химических реакций на поверхности катализатора; в разделе 4 исследуется статистическое отклонение эволюции при блочно-синхронных режимах от эволюции при асинхронном режиме для модели ZGB, показывается преимущество новых расширенных блочно-синхронных режимов; в разделе 5 описывается реализация и полученное ускорение на современных графических ускорителях.

1. Определение клеточного автомата

Асинхронная вероятностная КА модель описывается четвёркой

$$\text{КА} = \langle X, A, \Theta, \Upsilon \rangle,$$

X – множество координат клеток. Наиболее употребимый вариант множества X – прямоугольная решётка

$$X = \{(i,j) \,|\, i \in \{0,1,2,\dots N_I - 1\},\, j \in \{0,1,2,\dots,N_J - 1\}\}\,. \tag{1}$$

A – алфавит, множество состояний клеток. Клетка x есть пара $x = (\mathbf{x}, a)$, где $\mathbf{x} \in X$ называется координатой клетки, и $a \in A$ – её состоянием. Клеточный массив Ω есть множество клеток, $\Omega = \{(\mathbf{x}, a)\} \subset X \times A$, где ни одна пара клеток не имеет одинаковой координаты и $\{\mathbf{x} \mid (\mathbf{x}, a) \in \Omega\} = X$.

Поскольку между множеством координат клеток и множеством клеток в клеточном массиве существует взаимно однозначное соответствие, далее мы будем отождествлять клетку и её координаты.

Θ – локальный оператор перехода, далее просто *оператор*. Оператор Θ есть вероятностное отображение вида

$$\Theta : A^{|T|} \times A^{|T|} \xrightarrow{p} A^{|T|}, \tag{2}$$

где *шаблон* T есть список *именующих функций* $\phi : X \to X$, $T = \{\phi_1, \phi_2, \dots, \phi_{|T|}\}$. Шаблон T определяет соседство клетки $\mathbf{x}$: $T(\mathbf{x}) = \{\phi_1(\mathbf{x}), \dots, \phi_{|T|}(\mathbf{x})\}$. В КА рассматриваемого класса следующие шаблоны наиболее используемы:

$$T_{13}(\mathbf{x}) = \{\mathbf{x} + \mathbf{v_0}, \mathbf{x} + \mathbf{v_1}, \dots, \mathbf{x} + \mathbf{v_{12}}\}, \tag{3}$$

$$T_{9}(\mathbf{x}) = \{\mathbf{x} + \mathbf{v_0}, \mathbf{x} + \mathbf{v_1}, \dots, \mathbf{x} + \mathbf{v_8}\}, \tag{4}$$

где $V = \{\mathbf{v_0}, \mathbf{v_1}, \dots, \mathbf{v_{12}}\} = \{(0,0), (0,1), (1,0), (0,-1), (-1,0), (1,1), (1,-1), (-1,1), (-1,-1), (0,2), (2,0), (0,-2), (-2,0)\}$.

Применением оператора Θ к клетке $\mathbf{x}$ называется обновление состояний клеток соседства $T(\mathbf{x})$ состояниями $\Theta(H(\mathbf{x}))$, где $H(\mathbf{x})$ есть список состояний клеток соседства $T(\mathbf{x})$.

Υ – режим работы, который определяет порядок применения оператора к клеткам:

$$\Upsilon \in \{\sigma,\, \alpha,\, \beta,\, \gamma\}\,,$$

где σ означает синхронный режим, α – асинхронный, β – блочно-синхронный, и γ – новый расширенный блочно-синхронный режим. Первые КА модели [8] формулировались в синхронном виде, то есть использовался синхронный режим. Основной режим

работы рассматриваемого в данной работе класса КА моделей – асинхронный. Блочно синхронные режимы используется в качестве вспомогательных в параллельных реализациях, в том числе на графических ускорителях. Далее подробно определяются режимы работы и эволюция клеточного автомата.

2. Эволюция клеточного автомата

В ходе моделирования происходит изменение клеточного массива Ω. Процесс моделирования разбивается на итерации. *Эволюция клеточного автомата* есть последовательность $\Omega^* = \{\Omega_0, \Omega_1, \Omega_2, \ldots\}$, где Ω_0 – начальный клеточный массив, а Ω_t – клеточный массив после исполнения t-ой итерации.

Режим работы Υ задаёт порядок и количество применений оператора Θ к клеткам. Далее рассмотрим три режима наиболее часто используемых в КА моделях физико химических процессов: синхронный (σ), асинхронный (α) и блочно-синхронный (β и γ).

Определение. В синхронном режиме σ итерация есть синхронное, одновременное применение оператора Θ ко всем клеткам клеточного массива.

Определение. В асинхронном режиме α итерация состоит из $|X|$ последовательных шагов, на каждом шаге оператор Θ применяется к случайно выбранной клетке. На каждом шаге клетка выбирается независимо от номера итерации, номера шага, клеточного массива и предыдущих выбранных клеток. Разбиение процесса моделирования в асинхронном режиме на итерации, которые состоят из $|X|$ шагов, является условным, и вводится по аналогии с синхронным режимом.

Определяемый далее блочно-синхронный режим содержит как синхронную компоненту, что позволяет эффективно исполнять его на параллельных вычислителях, так и асинхронную, что позволяет в некоторой степени имитировать работу асинхронного режима. Именно поэтому в работах [4, 5] он вводится исключительно как вспомогательный, для эффективного и простого исполнения АКА моделей на параллельных вычислителях. В работах [4, 5, 6] на нескольких АКА моделях физико-химических процессов показано, что эволюции, полученные при асинхронном и блочно-синхронном режимах, не отличимы по анализируемым характеристикам. Полученные в этих статьях результаты не могут служить доказательством какой бы то ни было эквивалентности эволюции КА при асинхронном и блочно-синхронном режимах. Кроме того автором были найдены “контрпримеры” показывающие, что для некоторых КА моделей использование блочно-синхронного режима вместо асинхронного приводит к существенным изменениям в эволюции КА.

Определение. [4, 5] В блочно-синхронном режиме β итерация состоит из w стадий. На каждой стадии оператор Θ синхронно применяется ко всем клеткам случайно выбранного подмножества S_i, $S_i \subset X$, из семейства $S = \{S_1, S_2, \ldots, S_n\}$, причем $\forall k \ \ |S_k| = \frac{|X|}{w}$. Для обеспечения свойств корректности [4] синхронного применения, равноправности выбора клеток и эффективного параллельного исполнения [10] на семейство S накладываются следующие условия:

$$\bigcup_i S_i = X, \tag{5}$$

$$\forall i \neq j \ \ S_i \cap S_j = \emptyset, \tag{6}$$

а также условия с участием дополнительного шаблона T, покрывающего шаблон T_Θ оператора Θ, $T_\Theta \subseteq T$:

$$\forall S_i \bigcup_{\mathbf{x} \in S_i} T(\mathbf{x}) = X, \tag{7}$$

$$\forall S_i \ \forall \mathbf{x}_1, \mathbf{x}_2 \in S_i \ : \ T(\mathbf{x}_1) \cap T(\mathbf{x}_2) = \emptyset. \tag{8}$$

Увеличивая размер шаблона T мы приближаемся к асинхронному режиму: при условии, что $\forall \mathbf{x} \in X \ T(\mathbf{x}) = X$, блочно-синхронный и асинхронный режимы эквивалентны, $\text{КА}_\beta = \text{КА}_\alpha$.

Определение. В расширенном блочно-синхронном режиме γ итерация определяется так же, как и в блочно-синхронном режиме за следующим исключением. Условия (5) и (6), которые говорят, что S есть *одно* из возможных разбиений X, заменяются на следующее: S есть объединение *всех* разбиений X, для которых выполняются условия (7) и (8).

Таким образом, расширенный блочно-синхронный режим обладает свойствами корректности синхронного применения, за которое отвечает условие (8), а также эффективного параллельного исполнения, поскольку множество координат клеток покрывается шаблоном T плотно (7). Кроме того, увеличивается размер семейства S, что делает расширенный блочно-синхронный режим более похожим на асинхронный, по сравнению с блочно-синхронным. Тем не менее, условие равноправия при выборе клеток, как и в блочно-синхронном режиме, нарушается, что может привести к статистическим смещениям в моделируемом процессе. Исследование вносимых статистических смещений исследуется далее.

Перечислим элементы семейств S^9 и S^{13} (см. также Рис. 1 и 2) построенных с помощью шаблонов T_9 и T_{13}, соответственно:

$$S^{9,1}_{i,r_0\ldots r_{N_Y/3}} = \{(x,y) \,:\, y \equiv i\,(mod\,3)\ x \equiv r_{\lfloor y/3 \rfloor}(mod\,3)\}, \tag{9}$$

$$S^{9,2}_{i,r_0\ldots r_{N_X/3}} = \{(x,y) \,:\, x \equiv i\,(mod\,3)\ y \equiv r_{\lfloor x/3 \rfloor}(mod\,3)\}, \tag{10}$$

$$S^9 = \left(\bigcup_{i,r_0,\ldots r_{N_X/3}} S^{9,1}_{i,r_0,\ldots,r_{N_X/3}} \right) \cup \left(\bigcup_{i,r_0,\ldots r_{N_Y/3}} S^{9,2}_{i,r_0,\ldots,r_{N_Y/3}} \right), \tag{11}$$

$$S^{13,1}_i = \{(x,y) \,:\, x + 5y \equiv i\,(mod\,13)\}, \tag{12}$$

$$S^{13,2}_i = \{(x,y) \,:\, x + 8y \equiv i\,(mod\,13)\}, \tag{13}$$

$$S^{13} = \bigcup_i (S^{13,1}_i \cup S^{13,2}_i). \tag{14}$$

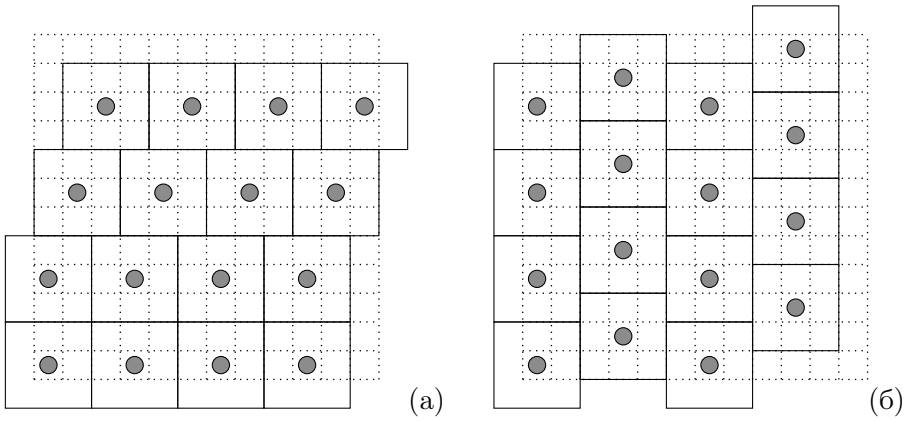


Рис. 1. Серыми кружками выделены клетки, координаты которых принадлежат множествам $S^{9,1}_{0,0,1,2,\ldots}$(а) и $\dot{S}^{9,2}_{0,1,0,2,\ldots}$(б).

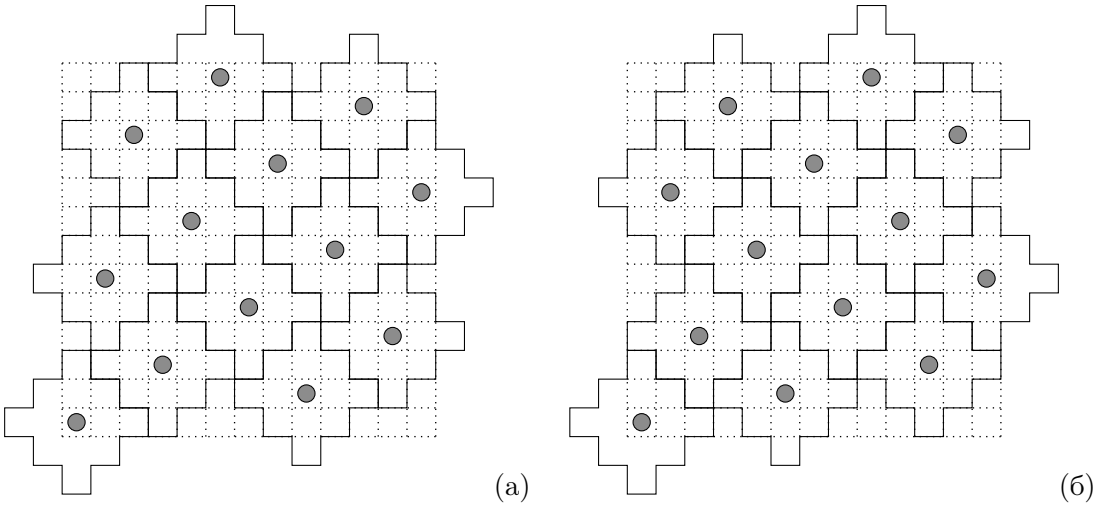


Рис. 2. Серыми кружками выделены клетки, координаты которых принадлежат множествам $S_0^{13,1}$(а) и $S_0^{13,2}$(б).

Таким образом, полностью определено семейство S расширенного блочно-синхронного КА для наиболее часто используемых шаблонов T_N, где $N \in \{9, 13\}$. Далее также используются блочно-синхронные режимы с семействами S, построенными с помощью шаблонов T_{25}(квадрат 5×5) и T_{49}(квадрат 7×7). Семейства S^{25} и S^{49} устроены аналогично семейству S^9, построенного с помощью шаблона T_9(квадрат 3×3).

3. Модель ZGB

Для проведения вычислительных экспериментов используется известная АКА модель ZGB (Ziff-Gulari-Barshad) [7] физико-химического процесса окисления углекислого газа на каталитической поверхности. Множество координат есть прямоугольная решётка (1). Алфавит $A = \{\emptyset, CO, O\}$, где состояние $\emptyset$ означает пустой активный центр катализатора, а состояния CO и O – нахождение соответствующих молекул в активном центре катализатора.

Шаблон T_Θ оператора Θ равен $T_\Theta = T_{13}$. Во время применения оператора Θ в начале случайным образом выбирается элементарный процесс с вероятностью, соответствующей его скорости протекания: процесс адсорбции CO, адсорбции O_2 и реакции $CO + O = CO_2$. Затем выполняются действия по изменению состояний клеток из соседства T_{13} в соответствии с выбранным элементарным процессом. Эти действия подробно описаны в работе [7]. Стоит заметить лишь то, что большую часть кода реализующего оператор перехода составляют условные переходы, присваивания и целочисленные операции.

В данной работе в качестве дополнительного элементарного процесса вводится диффузия для определения её влияния на степень отличия эволюций при различных режимах работы. Диффузия вводится таким же способом, как это делалось в более поздних работах по моделированию поверхностных каталитических реакций методом Монте-Карло (см., например, [1]).

4. Различие в поведении асинхронного и блочно-синхронных режимов

В работах [4, 5, 6] на трёх физико-химических моделях показывается, что изменение режима с асинхронного α на блочно-синхронный β не вносит существенных изменений в общую картину эволюции КА. Как это будет показано далее, не для всех моделей изменение режима не изменяет эволюцию КА.

Для анализа степени подобия эволюций клеточных автоматов KA_{α}, KA_{β_N} и KA_{γ_N} как стохастического объекта далее используется следующий подход: по эволюциям Ω^*_{α}, $\Omega^*_{\beta_N}$ и $\Omega^*_{\gamma_N}$ вычисляются характеристики $f\left(\Omega^*_{\alpha}\right)$, $f\left(\Omega^*_{\beta_N}\right)$ и $f\left(\Omega^*_{\gamma_N}\right)$, которые являются случайными величинами, после чего сравниваются их математические ожидания. Далее математическое ожидание характеристики для различных режимов будем обозначать через $m_{\Upsilon}\left(f\right)$, где $\Upsilon \in \{\alpha, \beta_N, \gamma_N\}$. Численную оценку величины m_{Υ} далее будем обозначать через m'_{Υ}. Для численной оценки m_{Υ} генерируется такое количество эволюций Ω^*_{Υ}, что $m_{\Upsilon} \in \left(m'_{\Upsilon} - 0.001,\, m'_{\Upsilon} + 0.001\right)$ с вероятностью больше 0.99.

Наиболее часто употребима следующая характеристика

$$f_{t,a}(\Omega^*) = \frac{|\{(\mathbf{x}, a) \in \Omega_t\}|}{N_I \cdot N_J}, \tag{15}$$

то есть $f_{t,a}(\Omega^*)$ равно доле клеток с состоянием a в клеточном массиве после исполнения t итераций. С помощью этой характеристики зачастую очень трудно отличить эволюции с различными режимами.

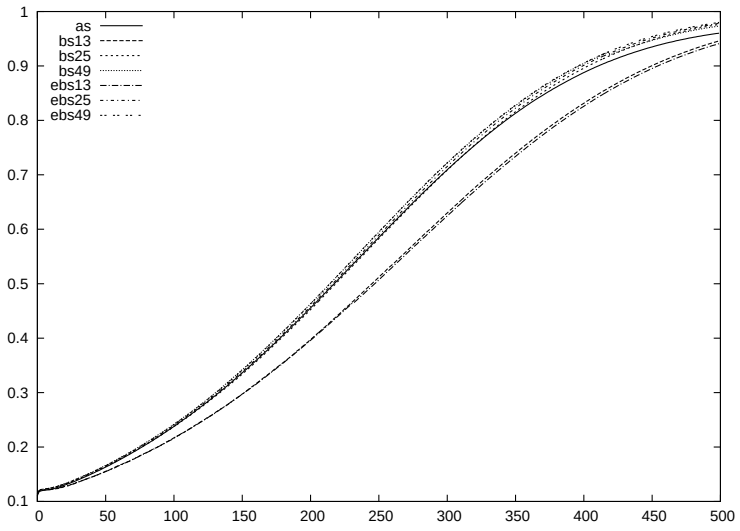


Рис. 3. Значение характеристики $f_{t,CO}$ для модели ZGB без диффузии для первых 500 итераций, $t < 500$. Графики as, bs13, bs25, bs49, ebs13, ebs25 и ebs49 соответствуют значению характеристики $f_{t,CO}$ при режимах $\alpha, \beta_{13}, \beta_{25}, \beta_{49}, \gamma_{13}, \gamma_{25}$ и γ_{49}.

На Рис. 3 представлено значение характеристики $f_{t,CO}$ для модели ZGB без диффузии. Видно, что часть графики, соответствующие режимам β_{13} и β_{25}, существенно откланяются от графика, соответствующего асинхронному режиму. При введении в модель диффузии, отличить графики невозможно.

В данной работе предлагается использовать следующую более информативную характеристику эволюции клеточного автомата

$$f_{t,\Delta\mathbf{x}}(\Omega^*) = \frac{|\{(\mathbf{x}, a) \in \Omega_t \,:\, (\mathbf{x} + \Delta\mathbf{x},\, a) \in \Omega_t\}|}{N_I \cdot N_J}, \tag{16}$$

то есть $f_{t,\Delta\mathbf{x}}(\Omega^*)$ равно доле пар клеток с координатами $\mathbf{x}$ и $\mathbf{x}+\Delta\mathbf{x}$ в одинаковых состояниях в клеточном массиве после исполнения t итераций. Предлагаемая характеристика $f_{t,\Delta\mathbf{x}}$ содержит в себе информацию не только о количестве, но и о пространственном расположении состояний, в отличие от часто используемой характеристики $f_{t,a}$.

В Табл. 1 представлены результаты численной оценки математического ожидания среднеквадратичных отклонений

$$g_{t,\Upsilon} = \sqrt{\frac{1}{26 \cdot 51} \sum_{\Delta \mathbf{x}} \left(m_\alpha \left(f_{t,\Delta \mathbf{x}} \right) - m_\Upsilon \left(f_{t,\Delta \mathbf{x}} \right) \right)^2} \tag{17}$$

при $k_{diff} \in \{0, 1\}$. Для численной оценки $g_{t,\Upsilon}$, обозначаемой как $g^{'}_{t,\Upsilon}$, генерируется такое количество эволюций Ω^*_α и Ω^*_Υ, что $g_{t,\Upsilon} \in \left(g^{'}_{t,\Upsilon} - 0.001,\ g^{'}_{t,\Upsilon} + 0.001 \right)$ с вероятностью больше 0.99.

	$\Upsilon = \beta_{13}$	$\Upsilon = \beta_{25}$	$\Upsilon = \beta_{49}$	$\Upsilon = \gamma_{13}$	$\Upsilon = \gamma_{25}$	$\Upsilon = \gamma_{49}$
$k_{diff} = 0$	0.0067	0.0039	0.0026	0.0054	0.001	0.0008
$k_{diff} = 1$	0.0028	0.0021	0.0015	0.0019	0.0006	0.0004

Таблица 1. Численные оценки математического ожидания среднеквадратичных отклонений $g_{t,\Upsilon}$ при $t = 100$ и варьирующихся Υ и коэффициенте k_{diff}. Значение $k_{diff} = 0$ соответствует оригинальной модели ZGB, значение $k_{diff} = 1$ – модели ZGB с введённой диффузией (скорость протекания процесса диффузии в этом случае равна сумме скоростей остальных элементарных процессов).

Перечислим основные результаты, которые можно наблюдать на представленном Рис. 3 и в Табл. 1:

1. Эволюции $\Omega^*_{\beta_{13}}$ и $\Omega^*_{\gamma_{13}}$ существенно отличаются от Ω^*_α по обеим характеристикам $f_{t,a}$ (15) и $f_{t,\Delta \mathbf{x}}$ (16)при $k_{diff} = 0$;
2. Эволюции Ω^*_α, $\Omega^*_{\beta_{25}}$, $\Omega^*_{\beta_{49}}$, $\Omega^*_{\gamma_{25}}$ и $\Omega^*_{\gamma_{49}}$ слабо различимы с точки зрения первой характеристики $f_{t,a}$ (15) при $k_{diff} = 0$;
3. Эволюции Ω^*_α, $\Omega^*_{\beta_{13}}$, $\Omega^*_{\beta_{25}}$, $\Omega^*_{\beta_{49}}$, $\Omega^*_{\gamma_{13}}$, $\Omega^*_{\gamma_{25}}$ и $\Omega^*_{\gamma_{49}}$ практически не различимы с точки зрения первой характеристики $f_{t,a}$ (15) при $k_{diff} = k_{adsCO} + k_{adsO_2}$;
4. Эволюции $\Omega^*_{\gamma_{25}}$ и $\Omega^*_{\gamma_{49}}$ практически неотличимы от эволюции Ω^*_α по всем представленным характеристикам, включая $g_{t,\Upsilon}$ (17), при всех значениях k_{diff}.

Перечисленные результаты, наблюдаемые автором и на многих других моделях, обобщаются в следующие два очень важных утверждения:

1. КА модели с диффузией существенно лучше поддаются моделированию блочно- синхронными режимами;
2. Предложенные автором расширенные блочно-синхронные режимы расширяют возможности эффективного параллельного исполнения КА моделей.

5. Реализация на CUDA

CUDA [9] расшифровывается как Compute Unified Device Architecture (унифицированная архитектура вычислительных устройств), описывает архитектуру аппаратной части и программного обеспечения. CUDA создана компанией Nvidia в 2007 году, позволяет разрабатывать программы общего назначения для графических ускорителей. Для тестирования использовались графические ускорители Nvidia GTX 280 и Nvidia GTX 680 (Kepler), их характеристики представлены в Таб. 2.

Стадии итерации блочно-синхронных режимов на графическом ускорителе исполняются последовательно, поскольку информационно зависимы между собой. Применение оператора Θ к клеткам выбранного на текущей стадии подмножества S_i может быть выполненно

	GTX 280	GTX 680
Тактовая частота (ГГц)	1.3	1.1
Мультипроцессоров (шт.)	30	8
Потоковых процессоров (шт.)	240	1536
Глобальная память (Мбайт)	768	4096
Разделяемая память (Кбайт)	16	48
Текстурный кэш (байт)	8192	8192
Кэш (Кбайт)	–	48
Регистры (шт.)	16384	32768
Планируемое число потоков на мультипроцессоре	1024	1536

Таблица 2. Параметры используемых графических ускорителей.

параллельно, согласно условию непересечения окрестностей различных клеток, входящих в S_i (8).

Параллельный алгоритм. Программа, исполняемая на центральном процессоре, используя генератор псевдо случайных чисел выбирает S_i из семейства S, после чего запускает функцию на графическом ускорителе, исполняющую одну стадию. Номер подмножества i передаётся на графический ускоритель через аргумент запускаемой функции. Число порождаемых потоков на графическом ускорителе равно числу клеток, к которым применяется оператор Θ на текущей стадии, $|S_i|$. По номеру подмножества i и своему номеру каждый поток в соответствии с формулами (9-14) вычисляет координаты клетки, к которой этот поток будет применять оператор Θ. После применения оператора к клетке поток завершает свою работу. После завершения всех порождённых потоков, с центрального процессора запускается следующая стадия.

Согласно рекомендациям разработчиков CUDA [9], для достижения высокой производительности необходимо порождать потоков в несколько раз больше, чем имеется ядер на графическом ускорителе. Данное условие будет выполняться на современных графических ускорителях для размеров клеточной области больше 700×700. Именно для таких размеров и требуется распараллеливание.

В Таб. 3 представлены результаты тестирования реализации описанного алгоритма на современных графических ускорителях при шаблоне $T = T_{13}$. Использование других шаблонов T_{25} и T_{49} влияет на время исполнения незначительно.

	1000×1000	2000×2000	8000×8000
Ускорение GTX 280/Core i7	$\times 25$	$\times 31$	$\times 35$
Ускорение GTX 680/Core i7	$\times 48$	$\times 60$	$\times 65$

Таблица 3. Результаты тестирования при размерах клеточной области в 1000×1000, 2000×2000 и 8000×8000 клеток на графических ускорителях GTX 280 и GTX 680. Показано ускорение по отношению ко времени работы последовательной программы на одном ядре процессора Core i7.

По результатам приведённым в Табл. 3 и работе [11] можно сказать, что один рядовой графический ускоритель GTX 280 на тестируемой задаче по производительности заменяет мощную 32-ядерную вычислительную систему над общей памятью (4 процессора Intel Xeon X7560 2.2Ghz, была доступна автору в рамках проекта Intel Manycore Testing Lab), а новый графический ускоритель GTX 680 заменит мощную 64-ядерную вычислительную систему (8 процессоров Intel Xeon X7560 2.2 Ghz, один узел в кластере Новосибирского Государственного Университета [12]).

Заключение

В работе показана принципиальная возможность использования графических ускорителей для моделирования физико-химических процессов на нано уровне кинетическими методами Монте-Карло. Предложены новые расширенные блочно-синхронные режимы, обеспечивающие существенно меньшее статистическое отклонение от “эталонного” асинхронного режима и имеющие ту же, что и блочно-синхронный режим, эффективность параллельного исполнения.

Работу следует продолжить в направлении (а) теоретического обоснования возможности использования расширенных блочно-синхронных режимов и (б) накопления статистики для других физико-химических моделей в основе которых лежит кинетический метод Монте-Карло.

Литература

1. Elokhin, V.I., Latkin, E.I., Matveev, A.V., and Gorodetskii, V.V. Application of Statistical Lattice Models to the Analysis of Oscillatory and Autowave Processes on the Reaction of Carbon Monoxide Oxidation over Platinum and Palladium Surfaces.
2. Overeinder B. J., Sloot P. M. A. Extensions to TimeWarp Parallel Simulation for Spatial Decomposed Applications // Proceedings of the Fourth United Kingdom Simulation Society Conference (UKSim 99) / Ed. by D. Al-Dabass, R. Cheng. Cambridge, UK: 1999. P. 67-73.
3. Lubachevsky B. Efficient parallel simulation of asynchronous cellular arrays // Complex Systems. 1987. Vol. 1. P. 1099-1123.
4. O.L. Bandman. Parallel Simulation of Asynchronous Cellular Automata Evolution. ACRI 2006, LNCS 4173, pp.41-47, 2006.
5. S. V. Nedea, J. J. Lukkien, P. A. J. Hilbers, A. P. J. Jansen. Methods for Parallel Simulations of Surface Reactions. Proceedings of the 17th Inter- national Symposium on Parallel and Distributed Processing, 2003.
6. Anastasia Sharifulina, Vladimir Elokhin: Simulation of Heterogeneous Catalytic Reaction by Asynchronous Cellular Automata on Multicomputer. PaCT 2011: 204-209
7. Ziff R.M., Gulari E., Barshad Y. "Kinetic phase transitions in an irreversible surface-reaction model". Phys Rev Lett 56 (24): 2553–56, 1986.
8. von Neumann J. Theory of self reproducing automata. - University of Illinois Urbana. USA. 1966
9. NVIDIA CUDA Programming Guide. http://www.nvidia.com/object/cuda_get.html
10. K. V. Kalgin. Comparative Study of Parallel Algorithms for Asynchronous Cellular Automata Simulation on Different Computer Architectures. ACRI-2010, LNCS-6350, pp. 399-408, 2010.
11. К. В. Калгин. Параллельная реализация асинхронных клеточных автоматов на 32-ядерной вычислительной системе. Сиб. журн. вычисл. матем., 15:1 (2012), 55-65.
12. Вычислительный кластер Новосибирского Государственного Университета, http://www.nusc.ru.

Исследование эффективности глобальной параллельной оптимизации функций многих переменных

А.Н. Коварцев, Д.А. Попова-Коварцева, П.В. Аболмасов

Самарский государственный аэрокосмический университет
им. академика С.П. Королёва (национальный исследовательский университет)

Рассматривается параллельный алгоритм глобальной оптимизации функций многих переменных модифицированным методом половинных делений, основанный на использовании локальной техники. Основное внимание уделяется рациональной организации вычислений за счет распределения вычислительных ресурсов между фазами глобальной и локальной оптимизации. При разработке алгоритмов использовалось визуальное средство автоматизации программирования параллельных вычислений PGRAPH. Представлены результаты численного тестирования предложенного алгоритма.

1. Введение

Проблема разработки эффективных методов решения задачи глобальной оптимизации функций многих переменных является чрезвычайно важной для науки и техники. В этой области накоплен значительный опыт и разработано большое количество алгоритмов и методов решения задач многоэкстремальной оптимизации. В то же время, преодоление принципиальной трудности задач глобальной оптимизации, связанной с экспоненциальным ростом их сложности в зависимости от размерности оптимизируемой функции, остаётся открытой проблемой, во всяком случае, для точных методов оптимизации.

Современные точные методы глобальной оптимизации совершенствуются, главным образом, для класса Липшицевых функций в двух направлениях: первое связано с разработкой новых критериев отсева неперспективных областей оптимизируемой функции [1], второе - с повышением эффективности методов редукции задачи многомерной оптимизации [2]. В работе [3] на примере метода неравномерных покрытий и его модификаций была предпринята попытка качественного анализа эффективности методов решения задачи глобальной оптимизации. В частности получена следующая оценка средней сложности задачи глобальной оптимизации: $\widetilde{N}(n) = \beta \cdot \left(\frac{1}{4\varepsilon} \right)^{n-\mu} - 1$, где $\beta = \frac{(\alpha+1)2^n - 2}{\alpha 2^n - 1}$, α $(\alpha > 1/2^n)$ – доля неотбракованных параллелепипедов ($(1-\alpha)$ параллелепипедов отбраковываются с помощью критериев отсева), $\mu = \lfloor \log_2 \alpha \rfloor$, ε - заданная точность решения задачи глобальной оптимизации. При $\alpha = 1$, когда критерии отсева параллелепипедов не работают, алгоритмы глобальной оптимизации имеют самую низкую производительность и практически «не работают», начиная с $n > 5$. Для предельно возможной эффективности правил отсева ($\alpha \approx 1/2^{n-1}$, $\mu \approx n-1$) практическое применение прямых методов глобальной оптимизации ограничено ($n < 20$). Не лучше складывается ситуация для методов редукции задачи оптимизации. В [3] показано, что в этом случае справедлива следующая оценка средней сложности задачи глобальной оптимизации: $\widetilde{N}(n) = \left(\beta \cdot \left(\frac{1}{4\varepsilon} \right)^{1-\mu} - 1 \right)^n$, т.е. сохраняется экспоненциальный характер роста этого показателя. В то же время, для реальных значений α ($\alpha \approx 0.4...0.8$) средняя сложность алгоритмов глобальной оптимизации для метода редукции на несколько порядков меньше, чем для методов, не использующих редукцию.

В последнее время решение проблемы высокой вычислительной сложности задач глобальной оптимизации связывают с использованием методов параллельных и распределенных вычислений [6]. Однако, использование этой относительно «новой» техники вычислений на прак-

тике позволяет получить как существенное ускорение алгоритма глобальной оптимизации, так и замедление его характеристик. Как для любого «нового дела», процесс распараллеливания алгоритмов глобальной оптимизации имеет свои «подводные камни», особенности и специфические трудности. В данной работе на примере модифицированного метода половинных делений [4] рассматриваются некоторые особенности организации параллельных алгоритмов глобальной оптимизации.

2. Постановка задачи и основные определения

Рассмотрим задачу безусловной глобальной оптимизации непрерывной функции $f : R^n \to R$, заданной на допустимом множестве $X \in R^n$ в следующей постановке:

$$f_* = glob\min_{x \in X} f(x) = f(x_*) . \quad (1)$$

Положим, что глобальный минимум x_* принадлежит множеству X_*, причем $X_* \subset X$, а $X = \overset{n}{\underset{i=1}{\otimes}} [0,1]$, является многомерным единичным гиперкубом. Предлагаемый далее алгоритм глобальной оптимизации является модификацией известного метода половинных делений (метода неравномерных покрытий) [1], основные положения которого заключаются в следующем.

Определим множество ε - решений задачи (1) как

$$X_*^{\varepsilon} = \{ x \in X : f(x) \le f_* + \varepsilon \} . \quad (2)$$

Нахождение приближенного решения задачи (1) заключается в поиске хотя бы одной точки из множества X_*^{ε}.

В методе неравномерных покрытий строится последовательность $\{X_i\} = \{X_1,...,X_k\}$ подмножеств множества X и точек $x_1,...,x_k$ ($x_i \in X_i$) (для метода половинных делений в качестве X_i обычно рассматриваются параллелепипеды). В каждой точке x_i вычисляется значение функции $f(x)$ и определяется её рекордное значение f_r:

$$f_r = \min_{1 \le i \le k} f(x_i) = f(x_r), \quad 1 \le r \le k . \quad (3)$$

На каждом из подмножеств X_i вводятся *миноранты*, т.е. такие функции $G_i(x)$, что $\forall x \in X_i \quad f(x) \ge G_i(x)$. Тогда последовательности подмножеств $\{X_i\}$ можно поставить в соответствие совокупность покрывающих подмножеств $\{Z_i\} = Z_1,..., Z_k$, определяемых из условий

$$Z_i = \{ x \in X_i : G_i(x) \ge f_r - \varepsilon \}, \; r \le i . \quad (4)$$

Множество Z_i строится таким образом, что $\forall x \in Z_i$ выполняется неравенство $f(x) \ge f_r - \varepsilon$, из чего следует, что глобальный минимум функции $f(x)$ на множестве Z_i не может быть меньше рекорда f_r более чем на ε. Таким образом, в процессе поиска глобального минимума подмножество Z_i можно исключить из рассмотрения и продолжить оптимизацию на множестве $X_i \setminus Z_i$. Алгоритм останавливается, когда допустимое множество оказывается покрытым подмножествами Z_i, т.е.

$$X \subseteq Z_1 \cup Z_2 \cup ... \cup Z_k . \quad (5)$$

Алгоритм неравномерных покрытий гарантирует нахождение ε-оптимального решения, что подтверждается соответствующей теоремой.

Предлагаемая модификация метода половинных делений связана с введением понятия *области притяжения локального минимума* и использованием локальной техники. Пусть $x_i^*, i = 1,.., m$ – локальные минимумы функции $f(x)$. Тогда область притяжения локального минимума x_i^* определим как множество точек начальных приближений алгоритма локальной оптимизации, из которых алгоритм сходится к x_i^*:

$$S_i = \{ x \in X : \ arg\ min f(x) = x_i^* \}. \quad (6)$$

Поскольку построение области притяжений в смысле (6) достаточно сложная задача, далее будем использовать следующее упрощенное определение области притяжения локального минимума:

$$S_i = \{ x \in X : \ || x - x_i^* || < \rho_i, \quad \rho_i > 0 \}, \quad (7)$$

где ρ_i – радиус области притяжения локального минимума.

2.1 Использование локальной техники

Для модифицированного алгоритма половинных делений сохраняется экспоненциальный рост сложности задачи оптимизации в зависимости от числа переменных. Одной из эффективных стратегий совершенствования алгоритмов глобальной оптимизации (ГО) является использование *локальной техники*, когда стратегия глобального поиска удачно сочетается с методами локальной оптимизации (ЛО).

При ограниченном количестве минимумов функции $f(x)$ области притяжения S_i имеют значительные размеры. Предположим, что относительно оптимизируемой функции известно количество локальных минимумов – r, включая глобальный, и размеры областей притяжения: $\rho_1,...,\rho_r$. Введем понятие «гарантированного радиуса» области притяжения минимумов функции, под которым будем понимать $\rho_m = min\ \rho_i, \quad 1 < i \le r$.

В этом случае, как показано в работе [3], среднюю сложность алгоритма глобальной оптимизации можно оценить величиной

$$\widetilde{N}(n) = 2\left(1/4\rho_m\right)^n - 1 + r\, log_2(\rho_m / \varepsilon). \quad (8)$$

С учетом (8) при $\rho_m >> \varepsilon$ этап глобальной оптимизации, связанный с поиском областей притяжения локальных минимумов функции, можно производить достаточно грубо, а следовательно, с меньшими затратами времени.

2.2 Двухфазный алгоритм метода половинных делений (ДАМПД)

Дополнительная информация о размерах областей притяжения локальных минимумов функции позволяет построить двухфазную схему алгоритма глобальной оптимизации, в которой выделим фазы глобальной и локальной оптимизации (см. рис. 1).

В ДАМПД фаза глобальной оптимизации реализуется с помощью модифицированного алгоритма, с той лишь разницей, что двоичное деление параллелепипедов реализуется до достижения заданных размеров их радиусов R_i. Величина R_i определяется размером «гарантированного радиуса» ρ_m областей притяжения минимумов функции. На рисунке 1 пунктирными линиями отмечены гарантированные области притяжения минимумов функции. Заштрихованные прямоугольники являются прямоугольниками, «отбракованными» с использованием константы Липшица. В итоге, исходя из условия $R_i \le \rho_m$, фаза глобальной оптимизации завершилась, породив, для приведенного на рисунке 2 примера, 24 прямоугольника заданного радиуса.

В фазе локальной оптимизации из точек, принадлежащих областям притяжения локальных минимумов, организуется поиск минимумов функции с помощью локальных алгоритмов оптимизации.

Такая схема организации процедуры поиска глобального минимума функции существенно сокращает трудоемкость фазы глобальной оптимизации.

В работе [4] исследована сходимость двухфазного алгоритма метода половинных делений, где показано, что для дважды дифференцируемой Липшицевой функции, вторая производная которой также удовлетворяет условию Липшица, ДАМПД обеспечивает нахождение ε-оптимального решения.

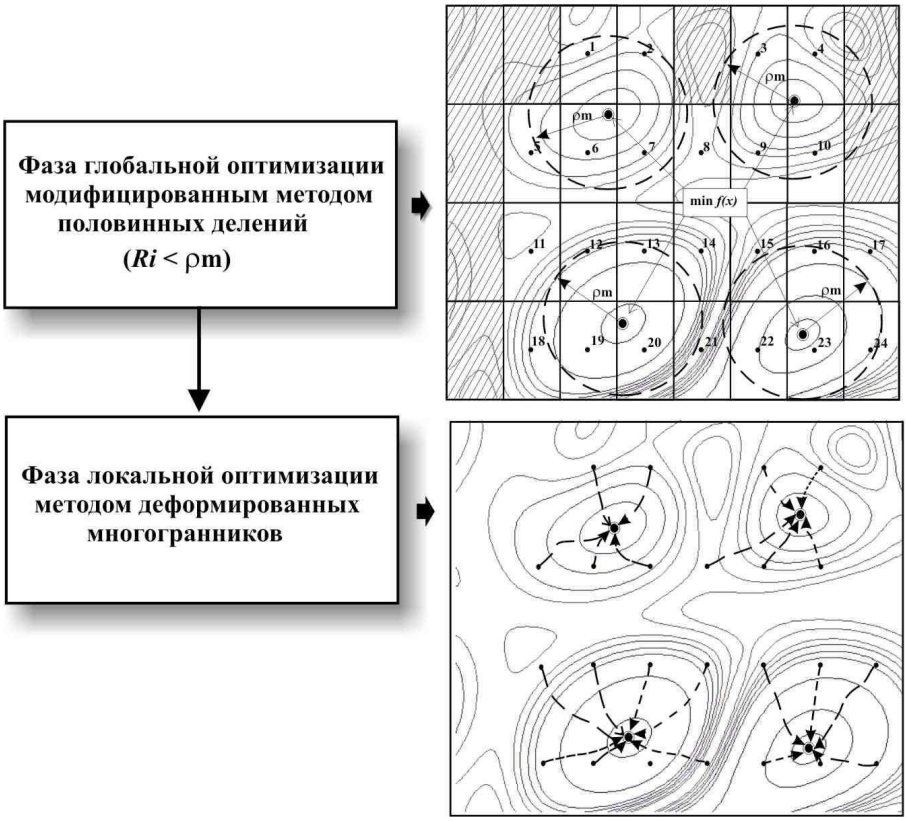


Рис. 1. Двухфазный алгоритм метода половинных делений

3. Алгоритм формирования множества точек начальных приближений для ДАМПД

Для больших размерностей вектора независимых переменных оптимизируемой функции в фазе глобальной оптимизации порождается значительное количество параллелепипедов заданного размера. В этом случае объем вычислений в фазе локальной оптимизации становится чрезмерно большим. Рассмотрим следующий адаптивный алгоритм формирования списка точек начальных приближений областей притяжения локальных минимумов оптимизируемой функции, осуществляющий сжатие количества точек начальных приближений, используемых во второй фазе алгоритма.

Будем считать, что область притяжения S_i определена, если она содержит, по крайней мере, одну точку из множества $C = \{c_1,...,c_k\}$, где c_k - центры параллелепипедов.

Пусть r – количество зон притяжения минимума функции; ρ_m – гарантированный радиус области притяжения минимума функции, обеспечивающий нахождение всех локальных минимумов.

Сформируем список областей притяжения локальных минимумов $V = \{V_1,V_2,...,V_m\}$, элементами которого являются структуры $V_i = (\widetilde{c}_i, \widetilde{f}_i)$, где $\widetilde{c}_i$ - координаты «представителя» i-й области притяжения, имеющего наилучшую, достигнутую для этой области, оценку $\widetilde{f}_i$. Вектор $\widetilde{c}_i$ условно считается центром i-ой области притяжения.

Первоначально список V пуст. По мере вычислений функции он наполняется элементами, но в конце этапа глобального поиска не может содержать больше m элементов (m – заданный размер списка, $m \geq r$). Размер списка m зависит от свойств оптимизируемой функции и выбирается из соображений попадания в него представителя области притяжения глобального минимума функции. Элементы списка V упорядочены таким образом, что $\widetilde{f}_1 < \widetilde{f}_1 < ... < \widetilde{f}_m$.

Пусть в процессе работы алгоритма глобальной оптимизации произведено очередное испытание $f_k = f(c_k)$. Эволюция содержимого списка V происходит по следующим простым правилам:

1. Проверяется принадлежность центра очередного параллелепипеда c_k окрестностям одной из имеющихся областей притяжения $V_i \quad i = 1,...,l$ (l – текущий размер списка).

1.1.Если выполняется условие

$$c_k \in \{ x \in X : ||(x - \tilde{c}_i)|| \le \rho_m \}. \qquad (9)$$

1.1.1. То при $f_k < \tilde{f}_i$ содержимое элемента V_i обновляется:

$\tilde{c}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} c_j$, $\tilde{f}_i := f_k$, где c_j - центры параллелепипедов «попавших» в область V_i.

Далее выполняется действие 1.1.3.

1.1.2. Иначе уточняется только значение $\tilde{c}_i$. Выполняется действие 1.1.3.

1.1.3. Список V упорядочивается в порядке возрастания $\tilde{f}_i$.

1.2. Если условие (9) не выполняется, то проверяем правило 2.

2. Определяется возможность включения нового элемента в список V.

2.1. Если $(f_k < \tilde{f}_1)$, то элемент V_k записывается в голову списка V.

2.2. Если $(\tilde{f}_j < f_k \le \tilde{f}_{j+1})$, то элемент V_k записывается между V_j и V_{j+1} элементами списка.

2.3. Если $(f_k > \tilde{f}_l)$, то элемент V_k записывается в конец списка V.

3. При превышении предельного числа элементов списка, из списка исключается последний элемент списка.

Предложенный алгоритм является эвристическим, однако, вычислительные эксперименты с тестовыми функциями показали его состоятельность. Более того, как правило, в первой сотне элементов списка содержится начальное приближение глобального минимума функции. На рисунке 2 представлены элементы списка областей притяжения. Нумерация соответствует весу элемента. Из рисунка видно, что каждый элемент списка сформировался, агрегируя большое количество точек вычисления функции. Например, элемент V_1 агрегировал 9 точек, V_2 - 8, и т.д.

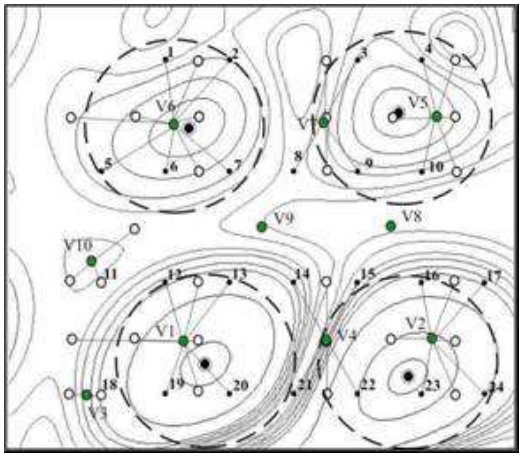


Рис. 2. Формирование списка областей притяжения

4. Визуальное средство реализации параллельных алгоритмов PGRAPH

Методы разработки параллельных алгоритмов существенно отличаются от методов последовательного программирования. Здесь большое значение приобретают средства автоматиза-

ции разработки моделей и кодов параллельных алгоритмов. Технология программирования (особенно в параллельном исполнении) должна быть понятна конечному пользователю, специалисту в предметной области.

4.1 Средство визуального программирования PGRAPH

При разработке и реализации двухфазного алгоритма метода половинных делений использовалась средство моделирования параллельных вычислений PGRAPH [5]. Система PGRAPH предоставляет пользователю возможность визуального формирования графических образов разрабатываемых алгоритмов, средства автоматизированного анализа проектируемых программ, а также средства автоматического синтеза исходных кодов программ. С помощью внешнего компилятора система позволяет компилировать и запускать на выполнение созданные в ней параллельные модели алгоритмов. При этом пользователь фактически «не видит» и не использует директивы MPI, которые автоматически «встраиваются» в текст программы при её компиляции.

Модель описания параллельных алгоритмов в системе PGRAPH представляется четверкой $< \Lambda, F, P, G >$ где Λ – множество данных некоторой предметной области, F – множество операторов, определенных над данными предметной области, P – множество предикатов, действующих над структурами данных предметной области, $G = (A, \Psi)$ – ориентированный помеченный граф, называемый агрегатом. $A = \{A_1, ..., A_n\}$ – множество вершин графа. Каждая вершина A_i помечена локальным оператором $\varphi_i(\Lambda) \in F$. На графе задано множество дуг управления $\Psi = \{\Psi_1, ..., \Psi_m\}$.

Дуга управления, соединяющая любые две вершины A_i и A_j, имеет три метки: *приоритет*; *предикат* – $p_{ij}(\Lambda) \in P$, управляющий ходом вычислительного процесса и *тип дуги*.

Граф G – инициальный. Работа алгоритма всегда начинается из начальной вершины. Дальнейшее развитие вычислений ассоциируется с переходами на графе из вершины в вершину по дугам управления. Для «последовательных» вершин, переход по дуге управления возможен лишь в случае истинности предиката, которым она помечена. Рассматриваются четыре типа дуг: *последовательная дуга* (описывает передачу управления на последовательных участках алгоритма) и обозначается на схеме прямоугольником; *параллельная дуга* (обозначает начало нового параллельного фрагмента алгоритма и помечается «кружочком»); *терминирующая дуга* (завершает параллельный фрагмент алгоритма, на схеме перечеркивается) и дуга синхронизации. Пример описания параллельного алгоритма представлен на рисунке 3.

4.2 Параллельный алгоритм глобальной оптимизации

Реализация алгоритма ДАМПД представлена на рисунке 3. Для удобства описания модели все вершины пронумерованы.

В вершине 1 устанавливаются значения параметров теста GKLS и производится его инициализация [7]. Алгоритм состоит из двух фаз. Первая фаза глобальной оптимизации состоит из вершин 2-5. В вершине 2 формируется начальный список параллелепипедов, которые впоследствии «раздаются» по процессорам кластера (см. вершину 3).

Глобальная оптимизация методом половинных делений реализуется в вершинах 4а-4г. На рисунке 3 для наглядности приведён вариант алгоритма для четырех процессоров, хотя несложно его масштабировать на любое число процессоров. Параллелепипеды делятся до тех пор, пока их радиус не станет меньше ρ_m. При этом на каждом процессоре формируются списки точек, лежащих в областях притяжения локальных минимумов. В вершине 5 сформировавшиеся списки объединяются.

Во второй фазе алгоритма осуществляет поиск локального минимума из точек, вычисленных на первом этапе. Данная фаза состоит из вершин 6-8. В вершине 6 происходит «раздача» точек начальных приближений процессорам для организации поиска локальных минимумов, который реализуется в вершинах 7а-7г. Вершина 8 завершает фазу локальной оптимизации. Алгоритм ДАМПД завершает свою работу по исчерпанию списка точек начальных приближений.

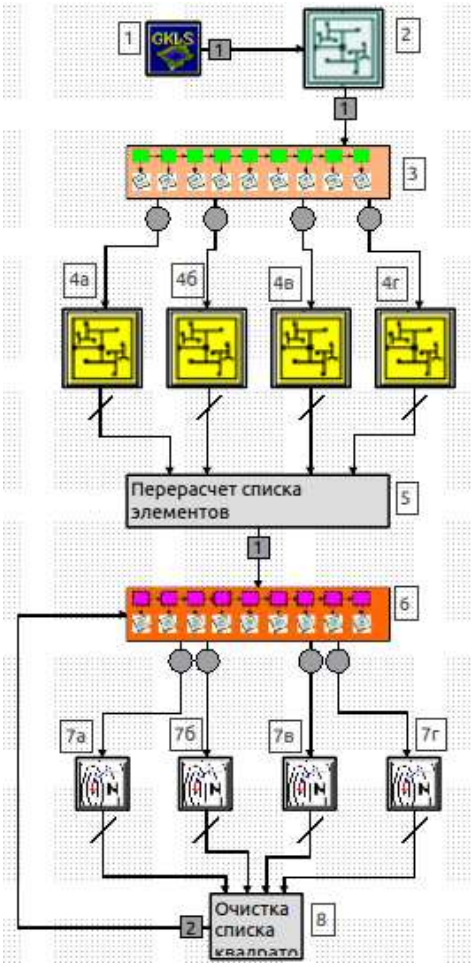


Рис. 3. Параллельная версия алгоритма ДАМПД

5. Вычислительные эксперименты

5.1 Условия тестирования

Для вычислительных экспериментов был выбран класс липшицевых функций, моделируемых известным генератором GKLS [7]. Генератор тестовых заданий GKLS порождает три класса тестовых функций: недифференцируемых, непрерывно дифференцируемых и дважды непрерывно дифференцируемых. Для каждого класса можно использовать 100 тестовых функций. В GKLS очень просто задать сложность генерируемых функций, определить количество локальных минимумов, размеры областей притяжения и многое другое.

Тестирование алгоритма ДАМПД проводилось на наиболее сложном для глобальной оптимизации, классе недифференцируемых функций. Для всех классов задач: число экстремумов равно десяти; радиус притяжения глобального оптимума – 0,33. Эксперименты проводились на суперкомпьютерном кластере СГАУ «Сергей Королев». Кластер построен на базе линейки оборудования IBM BladeCenter с использованием блейд-серверов HS22 и обеспечивает пиковую производительность более десяти триллионов операций с плавающей точкой в секунду. Общее

число процессоров/вычислительных ядер: 272/1184. Глобальный минимум вычислялся с точность $\varepsilon = 1{,}0 \cdot 10^{-8}$ (по аргументам функции), при этом использовалось 256 процессоров кластера.

5.2 Тестирование алгоритма ДАМПД

Результаты вычислительного эксперимента для алгоритма ДАМПД (при n=8) приведены в таблице 1. Общее ускорение алгоритма составило 91, хотя для фазы ЛО ускорение равно 139 . На рисунке 4 и 5 показана загрузка процессоров (количество обращений к оптимизируемой функции на каждом из них).

Таблица 1. Результаты эксперимента с базовой версией алгоритма ГО ДАМПД

Время работы алгоритма, сек	463,7
Число обращений к функции на этапе ГО (суммарно на всех процессорах)	262122 (23643168)
Число обращений к функции на этапе ЛО (суммарно на всех процессорах)	4922 (683665)

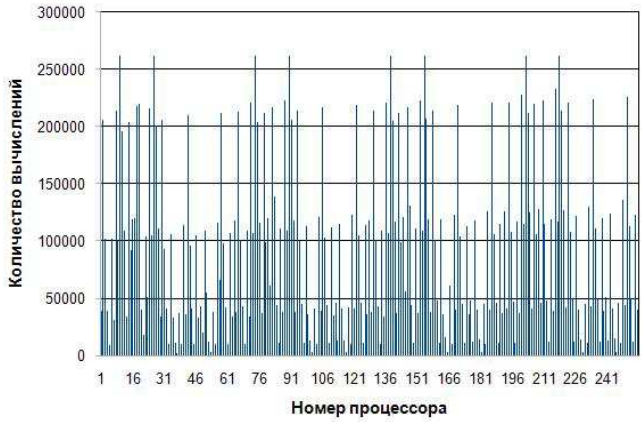


Рис. 4. Загрузка процессоров на этапе ГО в базовой версии

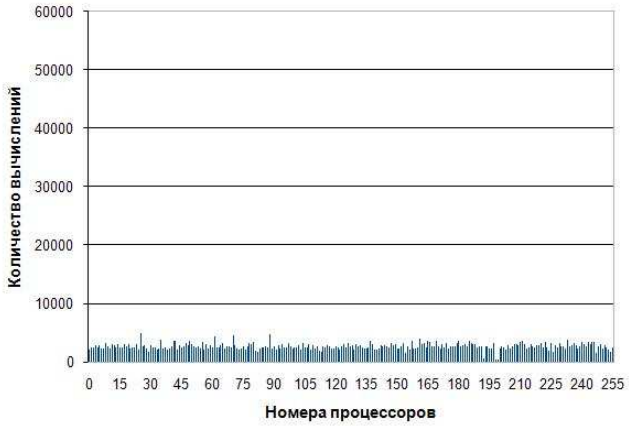


Рис. 5. Загрузка процессоров на этапе ЛО в базовой версии

Как видно из рисунков, наблюдается невысокая эффективность алгоритма ДАМПД для этапа глобальной оптимизации, поскольку процессоры кластера имеют неравномерную вычислительную нагрузку. Для фазы глобальной оптимизации количество делений параллелепипедов, реализуемое на каждом из процессоров, зависит от сложности доставшегося ему участка оптимизируемой функции. В итоге наблюдается нерациональное использование ресурсов кластера (используется только 35% потенциальной вычислительной мощности).

5.3 Тестирование алгоритма ДАМПД, реализованного по схеме «менеджер - исполнитель»

Для повышения эффективности алгоритма необходимо организовать адаптивное перераспределение нагрузки между процессорами, подкачивая по мере необходимости новые задания для «быстрых» процессоров.

Для этого в системе PGRAPH используются механизмы синхронизации параллельных процессов. Один из процессоров выделяется как управляющий (менеджер), а в его функции входит «раздача» заданий на оптимизацию для процессоров-исполнителей. Пример такой вычислительной схемы для этапа ЛО представлен на рисунке 6.

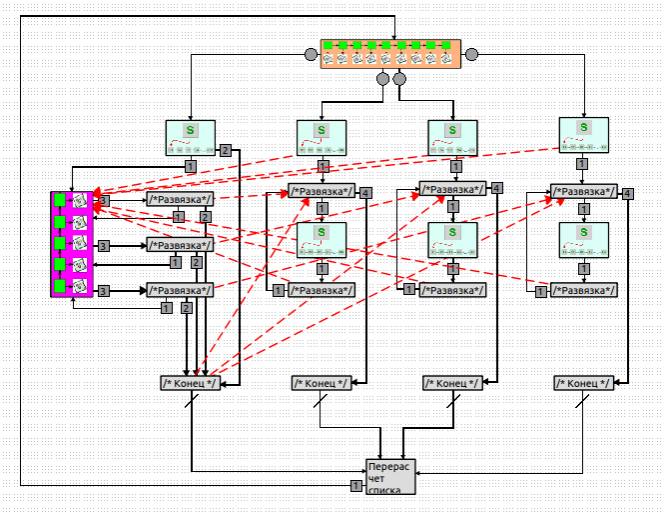


Рис. 6. Этап ЛО в режиме «менеджер – исполнитель»

Положим, что в фазе ЛО подготовлен список точек начальных приближений. Первоначально все процессоры, включая «менеджера», одновременно запускаются на поиск локального минимума. После завершения решения оптимизационной задачи ветвь «менеджера» переходит в режим диспетчера и ждет сообщения о завершении ЛО от остальных ветвей. На схеме передача сообщения изображается пунктирной стрелкой. После приёма сообщения от любой из параллельной ветви ветвь-менеджер передаёт соответствующему процессору новую точку начального приближения (фактически новое задание), запуская при этом очередной поиск локального минимума. Как только список точек начальных приближений заканчивается, ветвь-менеджер выходит из цикла приёма сообщений, и все ветви завершают свою работу.

Для фазы ГО асинхронное управление процессорами реализуется по аналогичной схеме. Необходимо лишь на этапе формирования начального списка параллелепипедов подготовить параллелепипедов больше, чем число процессоров, обеспечивая запас заданий для «быстрых» процессоров.

Результаты вычислительного эксперимента для данной версии алгоритма приведены в таблице 2.

Таблица 2. Результаты эксперимента для схемы «менеджер-исполнитель»

Время работы алгоритма, сек	268,2
Число обращений к функции на этапе ГО (суммарно на всех процессорах)	132662 (23715292)
Число обращений к функции на этапе ЛО (суммарно на всех процессорах)	12565 (2519983)

Увеличение ускорения для второго варианта ДАМПД объясняется тем, что для схемы «менеджер - исполнитель» в фазе глобальной оптимизации можно в разы увеличить начальный список параллелепипедов. При этом, естественно, уменьшаются их размеры (диаметры параллелепипедов) и, как следствие, уменьшается трудоемкость глобальной оптимизации на каждом из процессоров кластера, поскольку оптимизация производится до достижения заданного размера параллелепипеда сравнимого с «гарантированным радиусом» области притяжения. В этом случае получается, что максимальное время ГО на одном из процессоров во второй схеме

ДАМПД становится меньше, чем в первой схеме ($T_p^{(2)} < T_p^{(1)}$). Пусть трудоемкость последовательного алгоритма ГО составляет T_1. Если предположить, что обработка «избыточных» начальных параллелепипедов во второй схеме перераспределяется «менеджером» между освобождающимися процессорами, и суммарная загрузка процессоров не превышает $T_p^{(2)}$, то очевидно, что ускорение для второй схемы ДАМПД $S_p^{(2)} = T_1 / T_p^{(1)}$ будет больше, чем у первой схемы ($S_p^{(2)} > S_p^{(1)}$).

Общее ускорение составило 181. Загрузка процессоров для фаз ГО и ЛО показана на рисунке 7. В данном случае наблюдается более равномерное распределение нагрузки между процессорами. В частности общая эффективность алгоритма возросла до 70%, а фазы ЛО до 78%.

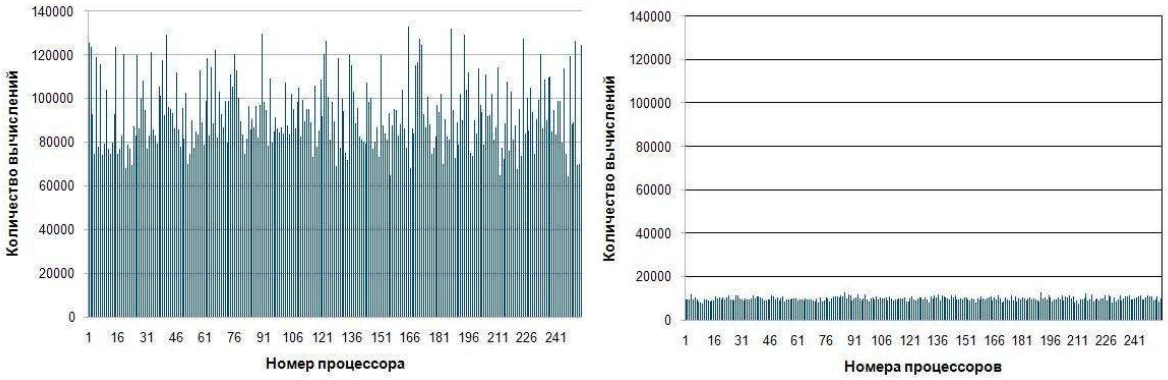


Рис. 7. Загрузка процессоров для этапов ГО и ЛО схемы «менеджер – исполнитель»

5.4 Зависимость сложности алгоритма ДАМПД от «гарантированного радиуса» притяжения

Несомненно, что трудоёмкость задачи глобальной оптимизации существенно зависит от величины «гарантированного радиуса» области притяжения. Последнее следует из формулы (8), поскольку с уменьшением радиуса ρ_m значительно увеличивается сложность фазы ГО, и немного уменьшается сложность фазы ЛО. На самом деле, сложность задачи оптимизации во многом определяется свойствами оптимизируемой функции. Например, «легче» отыскивается глобальный оптимум для функций, у которых области притяжения локальных экстремумов не пересекаются. Как показали вычислительные эксперименты, задача оптимизации становится более трудоёмкой, если области локальных притяжений её экстремумов вложены друг в друга.

Влияние радиуса притяжения на производительность алгоритма ДАМПД рассматривалась на примере задачи оптимизации функции 7 переменных для тестовой функции №4 GKLS (со стандартным набором установочных параметров) и для «хорошей» тестовой функции, описанной в работе [3]. В первом случае наблюдается вложенность областей локальных притяжений друг в друга. Во втором – области локальных притяжений экстремумов функции не пересекаются. Для того чтобы исключить влияние схемы реализации параллельных вычислений на оценку сложности задачи глобальной оптимизации, рассматривался последовательный алгоритм ДАМПД, а в качестве критерия оценки сложности использовалось число обращений к функции (N_1).

В таблицах 3 и 4 приведены результаты расчетов сложности последовательного алгоритма ДАМПД в зависимости от «гарантированного радиуса» области притяжения для тестовой функции GKLS и «хорошей» функции. Как видно из таблиц, в фазе ГО сложность задачи оптимизации стремительно увеличивается при уменьшении радиуса ρ_m. Однако значительно большее влияние на производительность алгоритма ДАМПД оказывают «топологические» свойства областей притяжения локальных экстремумов. Что же касается фазы ЛО, то сложность локальной оптимизации практически не зависит от радиуса притяжения.

Таблица 3. Сложность последовательного алгоритма ДАМПД для функции GKLS

Радиус притяжения, ρm	Число обращений к функции N1 на этапе	
	ГО	ЛО
0.31	1130068	115830
0.22	10361462	149928
0.16	60819698	291252

Таблица 4. Сложность последовательного алгоритма ДАМПД для «хорошей» функции [3]

Радиус притяжения, ρm	Число обращений к функции N1 на этапе	
	ГО	ЛО
0.31	45472	413199
0.22	725310	316044
0.16	4376072	283662

6. Заключение

В работе на примере задачи глобальной оптимизации продемонстрированы возможности средства визуального моделирования параллельных алгоритмов PGRAPH, ориентированного на конечного пользователя и освобождающего его от изучения непрофильных областей знаний, например, «тяжеловесного» стандарта MPI.

Очевидно, что общая эффективность параллельного алгоритма глобальной оптимизации зависит как от используемой схемы решения задачи ГО, так и от применяемой стратегии её реализации на высокопроизводительной ЭВМ. В частности, в данной работе в качестве концептуальной идеи повышения эффективности методов решения задачи ГО предлагалось использовать частичное замещение точных методов глобальной оптимизации, имеющих экспоненциальный рост сложности, методами локальной оптимизации, сходящимися к решению за полиномиальное время.

Что же касается реализации выбранной стратегии на высокопроизводительной ЭВМ, то здесь возможны самые разнообразные решения. В работе рассматривались 2 варианта реализации алгоритма ДАМПД. На самом деле, с учетом методов организации обмена данными между процессорами, их значительно больше. Причем невозможно заранее установить, какой из вариантов параллельного алгоритма окажется самым эффективным. В таких условиях большое значение приобретают средства автоматизации моделирования параллельных алгоритмов, к числу которых относится и PGRAPH. Следует иметь в виду, что на рисунках 3 и 6 показаны не схемы параллельных алгоритмов, а готовые программы, представленные в визуальной нотации, которые автоматически компилируются в тексты программ на языке C++, включая директивы MPI, синхронизацию и прочее.

Литература

1. Евтушенко Ю.Г., Посыпкин М.А. Параллельные методы решения задач глобальной оптимизации // PACO'2008: Труды четвертой международной конференции «Параллельные вычисления и задачи управления». М., 2008. С. 5 – 15
2. Квасов Д.Е., Сергеев Я.Д. Многомерный алгоритм глобальной оптимизации на основе адаптивных диагональных кривых // ЖВМ и МФ, 2003, Т.43, № 1. С. 42-59.
3. Коварцев А.Н., Попова-Коварцева Д.А. К вопросу об эффективности параллельных алгоритмов глобальной оптимизации функций многих переменных // Компьютерная оптика. 2011. Т. 35, № 2. С. 256 – 262.
4. Коварцев А.Н., Попова-Коварцева Д.А. Многомерный параллельный алгоритм глобальной оптимизации модифицированным методом половинных делений // В мире научных открытий. № 8.1(32), 2012. С. 80-108.
5. Коварцев А.Н., Жидченко В.В. Моделирование синхронных параллельных вычислений при построении математических моделей сложных систем // Труды первой международной

конференции: Системный анализ и информационные технологии. Том 2. М.: КомКнига, 2005. С.154-160.

6. Посыпкин М.А. Методы решения задач конечномерной оптимизации в распределенной вычислительной среде // САИТ-2009: Труды конференции. М., 2009. С. 729-740
7. Gaviano M., Kvasov D.E., Lera D., Sergeyev Ya.D. Software for generation of classes of test of functions with known local and global minima for global optimization // ASM Transactions on Mathematical Software, 2003. 29(4). pp. 469-480.

Параллельные алгоритмы метода дополнения Шура на гибридной архитектуре *

С.П. Копысов, И.М. Кузьмин, Н.С. Недожогин, А.К. Новиков

Институт механики УрО РАН

Эффективность применения метода дополнения Шура на гибридных (CPU/GPU) архитектурах зависит от распределения вычислений между центральным процессором и графическими ускорителями. Показано, что формирование матриц дополнения Шура для нескольких подобластей эффективно выполнять на GPU, а с ростом числа подобластей на CPU. Представлен параллельный алгоритм обращения матрицы при помощи решения матричной системы множеством параллельных потоков. Для решения интерфейсной системы предложен параллельный алгоритм метода сопряженных градиентов с явным предобуславливателем, позволяющий достигать существенного ускорения вычислений на нескольких GPU.

1. Введение

Блочное разложение исходной матрицы или системы уравнений на основе дополнения Шура позволяет ускорить вычисления за счет параллельного обращения и умножения подматриц меньшего размера, чем у исходной матрицы [1,2]. Первоначально метод дополнения Шура [3] рассматривался как метод декомпозиции области для вычислений на системах с ограниченными вычислительными мощностями. В настоящее время, вычисление дополнения Шура чаще рассматривают, как гибридный метод решения систем линейных алгебраических уравнений (СЛАУ) [4, 5], сочетающий преимущества как прямых, так и итерационных методов, и учитывающий различные архитектуры параллельных вычислительных систем. На его основе строятся эффективные предобуславливатели [6].

Появление программного обеспечения для вычислений общего назначения на графические устройствах (GPGPU) позволило на ряде задач, в том числе вычислительной линейной алгебры, получать ускорение вычислений в десятки и сотни раз, по сравнению с центральным процессором (CPU). Тысячи потоков (нитей) GPU могут эффективно выполнять одновременно большое число простых арифметических операций, что характерно для мультипликативных и аддитивных операций с векторами и матрицами. Вместе с тем, последовательные операции и ветвления, характерные для разложения матриц на треугольные множители, выполняются на GPU медленнее, чем ядрами CPU. Кроме того, графический ускоритель обладает существенно меньшим объемом собственной оперативной памяти, чем типичный современный вычислительный модуль, что приводит к необходимости использования в расчетах нескольких GPU, в том числе связанных вычислительной сетью.

Таким образом, эффективное применение графических ускорителей, тем более нескольких, для метода дополнения Шура связано с декомпозицией матриц и определением алгоритмов, которые более эффективно выполняются на CPU или GPU. Переход к параллельным вычислениям на нескольких графических ускорителях предполагает использование нескольких технологий параллельного программирования: CUDA — для вычислений на одном GPU, OpenMP — для распараллеливания вычислений между несколькими GPU внутри одного вычислительного модуля и MPI — для передачи данных при вычислениях на кластере с гибридной архитектурой.

В работе решаются системы уравнений, получаемые при конечно-элементом решении трехмерных задач теории упругости и использующие разделение расчетной сетки при формировании блочной структуры матриц системы. В данном методе можно реализовать два

*Работа выполнена в рамках программы Президиума РАН №18 при поддержке УрО РАН (проект 12-П-1-1005) и РФФИ (грант №11-01-00275-а, 12-07-31114-мол-а).

уровня распараллеливания: первый уровень связан с разделением вычислений между подобластями [7, 9], второй с параллельной реализацией методов, которые используются для формирования внутри отдельной подобласти. Помимо этого, распараллеливанию подлежит и решение системы для дополнения Шура (интерфейсной системы). В представленной работе рассматривается второй уровень распараллеливания, для которого предложены алгоритмы формирования матриц дополнения Шура, а также решение интерфейсной системы уравнений с использованием нескольких графических ускорителей.

2. Ресурсная эффективность метода дополнения Шура

Рассмотрим использование построения дополнения Шура, как один из вариантов методов декомпозиции области в виде алгоритма метода подструктур [3]. Для обеспечения независимости вычислений в отдельных подобластях и последующего их взаимодействия, все узлы области делятся на два множества: внешних и внутренних узлов. Неизвестные перемещения области рассматриваются в виде суперпозиции двух составляющих. Первая составляющая — перемещения, вызванные внешними силами при закреплении границ в подобластях. Перемещения каждой подобласти определяются из уравнений, включающих неизвестные, связанные только с данной подобластью. Вторая составляющая — перемещения, вызванные смещениями границ подобласти с исключенными внутренними узлами.

Пусть область Ω разбита на n_Ω непересекающихся подобластей:

$$\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \ldots \bigcup \Omega_{n_\Omega}, \quad \text{где} \quad \Omega_i \bigcap \Omega_j = \emptyset, \quad \Gamma_B = \bigcup_{i=1}^{n_\Omega} \partial\Omega_i \setminus \partial\Omega. \tag{1}$$

Разделение на подобласти наследуется от процесса разделения дуального графа расчетной сетки $G(V, E) = \bigcup_{i=1}^{n_\Omega} G_i(V_i, E_i)$, здесь множество вершин графа V — это множество конечных элементов расчетной сетки, множество ребер графа E — множество смежных конечных элементов, $V_i \subset V$ — множество конечных элементов, образующих подобласть. Далее полагается, что все подграфы $G_i(V_i, E_i)$ связные, в противном случае система уравнений (2) для подобласти Ω_i распадается на несвязанные системы уравнений.

Узлы расчетной сетки образуют множество $\hat{V}$ и условно разделяются на внешние $\hat{V}_{B_i}$ — принадлежат границе области и внутренние $\hat{V}_{I_i}$ — связанные с узлами подобласти сетки, соответствующей подграфу $G_i(V_i, E_i)$. Из множества внешних узлов выделяются интерфейсные $\hat{V}_{C_i} \subset \hat{V}_{B_i}$, связанные с узлами из других подобластей.

Для каждой подобласти Ω_i строятся системы уравнений, причем степени свободы, связанные с внутренними и внешними (граничными) узлами, разделяются:

$$\begin{pmatrix} A^i_{II} & A^i_{IB} \\ A^i_{BI} & A^i_{BB} \end{pmatrix} \begin{pmatrix} u^i_I \\ u^i_B \end{pmatrix} = \begin{pmatrix} f^i_I \\ f^i_B \end{pmatrix}, \tag{2}$$

где индексы I, B относятся к внутренним и граничным степеням свободы.

Система для интерфейсных узлов определяется как

$$S_{BB}\tilde{u}_B = \tilde{f}_B, \tag{3}$$

здесь $S_{BB} = \sum_i^{n_\Omega} \left(A^i_{BB} - A^i_{BI} A^{i^{-1}}_{II} A^i_{IB} \right)$ — матрица граничных жесткостей или дополнение Шура для подобласти i, вектор $\tilde{f}_B = \sum_i^{n_\Omega} \left(f^i_B - A^i_{BI} A^{i^{-1}}_{II} f^i_I \right)$ — вектор правых частей.

Как правило для расчетов задач используется усовершенствованный алгоритм метода подструктур. В его основе лежит использование свойств невырожденности и положительной определенности матриц подобластей. Для этих матриц существует разложение Холесского $A_{II} = L_{II} L^T_{II}$, где L — нижняя треугольная матрица с положительными диагональными элементами. Использование разложения Холесского значительно сокращает вычислительные затраты и используемый объем оперативной памяти.

Рассмотрим реализацию последовательного алгоритма вычисления дополнения Шура ($n_\Omega > n_p$ и число процессоров $n_p = 1$) и вычислительные затраты, связанные с каждым шагом выполнения (после шага алгоритма показано число необходимых операций с учетом симметрии матриц, причем операция сложения и умножения принимается как одна). Здесь

Алгоритм 1 Последовательный вариант дополнения Шура:

1: Выполним разложение Холесского матрицы A_{II} для соответствующей подобласти(индекс i опущен)
$$A_{II} = L_{II} L_{II}^T. \qquad \left(n_I^3/6\right)$$
2: Вычисляем вспомогательные переменные
$$A'_{IB} = A_{II}^{-1} A_{IB}. \qquad \left(n_B \cdot n_I^2\right)$$
3: Сформируем матрицы граничной жесткости подобластей
$$S_{BB} = A_{BB} - A_{BI} A'_{IB}. \qquad \left((n_I \cdot n_B^2)/2\right)$$
4: Формируем вектор правой части
$$\tilde{f}_B = f_B - A_{BI} A_{II}^{-1} f_I. \qquad (n_I \cdot n_B)$$
5: Собираем и решаем систему уравнений
$$S_{BB} \tilde{u}_B = \tilde{f}_B. \qquad (k(M^{-1} S_{BB}) \leq C(1 + \log(H/h)))$$
6: Определяем неизвестные для внутренних узлов
$$u_I = A_{II}^{-1} f_I - A'_{IB} \tilde{u}_B. \qquad \left(n_I^2\right).$$

$n_I = m \cdot |\hat{V}_{I_k}|$, $n_B = m \cdot |\hat{V}_{B_k}|$ — число внутренних и граничных степеней свободы, m — число степеней свободы в узле сетки, h — шаг сетки, H – размер подобласти, M — предобуславливатель для дополнения Шура. На пятом шаге **Алгоритма 1** приведена оценка обусловленности матрицы S_{BB}, а не вычислительная сложность, которая зависит от выбора метода решения системы уравнений. В данной работе, в качестве метода решения СЛАУ, использовался метод сопряженных градиентов с различными предобуславливателями. Матрицы S_{BB}, A_{II} положительно определены и симметричны. Отметим, что порядок матрицы S_{BB} значительно меньше, чем исходной матрицы, но достаточно большой, поэтому для системы с дополнением Шура применяются итерационные методы решения и компактные схемы хранения.

Для обеспечения эффективной работы с матрицами используются различные схемы хранения. Матрицы S_{BB}, A_{II} являются симметричными, поэтому возможно хранение только её части (верхний или нижний треугольник с диагональю).

Формат хранения (DCSR), используемый при вычислениях в данной работе, представляет массив, состоящий из списка упакованных строк, реализован в системе конечно-элементного анализа FEStudio [7, 8]. Каждая строка матрицы представляет структуру, состоящую из двух массивов. Первый массив хранит значения ненулевых элементов, второй — столбцовые индексы этих элементов. Размер каждого из массивов для строки i равен числу ненулевых элементов Nnz_i и меняется динамически по мере необходимости.

Предложенный формат DCSR является разновидностью распространенного формата хранения разреженных матриц — формат CSR (Compressed Sparse Row Storage Format). Для матрицы A, в формате CSR, выделяются три одномерных массива, в которых хранятся: ненулевые значения $\{a_{ij} \,|\, a_{ij} \neq 0\}$, $1 \leqslant i, j \leqslant Nnz$, их столбцовые индексы $\{j \,|\, a_{ij} \neq 0\}$, $1 \leqslant i, j \leqslant Nnz$ и позиции элементов a_{i1}, $1 \leqslant i \leqslant N + 1$ в двух первых массивах. Здесь N — размерность матрицы, $Nnz = \sum_{i=1}^{N} Nnz_i$ — число ненулевых элементов в матрице, Nnz_i — число ненулевых элементов в i-строке матрицы.

Сравним ресурсоёмкость предложенных схем хранения матриц по следующим параметрам: занимаемая память, алгоритмическая сложность доступа к элементу и его добавления. Оценка показывает, что алгоритмические сложности доступа к элементу $\mathcal{O}(2N_\beta + 1)$ и его добавления для ленточного формата составляет $\mathcal{O}((2N_\beta + 1)N)$, для DCSR —

$\mathcal{O}(Nnz^*)$ и $\mathcal{O}(Nnz^*)$ соответственно, а для формата CSR — $\mathcal{O}(Nnz^*)$ и $\mathcal{O}(Nnz)$, где $Nnz^* = \max_{i=1}^{N}\{Nnz_i\}$, $N_\beta = \max_{i=1}^{N}\{\max_{j=1}^{N}\{j-i \,|\, a_{ij} \neq 0\}\}$ — так называемая полуширина ленты, зависящая от нумерации неизвестных и уравнений. Необходимый объем памяти для ленточного формата — $8(2N_\beta + 1)N$, для формата DCSR — $\sum_{i=1}^{N}(12Nnz_i + 4)$, для формата CSR — $12Nnz + 4(N + 1)$ байт. В этом случае полагается, что при хранении вещественной величины используется 8 байт памяти, и 4 байта — для целого числа. В симметричном случае, величины $2N_\beta + 1$ в оценках сложности алгоритма (и приведенной далее частоты доступа) заменяются на $N_\beta + 1$, а Nnz и Nnz_i относятся к элементам треугольной матрицы.

Отметим, что формат DCSR более удобен, чем CSR при выполнении треугольного разложения матриц A_{II}, когда в строках появляются новые ненулевые элементы. В этом случае изменение в одной из строк не требует смещения всех последующих элементов в массивах, как в формате CSR. Поэтому процедура добавления нового элемента имеет меньшую алгоритмическую сложность $\mathcal{O}(Nnz^*)$, чем в формате CSR — $\mathcal{O}(Nnz)$. Кроме того, из представленных форматов, необходимый для DCSR объем памяти — $\sum_{i=1}^{N}(12Nnz_i + 4)$ байт, является минимальным. Преимущества ленточного формата, по сложности доступа и добавления элемента, компенсируются необходимым объемом памяти $8(2N_\beta + 1)N$ и частотой доступа к элементам матрицы $\mathcal{O}((2N_\beta + 1)N)$, вместо $\mathcal{O}(Nnz)$ для форматов DCSR и CSR.

Как показали эксперименты [9], схема хранения оказывает существенное влияние на время вычислений. Отметим, что время формирования S_{BB} с матрицами в ленточном формате составляет порядка 80% общего времени вычислений. Переход на формат DCSR для матриц A_{BB}, A_{IB}, A_{II} для одной и той же задачи дал сокращение затрат в четыре раза. Таким образом, в последовательном варианте, наиболее эффективным, с точки зрения ресурсоемкости и алгоритмической сложности, представляется использовать метод дополнения Шура с разложением Холесского матрицы A_{II} и хранением матриц в сжатом формате.

Обработка строк матрицы, хранящейся в формате DCSR, на графическом ускорителе (GPU) потребует для каждой строки: выделения памяти на GPU, копирования строки и далее, в ядре CUDA, выполнения вычислений над строкой и возвращения результата в оперативную память или кэш CPU. Таким образом, потребуется $2N$ выделений памяти и копирований массивов размера Nnz_i, вместо выделения памяти и копирования двух массивов размера Nnz, поэтому, для вычислений на GPU, матрица переводится в CSR формат.

3. Параллельная эффективность метода дополнения Шура

В методе декомпозиции области можно выделить два направления распараллеливания процесса решения. Первое связано с методами явного деления на подобласти, где параллельные вычисления осуществляются на уровне области, то есть число ветвей параллельного алгоритма равно числу подобластей. Как показывает практика, такой подход дает значительное ускорение, если каждый процессор решает в своей подобласти свою подзадачу. Второе направление — неявные методы внутри подобластей. В каждой подобласти различные задачи могут быть решены наиболее эффективным способом.

Рассмотрим **Алгоритм 1**, исходя из введенных вариантов распараллеливания. *Шаги 1–4, 6* выполняются для каждой подобласти независимо, что говорит о естественном параллелизме, который обеспечивает первый уровень распараллеливания — на уровне области.

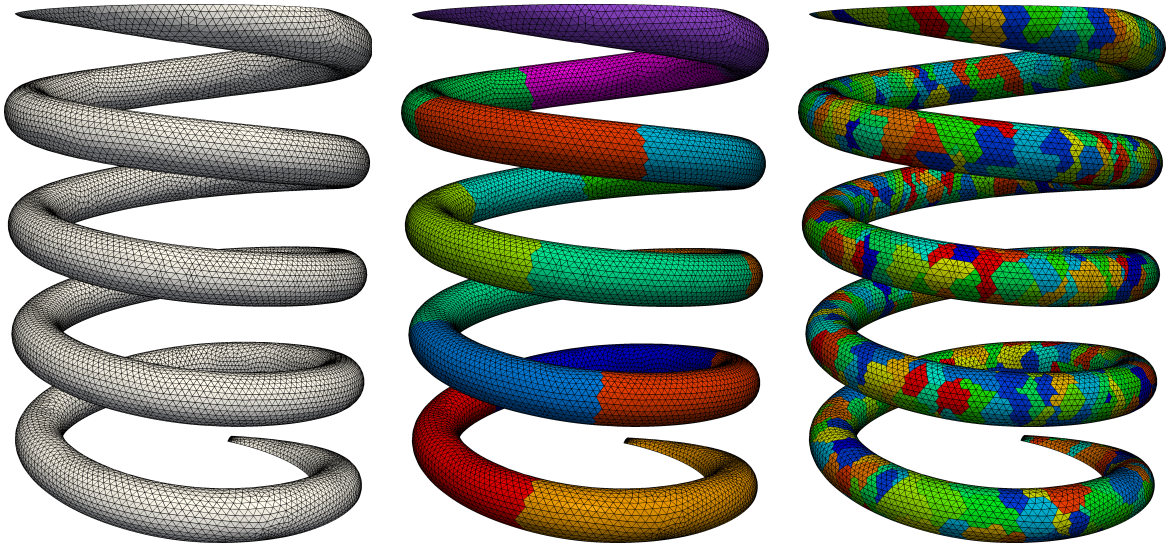
Наиболее трудоёмкими шагами алгоритма являются: процессы формирования матрицы дополнения Шура — *Шаги 1–3*, вектора правых частей — *Шаг 4*, а также решение системы — *Шаг 5*. Вычисления внутри каждой подобласти выполняются независимо от других подобластей, значит становится возможной их параллельная реализация.

При реализации параллельных алгоритмов неизбежно возникает проблема балансировки вычислительной нагрузки. На четвертом шаге **Алгоритма 1** формируются локальные матрицы дополнения Шура S_{BB}^i, независимо для каждой из подобластей. Далее из S_{BB}^i строится система уравнений с глобальной матрицей дополнения Шура (3), которая не мо-

жет быть решена, пока S^i_{BB} не сформируются всеми параллельными процессами i.

Таким образом, источников неравномерности нагрузки может быть несколько. Одним из них является неравномерное распределение количества подобластей на вычислительные узлы — этот недостаток легко исключить, разделив область на количество подобластей кратное количеству используемых вычислительных узлов. Другой источник — неравномерное распределение узлов сетки и конечных элементов по подобластям. В этом случае неравномерность исключается заданием дополнительных условий при разделении сетки.

Сбалансированное распределение вычислительной нагрузки при выполнении *Шагов 1–4* зависит не от общего числа узлов $|\hat{V}_i|$ в подобластях Ω_i, а от числа внутренних $|\hat{V}_{I_i}|$ и внешних $|\hat{V}_{B_i}|$ узлов в подобластях. Вместе с тем, сетки для трехмерных областей с развитой поверхностью (пружины, тонкие пластины и оболочки) содержат большое число конечных элементов, в которых все узлы, принадлежат границе расчетной области. Разделение дуальных графов таких сеток и выполнение условия $|V_i| \approx |V_j|, \forall i \neq j$ приводит к подобластям Ω_i: $|\hat{V}_{I_i}| = 0$. Наглядным примером является задача моделирования напряженно-деформированного состояния пружины, рассматриваемая в данной работе.



а) б) в)

Рис. 1. Сетка: а) исходная; б)разделенная на 16 подобластей; в)разделенная на 1024 подобласти двухуровневым разделением.

Геометрия пружины аппроксимируется неструктурированной расчетной сеткой (см. рисунок 1а) с ячейками в виде тетраэдров, число ячеек $|V| = 174264$, число узлов $|\hat{V}| = 40743$. Для получения подобластей $\Omega_i : |\hat{V}_{I_i}| > 0$, дуальный граф полагался взвешенным $G(V, E, W)$, с множеством весов $W = \{w_k\}$. Если хотя бы один из узлов конечного элемента не лежит на $\partial\Omega$, то вес соответствующей вершины графа $w_k = 3$, в другом случае $w_k = 1$.

Проведенные вычислительные эксперименты с различным числом подобластей ($n_\Omega = 16, 32, \ldots, 1024$) показали, что увеличение числа подструктур приводит к увеличению размера матрицы дополнения Шура. Отметим, что число подобластей увеличилось в 64 раза, а размер матрицы S_{BB} только в 1.44 раза ($N = 66030$ в случае $n_\Omega = 16$ и $N = 95523$ — для $n_\Omega = 1024$). Вместе с тем, число ненулевых элементов матрицы дополнения Шура уменьшилось в 18 раз (с $Nnz \approx 2.7 \cdot 10^8$ в случае 16-ти, до $Nnz \approx 1.5 \cdot 10^7$ — для 1024-х подобластей), как следствие, уменьшилась её заполненность с 6.11% до 0.16%. В среднем, примерно каждый шестнадцатый элемент в строке S_{BB} является ненулевым (4041 из 66030), при $n_\Omega = 16$ подобластей (см. рисунок 1 б), и примерно каждый шестисотый (154 из 95523) — при разделении на 1024 подобласти (см. рисунок 1 в).

Важно отметить, что размер системы (3) меньше размера исходной конечно-элементной системы (в рассмотренных случаях в 1.4–2.0 раза), тогда как заполненность матрицы S_{BB} на один-два порядка больше, чем заполненность глобальной матрицы жесткости (0.03%).

4. Формирование матрицы дополнения Шура

Одной из самых затратных операций формирования дополнения Шура является обращение матрицы A_{II}. Обычно, методами нахождения обратной матрицы являются плохо распараллеливаемые прямые методы, например метод обращения на основе LL^T-разложения.

Рассматриваемый в работе алгоритм вычисления обратной матрицы состоит из решений матричной системы вида $A_{II}X = E$, где E — единичная матрица $n_I \times n_I$. Система эффективно решается на GPU предобусловленным алгоритмом сопряженных градиентов (см. следующий параграф). Если в правой части системы вместо матрицы E брать A_{IB}, то её решением будет матрица $A'_{IB} = A_{II}^{-1} A_{IB}$. Такое представление позволяет заменить операции обращения матрицы и матричного произведения на решение n_B систем, каждая из которых решается независимо, что позволяет использовать одновременно несколько GPU. Дополнение Шура или матрица граничных жесткостей вычисляется по соотношениям (3), где $A_{BB} \in \mathbb{R}^{n_B \times n_B}$, $A_{II} \in \mathbb{R}^{n_I \times n_I}$ $A_{BI} \in \mathbb{R}^{n_B \times n_I}$, $A_{IB} \in \mathbb{R}^{n_I \times n_B}$.

Пусть $\tilde{n}_B^i$ — количество столбцов матрицы A_{BB} пересылаемых на i-ый GPU, $\tilde{A}_{BB}^i \in \mathbb{R}^{n_B \times \tilde{n}_B^i}$ матрица, состоящая из столбцов матрицы A_{BB} с номерами от $\sum_{j=0}^{i} \tilde{n}_B^j$ до $\sum_{j=0}^{i+1} \tilde{n}_B^j$, где i — номер GPU. Каждый графический ускоритель решает $\tilde{n}_B^i$ систем $A_{II}a^k = a_{IB}^k$, здесь a_{IB}^k — k-ый столбец матрицы A_{IB}, $k \in \left[\sum_{j=0}^{i} \tilde{n}_B^j, \sum_{j=0}^{i+1} \tilde{n}_B^j\right]$, а $A'_{IB} = \{a^1, a^2 \dots a^{n_B}\}$. В реализации подхода независимого решения систем уравнений на нескольких графических ускорителях используется технология OpenMP. Для этого создаются несколько нитей (число которых равно количеству доступных устройств GPU), определяется номер нити и каждой назначается графический ускоритель с тем же номером.

Ниже представлен **Алгоритм 2**, реализующий этот подход. На каждом GPU после

Алгоритм 2 Параллельный алгоритм формирования дополнения Шура на каждой подобласти Ω_i (индекс i опущен):

1: Решаем систему $A_{II}A'_{IB} = A_{IB}$; {матрицы хранятся на GPU в CSR формате, решение — по столбцам}
2: $S_{BB} = A_{BB} - A_{BI}A'_{IB}$; {$S_{BB}$ и результат произведения матриц — построчно, A_{BB} — CSR формате}
3: $\tilde{f}_B = f_B - A_{BI}x$; {$x$ — решение системы $A_{II}x = f_I$}
4: Формируем и решаем: $S_{BB}\tilde{u}_B = \tilde{f}_B$; {$S_{BB}$ формируется в DCSR на CPU, а затем копируется в CSR на GPU }
5: Определяем $u_I = x - A'_{IB}\tilde{u}_B$; {$x$ и A'_{IB} вычислены ранее, и хранятся на CPU}.

решения систем остаётся матрица $(A'_{IB})^i \in \mathbb{R}^{n_I \times n_B^i}$, состоящая из столбцов матрицы A'_{IB} с номерами от $\sum_{j=0}^{i} \tilde{n}_B^j$ до $\sum_{j=0}^{i+1} \tilde{n}_B^j$. Это позволяет без дополнительных коммуникаций выполнить оставшиеся операции (произведение и разность матриц, см. соотношения (3)) для каждой матрицы $(A'_{IB})^i$ независимо на нескольких GPU, которые используются при вычислении матриц S_{BB}^i. Далее формируется глобальная матрица дополнения Шура S_{BB} (*Шаг* 4 в **Алгоритме 2**), состоящая из локальных S_{BB}^i, принадлежащих i-подобласти.

Локальные матрицы дополнения Шура S_{BB}^i хранятся в несжатом формате. Матрица S_{BB}, после формирования из локальных S_{BB}^i, преобразуется из несжатого к тому формату, в котором будет наиболее удобно с ней работать на GPU, например CSR.

В таблице 1 приведены затраты времени на параллельное формирование S_{BB}, в зависимости от n_Ω и числа GPU. Во второй колонке представлены данные для последовательного алгоритма, выполняемого на центральном процессоре (**Алгоритм 1**).

Таблица 1. Время формирования дополнения Шура, мин. : сек.

n_Ω	CPU	1 GPU	2 GPU	4 GPU	6 GPU	8 GPU
16	26:23.6	31:52.6	19:25.5	13:09.9	10:57.6	09:49.4
32	08:00.6	19:33.9	11:01.8	06:41.7	05:14.0	04:26.5
64	02:25.1	11:18.8	06:18.2	03:44.8	02:54.7	02:29.0
128	—	—	03:57.2	02:25.8	01:58.5	01:42.5
256	—	—	03:14.0	02:01.0	01:37.7	01:26.0
512	—	—	02:48.3	01:45.7	01:26.0	01:15.4
1024	00:18.7	03:53.4	02:24.0	01:32.2	01:17.5	01:08.0

Ускорение параллельного алгоритма (**Алгоритм 2**), реализованного на GPU, по отношению к CPU, определим как $s(n_p)_{CPU} = t_{CPU}/t(n_p)_{GPU}$, где t_{CPU} — время выполнения последовательного алгоритма, реализованного на CPU, а $t(n_p)_{GPU}$ — время выполнения параллельной реализации на n_p GPU. Аналогичным образом введём ускорение $s(n_p)_{GPU} = t(1)_{GPU}/t(n_p)_{GPU}$. В рассмотренных вариантах максимальное ускорение составляет $s(8)_{GPU} = 2.7$ и достигается при числе подобластей $n_\Omega = 16$, при этом в каждой подобласти находится в среднем примерно по 11000 ячеек сетки. Формирование матрицы дополнения Шура на двух GPU приводит к ускорению $s(2)_{GPU} > 1.5$, в зависимости от числа подобластей. Использование восьми графических ускорителей даёт наименьшее время выполнения для реализации на GPU, но, для задач с небольшим числом ячеек в каждой подобласти (< 2500), CPU-реализация оказывается эффективнее. В этом случае, при решении большого числа систем уравнений малой размерности для каждой подобласти требуются время на копирование данных между GPU и CPU, которое становится больше, чем время решения систем. В рамках одного GPU (при использовании нескольких) затраты на решение сокращаются, но не покрывают затрат на инициализацию и копирование.

Полученные данные позволяют применять **Алгоритм 2** для обеспечения более сбалансированного использования GPU и CPU в гибридных вычислительных системах.

5. Решение интерфейсной системы уравнений на GPU

Матрица дополнения Шура $S_{BB} \in \mathbb{R}^{N\times N}$ (далее $S = [s_{ij}]$) имеет порядок и обусловленность меньше, чем у исходной матрицы и является симметричной, положительно определённой и разреженной. Решение интерфейсной системы (3) и систем уравнений из предыдущего параграфа выполняются предобусловленным методом сопряженных градиентов.

В большинстве работ, посвященных реализации итерационных методов на GPU, рассматривается предобуславливатель Якоби или его блочный аналог. Оптимальным выбором для вычислений на GPU представляются предобуславливатели, в которых считается, что известна аппроксимация обратной матрицы системы $\bar{M} \approx S^{-1}$ [10]. Тогда дополнительные операции, связанные с предобуславливанием, сводятся к произведению $z_{k+1} = \bar{M}r_{k+1}$.

В данной работе рассматриваются методы сопряжённых градиентов с диагональным предобуславливателем (DIAG), предобуславливанием по методу симметричной последовательной верхней релаксации (SSOR) и предобуславливателем, построенным масштабированием системы (DIP). К сожалению, в наших экспериментах предобуславливатели, основанные на аппроксимации обратной матрицы с использованием дополнения Шура, не показали

сравнимую эффективность вычислений на рассматриваемых матрицах.

Диагональный предобуславливатель имеет вид:

$$\bar{M} = M^{-1} = \{\bar{m}_{ij}\}_{ij=1}^{N}, \quad \bar{m}_{ij} = \begin{cases} s_{ij}^{-1}, & \text{если } i = j; \\ 0, & \text{в остальных случаях.} \end{cases}$$

Определим предобуславливатель SSOR следующим образом. Пусть матрица системы представима как $S = L + D + L^T$, тогда $M = KK^T$, $K = \frac{1}{\sqrt{2-\omega}}\tilde{D}(I + \tilde{D}^{-1}L)\tilde{D}^{-1/2}$, или $K^{-1} = \sqrt{2-\omega}\tilde{D}^{1/2}((I + \tilde{D}^{-1}L))^{-1}\tilde{D}^{-1}$, где $0 < \omega < 2$, $\tilde{D} = (1/\omega)D$.

Вычислим приближенно K^{-1} ограничиваясь первым членом в разложении

$$K^{-1} \approx \bar{K} = \sqrt{2-\omega}\tilde{D}^{-1/2}(I - L\tilde{D}^{-1}). \tag{4}$$

Тогда предобуславливатель SSOR примет вид $\bar{M} = \bar{K}^T\bar{K}$. Если положить $\omega = 1$ в (4) получим предобуславливатель вида $\bar{K} = D^{-1/2}(I - LD^{-1})$ и соответственно $\bar{M} = D^{1/2}\bar{K}\bar{K}^T D^{1/2}$. При построении предобуславливателя DIP масштабируем исходную систему

$$\tilde{S} = D^{-1/2}SD^{-1/2}; \quad \tilde{f} = D^{-1/2}f; \quad \tilde{u} = D^{1/2}u;$$

тогда предобуславливатель примет вид $\bar{M} = (I - \tilde{L}^T)(I - \tilde{L})$.

Решение интерфейсной системы методом сопряжённых градиентов распараллелено с помощью технологии CUDA для вычисления на GPU. Все вспомогательные массивы, в частности r, p, q, z, а также матрица системы, предобуславливатель, вектор правых частей и вектор решения, хранятся в памяти графического ускорителя (см. **Алгоритм 3**). После завершения работы метода сопряжённых градиентов, массив u, в котором хранится приближение вектора решения, копируется в память CPU.

Для реализации операций суммы, скалярного произведения, копирования векторов и умножения вектора на скаляр использовались функции библиотеки CUBLAS. При выполнении матрично-векторного произведения, вектор хранится в текстурной памяти, которая кэшируется, что даёт более быстрый доступ и уменьшает временные затраты. Для вы-

Алгоритм 3 Алгоритм метода сопряжённых градиентов с предобуславливателем:

1: $S, \bar{M} \in \mathbb{R}^{N\times N}$ {$\bar{M}$ формируется на GPU, матрицы хранятся в CSR формате}
2: $u, r, p, q, z \in \mathbb{R}^N$ {Вектора хранятся в памяти GPU, копии на CPU нет}
3: $r_0 \leftarrow f$ {копирование векторов осуществляется с помощью cublasDcopy}
4: $u_0 \leftarrow 0$ {инициализация выполняется на GPU}
5: $z_0 \leftarrow \bar{M}r_0$ {выполняется на GPU}
6: $p_0 \leftarrow z_0$ {копирование векторов осуществляется с помощью cublasDcopy}
7: $\rho_0 \leftarrow (r_0, z_0)$ {здесь и далее $(\cdot,\cdot) = \sum\limits_P (\cdot,\cdot)^{(P)}$}
8: **while** $||r_i||_2/||b||_2 > \varepsilon$ **do**
9: $q_i \leftarrow Sp_i$ {выполняется на GPU}
10: $\alpha_i \leftarrow (r_i, z_i)/(q_i, p_i)$ {вычисляется с помощью функции cublasDdot}
11: $u_{i+1} \leftarrow u_i + \alpha_i p_i$ {операция выполняется с помощью функции cublasDasxpy}
12: $r_{i+1} \leftarrow r_i - \alpha_i q_i$ {операция выполняется с помощью функции cublasDasxpy}
13: $z_{i+1} \leftarrow \bar{M}r_{i+1}$ {выполняется на GPU}
14: $\rho_{i+1} \leftarrow (r_{i+1}, z_{i+1})$ {вычисляется с помощью функции cublasDdot}
15: $\beta_{i+1} \leftarrow \rho_{i+1}/\rho_i$
16: $p_{i+1} \leftarrow z_{i+1} + \beta_{i+1}p_i$ {последовательное использование функций cublasDscal и cublasDasxpy}
17: **end while**

числения каждой координаты вектора результата, используется от 2-х до 32-х потоков, в

зависимости от разреженности матрицы. Вычисление предобуславливателя выполняется либо на GPU, либо на CPU, в зависимости от его типа.

Предобуславливатель SSOR вычисляется на центральном процессоре. Разреженность матрицы K равна разреженности матрицы S и поэтому, выгоднее её так же хранить в компактном формате. Теоретически, метод сопряжённых градиентов решает систему не более, чем за N итераций. Пусть система решается за $\tilde{N}$ итераций, значит при вычислении $\bar{K}^T\bar{K}$ требуется N матрично-векторных произведений, плюс $\tilde{N}$ матрично-векторных произведений $z_{k+1} = \bar{M}r_{k+1}$ для решения системы методом сопряжённых градиентов. Если не вычислять $\bar{M}$ явно, а произведение $z_{k+1} = \bar{M}r_{k+1}$ заменить на два последовательных друг за другом матрично-векторных произведения $\tilde{z}_{k+1} = \bar{K}r_{k+1}$ и $z_{k+1} = \bar{K}^T\tilde{z}_{k+1}$, то нам потребуется всего $2\tilde{N}$ матрично-векторных произведений, что меньше, чем $N + \tilde{N}$ при явном нахождении матрицы $\bar{M}$. Поэтому, в реализации метода сопряжённых градиентов с SSOR предобуславливателем, произведение $z_{k+1} = \bar{M}r_{k+1}$ заменено на $z_{k+1} = \bar{K}^T\bar{K}r_{k+1}$.

Масштабирование системы и вычисление предобуславливателя DIP выполняется на GPU. Как и в случае с SSOR, разряженности матриц совпадают, поэтому они хранятся в разряженном формате. Произведение $z_{k+1} = \bar{M}r_{k+1}$ также заменяем на два матрично-векторных произведения $y = (I - \tilde{L})r_{k+1}$ и $z_{k+1} = (I - \tilde{L}^T)y$, выполняемых на GPU.

Применение предобуславливателя SSOR сокращает число итераций при решении системы в полтора раза, что позволяет покрыть затраты на его создание и на дополнительное матрично-векторное произведение. В результате чего, нет выигрыша по времени вычислений в сравнении с диагональным предобуславливателем. Аналогичные результаты получены для систем разрывного метода Галёркина [10]. Отличие в свойствах матрицы S и матриц систем разрывного метода Галёркина, позволило значительно уменьшить число итераций при использовании предобуславливателя DIP, которое стало примерно равным с SSOR.

Для решения интерфейсной системы (3) реализован блочный вариант **Алгоритма 3**, использующий несколько GPU. При разделении на блоки, матрица S системы (3) представлялась графом, имеющим число вершин равное размерности S, где s_{ij} — элемент матрицы расположенный в i-ой строке и j-м столбце. Каждой вершине графа матрицы S, на основе вычисленного разделения многоуровневым алгоритмом [11], ставится в соответствие номер графического ускорителя, исходя из которого вершины графа делятся на внутренние и граничные (связанные хотя бы с одной вершиной, имеющей другой номер ускорителя).

При помощи полученного разделения, в каждом блоке формируется несколько матриц S_k, где k — номер блока. В каждом блоке выделяется несколько типов матриц: $S_k^{[i,i]}$ — матрица, элементы которой связывают внутренние вершины; $S_k^{[i,b]}$, $S_k^{[b,i]}$ — матрицы, связывающие внутренние вершины с граничными; $S_k^{[k,m]}$ — матрица, связывающая граничные вершины k-го блока с граничными вершинами m-го блока. Здесь $k, m \in [1, N]$, где N — число блоков. Используя эти обозначения, матрица S примет вид

$$S = \begin{pmatrix} S_1^{[i,i]} & S_1^{[i,b]} & 0 & 0 & \cdots & 0 & 0 \\ S_1^{[b,i]} & S_1^{[1,1]} & 0 & S_1^{[1,2]} & \cdots & 0 & S_1^{[1,N]} \\ 0 & 0 & S_2^{[i,i]} & S_2^{[i,b]} & \cdots & 0 & 0 \\ 0 & S_2^{[2,1]} & S_2^{[b,i]} & S_2^{[2,2]} & \cdots & 0 & S_2^{[2,N]} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & S_N^{[i,i]} & S_N^{[i,b]} \\ 0 & S_N^{[N,1]} & 0 & S_N^{[N,2]} & \cdots & S_N^{[b,i]} & S_N^{[N,N]} \end{pmatrix}.$$

При матрично-векторном произведении $q = Sp$ на каждом GPU вычисляются два вектора:

$$q_b^k = S_k^{[b,i]} p_k + \sum_{m=1}^{m \leqslant N} S_k^{[k,m]} p_b^m; \qquad q_k = S_k^{[i,i]} p_k + S_k^{[i,b]} p_b^k, \tag{5}$$

где k — номер GPU. Это позволяет снизить затраты связанные с обменом между блоками на каждой итерации метода сопряженных градиентов, так как для выполнения последующих операций с векторами требуется обмен q_b^k, которые при минимизации границ имеют размер гораздо меньший, по сравнению с размером q.

Решение СЛАУ методом сопряженных градиентов требует меньше временных затрат в алгоритме, использующем GPU (см. таблицу 2), при этом ускорение $s(1)_{CPU} = 72$, при разделении на 16 подобластей, и $s(1)_{CPU} = 94$ при разделении на 1024 подобласти.

Таблица 2. Время решения интерфейсной системы, часы : мин. : сек.

n_Ω	CPU	1 GPU	2 GPU	4 GPU	6 GPU	8 GPU
16	> 8:00:00	0:15:12	0:07:01	0:03:25	0:04:41	0:04:24
32	> 8:00:00	0:16:23	0:10:30	0:07:10	0:05:17	0:04:34
64	5:01:07	0:04:47	0:02:54	0:01:38	0:01:22	0:01:12
128	—	—	0:02:07	0:01:18	0:01:06	0:01:00
256	—	—	0:01:46	0:01:10	0:01:01	0:00:56
512	—	—	0:01:25	0:01:03	0:00:56	0:00:53
1024	1:48:22	0:01:09	0:01:16	0:00:59	0:00:54	0:00:52

Использование нескольких GPU даёт максимальные ускорения $s(8)_{CPU} = 251$ для $n_\Omega = 64$ и $s(8)_{GPU} = 3.5$ для $n_\Omega = 16$. С увеличением числа подобластей количество ненулевых элементов, в полученной матрице дополнения Шура, уменьшается — это приводит к тому, что эффективность использования нескольких GPU, для решения интерфейсной системы, снижается. Так, например, при разделении на 1024 подобласти $s(8)_{GPU} = 1.3$. *Шаг 6* в **Алгоритме 1**, отвечающий за нахождения решений на внутренних узлах, также выполняется на GPU. Произведение $A_{II}^{-1} f_I$ уже вычислено на *Шаге 4*, матрица A'_{IB} получена на *Шаге 2*. Поэтому, необходимо выполнить лишь две операции — матрично-векторного произведения и разности векторов, которые выполняются на GPU.

В ходе выполнения вычислительных экспериментов минимальные значения суммарного времени формирования и решения системы для дополнения Шура (3) получены при $n_\Omega = 1024$ (см. таблицы 1 и 2). При выполнении этих шагов только центральным процессором потребовался 1 час 48 мин. В случае использования только одного GPU, для формирования и решения СЛАУ, затраты сократились в 22 раза. Минимальное время вычислений на графических ускорителях получено на восьми GPU — $t(8)_{GPU} = 2$ мин. Формирование системы (3) на центральном процессоре и решение на одном графическом ускорителе выполнено за полторы минуты. Наименьшие суммарные затраты потребовались при формировании системы на CPU и её решении на восьми GPU.

Система (3) решалась также на гибридном кластере, оснащенном графическими ускорителями. Следующие варианты рассмотрены: 8 процессов MPI, выполняемые в одном модуле; по 4 процесса в двух модулях; по 2 процесса в четырех модулях и по одному процессу

в 8 модулях. Коммуникации обеспечивались коллективными функциями, которые вызывались одним из потоков OpenMP, запускаемых внутри каждого процесса MPI.

Таблица 3. Время решения (3) на нескольких вычислительных модулях, мин. : сек.

n_Ω	Nnz/N	1×8	2×4	4×2	8×1
16	266826696/66030	5:21	4:28	3:28	3:53
64	90931750/70620	1:18	1:12	1:11	1:35
128	58592886/75732	1:08	1:00	0:59	1:26
256	38066007/81753	1:06	0:57	1:06	1:27
512	23955996/88248	1:03	0:55	1:08	1:42
1024	14749329/95523	1:05	0:54	0:54	1:50

Результаты экспериментов показывают, что затраты на решение (3) существенно зависят от распределения вычислений между CPU и GPU при выполнении матрично-векторного произведения. В случае $n_\Omega = 16$ (таблица 3) основные затраты приходятся на слагаемое $\hat{q}_k^b = \sum\limits_{m=1, m\neq k}^{m<n_p} S_k^{[b_k,b_m]} p_m^b$ из (5), которое вычисляется на CPU. Основной причиной этого являются большие размеры блоков $S_k^{[b_k,b_m]}$ и неравномерное распределение их между параллельными процессами/потоками. В подобных случаях можно перенести эту операцию на графический ускоритель. Приведенные в таблице 3 результаты показывают, что размещение вычислений по разным вычислительным модулям привело к ускорению вычислений в 1.2–1.5 раза, вызванное скорее всего конкуренцией за кэш-память CPU.

Более равномерное разделение граничных вершин и много меньшие (на два, три порядка) размеры $S_k^{[b_k,b_m]}, m \neq k$, при $n_\Omega = 1024$, приводят к эффективному вычислению $\hat{q}_k^b$ на CPU, а в результате — к доминированию затрат на операции над векторами (см. **Алгоритм 3**). В этом случае обращения к функции `cublasDdot` при вычислении скалярных величин составляют около 90% времени вычислений (варианты 1×8, 2×4, 4×2).

Ускорение вычислений при увеличении числа подобластей связано, в основном, с увеличением разреженности систем за счет уменьшения Nnz и увеличения N. В результате уменьшается общее число арифметических операций, приходящихся на матрично-векторные произведения при решении системы (3).

Распределение блоков матрицы между вычислительными модулями, при помощи обмена сообщениями MPI также позволило снять ограничение (объем памяти ускорителей в пределах одного вычислительного модуля) на размер решаемой интерфейсной системы (3).

6. Заключение

Метод дополнения Шура позволяет распределить вычисления между CPU и графическими ускорителями при решении СЛАУ на гибридных вычислительных системах, чем обеспечивает их более сбалансированное и эффективное использование.

Полученные результаты показывают, что при использовании метода дополнения Шура оптимальный выбор алгоритма формирования матрицы дополнения Шура зависит от размера подобластей/подматриц, на которые делится расчётная сетка или матрица коэффициентов системы. Если в одной подобласти находится относительно небольшое число ячеек

сетки (< 5000) или неизвестных (< 1500 — для внутренних и < 2500 — для граничных), то, для формирования матрицы дополнения Шура, эффективнее использовать прямые методы нахождение обратных матриц, и, как следствие, задействовать только CPU. В другом случае наиболее эффективны итерационные алгоритмы, использующие для вычислений несколько графических ускорителей.

Интерфейсная система уравнений дополнения Шура эффективно решается на нескольких GPU итерационными методами, в этом случае время решения сокращается в десятки и сотни раз.

Вычислительные эксперименты выполнены на вычислительных модулях гибридного кластера «Уран» ИММ УрО РАН, которые содержат два процессора Intel Xeon E5675 и восемь графических ускорителей NVIDIA Tesla M2090.

Литература

1. Haynsworth E.V. On the Shur Complement // Basel Mathematical Notes. – 1968. – №20. – 17 p.

2. Фаддеев Д.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. – М. Физматгиз, 1960. – 656 с.

3. Przemieniecki J.S. Theory of Matrix Structural Analysis. – New York: McGaw-Hill, 1968. – 480 p.

4. Giraud L., Haidar A., Saad Y. Sparse approximations of the Schur complement for parallel algebraic hybrid solvers in 3D // Numerical Mathematics. – 2010. – V.3 – P. 276-294.

5. Rajamanickam S., Boman E.G., Heroux M.A. ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms // IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS), 21-25 May 2012. – P. 631-643.

6. Корнеев В.Г., Енсен С. Эффективное предобуславливание методом декомпозиции области для p-версии с иерархическим базисом // Известия вузов. Математика. – 1999. – Т. 444, №5. – С. 37–56.

7. Копысов С.П., Красноперов И.В., Рычков В.Н. Объектно-ориентированный метод декомпозиции области // Вычислительные методы и программирование. – 2003. – Т.4, №1. – С. 176–193.

8. Kopysov S.P., Krasnopyorov I.V., Novikov A.K., Rychkov V.N. Parallel Distributed Object-Oriented Framework for Domain Decomposition // Domain Decomposition Methods in Science and Engineering. – Springer, 2005. – V. 40. — P. 605-614.

9. Копысов С.П. Оптимальное разделение области для параллельного метода подструктур // Сб. трудов Пятого Всероссийского семинара «Сеточные методы для решения краевых задач и приложения». – Казань: Изд-во КГУ, 2004. – С. 121–124.

10. Копысов С.П., Новиков А.К., Сагдеева Ю.А. Решение систем уравнений метода Галёркина с разрывными базисными функциями на графическом ускорителе // Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. – 2011. – №3. – С. 137-147

11. Karypis G., Kumar V. Parallel multilevel k-way partitioning scheme for irregular graphs // SIAM Rev. 1999. – V.41, №2. – P. 278 – 300.

Эффективный запуск гибридных параллельных задач в гриде*

А.П. Крюков[1,2,+], М.М. Степанова[3], Н.В.Приходько[4],
Л.В. Шамардин[1], А.П. Демичев[1,2]

[1]Московский государственный университет имени М.В. Ломоносова,
[2]Национальный исследовательский центр «Курчатовский институт»,
[3]Санк-Петербургский государственный университет,
[4]Новгородский государственный университет имени Ярослава Мудрого

В работе рассматривается способ эффективного запуска в гриде гибридных задач, совместно использующих технологии MPI и OpenMP. Для гибкого управления параметрами запуска параллельных задач на суперкомпьютерных (СК) ресурсах была расширена спецификация языка описания задач. Поддержка новых атрибутов реализована для всех ключевых компонентов инфраструктуры. Взаимодействие вебсервиса запуска с локальным менеджером ресурсов организовано через специальные обработчики разных типов заданий (single/openmp/mpi/hybrid), что обеспечивает передачу локальному менеджеру СК правильных параметров для резервирования ресурсов и запуска задачи. Представленное решение было опробовано на гридполигоне, развернутом на базе промежуточного ПО ГридННС.

1. Введение

Современные прикладные и фундаментальные исследования требуют выполнения высокопроизводительных расчетов, использующих параллельные вычисления, что подразумевает применение вычислительных кластеров, которые поддерживают параллельный режим выполнения программ. Для получения максимальной производительности и скорости исполнения программ в режиме пакетной обработки необходимо запускать такие программы оптимальным образом с учетом особенностей конкретного кластера. При непосредственной работе на кластере пользователю хорошо известна его архитектура и конфигурация, поэтому он может точно указать все параметры ресурсов и запуска задачи. В гриде все сложнее – среда является гетерогенной, а ресурс, на котором будет выполняться задание, в общем случае, заранее неизвестен. Кроме того, сам процесс запуска в гриде существенно сложнее – это многоуровневая процедура, в которой участвует большое количество сервисов и компонентов промежуточного ПО. Выполнение задания в гриде включает:

- определение характеристик задания в описании;
- отправка пользователем задания в грид на сервис распределения нагрузки;
- поиск и выбор подходящего ресурса для выполнения конкретных задач композитного задания;
- передача задания на выбранный сайт;
- резервирование ресурса для задания;
- запуск и выполнение на конкретном ресурсе.

В общем случае, описание задания для запуска в гриде имеет довольно абстрактный вид и не зависит от типа ресурса. Конечным итогом прохождения заданием всех грид-уровней является формирование скрипта задачи, который будет запущен конкретным локальным менедже-

* Работа поддержана грантом РФФИ (№ 11-07-00434-а) и грантом Президента РФ (НШ-3920.2012.2)
+ E-mail: kryukov@theory.sinp.msu.ru

ром (ЛРМ) на конкретном ресурсе. Чтобы ЛРМ мог выполнить корректный запуск, ожидаемый пользователем, описание задачи должно быть исчерпывающим, а алгоритмы его обработки на всех уровнях очень тщательно проработаны.

На сегодняшний день многие грид-инфраструктуры позволяет проводить параллельные расчеты, однако, в каждой из них имеются свои ограничения по запуску параллельных задач по сравнению с обычным кластером. Сравнение возможностей и тестирование ППО современных грид-проектов показывает, что очень трудно обеспечить максимально простой и универсальный механизм, который был бы эффективен для оптимального резервирования ресурсов и обработки разных типов задач в гетерогенной среде [1]. Можно выделить следующие основные моменты, которыми определяются присущие всем гридам ограничения по сравнению с обычным кластером:

- возможности формата описания заданий;
- полнота публикуемой информации и гибкостью алгоритма поиска ресурса;
- различия в типах и конфигурации ЛРМ на сайте;
- реализация интерфейса к ЛРМ;
- специфичность требований к способу запуска и установке окружения для разных типов задач.

Общей проблемой в гриде при выполнении параллельных задач является генерация набора параметров для $MPIEXEC. С одной стороны, набор должен быть согласован с запрошенными ресурсами, с другой – весьма гибким для учета особенностей ПО и правильного запуска. Все существующие реализации с фиксированным набором типов заданий, где параметры запуска автоматически рассчитываются из требований к ресурсам, привлекают простотой с точки зрения описания заданий, но подходят лишь под ограниченный круг задач. В частности, это относится к разработкам, в основе которых лежит ПО GlobusToolkit, в том числе и к российскому проекту ГридННС [2].

Другие известные проекты без дополнительной конфигурации также пока обеспечивают лишь базовую функциональность для простейших MPI- и OpenMP-задач.

Так, в gLite в качестве языка описания заданий используется JDL. На грид-шлюзе (CreamCE), начиная с версии glite-ce-cream v.1.13, для резервирования ресурсов введен новый расширенный набор атрибутов: CPUNumber (полное число запрашиваемых cpu), SMPGranularity (минимальное число ядер на узел), HostNumber (полное число запрашиваемых узлов), WholeNodes (полное резервирование узла, т.е. все ядра). Для запуска параллельных заданий предназначен набор скриптов MPI-Start [3] – пользователь должен передать скрипт с явно установленными параметрами и вызовом MPI-Start. Итерфейс MPI-Start в сочетании с новым набором атрибутов, по сути, дает унифицированную связку с любым ЛРМ. Теоретически можно запустить все, но требуется очень аккуратная настройка как сайта, так и параметров запуска пользователем.

В проекте NorduGrid ARC описание заданий выполняется на языке xRSL. Для резервирования cpu в описании имеется единственный атрибут count, который в зависимости от настройки сайта может интерпретироваться как число узлов или число ядер. Более гибкий вариант резервирования на сайте может быть доступен путем настройки runTimeEnvironment (RTE). RunTimeEnvironment – это механизм установки среды на основе shell-скриптов. Три вызова (до создания PBS submit-скрипта, перед запуском задачи и после завершения) позволяют очень гибко настроить исполняющую среду для любых приложений. Однако, использование такого способа для резервирования ресурсов не является лучшим вариантом, поскольку требует либо унифицированной поддержки в рамках VO, либо знания пользователем деталей конфигурации конкретных сайтов

Самым законченным решением на сегодняшний день следует признать проект UNICORE, который имеет наиболее продуманную спецификацию описания заданий и ее полную поддержку на грид-шлюзе. Характерная особенность – четкое разделение функциональности, а именно: параметры описания из раздела Resources полностью определяют резервирование, а через механизм программных окружений ExecutionEnv можно точно задать параметры и опции запуска

задачи. Это позволяет обеспечить поддержку корректного запуска практически любых параллельных задач и специализированных программных пакетов.

Данное исследование направлено на разработку для инфраструктуры ГридННС универсального способа запуска сложных параллельных задач. Предлагаемое решение позволяет использовать грид-среду для эффективного запуска не только распространенных вариантов на базе MPI- и OpenMP-технологий, но и наиболее сложного комбинированного типа - гибридных (MPI+OpenMP)-задач.

Структура работы следующая. Во втором разделе описаны особенности гибридных задач и используемые методы их запуска. Третий раздел посвящен описанию методов запуска разных типов параллельных задач в среде ГридННС. Четвертый и пятый разделы описывают некоторые аспекты реализации предложенных методов. В шестом разделе представлены результаты тестов. В заключении перечислены основные результаты работы и возможные направления их развития.

2. Особенности гибридных задач и методы их запуска

Классические параллельные приложения реализуются на основе технологий MPI [4] или OpenMP [5]. Вариант на базе MPI хорошо зарекомендовал себя для работы на традиционных кластерных системах. OpenMP предназначен для распараллеливания на многоядерных узлах с общей памятью. Наиболее типичные варианты запуска параллельных задач на кластере из многоядерных узлов обсуждаются в [1].

Самым сложным из параллельных типов являются гибридные задачи. Особый интерес к ним вызван тенденцией к использованию для высокопроизводительных вычислений многоядерных архитектур и SMP-кластеров. Одним из наиболее эффективных подходов программирования для таких кластеров является гибридный, основанный на комбинированном использовании MPI и OpenMP. Гибридный подход предполагает, что алгоритм разбивается на параллельные процессы, каждый из которых сам является многопоточным. Таким образом, имеется два уровня параллелизма: параллелизм между MPI процессами и параллелизм внутри MPI процесса на уровне потоков. В такой модели программирования MPI используется для организации обмена между процессами, а OpenMP для многопоточного взаимодействия внутри процесса (в рамках одного узла). Как показывает практика [6-10], за счет укрупнения MPI-процессов и уменьшения их числа гибридная модель может устранить ряд недостатков MPI, таких как большие накладные расходы на передачу сообщений и слабая масштабируемость при увеличении числа процессов. Однако, производительность гибридной задачи очень сильно зависит от режима ее запуска и выполнения, который определяет соотношение числа MPI-процессов и OpenMP-потоков на одном вычислительном узле, а также способа привязки MPI-процессов к физическим процессорам системы. В случае неправильного запуска или некорректного выделения ресурсов производительность таких программ может существенно снижаться.

Стандартный способ организации многопользовательского доступа к вычислительным кластерам – использование системы управления пакетной обработкой заданий (например, Torque, SLURM, PBSPro и др.) Важнейшим аспектом для запуска параллельных и особенно гибридных задач является поддержка локальным менеджером корректного выделения и резервирования вычислительных ресурсов на время выполнения задачи. В частности, должно обеспечиваться управление динамическим распределением задач в больших системах и строгое ограничение процессов на подмножестве процессоров и памяти узла. Один из возможных механизмов такого типа – cpuset, реализованный на уровне ядра ОС Linux и доступный в Torque. Данный механизм позволяет "на лету" распределять вычислительные ресурсы между различными задачами, ограничивая использование оперативной памяти и процессоров/ядер "вычислительным доменом", которому присвоена задача.

Гибридные задачи являются разновидностью MPI-задач, поэтому их запуск выполняется с помощью утилиты mpirun (или mpexec):

```
mpirun [ options ] <program> [ <args> ]
```

Предварительно должно быть определено необходимое окружение (пути к библиотекам, переменные окружения и т.п.), а так же обеспечено количество свободных ресурсов в соответствии с указанными при запуске опциями [options].

Требования к ресурсам и, соответственно, оптимальный способ запуска гибридной задачи в значительной степени определяется ее кодом. Большинство программ разрабатываются без привязки к архитектуре кластера, на котором она будет компилироваться и выполняться. Для их нормального выполнения достаточно задать полное количество MPI-процессов (M) и количество потоков на один процесс (K), а также обеспечить монопольное резервирование на время выполнения MxK процессоров/ядер.

В Таблице 1 приведены различные варианты запуска гибридных задач на кластере. Наиболее универсальный метод состоит в том, чтобы обеспечить распределение одного MPI-процесса на один узел, при этом число OpenMP-потоков внутри процесса не должно превышать количество ядер на этом узле (строка 1 в Таблице 1). Такой способ приемлем на любых ЛРМ, в том числе, без продвинутой поддержки управления динамическим распределением. Однако, его нельзя считать эффективным на кластерах, состоящих из узлов с большим количеством cpu, поскольку в этом случае задача будет потреблять неоправданно большие ресурсы (занимает узлы полностью). Строка 2 из Таблицы 1 содержит другой вариант запуска, когда на одном узле размещается несколько многопоточных процессов. Такой способ повышает полезную загрузку кластера, но уже предъявляет высокие требования к ЛРМ. В третьей строке Таблицы 1 приведен общий формат запуска задачи со строгой привязкой к ресурсам. Это необходимо для программ, код которых оптимизирован под конкретную архитектуру вплоть до точной привязки распределения процессов к топологии сокетов и ядер на узлах. Такой вариант требует, во-первых, знания архитектуры на которой код компилируется и запускается, во вторых, также предъявляет высокие требования к ЛРМ.

Таблица 1. Методы запуска гибридных (MPI+OpenMP) задач на вычислительном ресурсе

	Метод запуска	Резервирование ресурсов (cpu) и параметры запуска
1	Запуск в режиме один MPI-процесс на узел при полном резервировании узла	`#PBS -l nodes=N:ppn=K, где K=SMPSize;` `$MPIEXEC -pernode ./hyb_task`

2	Запуск в режиме на один узел M MPI-процессов, каждый с числом потоков K	`#PBS -l nodes=N:ppn=K,`
		`export OMP_NUM_THREADS=L, где M*L=K`
		`$MPIEXEC -npernode M ./hyb_task`

3	Запуск в режиме отображения на ресурсы, задаваемого строкой опций	`#PBS -l nodes=N:ppn=K,`
		`export OMP_NUM_THREADS=L, где M*L=K`
		`$MPIEXEC[special_option_string] ./hyb_task`

3. Запуск параллельных задач в ГридННС

3.1 Общая архитектура ГридННС с точки зрения прохождения задания

ГридННС включает ряд базовых компонентов ПО GlobusToolKit-4, а также набор ключевых инфраструктурных RESTful-грид-сервисов оригинальной разработки. Подробности архитектуры среды ГридННС и детали реализации отдельных сервисов представлены в [2, 11].

Рассмотрим ее с точки зрения функционирования компонент, определяющих запуск задания. Общая схема прохождения задания представлена на Рисунке 1.

1. Пользователь составляет описание задания в формате JSON. Синтаксис описания задания является унифицированным и не зависит от типа локального менеджера ресурсов.
2. С помощью интерфейса управления заданиями pilot-cli (или веб-интефейса ВИГ) передает его системе распределения и контроля заданий – "Пилот", который выполняет все

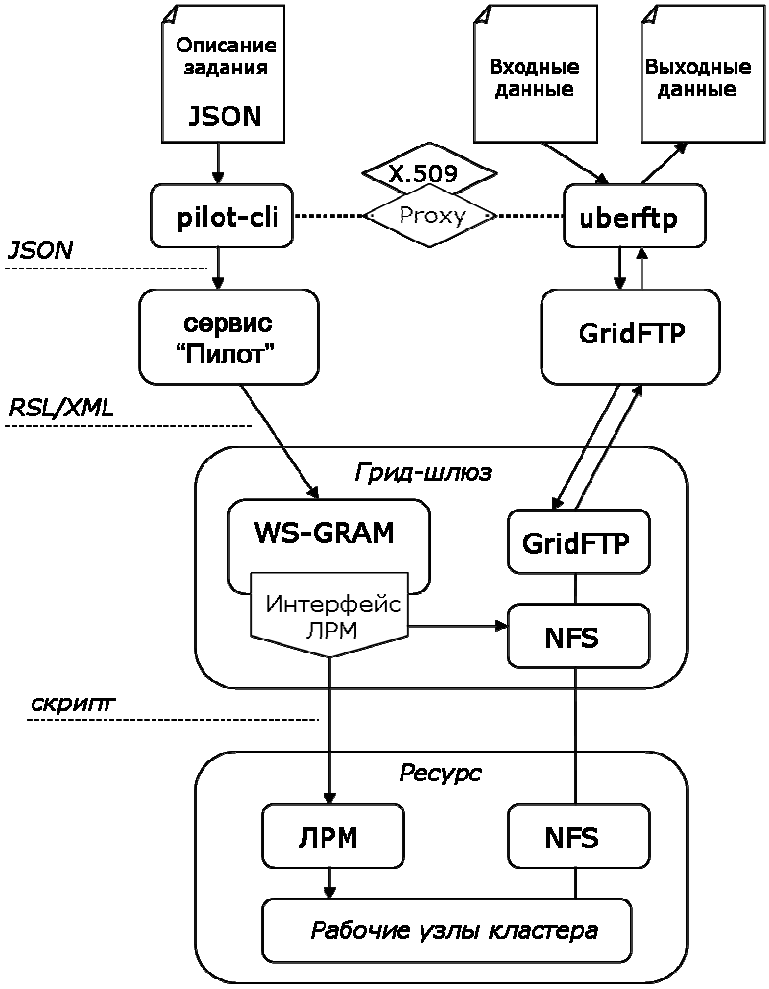


Рис. 1. Схема прохождения задания в ГридННС

функции брокера. Проверка корректности синтаксиса описания выполняется на уровне pilot-cli.

3. "Пилот" [12] реализован в виде комплекса RESTful-грид-сервисов, который обеспечивает запуск и координацию выполнения задач вычислительных ресурсах конкретного сайта. В частности, выполняются следующие действия:

   - аутентификация и авторизация на основе сертификата пользователя и его членства в ВО;
   - разбор JSON-описания задания;
   - запрос к информационному сервису о наличии ресурсов-кандидатов, удовлетворяющих критериям из описания задания;
   - выбор конкретного ресурса и подбор необходимых параметров запуска;
   - преобразование JSON-описания с учетом выбранных дополнительных параметров задачи в формат RSL/XML, используемый грид-менеджером ресурсов на грид-шлюзе;
   - передача на грид-шлюз для запуска;
   - отслеживание статуса состояния задач.

4. Грид-менеджер на шлюзе (построен на основе WS-GRAM из GlobusToolkit) принимает RSL-описание задачи, по которому интерфейс ЛРМ формирует скрип запуска для локального менеджера и передает его на ЛРМ.
5. ЛРМ запускает задачу на выполнение на вычислительных узлах кластера.

Исходная реализация ГридННС обеспечивает запуск задач только двух типов:

- "single": простая задача с последовательным кодом; запускается на одном ядре;
- "mpi": параллельная MPI-задача; запускается в режиме один MPI-процесс на одно ядро; для резервирования ресурса и запуска пользователю доступен один параметр описания – count, который определяет количество запрашиваемых ядер и задает число запускаемых процессов.

Отметим, что тип задачи пользователь не задавал явным образом в описании, он присваивался на стадии ее обработки сервисом "Пилот" в зависимости от значения параметра `count`: если `count` >1, то тип "mpi", если `count=1` или не задан, то "single". При формировании скрипта запуска для ЛРМ на грид-шлюзе выполнялась автоматическая подстановка способа запуска. Так, в случае "mpi" - посредством "`$MPIEXEC –n count <executable>`".

Очевидно, что поддержка более сложных вариантов запуска параллельных задач требует расширения возможностей языка описания заданий и новых алгоритмов их обработки для обеспечения резервирования ресурсов и запуска задачи. Наиболее простой путь состоит в расширении функциональности грид-сервисов для поддержки ряда новых типов заданий. Конкретная реализация необходимых новых компонентов определяется как текущим программным решением, так и используемыми внешними стандартами и спецификациями.

Синтаксис языка описания задания с представлением в формате JSON достаточно гибкий, и его расширение новыми атрибутами не вызывает затруднений. Однако, следует учитывать, что в дальнейшем это описание транслируется в RSL для обработки на грид-шлюзе сервисом WS-GRAM, а спецификация RSL допускает это только через механизм extensions.

Это потребовало существенной доработки в части алгоритмов обработки JSON-описания, поиска ресурса, подбора параметров запуска и формирования RSL важнейшего инфраструктурного сервиса "Пилот".

При выборе ресурса необходима исчерпывающая информация о текущем состоянии сайтов, которая предоставляется посредством информационного сервиса, использующего модифицированный вариант XML-реализации схемы GLUE 1.3. Добавление, при необходимости, новых элементов в публикуемую сайтами информацию о ресурсах не представляет больших сложностей.

Пользовательский интерфейс требует изменений в части проверки корректности синтаксиса JSON-описания задания.

Наконец, существенная модификация необходима для интерфейсов ЛРМ. Во-первых, нужен алгоритм для обработчика нового типа заданий. Во-вторых, расширение набора атрибутов описания задачи, усложнит и потребует модификации кода существующих обработчиков.

Принципы, положенные в процессе проектирования ГридННС позволили достаточно просто провести доработку ПО, чтобы обеспечить запуск дополнительных типов задач. В следующих разделах будут определены базовые типы задач и спецификации для их поддержки.

3.2 Типы задач

Для адекватной работы большинства пользовательских приложений и специализированных пакетов можно выделить следующий набор базовых типов задач, которые должна обеспечивать современная грид-инфраструктура:

- "single" – запуск задачи с последовательным кодом на одном ядре узла вычислительного кластера.
- "openmp" – запуск многопоточной (например, использующей технологию OpenMP) задачи на одном узле с возможностью резервирования под задачу узла либо целиком,

либо частично – резервирование запрошенного числа ядер на узле для монопольного использования задачей.

- "mpi" – запуск MPI-задачи с одним процессом на ядро без каких-либо требований к распределению ядер по узлам кластера.
- "hybrid" – запуск гибридной (MPI+OpenMP) задачи в режиме один многопоточный MPI-процесс на один узел.

3.3 Расширение языка описания задач и общие спецификации на реализацию сервисов

Спецификация описания задач была расширена следующим образом:

```
Секция definition:
   jobtype [single|openmp|mpi|hybrid]
   nodes [int]
   ppn [int]

Секция extensions:
   mpi_extra_arg [string]
```

Рис. 2. Расширение набора атрибутов описания задач

В секцию definition описания задачи вводится атрибут jobtype. Разрешенными типами заданий являются "single", "mpi", "openmp", "hybrid". В случае указания параметра jobtype сервис "Пилот" передает этот параметр на шлюз без изменений. Фактическая обработка параметра jobtype осуществляется на стороне шлюза в модуле стыка шлюза и ЛРМ. В этой же секции definitions добавляется возможность указания атрибутов для описания требований к ресурсам: nodes – число запрашиваемых задачей узлов и ppn – необходимое количество ядер на узле.

В случае, когда значения этих параметров заданы пользователем в описании задачи, они учитываются сервисом "Пилот" и шлюзом на разных стадиях жизни задания.

Так сервис "Пилот" использует параметры nodes и ppn на стадии выбора ресурса и, в общем случае, они должны учитываться совместно с count (общее число ядер, запрашиваемое для задачи). Для каждого из заданных параметров в описании задачи "Пилот" подходящими считаются только ресурсы со значением этого параметра >= "заданного". Если запрошенный набор не может быть удовлетворен хотя бы по одному из параметров, то задание не будет запущено, а пользователь получит сообщение - "нет подходящих ресурсов".

При поступлении задачи на грид-шлюз эти параметры учитываются им на стадии обработки модулем стыка шлюза с ЛРМ. Способ их обработки при формировании скрипта запуска на ЛРМ определяется атрибутом jobtype.

На стороне шлюза также реализуется поддержка двух вариантов обработки параметра ppn для целей резервирования – либо в точном соответствии с указанным ppn, либо резервирование узлов целиком, в зависимости от наличия поддержки cpuset в ЛРМ.

Также в секцию extensions добавляется поддержка (в настоящее время в статусе "experimental") расширения mpi_extra_args, в которой может передаваться строка альтернативных аргументов для $MPIEXEC. Для включения поддержки на шлюзе в jobmanager-pbs.conf указывается параметр use_mpi_extra_args (значения "no" или "yes", по умолчанию "no"). Использование значения "yes" для этой опции не рекомендуется для сайтов без поддержки cpuset, так как это может привести к потере контроля над ресурсами. Данный параметр предназначен для квалифицированных пользователей, которые хорошо представляют последствия своих действий.

В Таблице 2 приведены наборы атрибутов, допустимые к использованию для разных типов задач. Если в описании задания есть несоответствие в наборе обязательных параметров, то задание не будет отправлено в грид, а пользователь получит сообщение об ошибке на стадии валидации JSON-описания задания.

Таблица 2. Типы задач, поддерживаемые в ГридННС

Тип задач	Обязательные параметры в описании задачи	Дополнительные (необязательные) параметры в описании задачи
single	Нет	Нет
openmp	jobtype="openmp" ppn или count	environment: {"OMP_NUM_THREADS": <number>}
mpi	jobtype="mpi" count или пара {ppn,nodes}	Нет
hybrid	jobtype="hybrid" любая пара из {ppn,nodes,count} */ рекомендуется {ppn,nodes}	mpi_extra_args: <string> environment: {"OMP_NUM_THREADS": <number>}

4. Адаптация ПО "Пилота" для запуска гибридных задач

Базовая версия грид-сервиса "Пилот", распространяемая в составе комплекса программного обеспечения ГридННС, поддерживает две разновидности задач: однопроцессорные ("single") и обычные MPI-задачи ("mpi"). Добавление новых типов задач "openmp" и "hybrid" потребовало изменение алгоритма работы сервиса. Также был модифицирован алгоритм обработки задач типа "mpi" для учета дополнительных атрибутов ppn и nodes в процессе подбора ресурсов, удовлетворяющих требованиям задачи.

В схемы описания задач JSON Schema были добавлены определения новых полей ppn и nodes. Поскольку JSON Schema не позволяет задать сложные отношения между атрибутами, были также расширены правила верификации задач используемые загрузчиком описаний задач "Пилот". Проверяется выполнение следующих правил:

- для задач типа "single" проверяется отсутствие атрибутов ppn, nodes и count;
- для задач типа "mpi" проверятся наличие атрибута count, либо пары атрибутов nodes и ppn;
- для задач типов "openmp" и "hybrid" проверяется наличие атрибутов nodes и ppn.

Данная проверка правил выполняется как на стороне клиента при запуске задачи, так и на стороне сервера.

Также модификации потребовали модули генерации описания задач и выбора подходящих ресурсов на стороне сервера "Пилот". Поскольку язык промежуточного уровня RSL, используемый Globus Toolkit 4 не поддерживает атрибутов задач, логически соответствующих параметрам ppn и nodes, все атрибуты, связанные с запуском гибридных параллельных задач, передаются через расширения в описании задачи, что делает их доступными для интерпретации ПО грид-шлюза.

В модуле выбора подходящих ресурсов были внесены изменения, накладывающие следующие ограничения на выбор подходящих ресурсов. Во-первых, для всех задач рассчитываются значения недостающих атрибутов, если один из атрибутов count, ppn или nodes не указан. Во-вторых, выполняется проверка, что:

- количество процессов, запускаемых на одном узле (ppn) не превышает количества логических процессоров на узле ("logical slots" в информационной системе);
- суммарное количество узлов, необходимое для запуска задачи не превышает общее количество узлов в кластере ("physical slots").

Для задач типа "mpi", в которых указание ppn и nodes является опциональным, предполагается максимально плотная упаковка задач на узлах.

5. Взаимодействие грид-шлюза с менеджером локальных ресурсов

Взаимодействия веб-сервиса запуска с локальным менеджером ресурсов организовано через грид шлюз, который использует GRAM ПО Globus Toolkit. Из требуемых для запуска гибридных задач параметров count, ppn и nodes Globus Toolkit непосредственно поддерживает только передачу параметра count. Поэтому в спецификацию (см. п.3.3) введены атрибуты nodes и ppn, по которым "Пилот" генерирует расширение в RSL-описании, а на грид-шлюзе реализована его обработка в соответствии с типом заданий.

При формировании задачи для ЛМР на стороне шлюза реализована поддержка двух вариантов обработки параметра ppn для целей резервирования:

- обработка заданного значения ppn, то есть резервирование части узла; данный вариант может использоваться только в случае ЛРМ с поддержкой cpuset;
- замена заданного ppn на ppn_max, то есть резервирование узлов полностью; данный вариант должен использоваться в случае ЛРМ без поддержки cpuset.

Способ обработки ppn задается в конфигурационном файле jobmanager-pbs.conf через параметр use_full_node (значения "no" или "yes"; по умолчанию "yes").

С целью поддержки дополнительных опций $MPIEXEC также добавлена поддержка дополнительного параметра mpi_extra_args, в которой передается строка альтернативных аргументов. Возможность использования данного параметра определяется через параметр use_mpi_extra_args в конфиграционном файле jobmanager-<lrm>.conf.

Переданные параметры count, ppn, nodes, mpi_extra_args, учитываются шлюзом на стадии обработки модулем стыка шлюза с локальным менеджером ресурсов. Способ их обработки при формировании скрипта запуска на ЛМР определяется заданным jobtype. Рассмотрим различные значения параметра jobtype.

- **jobtype=single**

Запуск задания предполагает запуск одного однопоточного процесса на одном ядре рабочего узла кластера. Схема запуска повторяет стандартную схему запуска Globus Toolkit.

- **jobtype=mpi**

В этом случае выполняется запуск MPI-задачи с одним процессом на ядро без каких-либо требований к распределению ядер по узлам кластера. При генерации задания учитываются параметры ppn, nodes, count. Схема резервирования ресурсов ЛМР и запуска определяется следующим образом:

- если задан только count или только nodes, то резервирование и запуск осуществляется по стандартному алгоритму Globus Toolkit;
- если задан только ppn, то производится резервирование "1:<ppn_max>" и запуск "$MPIEXEC –n <ppn>";
- если заданы count+ppn, то производится резервирование "1:<ppn>" и запуск "$MPIEXEC –n <count>";
- если заданы count+nodes+ppn, count+nodes, nodes+ppn, то производится резервирование "<nodes>:<ppn>" и запуск "$MPIEXEC –n <count>". При этом неопределенный параметр определяется на основании соотношения count=nodes*ppn.

- **jobtype=openmp**

Запуск задания предполагает запуск одного многопоточного процесса на одном рабочем узле кластера. При генерации задания учитываются параметры ppn и count. Последний используется, если параметр ppn не указан. Дополнительно учитывается параметр

OMP_NUM_THREADS, переданный в разделе "extension.environment". Задание Torque/PBS запускается с резервированием nodes=1:<ppn>. В задание передается переменная среды OMP_NUM_THREADS со значением ppn или значением, переданным в "extension.environment", если оно указано.

- **jobtype=hybrid**

В этом случае предполагается, что будет запущена гибридная (MPI+OpenMP) задача в режиме один многопоточный mpi-процесс на один рабочий узел кластера. При генерации задания учитываются параметры ppn, nodes, count. При этом необходимо указание двух из трех параметров, оставшийся параметр автоматически рассчитывается из соотношения count=nodes*ppn. Задание Torque/PBS запускается с резервированием nodes=<nodes>:<ppn>. Дополнительно в задание передается переменная окружения OMP_NUM_THREADS со значением ppn или значением, переданным в "extension.environment", если оно указано. Если в задании указан параметр mpi_extra_args и его поддержка доступна на ресурсе, то запуск производится командой "$MPIEXEC $mpi_extra_args", в противном случае выполняется "$MPIEXEC -pernode".

6. Проверка методов запуска гибридных задач на полигоне ГридННС

Разработанное программное решение прошло тестирование на полигоне ГридННС, включающем:

- компьютер с пользовательским интерфейсом командной строки (pilot-cli);
- сервер с сервисом "Пилот";
- два сайта в конфигурации: шлюз ГридННС + ЛРМ Torque +кластер из 8-ядерных узлов. На одном сайте использовалась сборка Torque 3.0.4 с поддержкой cpuset, на другом – без такой поддержки.

В состав тестового набора входили исходные коды типичных параллельных программ (OpenMP-, MPI- и гибридных(MPI+OpenMP)), разработанных таким образом, чтобы выходные данные содержали подробную диагностику о количестве запущенных процессов/потоков и используемых ресурсах.

Все тестовые задания оформлялись в виде группы из двух задач – компиляции и параллельного запуска программы. Такое объединение задач компиляции и запуска в группу, означает, что они будут выполнены на одном и том же сайте, причем вторая задача является потомком первой. Резервирование ресурсов для этих задач происходит независимо. Задача компиляции запускалась как "single", а вторая - в соответствие с типом, указанным в описании ("openmp/mpi/hybrid").

Пример описания теста гибридной задачи представлен на рисунке 2. В данном примере задача COMPILE будет обработана по типу "single", а задача RUN в соответствие с типом "hybrid". Критерий корректного резервирования ресурсов и правильного запуска для этого теста следующий: количество запущенных mpi-процессов совпадает с числом выделенных узлов кластера (nodes), причем на каждом узле исполняется ровно один процесс, а количество omp-потоков, которое порождает каждый mpi-процесс, не превышает полного числа ядер узла.

Независимо от особенностей конфигурации ЛРМ сайтов (см. выше) разработанные алгоритмы обеспечивают правильный запуск. При этом сайт с поддержкой cpuset может более рационально использовать узлы кластера, а именно задействовать их не целиком, а в точном соответствии с заданными требованиями задачи.

```
{ "version": 2,
  "default_storage_base": "gsiftp://phys27.gridzone.ru/tmp/",
  "tasks": [
      { "id": "COMPILE",
        "children": [ "RUN" ],
        "definition":
        { "version": 2,
          "requirements": { "software": "mpich2" },
          "executable": "/bin/bash",
          "arguments":["-c",
             "mpicc -fopenmp -o test-hybrid test-hybrid.c"
                      ],
          "input_files": { "test-hybrid.c": "test-hybrid.c" },
          "output_files": { "test-hybrid": "test-hybrid" }
        },
       },
       { "id": "RUN",
         "definition":
         { "version": 2,
           "requirements": { "software": "mpich2" },
           "jobtype": "hybrid",
           "nodes": 4,
           "ppn": 3,
           "executable": "/bin/bash",
           "arguments": [ "-c", "./test-hybrid" ],
           "input_files": { "test-hybrid": "test-hybrid" },
           "stdout": "test-hybrid.out"
         }
       }
          ],
  "groups": [ [ "COMPILE", "RUN" ] ] }
}
```

Рис. 3. Пример описания тестового задания, включающий группу из двух задач – компиляция и запуск гибридной программы

Результаты проведенных тестов вместе с мониторингом состояния очередей и реальной загрузки кластерных узлов показали, что запуски всех типов задач приводили к корректному резервированию ресурсов кластера в соответствии с указанными в описании параметрами.

7. Заключение

Целью выполнения данной работы являлось повышение эффективности использования вычислительных ресурсов суперкомпьютерных центров, объединенных в грид-инфраструктуру с использованием ГридННС.

В ходе выполнения работы были систематизированы и формализованы методы запуска гибридных задач на вычислительном ресурсе, определены расширения спецификации языка описания заданий набором атрибутов, необходимых и достаточных для определения требований к выделяемым ресурсам, программной среде и параметрам запуска.

На основании этих результатов были разработаны спецификации и алгоритмы обработки новых атрибутов компонентами грид-сервисов на всех уровнях и выполнена программная реализация этих алгоритмов для грид-сервисов в среде ГридННС.

Проведенные тестовые испытания на полигоне ГридННС показали, что предложенные методы обеспечивают корректное резервирование ресурсов на вычислительном кластере в соответствии с типом запускаемой задачи.

Предложенное решение является достаточно общим подходом и может быть перенесено в другие грид-инфраструктуры. Разработанные методы позволяют существенно расширить спектр пользовательских грид-приложений, а также повысить качество обработки заданий и эффективность использования СК ресурсов.

Таким образом, полученные результаты позволяют повысить эффективность использования суперкомпьютеров, подключенных к гриду, за счет учета специфики параллельных задач и, как следствие, позволят пользователям более быстро выполнять инженерные и научные исследования.

В дальнейшем предполагается, что данная методика будет распространена на задачи с использованием графических процессоров, которые получают широкое распространение на современных вычислительных установках.

Литература

1. M.M. Stepanova, O.L. Stesik. Running Parallel Jobs on the Grid // Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 5rd Intern.Conf. (Dubna, 16-21 July, 2012).– Dubna: JINR, 2012, pp.383-387, ISBN -5-9530-0345-2
2. В.А. Ильин, В.В. Кореньков, А.П. Крюков. ГридННС: состояние и перспективы // Труды 5-й международной конференции "Распределенные вычисления и Грид-технологии в науке и образовании" (Дубна, 16-21 июля, 2012 г.).-Дубна: ОИЯИ, с. 332-336
3. MPI-Start // URL: http://grid.ifca.es/wiki/Middleware/MpiStart/
4. OpenMP Application Program Interface Version 3.1, July 2012 // URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf
5. MPI: A Message-Passing Interface Standard Version 2.2, September 2009 // URL: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf
6. Makris I. Mixed Mode Programming on Clustered SMP Systems // MSc in. High Performance Computing. The University Of Edinburgh, 2005.
7. R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes // In Proc. of 17th Euromicro Int'l Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009), pages 427–236, 2009.
8. A. Rane and D. Stanzione. Experiences in tuning performance of hybrid MPI/OpenMP applications on quad-core systems // In Proc. of 10th LCI Int'l Conference on High-Performance Clustered Computing, 2009.
9. Глазкова Е.А., Попова Н.Н.Анализ эффективности гибридного параллельгого программированиия на примере системы BLUE GENE/P 2009 // URL: http://agora.guru.ru/abrau2009/pdf/36_NSSI_2009_Abrau-2009.pdf
10. MPI Forum Hybrid Programming Working Group (Lead: Pavan Balaji) // URL: http://meetings.mpi-forum.org/mpi3.0 hybrid.php (July 2010).
11. Крюков А.П., Демичев А.П., Ильин В.А., Шамардин Л.В., Основные подходы к построению грид-инфраструктуры национально нанотехнологической сети // В сборнике Вычислительные технологии в естественных науках. Перспективные компьютерные системы: устройства, методы и концепции. Труды семинара, Таруса 24 марта 2011 г., Под ред. Р.Р.Назирова, Л.Н.Щура, ИКИ РАН, серия Механика, управление и информатика, с. 51-68
12. Демичев А.П., Ильин В.А., Крюков А.П., Шамардин Л.В., Реализация программного интерфейса грид-сервиса Pilot на основе архитектурного стиля REST // Вычислительные методы и программирование, том 11, с. 62-65

KernelGen – прототип распараллеливающего компилятора C/Fortran для GPU NVIDIA на основе технологий LLVM*

Н.Н. Лихогруд[1], Д.Н. Микушин[2]

[1]Факультет Вычислительной математики и кибернетики
Московского Государственного Университета им. М.В. Ломоносова,
[2]Institute of Computational Science, Università della Svizzera italiana

Проект KernelGen (`http://kernelgen.org/`) имеет цель создать на основе современных открытых технологий компилятор Fortran и C для автоматического портирования приложений на GPU без модификации их исходного кода. Анализ параллелизма в KernelGen основан на инфраструктуре LLVM/Polly и CLooG, модифицированной для генерации GPU-ядер и alias-анализе времени исполнения. PTX-ассемблер для GPU NVIDIA генерируется с помощью бекенда NVPTX. Благодаря интеграции LLVM-части с GCC с помощью плагина DragonEgg и модифицированного линковщика, KernelGen способен, при полной совместимости с компилятором GCC, генерировать исполняемые модули, содержащие одновременно CPU- и GPU-варианты машинного кода. В сравнительных тестах с OpenACC-компилятором PGI KernelGen демонстрирует бо̀льшую гибкость по ряду возможностей, обеспечивая при этом сравнимый или незначительно более высокий уровень производительности.

1. Введение

Широкое использование GPU в кластерных вычислительных системах требует массовой адаптации множества сложных приложений. Программные модели CUDA и OpenCL достаточно хорошо подходят для небольших программ с ярко выраженным вычислительным ядром. Однако для сложных приложений, состоящих из множества отдельных блоков, таких как математические модели, сложность настройки взаимодействия оригинального кода и кода для GPU многократно возрастает. Многие компании и научные группы по-прежнему откладывают портирование своих приложений, так как это приводит к возрастанию издержек на сопровождение и развитие нескольких различных версий одной и той же функциональности для CPU и GPU. Увеличить эффективность портирования призваны следующие типы технологий разработки:

- **Директивные расширения существующих языков высокого уровня с ручным управлением параллелизмом, по аналогии с OpenMP.** Данный тип технологий позволяет автоматически преобразовывать исходный код программы в гибридный CPU-GPU код, основываясь на заданных пользователем управляющих директивах. Для стандартизации набора директив в языках C/C++/Fortran коммерческими разработчиками подобных решений созданы консорциумы OpenACC [8] и OpenHMPP [9]. Аналогичный набор директив, но уже для ускорителей архитектуры Many Integrated Core (MIC) развивается компанией Intel [10]. В рамках систем F2C-ACC [11] и САПФОР [12] предложены наборы директив для преобразования исходного кода на языке Fortran в гибридную форму, причём САПФОР проводит распараллеливание как на уровне GPU, так и на уровне нескольких GPU-узлов, объединённых в MPI-кластер.

*Работа поддержана грантом HP2C (`hp2c.ch`), тестирование велось на оборудовании компании NVIDIA, кластере «Ломоносов» МГУ им М.В. Ломоносова и кластере «Tödi» Швейцарского национального суперкомпьютерного центра (CSCS).

В целом, несмотря на бо̀льшую гибкость, директивные расширения все же требуют значительного участия программиста в организации корректных и эффективных вычислений. Компиляторы некоторых из приведённых директивных расширений реализуют проверку параллельности циклов и непротиворечивости директив, в других такая проверка отсутствует. Часто возникают ситуации, в которых компилятор слишком «осторожен» при принятии решений на основе внутреннего анализе циклов и производит распараллеливание только при наличии дополнительных указаний от пользователя. Большинство директивных расширений не поддерживают генерацию GPU-ядер для циклов, в которых присутствуют вызовы функций из других модулей компиляции или библиотек, что существенно ограничивает применимость подобных технологий в больших проектах.

- **Специализированные языки (domain-specific languages, DSL), со встроенными средствами параллелизма, ориентированные на определённый класс задач.** В последние годы было предложено множество различных DSL- и Embedded DSL-языков со встроенными средствами параллелизма. Их основная идея состоит в том, чтобы приблизить синтаксис к характерным объектам и действиям задачи, в то же время исключив из языка конструкции, привязывающие реализацию к конкретной архитектуре. Обработка возникающего нового уровня абстракции производится компилятором или source-to-source процессором, генерирующим код для всех целевых архитектур. Так, в работе [1] предложен Си-подобный DSL-язык для выражения вычислений на сетках с учетом начальных и граничных условий, а также соответствующий компилятор с бекендами для различных CPU (SSE, AVX). В работе [5] аналогичная задача решается при помощи eDSL, основанного на шаблонах C++. Поддерживается генерация кода для CPU и GPU NVIDIA. Другой eDSL на основе C++ – Halide [6], с поддержкой x86-64/SSE, ARM v7/NEON и GPU NVIDIA, нацелен на эффективную реализацию методов обработки изображений. К классу DSL/eDSL можно отнести систему Nemerle Unified Device Architecture (NUDA) [7], позволяющую создавать новые расширения языка Nemerle и соответствующие плагины для компилятора.

  Тестирование DSL/eDSL как правило проводится в сравнении с программами, написанными вручную, что не позволяет судить о том, насколько существенен может быть выигрыш в эффективности DSL-языков по сравнению с директивными расширениями. Кроме того специализация ограничивает конкурентную среду, так как у каждого языка как правило существует только один разработчик. Глубокое сравнительное тестирование затруднено необходимостью реализации бенчмарков на каждом используемом языке.

- **Автоматический анализ параллельности кода с помощью эвристик или методов многогранного анализа.** Технологии данного типа предназначены для вычисления зависимостей данных и пространств итераций с помощью точных методов или эвристик. Эвристики в настоящее время являются частью большинства коммерческих компиляторов, когда в составе открытых и экспериментальных решений можно найти более сложные методы, такие как многогранный анализ (polyhedral analysis). В работе [13] для компилятора GCC реализовано расширение для автоматической идентификации параллельных циклов и генерации для них кода на OpenCL. Аналогичное расширение PPCG для компилятора Clang (LLVM) [14] способно преобразовывать код на C/C++ в CUDA-ядра. Обе технологии преобразуют вычислительные циклы из внутреннего представления компилятора в код на OpenCL или CUDA при помощи системы многогранного анализа Chunky Loop Generator (CLooG) [15]. Source-to-source компилятор Par4all [16] преобразует код на языке C или Fortran в код CUDA, OpenCL или OpenMP с помощью системы многогранного анализа PIPS.

Явное программирование на CUDA, директивные расширения и DSL-языки в любом случае предполагают модификацию или переработку исходного кода программы. По этой причине портирование больших приложений на GPU с помощью этих технологий сильно затруднено. Если же приложение портировано лишь частично, то синхронизация данных между хостом и GPU может значительно влиять на общую производительность. Так, при портировании только одного блока WSM5 модели WRF с помощью директив PGI Accelerator, время обменов данными составляет 40-60% общего времени [21].

На основе сопоставления свойств существующих технологий с требованиями, возникающими при портировании на GPU типичного вычислительного приложения, можно выделить ряд возможностей, имеющих потенциально наиболее важную роль при планировании и разработке программных систем следующего поколения:

- Поддержка широкого множества *существующих* языков программирования;
- Автоматическая оценка параллельности вычислительных циклов, не требующая внесения изменений в исходный код или каких-либо дополнительных действий со стороны пользователя;
- Генерация кода, полностью совместимая со стандартной хост-компиляцией;
- Минимизация обмена данными между памятью системы и GPU;
- Встраивание в существующие схемы распараллеливания, в первую очередь – MPI.

Целью проекта KernelGen является создание компилятора, удовлетворяющего всем перечисленным условиям и проработка стратегии развития необходимых для этого технологий. Очевидно, что подобная система не может быть построена ни на основе директивных расширений, ни на основе DSL, в то же время в ней вполне могли бы быть использованы наработки исследовательских решений по автоматическому анализу циклов. KernelGen находится на стыке математических методов, теории компиляторов и практической реализации новой технологии. Работа такого рода требует именно смешанного исследовательского формата, поскольку развитие существующих компиляторов слишком инертно для значительных нововведений, а недостаточная ориентация на практику ведёт к преждевременной стандартизации методов, не отвечающим реальным потребностям.

Данная статья организована следующим образом. В разделе 2 предлагаются решения по организации процессов компиляции, линковки и генерации кода, а также нестандартная модель исполнения, позволяющая естественным образом обеспечить более эффективное взаимодействие параллельных частей кода на GPU. В части 3 излагается способ модификации существующей технологии анализа параллельности циклов для генерации GPU-кода. Разделы 4 и 5 посвящены, соответственно, необходимым дополнительным подсистемам исполнения приложений и сравнительному анализу работы тестовых задач.

2. Этапы преобразования кода

При разработке системы компиляции на основе существующих наработок значительную роль играет выбор наиболее подходящей базовой инфраструктуры по большому числу критериев: наличие фронтендов для различных языков, полнота и гибкость внутреннего представления, существование базового набора оптимизирующих преобразований и эффективных бекендов для целевых архитектур, динамика развития и поддержка со стороны сообщества разработчиков. Наиболее развиты по этим критериям компиляторы GCC, LLVM и Open64. Компилятор GCC поддерживает наибольшее число языков программирования, но не имеет бекендов для GPU, когда как LLVM и Open64 имеют бекенды для NVIDIA PTX ISA. Компилятор Open64 имеет фронтенды для C, C++ и Fortran, генерирует качественный

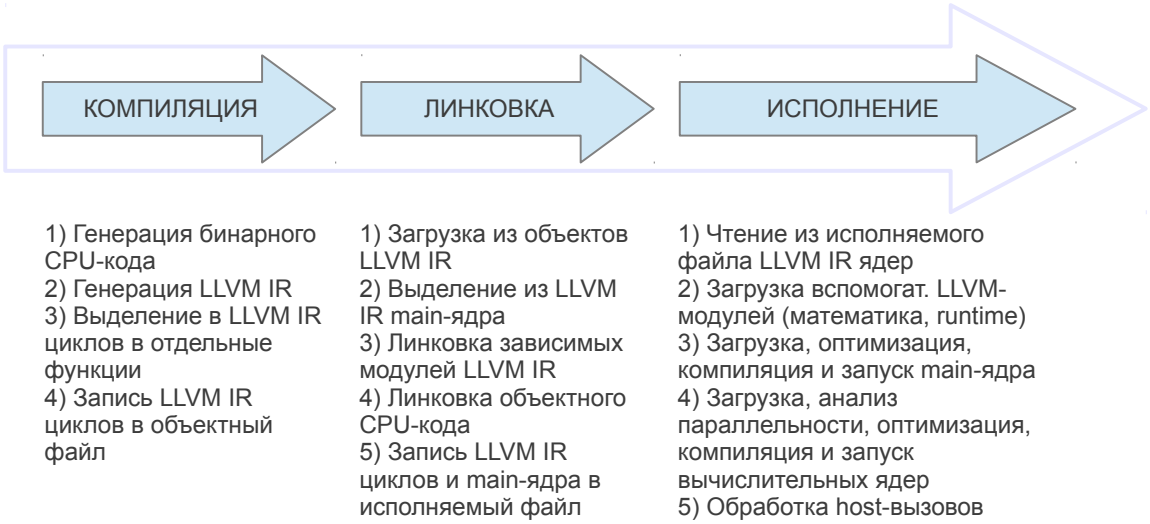


Рис. 1. Этапы преобразования кода компилятором KernelGen

код, но при этом, к сожалению, имеет сильно сегментированное сообщество разработчиков, развивающих множество отдельных веток кода в интересах коммерческих компаний и исследовательских организаций. Компилятор LLVM не имеет собственного фронтенда для языка Fortran, но способен при помощи плагина DragonEgg [17] использовать фронтенды компилятора GCC. При этом он имеет собственный GPU-бекенд NVPTX, имеет простое внутреннее представление (LLVM IR – intermediate language) и развивается намного более интенсивно, чем GCC и Open64. Из этих соображений, за основу для KernelGen был выбран LLVM.

Компилятор KernelGen работает напрямую с оригинальным приложением, не требуя каких-либо изменений ни в исходном коде, ни в системе сборки. За счёт использования фронтенда незначительно модифицированной версии GCC, он полностью совместим с его опциями, что гарантирует высокий уровень поддержки большого числа приложений. Чтобы обеспечить стандартный процесс сборки, в KernelGen используется схема, напоминающая LTO (link time optimization – инфраструктура компилятора для дополнительной оптимизации кода во время линковки): код для GPU сначала добавляется в отдельную секцию объектных файлов, затем объединяется и снова разделяется на отдельные ядра на этапе линковки. Окончательная компиляция GPU-ядер в ассемблер происходит при необходимости, уже во время работы приложения (JIT, just-in-time compilation). Схема основных этапов преобразования кода приведена на рис. 1.

В результате работы компилятора, исходное приложение преобразуется во множество GPU-ядер: одно или несколько *основных* ядер и множество *вычислительных* ядер. Основные ядра исполняются на GPU в одном потоке. Их задача – хранить данные, исполнять небольшие последовательные участки кода и производить вызовы вычислительных ядер и отдельных CPU-функций, которые невозможно или неэффективно переносить на GPU. Вычислительные ядра исполняются на GPU множеством параллельных нитей с полной загрузкой мультипроцессоров. Таким образом, максимальная доля кода выполняется на GPU, а CPU лишь координирует исполнение. В частности, при работе MPI-приложения каждый рабочий процесс в данном случае будет представлять собой GPU-ядро с небольшим числом CPU-вызовов MPI. Использование MPI дополнительно облегчается за счёт поддержки GPU-адресов в командах обмена данными [19]. В целом, такая модель исполнения имеет много общего с native-режимом Intel MIC, но работает на GPU, где скалярные вычислительные блоки способны достигать высокой эффективности без необходимости векторизации.

2.1. Компиляция

При компиляции отдельных объектов, генерируется как x86-ассемблер (таким образом, приложение по-прежнему работоспособно при отсутствии GPU), так и представление LLVM IR. Для разбора исходного кода используется компилятор GCC, чьё внутренее представление *gimple* преобразуется в LLVM IR с помощью плагина DragonEgg. Затем в IR-коде производится выделение тел циклов в отдельные функции, вызываемые через универсальный интерфейс вида

```
__device__ int kernelgen_launch(
    unsigned char* name, unsigned long long szargs,
    unsigned long long szargsi, unsigned int* args);
```

где name – имя или адрес функции (вместо имён в начале работы программы подставляются адреса), *args* – структура, агрегирующая аргументы вызова, *szarg* и *szargi* – размер списка аргументов и списка целочисленных аргументов (последний используется для вычисления хеша функции и поиска ранее скомпилированных ядер во время исполнения).

Стандартный механизм выделения каскадов вложенных циклов в функции LLVM *LoopExtractor* расширен, так чтобы цикл не заменялся, а дополнялся вызовом функции по условию:

```
if (kernelgen_launch(name, szargs, szargsi, args) == -1) {
    // Launch original loop.
}
```

С помощью данного условия runtime-библиотека KernelGen может переключать выполнение между различными версиями цикла. Например, если цикл определён как непараллельный, то *kernelgen_launch* возвращает -1, и код цикла начинает выполняться основным ядром в последовательном режиме. Тем не менее, данный цикл может содержать вложенные параллельные циклы, обработка которых будет проведена аналогичным образом. В конце концов, если весь каскад тесно вложенных циклов непараллелен, несовместим (например, содержит вызовы внешних CPU-функций) или оценен как неэффективный для GPU, то вся функция выгружается для работы на хосте с помощью вызова *kernelgen_hostcall*:

```
__device__ void kernelgen_hostcall(
    unsigned char* name, unsigned long long szargs,
    unsigned long long szargsi, unsigned int* args);
```

при котором GPU-приложение останавливает свою работу и передаёт данные и адрес функции для выполнения на CPU. Функции *kernelgen_launch* и *kernelgen_hostcall* работают в GPU-ядре и вызывают остановку его выполнения. После завершения работы другого ядра или CPU-функции, основное ядро продолжает работу. Хост-часть управляющих функций компилирует и выполняет заданную функцию с помощью интерфейса FFI (Foreign Function Interface).

Одним из специфических свойств KernelGen является хранение всех данных приложения в памяти GPU. Для того чтобы обеспечить его совместимость с наличием CPU-вызовов, реализована простая система синхронизации памяти. При попытке CPU-функции обратиться к памяти по адресу из диапазона GPU возникающий сигнал сегментации обрабатывается дублированием страниц из памяти GPU в страницы CPU-памяти, расположенные по тем же адресам. После завершения работы CPU-функции, изменённые CPU-страницы синхронизируют изменения с памятью GPU.

2.2. Линковка

Во время линковки отдельных объектов в приложение или библиотеку, LLVM IR также линкуется в один общий IR-модуль для main-ядра и по одному IR-модулю на каждый вычислительный цикл. IR-код погружается в исполняемый файл и в дальнейшем оптимизируется и компилируется в GPU код по мере необходимости во время работы приложения.

Специальной обработки требуют глобальные переменные. Синхронизация глобальных переменных между ядрами потребовала бы разработки для GPU динамического линковщика. Вместо этого, в начале работы программы на CPU передаются адреса всех глобальных переменных. Во время выполнения, виртуальные глобальные переменные заменяются на соответствующие фактические адреса. Это корректно, т.к. в LLVM глобальная переменная реализована как указатель на память, содержащую её логическое значение.

2.3. Модель исполнения

Основное ядро запускается в самом начале выполнения приложения и работает на GPU постоянно. Во время работы вычислительного ядра или CPU-функции основное ядро переходит в состояние активного ожидания и продолжает работу после завершения внешнего вызова. Для реализации данной схемы GPU должно поддерживать одновременное исполнение нескольких ядер (concurrent kernel execution) или временную выгрузку активного ядра (kernel preemption). Одновременное исполнение ядер доступно в GPU NVIDIA, начиная с Compute Capability 2.0, в GPU AMD такой возможности нет, но есть вероятность появления kernel preemption в одной из следующих версий OpenCL. По этой причине в данный момент KernelGen работает только с CUDA.

Вызовы *kernelgen_launch* и *kernelgen_hostcall* состоят из двух частей: device-функции на GPU и одноимённого вызова в CPU-коде, который выполняет, соответственно, окончательную генерацию кода и запуск вычислительного ядра или загрузку данных с GPU и запуск CPU-функции средствами Foreign Function Interface (FFI). Взаимодействие между частями может быть организовано посредством глобальной памяти GPU или pinned-памяти хоста. Однако для гарантированной передачи корректного значения необходимо обеспечить *атомарный* режим операций чтения и записи, доступность которого является определяющим фактором. По этой причине был реализован метод, использующий глобальную память.

Дополнительное препятствие взаимодействию GPU-ядра с другим ядром или CPU состоит в том, что данные нити (CUDA thread) хранятся в регистрах или локальной памяти. Это означает, что аргументы, переданные из основного GPU-ядра не могут быть использованы где-либо, кроме как в нём самом. Для преодоления этого ограничения, бекенд NVPTX изменён так, чтобы локальные переменные помещаются не в *.local*-секцию, а в *.global*, делая их доступными всем GPU-ядрам и хосту.

3. Генерация CUDA-ядер для параллельных циклов

Частью инфраструктуры LLVM является библиотека Polly [3] (от polyhedral analysis – многогранный анализ) – оптимизирующее преобразование циклов, основанное на CLooG. Оно способно распознавать параллельные циклы в IR-коде, оптимизировать кеширование за счёт добавления блочности, оптимизировать доступ к памяти за счёт перестановки циклов и генерировать код, использующий OpenMP. Для заданного кода CLooG строит *абстрактное синтаксическое дерево* (AST), а затем проводит расщепление циклов по некоторым измерениям. Благодаря возможности расщепления частично-параллельных измерений, для исходного цикла может быть найдено эквивалентное представление из одного или нескольких циклов, часть которых параллельна. Подобный подход используется довольно редко, большинство современных компиляторов ограничиваются проверкой параллельности измерений существующих циклов без глубокого анализа.

Polly работает с частями программы, для которых можно статически (без выполнения) предсказать поток управления и доступ в память в зависимости от фиксированного набора параметров. Такие части принято называть *статическими частями потока управления* (static control parts – SCoPs). Часть программы представляет собой SCoP при выполнении следующих условий:

Рис. 2. Этапы генерации CUDA-ядер для параллельных циклов компилятором KernelGen. Оптимизация происходит сразу для всего SCoP, генерация – для каждой функции в отдельности

1. Поток управления формируется условными операторами и циклами-счётчиками;
   (a) Каждый цикл-счётчик имеет одну индексную переменную с константным шагом изменения, верхняя и нижняя границы цикла заданы афинными выражениями, зависящими от параметров и индексных переменных внешних циклов;
   (b) Условные операторы сравнивают значения двух афинных выражений, зависящих от параметров SCoP и индексных переменных;
2. Обращения в память происходят со смещениями от указателей-параметров SCoP. Смещения задаются афинными выражениями от параметров SCoP и индексных переменных циклов.
3. Содержит вызовы только функций без побочных эффектов

Первое условие означает структурированность потока управления: код можно логически разбить на иерархию вложенных блоков, имеющих один вход и один выход, каждый блок полностью вложен в объемлющий. Запрещены конструкции, нарушающие структуру потока управления (*break*, *goto*). Использование афинных выражений позволяет применять аппарат целочисленного программирования для расчёта границ циклов и обращений в память в зависимости от параметров SCoP.

Если работать с высокоуровневым представлением программы (код на языке высокого уровня, абстрактное синтаксическое дерево), то множество приёмов программирования (например, арифметика указателей, циклы *while*, операторы *goto*) будут нарушать описанные требования. Если же перейти к промежуточному, близкому к ассемблеру представлению, то арифметика указателей будет реализована как набор арифметических операций с регистрами, и любой цикл, вне зависимости от типа (*for*, *while*), будет реализован как обычный условный переход. Следствием этого являются две полезные возможности KernelGen:

- распараллеливать *while*-циклы, когда как, например, стандартом OpenACC такая возможность не предусмотрена;
- распараллеливать циклы с адресной арифметикой, что, например, не поддерживается в PGI OpenACC

При адаптации Polly для получения GPU-ядер был использован существующий генератор кода OpenMP, работающий следующим образом. Если внешний цикл является параллельным, то его содержимое перемещается в отдельную функцию, с добавлением вызовов функций библиотеки libgomp – GNU реализации OpenMP. При этом распределение итераций по ядрам производит среда исполнения, а распараллеливается только самый внешний цикл. Для KernelGen в эту логику были внесены следующие изменения:

1. Отображение пространства итераций на нити GPU, с учётом необходимости объединения запросов в память нитей варпа (coalescing transaction);

2. Рекурсивная обработка вложенных циклов с целью использования возможностей GPU по созданию многомерных сеток нитей.

Пусть в заданной группе циклов можно распараллелить N тесно-вложенных циклов. Тогда ядро может быть запущено на решётке с числом измерений N (для CUDA $N \leq 3$). Для каждого измерения, распределяемого между нитями GPU, генерируется код, рассчитывающий положение нити в блоке и блока в сетке. Каждому параллельному циклу ставится во взаимно однозначное соответствие измерение решетки, причём в обратном порядке – внутреннему циклу соответствует измерение X (это позволяет объединять запросы в память). Для каждого параллельного цикла генерируется код, определяющий нижнюю и верхнюю границы части пространства итераций, которая должна быть выполнена нитью. Затем генерируется последовательный код цикла с изменёнными границами и шагом.

Схема этапов работы Polly, анализа и генерации кода приведена на рис. 2.

4. Дополнительные средства времени исполнения

Включение бекенда NVPTX для генерации GPU-кода в LLVM позиционировалось компанией NVIDIA как «открытие компилятора». Тем не менее, помимо того, что закрытым остаётся C/C++/CUDA-фронтенд, а clang обладает лишь минимальной поддержкой некоторых ключевых слов CUDA, недоступной также остаётся часть компилятора существенно важная для его применения в LLVM: библиотека математических функций C99. Поскольку в рамках закрытого компилятора CUDA эти функции реализованы в виде заголовочных файлов C/C++, их использование с другими языками на уровне LLVM невозможно, и пользователю NVPTX-бекенда доступны только функции, встроенные в аппаратуру (builtins), среди которых, например, нет точных версий функций sin, cos, pow. В KernelGen данная проблема решена путём конвертации заголовочных файлов в LLVM IR одним из двух способов: с помощью clang (требуется множество модификаций, полученный IR-код предположительно содержит некорректные части) или с помощью cicc (вызывается из nvcc). В последнем случае, IR-модуль можно получить, отправив на компиляцию пустой .cu-файл и выгрузив код IR-модуля из cicc с помощью отладчика. IR, сгенерированный cicc совместим с актуальной версией LLVM и позволяет производить линковку математической библиотеки и использующего её приложения на уровне IR-кода, вне зависимости от начального языка. В частности, благодаря этому KernelGen позволяет генерировать GPU-код для программ на языке Fortran, использующих любые стандартные математические функции.

Некоторые типы функций CUDA API, такие как выделение GPU-памяти и загрузка другого CUDA-модуля, всегда приводят к неявной синхронизации асинхронных операций. Поскольку схема работы KernelGen требует постоянного поддержания основного ядра в состоянии выполнения, необходимость выделения памяти или загрузки новых ядер в процессе его работы приведёт к блокировке. В этом отношении существующие версии CUDA создают для развития KernelGen определённые препятствия, вынуждая реализовывать нестандартные эквиваленты базовой функциональности.

Если синхронность выделения памяти на GPU со стороны хоста ещё можно считать разумным ограничением, то подтверждённая экспериментами синхронность вызовов *malloc*

внутри GPU-ядер явно избыточна, так как память для индивидуальных потоков выделяется заранее. Так как оба стандартных варианта не могут быть использованы, KernelGen выполняет начальную преаллокацию памяти для собственного динамического пула и управляет его работой.

JIT-компиляция вычислительных ядер предполагает, что вновь скомпилированные GPU-ядра будут загружаться на GPU в фоне работающего основного ядра. Обычно динамическую загрузку ядер можно произвести с помощью стандартных функций *cuModuleLoad* и *cuModuleGetFunction* CUDA Driver API. Однако, обе эти функции являются синхронными, предположительно, из-за неявного выделения памяти для хранения кода и статических данных. В данной ситуации при разработке KernelGen не оставалось иного выбора, кроме как реализовать загрузчик кода новых ядер вручную, предварительно создав для них пустую функцию-контейнер. Загрузчик основан на технологиях проекта AsFermi [4] и действует следующим образом. В начале работы приложение на GPU загружается достаточно больше пустое ядро (содержащее инструкции NOP). По мере того, как в процессе работы приложения требуется запускать вновь скомпилированные ядра, их код копируется как данные в адресное пространство контейнера, которое известно благодаря инструкции LEPC (получить значение Effective Program Counter). Контейнер размещает код множества небольших ядер друг за другом, создавая своеобразный динамический пул памяти для кода. При этом необходмо учитывать, что различные ядра могут использовать различное число регистров. Для этого загрузчик создаёт 63 фиктивных ядра (точки входа), использующих от 1 до 63 регистров с единственной инструкцией JMP для перехода по адресу начала требуемого ядра в контейнере.

Система синхронизации памяти между GPU и хостом использует вызов *mmap*, ограничивающий возможные диапазоны адресов величинами, кратными размеру страниц (4096 байт). Поэтому выравнивание всех данных GPU по границе 4096 было бы очень удобным упрощением на данном этапе. К сожалению, текущая реализация CUDA (5.0) учитывает настройки выравнивания данных при компиляции, но при этом игнорирует их во время исполнения. Обход этого дефекта реализован посредством выравнивания размеров всех данных по границе 4096 вручную с помощью функций библиотеки *libelf*.

5. Тестирование

KernelGen тестируется на трех типах приложений: тесты корректности, тесты производительности и работа на реальных задачах. Тесты корректности предназначены для контроля регрессивных изменений в генераторе кода, тесты производительности позволяют анализировать эффективность текущей версии KernelGen в сравнении с предыдущими сборками и другими компиляторами. При тестировании производительности предпочтение отдаётся сравнению с результатами других распараллеливающих компиляторов, поскольку в отличие от сравнения с кодом, оптимизированным вручную, это позволяет проанализировать достоинства и недостатки компилятора в своём классе систем.

Для тестирования были выбраны приложения, реализующие различные типовые алгоритмы на двумерной или трехмерной регулярной сетке с одинарной точностью (часть тестов адаптировано из материалов работы [2], описание и исходный код приведены в [22]). На рис. 3 показано насколько меняется скорость работы тестов, скомпилированных с помощью KernelGen по сравнению с версией PGI OpenACC. Соответствующие абсолютные времена и число регистров для каждой версии теста приведены в Табл. 1. Тесты *jacobi*, *matmul* и *sincos* реализованы на языке Fortran, остальные тесты – на C (также были проверены реализации тестов *wave13pt* и *laplacian* на языке C++, однако для данного сравнения они не пригодны, поскольку компилятор PGI не поддерживает директивы OpenACC в C++). KernelGen автоматически распознаёт наличие вложенных параллельных циклов внутри непараллельного цикла по числу итераций, когда как PGI делает это только при со-

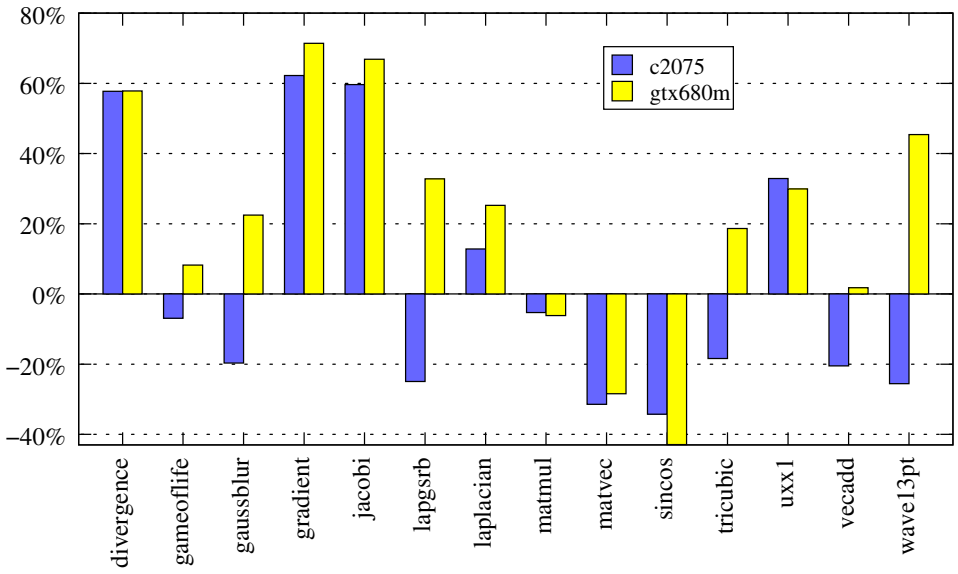


Рис. 3. Сравнение производительности вычислительных ядер некоторых тестовых приложений, скомпилированных KernelGen r1578 и PGI 12.10 на GPU NVIDIA Tesla C2075 (Fermi sm_20) и GTX 680M (Kepler sm_30)

ответствующей ручной расстановке директив OpenACC. Более низкая производительность KernelGen на тесте *matmul* обусловлена тем, что компилятор PGI реализует частичную раскрутку внутреннего цикла с редукцией на регистрах. Более низкая производительность ряда других тестов требует отдельного изучения.

Тестирование на больших вычислительных приложениях COSMO [20] и WRF [21] показало, что KernelGen способен генерировать корректные исполняемые файлы с поддержкой GPU за разумное время.

6. Заключение

В проекте KernelGen реализована оригинальная схема автоматического портирования кода на GPU, подходящая для сложных приложений. Не требуя никаких изменений в исходном коде, компилятор переносит на GPU максимально возможную часть кода, включая выделение памяти, тем самым создавая эффективную схему для преимущественно GPU-вычислений. KernelGen реализует средства автоматического анализа параллелизма циклов, основанные на LLVM, Polly и других проектах, расширяя их поддержкой генерации кода для GPU. Генератор GPU-кода основан на NVPTX-бекенде для LLVM, совместно развиваемом компанией NVIDIA и силами сообщества LLVM. Тестирование показало, что GPU-код, генерируемый KernelGen по своей эффективности сравним с коммерческим компилятором PGI.

Для того чтобы начать использовать компилятор в прикладных задачах, остается реализовать некоторые функциональные элементы. В частности, в нынешней версии отсутствует механизм накопления статистики эффективности исполнения GPU-кода для принятия решений о переключении на CPU-реализацию в неэффективных случаях. В генераторе параллельных циклов желательно добавить возможность использования разделяемой памяти и распознавание в коде идиомы редукции. Запуск вычислительных ядер на архитектуре Kepler может быть организован более эффективно за счёт использования динамического параллелизма.

Код KernelGen распространяется по лицензии University of Illinois/NCSA (за исключением плагина для GCC) и доступен на сайте проекта: `http://kernelgen.org/`.

Таблица 1. Сравнение времени исполнения (сек) и числа регистров (nregs) для вычислительных ядер некоторых тестовых приложений, скомпилированных KernelGen r1578 и PGI 12.10 на GPU NVIDIA Tesla C2075 (Fermi sm_20) и GTX 680M (Kepler sm_30)

Тест	NVIDIA Tesla C2075				NVIDIA GTX 680M			
	KernelGen		PGI		KernelGen		PGI	
	время	nregs	время	nregs	время	nregs	время	nregs
divergence	0.010920	18	0.017224	36	0.009811	20	0.015487	48
gameoflife	0.011383	21	0.010597	27	0.014631	21	0.015831	27
gaussblur	0.016835	56	0.013521	34	0.020240	51	0.024789	40
gradient	0.012687	21	0.020579	35	0.009314	22	0.015964	47
jacobi	0.009008	24	0.014380	26	0.007355	23	0.012274	31
lapgsrb	0.034975	55	0.026247	63	0.019294	40	0.025616	63
laplacian	0.009970	18	0.011246	32	0.008415	22	0.010537	48
matmul	0.001514	13	0.001434	33	0.001631	14	0.001531	37
matvec	0.049047	12	0.033639	27	0.062920	16	0.045024	27
sincos	0.012112	22	0.007962	25	0.009351	22	0.005341	29
tricubic	0.074085	60	0.060450	63	0.086248	61	0.102335	63
uxx1	0.024248	32	0.032225	53	0.019377	32	0.025174	62
vecadd	0.006269	12	0.004983	28	0.005046	12	0.005135	36
wave13pt	0.025364	34	0.018885	63	0.013564	34	0.019723	63

Литература

1. M. Christen, “Generating and Auto-Tuning Parallel Stencil Codes,” Ph.D. dissertation, University of Basel, Switzerland, 2011.

2. M. Christen, O. Schenk, Y. Cui, PATUS for Convenient High-Performance Stencils: Evaluation in Earthquake Simulations. SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.

3. T. Grosser, H. Zheng, R. Aloor, A. Simbürger, A. Größlinger, L.-N. Pouchet, “Polly – Polyhedral Optimization in LLVM,” *IMPACT 2011 (at CGO 2011), Charmonix France*, April 2011.

4. Y. Hou et al, “AsFermi: An assembler for the NVIDIA Fermi Instruction Set,” `http://code.google.com/p/asfermi/` (дата обращения 03.12.2012).

5. T. Gysi, “HP2C Dycore,” Presentation, `http://mail.cosmo-model.org/pipermail/pompa/attachments/20120306/079fadc1/DWD_HP2C_Dycore_120305.pdf`, *Workshop on COSMO dynamical core rewrite and HP2C project*, March 2012, Offenbach, Germany.

6. J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, F. Durand. Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, SIGGRAPH 2012.

7. А.В. Адинец, Программирование графических процессов при помощи расширяемых языков. // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование, т. 25 (242), с. 52-63, 2011.

8. The OpenACC^{TM}Application Programming Interface. Version 1.0, November, 2011, `http://www.openacc-standard.org` (дата обращения 03.12.2012).

9. OpenHMPP, New HPC Open Standard for Many-Core, http://www.openhmpp.org/en/OpenHMPPConsortium.aspx (дата обращения 03.12.2012).

10. The Heterogeneous Offload Model for Intel® Many Integrated Core Architecture, http://software.intel.com/sites/default/files/article/326701/heterogeneous-programming-model.pdf (дата обращения 03.12.2012).

11. M. Govett, "Development and Use of a Fortran → CUDA translator to run a NOAA Global Weather Model on a GPU cluster," *Path to Petascale: Adapting GEO/CHEM/ASTRO Applications for Accelerators and Accelerator Clusters*, Presentation, http://gladiator.ncsa.uiuc.edu/PDFs/accelerators/day2/session3/govett.pdf, April 2009, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.

12. В.А. Бахтин, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями // Труды международной научной конференции ПаВТ'2012, Новосибирск, с. 373-379, 2012.

13. A. Kravets, A. Monakov, A. Belevantsev, "GRAPHITE-OpenCL: Automatic parallelization of some loops in polyhedra representation," *GCC Developers' Summit,* October 2010, Ottawa, Canada.

14. S. Verdoolaege et al, "PPCG – C to CUDA processor," http://repo.or.cz/w/ppcg.git (дата обращения 03.12.2012).

15. C. Bastoul, "Code Generation in the Polyhedral Model Is Easier Than You Think," *PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques*, September, 2004, Juan-les-Pins, France.

16. M. Torquati, M. Venneschi, M. Amini, S. Guelton, R. Keryell, V. Lanore, F.-X. Pasquier, M. Barreteau, R. Barrère, C.-T. Petrisor, É. Lenormand, C. Cantini, F. De Stefani. An innovative compilation tool-chain for embedded multi-core architectures. Embedded World Conference 2012. Nuremberg, Germany, 2/2012.

17. D. Sands, "Reimplementing llvm-gcc as a gcc plugin," *Third Annual LLVM Developers' Meeting,* Presentation, http://llvm.org/devmtg/2009-10/Sands_LLVMGCCPlugin.pdf, October 2009, Apple Inc. Campus, Cupertino, California.

18. M. Wolfe, C. Toepfer, "The PGI Accelerator Programming Model on NVIDIA GPUs Part 3: Porting WRF," http://www.pgroup.com/lit/articles/insider/v1n3a1.htm (дата обращения 03.12.2012).

19. J. Squyres, G. Bosilca, S. Sumimoto, R. vandeVaart, "Open MPI State of the Union," *Open MPI Community Meeting*, Presentation, http://www.open-mpi.org/papers/sc-2011/Open-MPI-SC11-BOF-1up.pdf, Supercomputing 2011.

20. Consortium for Small-scale Modeling, http://www.cosmo-model.org/ (дата обращения 03.12.2012).

21. The Weather Research & Forecasting Model, http://www.wrf-model.org/index.php (дата обращения 03.12.2012).

22. KernelGen Performance Test Suite, https://hpcforge.org/plugins/mediawiki/wiki/kernelgen/index.php/Performance_Test_Suite (дата обращения 27.01.2013).

Параллельная реализация метода вихревых элементов с использованием модели симметричного вортона-отрезка*

И.К. Марчевский, Г.А. Щеглов

Московский государственный технический университет им. Н. Э. Баумана

На примере задачи о моделировании эволюции пространственных вихревых структур в идеальной несжимаемой среде исследованы вопросы ускорения вычислений в методе вихревых элементов. В качестве вихревого элемента использована разработанная авторами модель симметричного вортона-отрезка, позволяющая моделировать такие эффекты, как деформация и перезамыкание вихревых линий. Возможны несколько способов ускорения вычислений: распараллеливание исходного алгоритма с помощью средств MPI и OpenMP, использование метода мультипольных разложений, а также комбинация указанных подходов. Приведены результаты тестовых расчетов. Для метода мультипольных разложений построены оценки оптимальных параметров.

1. Введение

Численное моделирование пространственных течений среды и определение нестационарных аэродинамических нагрузок, действующих на плохообтекаемые тела, является достаточно трудоемкой задачей. В случае внешних течений с малыми дозвуковыми скоростями, когда сжимаемостью среды можно пренебречь, наиболее эффективными с вычислительной точки зрения являются бессеточные лагранжевы методы: метод дискретных вихрей, метод вихревых частиц, метод вихревых элементов [1–2]. Первичной расчетной величиной в этих методах является завихренность, переносимая вихревыми элементами (ВЭ), а поле скоростей и давление восстанавливаются при помощи закона Био — Савара и аналога интеграла Коши — Лагранжа [3]. Определение скоростей, необходимых для интегрирования уравнений движения ВЭ, основано на расчете парных взаимодействий всех ВЭ, что делает эту задачу аналогичной гравитационной задаче N тел. Понятно, что с повышением точности моделирования при увеличении количества вихревых элементов вычислительная сложность возрастает пропорционально квадрату числа элементов.

Ускорению вычислений методом вихревых элементов посвящено значительное количество работ, среди которых можно выделить два основных подхода: использование параллельных алгоритмов и применение приближенных «быстрых» алгоритмов решения задачи N тел. Для различных классов задач и используемых моделей вихревых элементов в литературе, как правило, используется либо один, либо другой подход.

В работе [4] рассмотрен параллельный алгоритм расчета пространственного обтекания тел методом вихревых элементов с использованием модели симметричного вортона-отрезка. На основе анализа трудоемкостей всех операций алгоритма показано, что распараллеливание только одной операции вычисления парных влияний не позволяет получить существенного ускорений вычислений при проведении расчетов на многопроцессорных ЭВМ. Распараллеливание всех операций позволяет довести долю параллельного кода до 98...99 %, что в соответствии с законом Амдала делает алгоритм решения задачи более масштабируемым за счет сохранения относительных трудоемкостей всех операций алгоритма.

При решении практических задач этот подход позволил сократить время вычислений в 9–10 раз при проведении расчетов на 16-ядерном кластере, при этом около половины

*Работа выполнена при частичной финансовой поддержке гранта РФФИ № 11-08-00699-а и гранта Президента РФ для государственной поддержки молодых российских ученых — кандидатов наук МК-6482.2012.08

времени счета приходится на вычисление парных влияний ВЭ. Существенное сокращение времени выполнения данной операции, как представляется, может быть получено внедрением в параллельный алгоритм «быстрого» метода решения задачи N тел, вычислительная сложность которого пропорциональна $N \lg N$. Такой алгоритм применительно к расчету пространственной эволюции завихренности в методе вихревых элементов при использовании модели симметричного вортона-отрезка реализован авторами и рассмотрен в настоящей статье.

2. Описание модельной задачи

В качестве тестовой задачи вычислительной гидродинамики рассматривается эволюция вихревых структур в безграничном объеме идеальной жидкости, описываемая уравнением неразрывности и уравнением Эйлера (уравнением сохранения импульса)

$$\nabla \cdot \vec{V} = 0,$$
$$\frac{\partial \vec{V}}{\partial t} + \left(\vec{V} \cdot \nabla\right) \vec{V} = -\nabla \left(\frac{p}{\rho_\infty}\right),$$

где $\vec{V}(\vec{r}, t)$ — поле скоростей, $p(\vec{r}, t)$ — давление, ρ_∞ — плотность среды, $\vec{r}$ — радиус-вектор в неподвижной декартовой системе координат, $\nabla = \vec{i}\,\partial/\partial x + \vec{j}\,\partial/\partial y + \vec{k}\,\partial/\partial z$. В качестве граничных условий используется условие затухания возмущений на бесконечности

$$\lim_{r\to\infty} \vec{V} = 0, \quad \lim_{r\to\infty} p = p_\infty.$$

В качестве начального условия задается поле скоростей $\vec{V}(\vec{r}, t_0) = \vec{V}_0$, индуцируемое начальным распределением завихренности $\vec{\Omega}(\vec{r}, t_0) = \nabla \times \vec{V}_0$.

Использование для решения поставленной задачи лагранжева метода вихревых элементов позволяет обеспечить тождественное выполнение уравнения неразрывности, а уравнение Эйлера записать в форме Лагранжа

$$\frac{D\vec{r}}{Dt} = \vec{V}, \quad \frac{D\vec{\Omega}}{Dt} = \left(\vec{\Omega} \cdot \nabla\right) \vec{V}, \tag{1}$$

где $\frac{D}{Dt} = \frac{\partial}{\partial t} + (\vec{V} \cdot \nabla)$ — субстанциональная (материальная) производная.

Вихревые структуры, образующие поле завихренности $\vec{\Omega}(\vec{r}, t)$ в (1), могут быть приближенно представлены виде суперпозиции N_V элементарных полей завихренности вихревых элементов (ВЭ)

$$\vec{\Omega}\,(\vec{r}, t) \approx \sum_{i=1}^{N_V} \vec{\Omega}_{0i}.$$

В этом случае дифференциальные уравнения в частных производных (1) могут быть сведены к системе обыкновенных дифференциальных уравнений относительно параметров ВЭ [1]. В случае моделирования пространственного течения выбор ВЭ представляет собой нетривиальную задачу, поскольку возникает необходимость моделирования эволюции вихревых линий (растяжения, перезамыкания и пр.). В данной работе в качестве модели ВЭ используется модель симметричного вортона-отрезка [5], который можно рассматривать как цепочку, составленную из точечных вортонов [6].

Симметричный вортон-отрезок, как показано на рис. 1, представляет собой отрезок вихревой линии циркуляции Γ длиной $2h$ с заданным центром $\vec{r}_0$ и вектором $\vec{h}$, определяющим его «полуразмах». Вектор вортона может быть представлен в виде $\vec{h} = h\vec{e}$, где $\vec{e} = \vec{h}/|\vec{h}|$ — направляющий вектор вортона. На рис. 1 также обозначены $\vec{s}_0 = \vec{r} - \vec{r}_0$ — вектор, соединяющий центр вортона и точку наблюдения, т.е. точку, в которой вычисляется индуцированная им скорость; $\vec{s}_1 = \vec{s}_0 - \vec{h} = \vec{r} - (\vec{r}_0 + \vec{h})$ – вектор, соединяющий конец

вортона с точкой наблюдения; $\vec{s}_2 = \vec{s}_0 + \vec{h} = \vec{r} - (\vec{r}_0 - \vec{h})$ – вектор, соединяющий начало вортона и точку наблюдения.

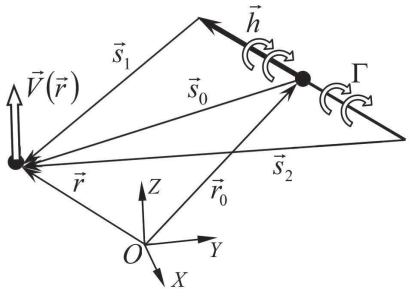


Рис. 1. Симметричный вортон-отрезок

Система уравнений, описывающая эволюцию параметров ВЭ, имеет вид

$$\frac{d\vec{r}_{0i}}{dt} = \vec{V}(\vec{r}_{0i}, t), \quad \frac{d\vec{h}_i}{dt} = [H] \cdot \vec{h}_i, \quad \frac{d\Gamma_i}{dt} = 0, \qquad i = 1, \ldots, N_V. \tag{2}$$

Поле скоростей $\vec{V}$ в (2) определяется суммарным влиянием всех ВЭ

$$\vec{V}(\vec{r}) = \sum_{i=1}^{N_V} \Gamma_i \int_{-1}^{1} \vec{v}(\vec{r}, \vec{r}_{0i} + s\vec{h}_i, \vec{h}_i, \varepsilon)\, ds, \tag{3}$$

вычисляемым по закону Био — Савара с учетом цилиндрически симметричной функции сглаживания

$$\begin{aligned}
&\vec{v}(\vec{r}, \vec{r}_i, \vec{h}_i, \varepsilon) = \begin{cases} \frac{1}{4\pi} c_v\, \vec{a}_v, & R \geq \varepsilon, \\ \frac{1}{4\pi} \frac{R}{\varepsilon} c_v^* \vec{a}_v^*, & R < \varepsilon, \end{cases} \\
&\vec{r}^* = \vec{r} + \left(\tfrac{\varepsilon}{R} - 1\right) \left(\vec{h}_i \tfrac{\vec{s}_0 \vec{h}_i}{h_i^2} - \vec{s}_0 \right), \quad R = |\vec{e} \times \vec{s}_0|, \\
&\vec{a}_v = \vec{h}\, \times \vec{s}_0, \quad c_v = |\vec{a}_v|^{-2} \left[\left(\tfrac{\vec{s}_2}{|\vec{s}_2|} - \tfrac{\vec{s}_1}{|\vec{s}_1|} \right) \cdot \vec{h} \right], \\
&\vec{a}_v^* = \vec{h}\, \times (\vec{r}^* - \vec{r}_i), \quad c_v^* = |\vec{a}_v^*|^{-2} \left[\left(\tfrac{\vec{r}^* - \vec{r}_i + \vec{h}}{|\vec{r}^* - \vec{r}_i + \vec{h}|} - \tfrac{\vec{r}^* - \vec{r}_i - \vec{h}_i}{|\vec{r}^* - \vec{r}_i - \vec{h}_i|} \right) \cdot \vec{h}_i \right],
\end{aligned} \tag{4}$$

где ε — радиус сглаживания поля скоростей вортона-отрезка.

Тензор деформации вектора ВЭ в (2) вычисляется по симметричной схеме [7]

$$[H] = \frac{[\mathrm{B}(\vec{r}_{0i})] + [\mathrm{B}(\vec{r}_{0i})]^T}{2},$$

где тензор $[\mathrm{B}(\vec{r}_{0i})] = \nabla \vec{V}$ определяется формулой

$$[\mathrm{B}(\vec{r}_{0i})] = \sum_{j=1}^{N_V} \int_{-1}^{1} \left[\mathrm{B}\left(\vec{r}_{0j} + s\vec{h}_j, \vec{r}_{0i} \right) \right] ds, \tag{5}$$

$$[\mathrm{B}(\vec{r}_{0j}, \vec{r}_i)] = \begin{pmatrix} \partial \left(V_{0j}(\vec{r}_i)\right)_x / \partial x_i & \partial \left(V_{0j}(\vec{r}_{0i})\right)_x / \partial y_i & \partial \left(V_{0j}(\vec{r}_i)\right)_x / \partial z_i \\ \partial \left(V_{0j}(\vec{r}_i)\right)_y / \partial x_i & \partial \left(V_{0j}(\vec{r}_{0i})\right)_y / \partial y_i & \partial \left(V_{0j}(\vec{r}_i)\right)_y / \partial z_i \\ \partial \left(V_{0j}(\vec{r}_i)\right)_z / \partial x_i & \partial \left(V_{0j}(\vec{r}_{0i})\right)_z / \partial y_i & \partial \left(V_{0j}(\vec{r}_i)\right)_z / \partial z_i \end{pmatrix}.$$

Частные производные компонентов вектора скорости $\vec{V}_{0j}(\vec{r}_i)$ по компонентам радиус-вектора $\vec{r}_i$ найдены по аналитическим формулам, полученным дифференцированием выражений для $\vec{a}_v$ и c_v [8].

При численном определении градиента скорости в правой части (2) необходимо вводить сглаживание особенности на отрезке ВЭ. Сглаживание производится аналогично (4) с использованием функций c_v^*, $\vec{a}_v^*$, вычисленных в точке $\vec{r}^*$:

$$[\mathrm{B}\,(\vec{r}_{0j},\vec{r}_i)] = \begin{cases} [\mathrm{B}\,(\vec{r}_{0j},\vec{r}_i)]\,, & R \geq \varepsilon, \\ \dfrac{R}{\varepsilon}\,[\mathrm{B}\,(\vec{r}_{0j},\vec{r}_i,\vec{r}^*,c_v^*\,\vec{a}_v^*)]\,, & R < \varepsilon. \end{cases}$$

Вычисление интегралов в (3) и (5) проводится с использованием квадратурных формул Гаусса. Методические расчеты показывают, что использование 3 гауссовых точек обеспечивает необходимую точность вычисления интеграла.

Представление вихревых структур (вихревых колец, рамок и нитей) с помощью симметричных вортонов-отрезков оказывается более эффективным по сравнению с использованием других типов ВЭ (точечные вортоны, вихревые сгустки и др.), поскольку использование модели вортона-отрезка позволяет точнее описывать процесс удлинения и изгибания вихревых нитей с использованием меньшего числа вихревых элементов.

Ранее в работе [9] на примере задачи о моделировании явления чехарды вихревых колец было показано, что с помощью симметричных вортонов-отрезков удается сохранять строгую периодичность решения на протяжении большего времени расчета по сравнению с другими типами ВЭ. Это свидетельствует о высокой точности расчета растяжения вихревых нитей с помощью симметричных вортонов-отрезков.

В данной работе задача о чехарде вихревых колец используется для тестирования предлагаемого метода ускорения вычислений. В качестве начального поля завихренности $\vec{\Omega}(\vec{r},t_0)$ при $t_0 = 0$ в идеальной жидкости с единичной плотностью рассматриваются два тороидальных вихревых кольца одинакового радиуса $R_1 = R_2 = 1$ имеющих круглое поперечное сечение с радиусом $r_R = 0{,}1$ и отстоящие друг от друга на расстоянии $z = 1{,}2$ как показано на рис. 2.

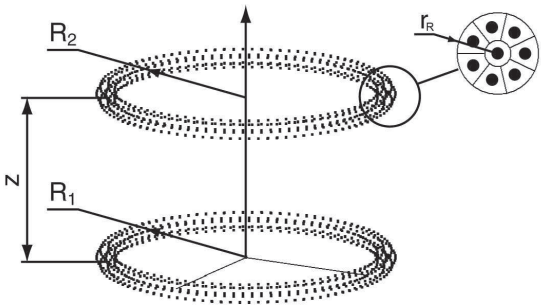


Рис. 2. Расчетная схема для моделирования чехарды вихревых колец

Распределение завихренности в сечении вихревого кольца задавалось соотношением $|\Omega_0| = A\,(r_R - r)\,/r_R$, где A — константа, выбираемая из условия единичной циркуляции вектора скорости по контуру, охватывающему сечение тора. Аппроксимация завихренности в торе при помощи N_V вихревых элементов производилась путем разбиения тора на ячейки примерно равного объема (рис. 3) и замены завихренности в ячейке эквивалентным по циркуляции ВЭ.

Интегрирование системы обыкновенных дифференциальных уравнений (2) производилось методом второго порядка точности с шагом по времени $\Delta t = 0{,}005$. Радиус сглаживания вортона принимался равным $\varepsilon = 0{,}10$.

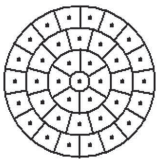


Рис. 3. Разбиение поперечного сечения кольца на ячейки

3. Актуальность применения «быстрого» метода

При использовании «прямого» метода расчета парных взаимодействий, даже используя параллельные вычислительные алгоритмы, число ВЭ в расчетных схемах на практике не превышает 10–20 тысяч. Однако для повышения точности моделирования количество ВЭ необходимо увеличивать, при этом желательно это делать не путем повышения эффективности распараллеливания и задействования в расчетах все большего числа вычислительных ядер, а за счет использования приближенных «быстрых» алгоритмов. При этом ресурс распараллеливания вычислений также может быть использован для еще большего ускорения вычислений. Представляется, что для эффективного решения актуальных на сегодня инженерных задач необходимо обеспечить возможность проведения расчетов для $N_V \sim 10\,000$ ВЭ, используя возможности персональных ЭВМ с многоядерными процессорами.

Эффективный «быстрый» метод приближенного решения гравитационной задачи N тел предложен и описан в работе [10], его модификация применительно к двумерным задачам моделирования обтекания профилей методом вихревых элементов рассмотрена в работе [11]. Вычислительная сложность данного метода пропорциональна $N_V \lg N_V$ против N_V^2 для «прямого» метода. Использование «быстрого» метода позволило довести число ВЭ (в двумерных задачах — бесконечных вихревых нитей, перпендикулярных плоскости течения) до нескольких миллионов. При этом расчет одного парного взаимодействия «плоских» ВЭ требует выполнения всего 6 операций умножения/деления, тогда как расчет взаимодействия двух вортонов-отрезков предполагает выполнение 170...850 операций в зависимости от выбираемой схемы интегрирования системы (2). Таким образом, для схемы интегрирования с 3 гауссовыми точками одно вычисление парного влияния вортонов-отрезков является почти на 2 порядка более затратной операцией по сравнению с плоским случаем, поэтому актуальность разработки эффективного алгоритма быстрого метода для решения подобных задач становится очевидной.

Отметим, что существуют и асимптотически более эффективные методы, вычислительная трудоемкость которых порядка N_V, однако они становятся эффективными лишь при экстремально больших N_V (порядка миллионов), тогда как актуальной является задача ускорения вычислений при N_V порядка нескольких тысяч либо десятков тысяч.

4. Описание «быстрого» метода решения задачи

Первым этапом алгоритма «быстрого» метода вычисления скоростей ВЭ является построение дерева — иерархической структуры областей, имеющих форму прямоугольного параллелепипеда (рис. 4).

Параллелепипед нулевого уровня содержит все ВЭ. Он делится по ребру наибольшей длины на два одинаковых параллелепипеда первого уровня. Путем перебора ВЭ определяется их принадлежность к одному из них. После этого каждый параллелепипед «обрезается» по всем измерениям, чтобы исключить из него области, не содержащие ни одного ВЭ. Заметим, что такая простая процедура позволяет с одной стороны существенно повысить эффективность метода, а с другой — упростить алгоритм.

Далее аналогичным образом эти параллелепипеды делятся пополам, образуя области второго уровня. Деление прекращается при выполнении заданного критерия по размеру

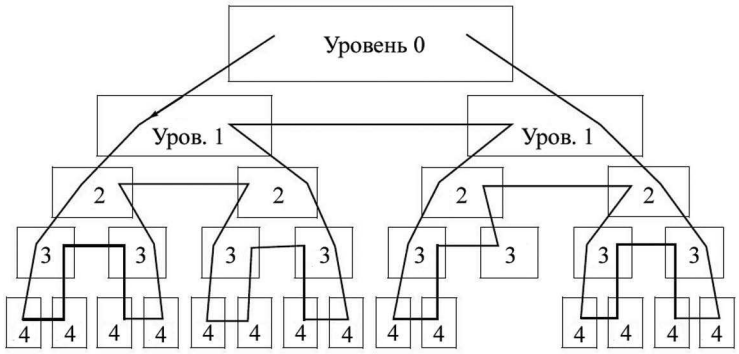


Рис. 4. Дерево, имеющее максимальную глубину 4 уровня, и направление его обхода

параллелепипеда, числу ВЭ в нем и (или) номеру уровня. Отметим, что «обрезание» всех параллелепипедов обеспечивает непустоту областей следующего уровня.

На втором этапе для каждой области определяются положение и вектор эквивалентного вортона, а также его интенсивность. Положение и вектор эквивалентного вортона вычисляются в 2 этапа: сначала находится «средний» радиус-вектор и «средний» вектор вортонов ячейки

$$\vec{\xi}_e = \frac{\sum \vec{r}_i \, |\Gamma_i|}{\sum |\Gamma_i|}, \quad \vec{\eta}_e = \frac{\sum \vec{h}_i \, |\Gamma_i|}{\sum |\Gamma_i|},$$

где суммирование производится по всем вортонам, находящимся в соответствующей ячейке.

Начало и конец эквивалентного вортона находятся в точках пересечения прямой, проходящей через точку $\vec{\xi}_e$ в направлении вектора $\vec{\eta}_e$, с гранями параллелепипеда. Центр этого отрезка определяет положение $\vec{r}_e$ эквивалентного вортона; вектор эквивалентного вортона $\vec{h}_e$ оказывается коллинеарным вектору $\vec{\eta}_e$, а его длина равна половине длины эквивалентного вортона (рис. 5).

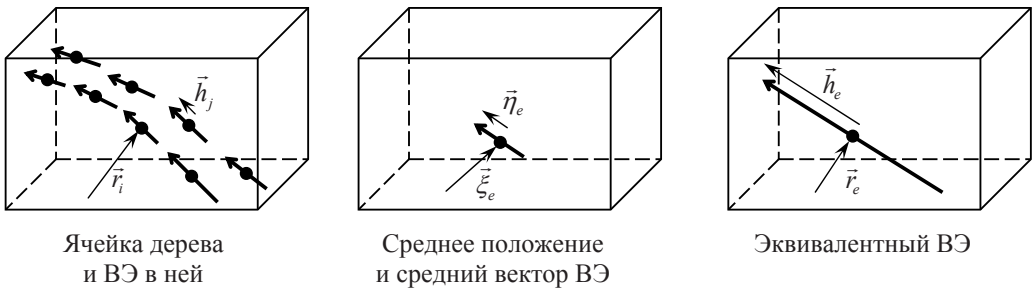


Рис. 5. Алгоритм построения эквивалентного вортона

Интенсивность эквивалентного вортона вычисляется по формуле

$$\Gamma_e = \frac{|\vec{\eta}_e|}{|\vec{h}_e|} \cdot \sum \Gamma_i.$$

Отметим, что для вершин дерева (тех ячеек, которые не имеют потомков следующего уровня), указанные величины вычисляются непосредственно. В дополнение к ним запоминаются также сумма модулей интенсивностей ВЭ и так называемые моменты:

$$\Gamma_e^* = \frac{|\vec{\eta}_e|}{|\vec{h}_e|} \cdot \sum |\Gamma_i| \,, \quad \vec{M}_e^r = \sum \vec{r}_i \, |\Gamma_i| \,, \quad \vec{M}_e^h = \sum \vec{h}_i \, |\Gamma_i| \,.$$

Для ячеек более высокого уровня моменты, суммарные интенсивности и суммарные модули интенсивностей ВЭ вычисляется простым суммированием соответствующих характеристик

дочерних областей, тогда «среднее» положение и «средний» вектор вортона будут определяться по формулам

$$\vec{\xi}_e = \frac{\vec{M}_e^r}{\Gamma_e^*}, \quad \vec{\eta}_e = \frac{\vec{M}_e^h}{\Gamma_e^*}.$$

Важно отметить, что процедура построения дерева и вычисления характеристик всех ячеек является исключительно малозатратной, при этом расчеты показывают, что время построения дерева и вычисления его характеристик растет практически линейно с увеличением количества ВЭ. Ниже в табл. 1 приведено отношение времени построения дерева ко времени вычисления парных влияний «прямым» методом для различного количества ВЭ в расчетной схеме.

Таблица 1. Временные затраты на построение дерева по отношению ко времени счета «прямым методом» (в %)

N_V	2 000	4 000	8 000	12 000	16 000	20 000
$\frac{t_{tree}}{t_{slow}} \cdot 100\,\%$	0,130	0,049	0,025	0,023	0,020	0,015

Таким образом, труднораспараллеливаемая операция построения дерева не является «узким местом» алгоритма быстрого метода. Следует также отметить, что в программной реализации процедуры построения дерева активно используется работа с динамической памятью, поэтому при работе с вычислительными машинами с распределенной памятью рассылка дерева на все вычислительные узлы является нетривиальной задачей. Наиболее рациональным способом решения возникающей проблемы представляется рассылка на все вычислительные узлы параметров всех вортонов и независимое построение деревьев на всех узлах. В силу полной детерминированности алгоритма и идентичности исходных данных все деревья получатся одинаковыми.

На рис. 6 приведены параллелепипеды 1…8 уровня структуры дерева для задачи о моделировании эволюции вихревых колец.

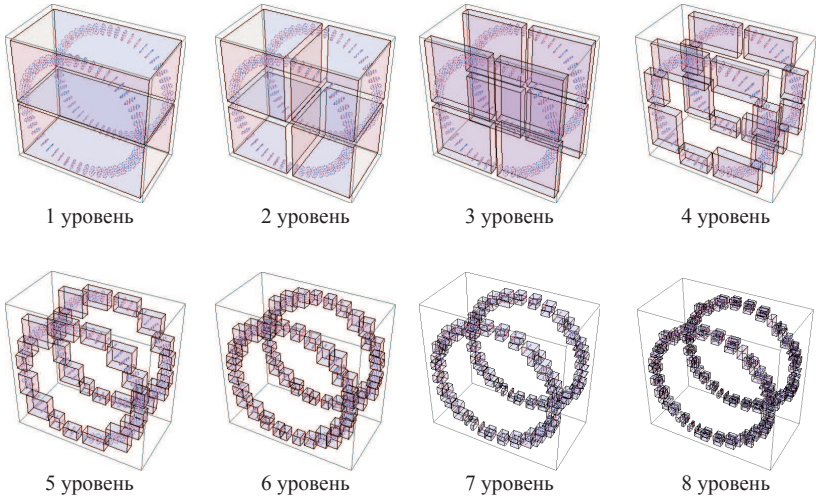


Рис. 6. Параллелепипеды 1…8 уровней

Третьим этапом является вычисление скоростей ВЭ в областях нижнего уровня (т.е. не имеющих дочерних областей): влияние ВЭ, находящихся в той же области, а также близкорасположенных ВЭ из других областей рассчитывается непосредственно по закону Био —

Савара, а затем осуществляется обход дерева, и влияние ВЭ, расположенных в достаточно удаленных областях, учитывается приближенно как влияние эквивалентных вортонов.

На рис. 7 представлена схема, на которой показано, как осуществляется вычисление влияния на один из вортонов (обозначенный кружком) от всех остальных вортонов, образующих вихревые кольца.

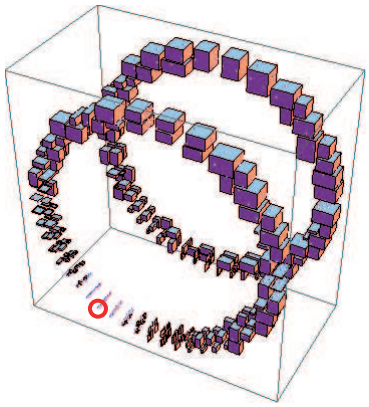


Рис. 7. Схема вычисления вихревого влияния на один из вортонов (обозначен кружком)

Непосредственное вычисление влияния происходит лишь от сравнительно небольшого числа вортонов, находящихся рядом с контрольным; от элементов дерева, изображенных на рис. 7 параллелепипедами, влияние вычисляется как от эквивалентных вортонов.

Критерием близости ячеек является отношение суммы длин их диагоналей к расстоянию между положениями эквивалентных вортонов. На рис. 7 критерием близости являлось превышение указанного отношения величины $\theta = 0{,}25$. На рис. 8 показаны схемы вычисления вихревого влияния в этой же задаче при $\theta = 0{,}10$ и $\theta = 0{,}50$.

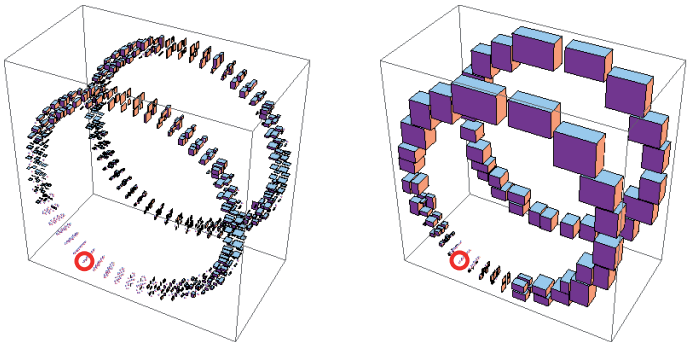


Рис. 8. Схема вычисления вихревого влияния на один из вортонов (обозначен кружком) для $\theta = 0{,}10$ и $\theta = 0{,}50$

Ясно, что при увеличении величины θ скорость счета увеличивается, однако точность получаемых результатов снижается. Значение $\theta = 0$ формально означает, что критерий дальности никогда не выполняется, следовательно алгоритм «быстрого» метода не работает и расчет производится «прямым» методом. Ниже на графике на рис. 9 представлены зависимости времени счета от величины θ для 2 000, 4 000, 8 000 и 20 000 ВЭ в расчетной схеме (все результаты отнесены к времени счета прямым методом в соответствующей задаче). Видно, что чем выше вычислительная сложность задачи, тем эффективнее оказывается использование быстрого метода.

Следует отметить, что при фиксированном значении θ (т.е. при одном и том же предельном уровне погрешности) скорость расчета существенно зависит от максимального числа

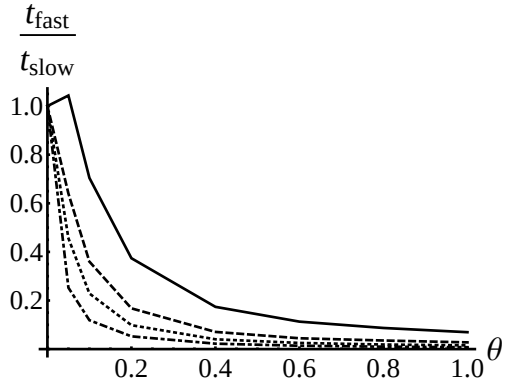


Рис. 9. Время счета с использованием быстрого метода в зависимости от величины параметра θ (сплошная линия – 2 000 ВЭ, пунктир – 4 000 ВЭ, точки – 8 000 ВЭ, штрихпунктир – 20 000 ВЭ)

уровней дерева, которое задается в алгоритме его построения. При этом ошибка на 2-3 уровня от оптимального значения приводит к очень существенному росту затрат времени.

Расчеты показывают, что для обеспечения необходимой для практических целей точности величину критерия θ дальности двух ячеек дерева необходимо выбирать не превышающей 0,2 . . . 0,3. Далее во всех расчетах полагалось $\theta = 0{,}2$.

5. Использование параллельных вычислительных технологий

Применение «быстрого» метода расчета парных взаимодействий ВЭ позволяет существенным образом сократить время счета. Ниже в табл. 2 приведено отношение времени выполнения одного шага расчета в задаче о моделировании эволюции вихревых колец при использовании «быстрого» и «прямого» методов при достаточно большом числе ВЭ в расчетной схеме.

Таблица 2. Отношение времени счета «быстрым» методом ко времени счета «прямым» методом

N_V	10 000	20 000	30 000	40 000	50 000
$\frac{t_{fast}}{t_{slow}}$	0,090	0,053	0,033	0,029	0,022

Видно, что скорость решения задач с расчетными схемами, содержащими десятки тысяч ВЭ, представляющими интерес на практике, возрастает в десятки раз, однако время выполнения одного шага расчета, к примеру, при $N_V = 50\,000$ на персональной ЭВМ составляет более 20 секунд, что все равно неприемлемо, поскольку для решения представляющих практический интерес задач требуется выполнять не менее нескольких сотен, а чаще даже тысяч шагов.

Единственным путем дальнейшего ускорения вычислений является использование параллельных вычислительных технологий. Предпочтительным представляется использование технологии MPI, поскольку она является универсальной и позволяет производить расчеты как на системах с общей памятью (в частности, на персональных ЭВМ с многоядерными процессорами), так и на кластерных системах с распределенной памятью. Отметим, что при распараллеливании с использованием OpenMP для систем с общей памятью с 2–8 вычислительными ядрами выигрыш во времени счета по сравнению с применением MPI

составлял не более 2–5 %.

При использовании «прямого» метода расчета [4] идея распараллеливания операции вычисления вихревого влияния сводилась к разделению всего множества ВЭ на равные блоки по числу задействованных вычислительных узлов и последующему независимому вычислению скоростей всех ВЭ. При этом, очевидно, перед каждым шагом требуется рассылка на все вычислительные узлы полной информации о всех ВЭ. При достаточно большом числе ВЭ ускорение вычислений оказывается близким к линейному, например, для расчета на 16 вычислительных узлах при $N_v = 50\,000$ ускорение составляет 15,97, однако общее время выполнения шага при таком расчете превышает 60 секунд.

В табл. 3 приведены значения ускорения расчета при использовании «прямого» метода вычисления вихревого влияния для различного числа ВЭ в расчетной схеме.

Таблица 3. Ускорение при использовании MPI для решения задачи «прямым» методом

N_V	Число вычислительных узлов					
	1	2	4	8	12	16
5 000	1,00	2,00	3,90	7,81	11,53	14,92
10 000	1,00	2,00	3,97	7,90	11,85	15,45
50 000	1,00	2,00	3,98	7,93	11,88	15,97

Распараллеливание вычислений при использовании «быстрого» метода расчета вихревого влияния основано на сходной идее, однако по вычислительным узлам распределяются не отдельные ВЭ, а вершины дерева, т.е. те его ячейки, которые не имеют потомков. При этом после рассылки на все вычислительные узлы в начале каждого шага полной информации о всех ВЭ на них выполняются процедуры построения дерева. В случае неудачно выбранного критерия окончания деления ячеек-параллелепипедов в алгоритме построения дерева, а также при сильной несимметрии распределения завихренности в пространстве предлагаемый метод разделения задачи на «подобласти» может приводить к существенному снижению равномерности загрузки вычислительных узлов, однако в практически интересных случаях получаемое ускорение оказывается довольно высоким.

Ниже в табл. 4 показаны значения ускорения расчета при использовании «быстрого» метода вычисления вихревого влияния ($\theta = 0{,}2$) для того же числа ВЭ в расчетной схеме, что и в табл. 3, а также для $N = 100\,000$.

Таблица 4. Ускорение при использовании MPI для решения задачи «быстрым» методом

N_V	Число вычислительных узлов					
	1	2	4	8	12	16
5 000	1,00	1,95	3,62	6,33	8,94	10,13
10 000	1,00	1,96	3,66	6,54	8,93	11,44
50 000	1,00	1,96	3,69	6,67	9,29	11,95
100 000	1,00	1,97	3,76	6,96	9,64	12,36

В случае $N_V = 50\,000$ при использовании 16 вычислительных узлов время выполнения одного шага расчета составляет менее 2 секунд, а при $N_V = 100\,000$ — менее 4 секунд, что позволяет использовать подобные алгоритмы на практике. Дальнейшее увеличение числа задействованных вычислительных узлов позволяет при необходимости еще сильнее сократить время счета, поскольку, как видно из табл. 4, алгоритм далек от насыщения.

На графике на рис. 10 показано время счета «прямым» методом без распараллеливания и «быстрым» методом на 16-ядерном кластере при сохранении приемлемой точности вычислений (при $\theta = 0{,}2$). Время счета на обоих графиках отнесено ко времени счета в задаче при $N_V = 10\,000$, при этом на левый график на рис. 10 наложен график функции $\bar{t} = 10^{-8} N_V^2$, а на правый — график функции $\bar{t} = 1{,}2 \cdot 10^{-4} N_V$.

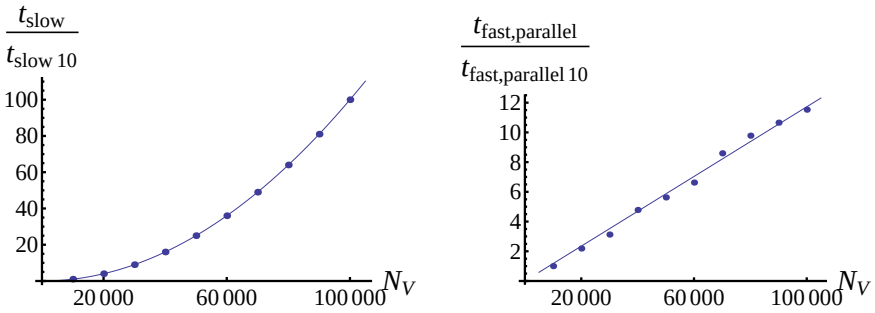


Рис. 10. Время счета при различных значениях N_V с использованием «прямого» метода (слева) и «быстрого» метода (справа)

Линейный по числу ВЭ рост временных затрат при использовании «быстрого» метода объясняется разнонаправленным влиянием двух факторов: сам по себе алгоритм «быстрого» метода обеспечивает сложность алгоритма, пропорциональную $N_V \lg N_V$, а эффективность распараллеливания с ростом N_V несколько повышается.

На рис. 11 приведен график суммарного ускорения за счет использования «быстрого» метода и параллельного алгоритма. В качестве базового выбиралось время решения задачи «прямым» методом без распараллеливания. Видно, что ускорение растет с ростом N_V практически линейно, на график наложен график функции $u = 0{,}011 N_V$.

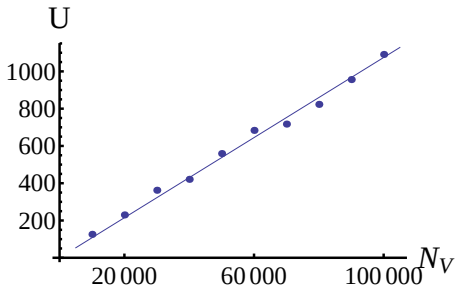


Рис. 11. Достигнутое ускорение при различных значениях N_V по сравнению с расчетом «прямым» методом без распараллеливания

6. Заключение

В работе рассмотрены методы ускорения вычислений при проведении расчетов динамики пространственных вихревых структур методом вихревых элементов. Использование модели симметричного вортона-отрезка дает возможность достаточно точного моделирования эволюции вихревых структур, при этом для решения актуальных задач требуется вводить в расчетные схемы тысячи и десятки тысяч вихревых элементов. Показано, что известные

подходы, связанные с использованием параллельных вычислительных алгоритмов, а также приближенных «быстрых» методов, аналогичных методам решения гравитационной задачи N тел, по отдельности не позволяют получить ускорение расчетов, которое сделало бы возможным решение таких задач на практике. В то же время одновременное применение обоих подходов дает возможность достичь суммарного увеличения производительности в $100 \ldots 1000$ раз при проведении расчета на 16-ядерном вычислительном кластере [12]. При этом время выполнения одного шага расчета становится порядка нескольких секунд, что позволяет решать большое число актуальных инженерных задач за приемлемое время.

Использование технологии MPI делает разработанный алгоритм и созданный на его основе программный модуль переносимым и универсальным: расчеты могут проводиться как на многоядерных системах с общей памятью (при этом снижение эффективности по сравнению с применением технологии OpenMP не превышает нескольких процентов), так и на кластерных системах.

Литература

1. Cottet G.-H., Koumoutsakos P. Vortex methods: theory and practice. Cambridge University Press, 2000.
2. Lewis R.I. Vortex Element Methods For Fluid Dynamic Analysis Of Engineering Systems. Cambridge University Press, 2005.
3. Андронов П.Р., Гувернюк С.В., Дынникова Г.Я. Вихревые методы расчета нестационарных гидродинамических нагрузок. М.: Изд-во МГУ, 2006. 184 с.
4. Марчевский И.К., Щеглов Г.А. Применение параллельных алгоритмов при решении задач гидродинамики методом вихревых элементов // Вычислительные методы и программирование. 2010. Т. 11. С. 105–110.
5. Марчевский И.К., Щеглов Г.А. Модель симметричного вортона-отрезка для численного моделирования пространственных течений идеальной несжимаемой среды // Вестник МГТУ им. Н.Э. Баумана. Сер. Естественные науки. 2008. № 4. С. 62–71
6. Новиков Е.А. Обобщенная динамика трехмерных вихревых особенностей (вортонов) // Журнал эксп. и теор. физики. 1983. Т. 84, Вып. 3. – С. 975–981.
7. A. J. Q. Alkemade. “On vortex atoms and vortons”. PhD Thesis, TU-Delft, April 1994.
8. Marchevsky I.K., Shcheglov G.A. 3D vortex structures dynamics simulation using vortex fragmentons // 6th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012), September 10-14, 2012, Vienna, Austria, Proceedings. Vienna University of Technology. 20 p.
9. Богомолов Д.В., Марчевский И.К., Сетуха А.В., Щеглов Г.А. Численное моделирование движения пары вихревых колец в идеальной жидкости методами дискретных вихревых элементов // Инженерная физика. 2008. № 4. С. 8–14.
10. Barnes J., Hut P. A hierarchical $O(N \log N)$ force-calculation algorithm // Nature. 1986. V. 324. P. 446–449.
11. Дынникова Г.Я. Использование быстрого метода решения «задачи N тел» при вихревом моделировании течений // ЖВМиМФ. 2009. Т. 49, № 8. С. 1458–1465.
12. Лукин В.В., Марчевский И.К. Учебно-экспериментальный вычислительный кластер. Ч.1. Инструментарий и возможности // Вестник МГТУ им. Н.Э. Баумана. Сер. Естественные науки. 2011. № 4. С. 28–44.

Моделирование осаждения мелкодисперсной взвеси из воздуха при прохождении волн давления

К.И. Михайленко, Ю.Р. Валеева

Институт механики им. Р.Р. Мавлютова УНЦ РАН, Уфа

В работе исследован процесс осаждения мелкодисперсной среды под воздействием проходящих волн давления. Записана математическая модель, предусматривающая коагуляцию дисперсных частиц с ростом концентрации. На дисперсные частицы действуют силы Стокса со стороны дисперсионной среды и сила тяжести. Приведены результаты вычислительного моделирования процессов осаждения дисперсной взвеси. Показано, что одним из механизмов осаждения взвесей может быть коагуляция частиц при прохождении волн давления. Произведено распараллеливание задачи на основе MPI, что позволило проводить численные эксперименты на больших пространственно-временных сетках.

1. Введение

В представленной работе рассматривается один из аспектов поведения дисперсной системы — особого соединения двух фаз, когда одна из фаз образует непрерывную дисперсионную среду, в объеме которой распределена дисперсная фаза в виде мелких кристаллов, твердых аморфных частиц, капель или пузырьков. Системы с газообразной дисперсионной средой играют важную роль в природе, быту и производственной деятельности человека. Природными источниками таких смесей являются землетрясения, извержения вулканов, метеоритная и космическая пыль, туман. Источниками техногенных дисперсных систем могут быть аварии, выбросы предприятий (в том числе вредные), пожары, нефтяная и газовая промышленности.

Нередко возникает необходимость в создании эффективной защиты помещений и открытых пространств от распыленных частиц вредных веществ. Одним из способов, позволяющих решить поставленную задачу, является использование акустического поля для осаждения дисперсной взвеси [1].

Экспериментальное исследование осаждения дисперсных сред и аэрозолей активно ведется, например, в Институте механики и машиностроения КазНЦ РАН [1, 2], в представленных работах описываются особенности нелинейных колебаний и осаждения аэрозоля в безударно-волновом режиме.

Нами сделана попытка построения математической модели процессов, связанных со стратификацией дисперсной фазы при прохождении волн давления в несущей газовой фазе и последующего численного исследования процессов осаждения.

2. Математическая модель

Рассматривается двухфазная система газ–дисперсная среда, где дисперсная среда представляется как множество гладких сферических частиц с небольшой концентрацией. Динамика системы описывается на базе основных положений механики сплошной среды [3]. Межфазное взаимодействие определяется осреднённой по пространству силой Стокса [4, 5]. Кроме того, учтено влияние силы тяжести на движение частиц дисперсной среды.

Таким образом, математическая модель, описывающая движение двухфазной смеси газа и дисперсной среды, при наличии коагуляции дисперсных частиц может быть записана исходя из следующих предположений:

- гранулированная среда состоит из гладких сферических частиц одинакового размера, для которых выполняются предположения о малости частиц по сравнению с харак-

терными линейными масштабами течений;

- время коагуляции пары частиц при их соударении пренебрежимо мало по сравнению со средним временем между столкновениями;
- после соединения частицы гранулированной среды вновь образуют сферическую частицу с вдвое бóльшим объемом;
- воздействие газовой фазы на дисперсную описывается осредненной по пространству силой Стокса;
- воздействие на газовую фазу со стороны дисперсной при рассматриваемых концентрациях ($\alpha_2 \ll 1$) достаточно мало и им можно пренебречь;
- на гранулированную среду действует осредненная по пространству сила тяжести.

С учетом высказанных предположений система уравнений математической модели может быть записана в виде:

$$\frac{\partial \rho_i}{\partial t} + \nabla_\ell \cdot \rho_i v_i^\ell = 0\,,$$

$$\rho_i \frac{\partial v_i^l}{\partial t} + \rho_i v_i \frac{\partial v_i^l}{\partial x^\ell} = -\alpha_i \nabla_\ell p + \nabla_k \tau_i^{\ell k} + \rho_i g + F_{ji}^\ell\,,$$

где v_i^ℓ — ℓ-ая составляющая вектора скорости i-ой фазы, в нашей модели принято, что $i = 1$ соответствует дисперсионной газовой фазе, а $i = 2$ — гранулированной; ρ_i — эффективная плотность i-ой фазы, связанная полной плотностью ρ_i^0 соотношением $\rho_i = \alpha_i \rho_i^0$; α_i — объемная концентрация i-ой фазы, $\sum_i \alpha_i = 1$; $\tau_i^{\ell k}$ — тензор сдвиговых напряжений i-ой фазы; p — давление; F_{ji}^ℓ — ℓ-ая составляющая вектора силы межфазного взаимодействия, действующего со стороны j-ой фазы на i-ую; g — ускорение свободного падения.

В соответствии с введенными выше предположениями, принимаем

$$\rho_1 g = 0, \qquad F_{21}^\ell = 0, \qquad \tau_2^{\ell k} = 0.$$

Обсудим подробнее силы, действующие на частицу, движущуюся в дисперсионной среде. Это силы трех категорий: внешние силы (сила тяжести), силы сопротивления среды и силы взаимодействия между частицами. Последние, в случае электрически нейтральных частиц, и при условии $\alpha_2 \ll 1$ достаточно малы, чтобы ими можно было пренебречь.

Сила сопротивления среды описывается формулой Стокса:

$$\mathbf{F}_{st} = 6\pi a \mu (\mathbf{v_1} - \mathbf{v}_p),$$

где a — радиус сферической частицы; μ — вязкость дисперсионной среды; $\mathbf{v}_1$ — скорость дисперсионной фазы; $\mathbf{v}_p$ — скорость частицы.

Сила тяжести, действующая на частицу сферической формы:

$$\mathbf{F}_g^0 = \frac{4}{3}\pi a^3 \rho_2^0 \mathbf{g}.$$

Здесь ρ_2^0 — плотность вещества дисперсной среды; g — ускорение свободного падения.

Взаимодействие взвешенных частиц с колеблющимся потоком газа приводит к нарушению их равномерного распределения в пространстве, сближению друг с другом и последующей коагуляции. В процессе коагуляции образуются частицы бóльшего размера, причем для упрощения модели мы считаем, что вновь образованная частица имеет удвоенную массу, но при этом остается сферической.

При достижении радиусом частицы определенного значения, величина силы тяжести начинает превышать значение силы Стокса, как это показано на рис. 1, что приведет к усиленному осаждению дисперсной фазы.

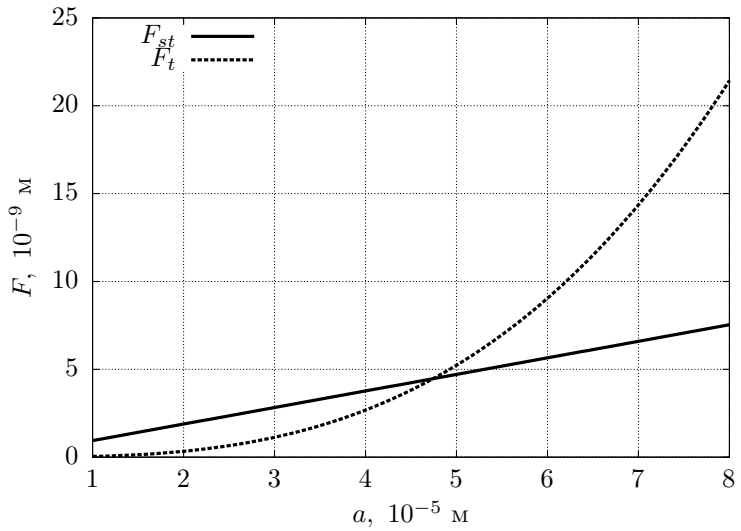


Рис. 1. Сравнение величины силы Стокса F_{st} при некоторой заданной разнице в скорости между частицей и газом и силы тяжести F_g, действующих на отдельную частицу, в зависимости от радиуса частицы a

Так как поведение дисперсной среды описывается с помощью континуальной модели, действие внешних сил необходимо записать в форме действия на единицу объема с заданной концентрацией дисперсной фазы α_2.

Силу межфазного взаимодействия, можно записать в виде:

$$\mathbf{F}_{12} = \eta_\mu \alpha_1 \alpha_2 a^{-2} (\mathbf{v}_1 - \mathbf{v}_2),$$

где a — радиус частиц дисперсной фазы; η_μ — структурный коэффициент, определяемый формой и поверхностью частиц. Для случая гладких сфер $\eta_\mu = \frac{9}{2}\mu_1$.

Аналогичным образом записывается сила тяжести, действующая на дисперсную фазу:

$$\mathbf{F}_g = \alpha_2 \rho_2^0 \mathbf{g}.$$

Для моделирования коагуляции вводится понятие эффективного радиуса дисперсной частицы $\tilde{a}$. Поясним введённое понятие на примере. Начальная концентрация дисперсной фазы равна α_0, при этом все частицы имеют одинаковый радиус, равный эффективному $\tilde{a}_0 = a$. При увеличении концентрации частицы начинают коагулировать и при достижении некоторого значения $\xi\alpha_0$ достигаем максимума коагуляции исходных частиц, когда все частицы считаются попарно объединившимися. Новые частицы имеют эффективный радиус, определяемый их удвоенным от начального объемом $\tilde{a}_1 = 2^{1/3}a$. При достижении концентрации $\xi^2\alpha_0$ мы будем иметь частицы с эффективным радиусом $\tilde{a}_2 = 2^{2/3}a$ и т.д. Таким образом, имеется следующая зависимость:

$$\alpha_i = \xi^i \alpha_0 \Rightarrow \tilde{a}_i = 2^{i/3} a.$$

Другими словами, эффективный радиус является функцией концентрации:

$$\tilde{a} = f(\alpha),$$

ее графическое представление приведено на рис. 2.

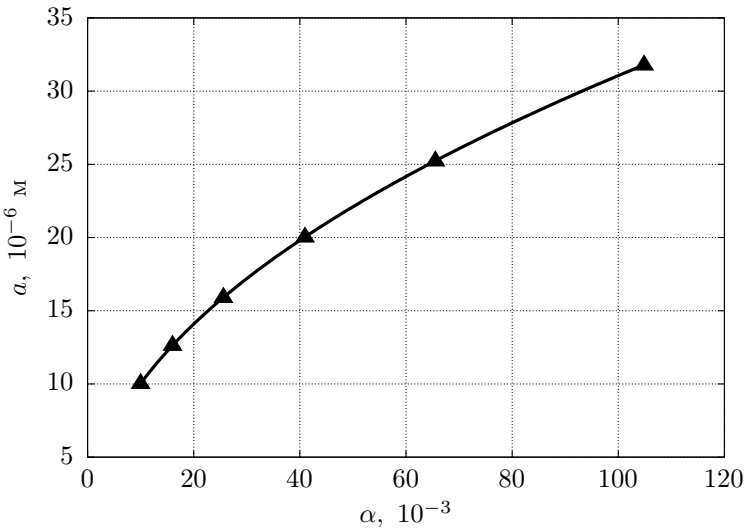


Рис. 2. Зависимость эффективного радиуса $\tilde{a}$ частиц гранулированной фазы от концентрации α_2; треугольниками обозначены значения размеров дисперсных частиц при заданных α_i

3. Численный метод

Уравнения модели решаются численно с использованием метода крупных частиц [6]. Выбор данного метода определяется его консервативностью и хорошей устойчивостью при решении задач со слабыми ударными волнами.

Основная идея метода крупных частиц заключается в расщеплении по физическим процессам системы уравнений, записанной в форме законов сохранения. Среда моделируется системой из крупных частиц, совпадающих в рассматриваемый момент времени с ячейкой эйлеровой сетки. Расчет каждого временного шага разбивается на три этапа.

Эйлеров этап, на котором пренебрегаем всеми эффектами, связанными с перемещением элементарной ячейки (потока массы через границы ячеек нет), и учитываем эффекты ускорения жидкости лишь за счет давления; здесь для крупной частицы определяются промежуточные значения искомых параметров потока $(\tilde{u}, \tilde{v}, \tilde{E})$.

Лагранжев этап, во время которого вычисляются потоки физических величин через границы ячеек.

Заключительный этап для определения в новый момент времени окончательных значений гидродинамических параметров потока (u, v, E, ρ) на основе законов сохранения массы, импульса и энергии для каждой ячейки и всей системы в целом на фиксированной расчетной сетке.

Разностная схема для расчета течения двухфазных сред является модификацией схемы для расчета течения однофазного газа, приведенной выше. Изменяется заключительный этап. В новой схеме добавляются концентрация и силы межфазного взаимодействия. Для рассматриваемой задачи была принята модель с общим давлением: $p_1 = p_2 = p$. В силу того, что $\alpha_2 \ll \alpha_1$, давление газа является определяющим, оно и принимается в качестве общего давления системы. Концентрации газа и дисперсных частиц рассчитываются для каждой ячейки. Также были добавлены функция пересчета концентрации фаз системы и функция, вычисляющая значение силы Стокса.

4. Результаты

4.1. Моделирование процесса осаждения

Описанные математическая модель и численный метод были использованы для решения поставленной задачи об осаждении дисперсной фазы при прохождении через среду волн давления.

Расчетная область представляет собой закрытую с одного конца трубу прямоугольного сечения. В качестве граничных условий на боковых и верхней границах используется условие прилипания (стенка). На нижней границе ставится условие периодического изменения давления (акустическое воздействие).

На рис. 3–4 показаны результаты моделирования осаждения дисперсной среды при прохождении волн давления. Приведенные результаты были получены при следующих параметрах рассматриваемой системы:

- исходный размер дисперсных частиц $a = 10^{-5}$;
- исходная объёмная концентрация дисперсной среды $\alpha = 0.01$;
- исходная плотность газовой фазы $\rho_1^0 = 1$ кг/м^3;
- плотность дисперсной фазы $\rho_2^0 = 1000$ кг/м^3;
- вязкость газовой фазы газ $\mu = 10^{-5}$ П;
- геометрические размеры системы $1 \times 0.15 \times 0.15$ м^3.

На рис. 3 показано продольное распределение концентрации дисперсной фазы в некоторые моменты времени. Здесь показано, что в начальный момент времени мы имеем равномерно распределенную по области концентрацию дисперсной среды. Под действием непрерывно проходящих волн давления равномерное распределение нарушается и концентрация дисперсной фазы начинает увеличиваться вблизи дальнего, закрытого конца области (рис. 4 (А)).

Одновременно с ростом концентрации происходит также и рост размеров частиц, в результате чего они, под действием силы тяжести, начинают смещаться вниз, повышая

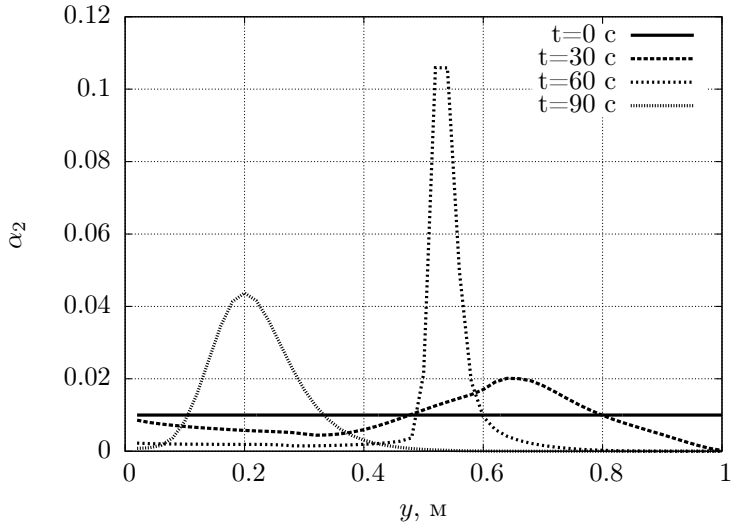
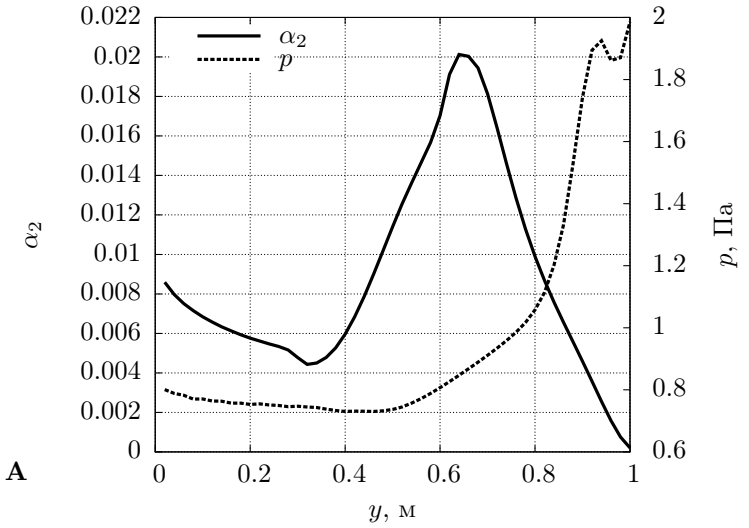
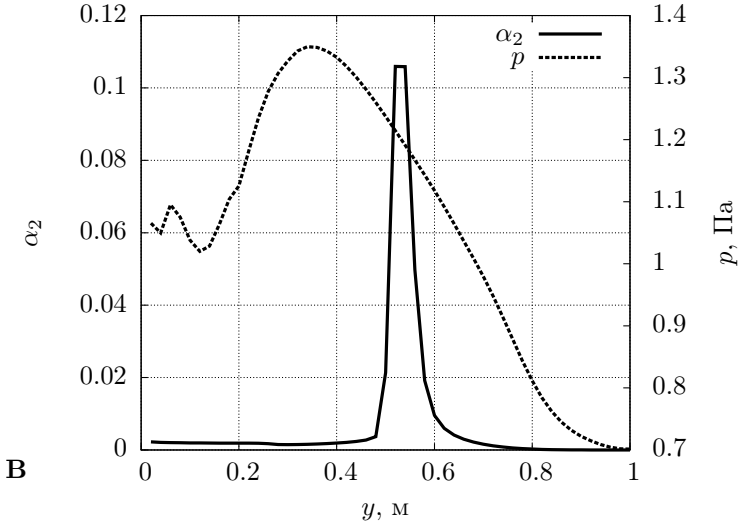


Рис. 3. Изменение концентрации дисперсной фазы α_2 с течением времени





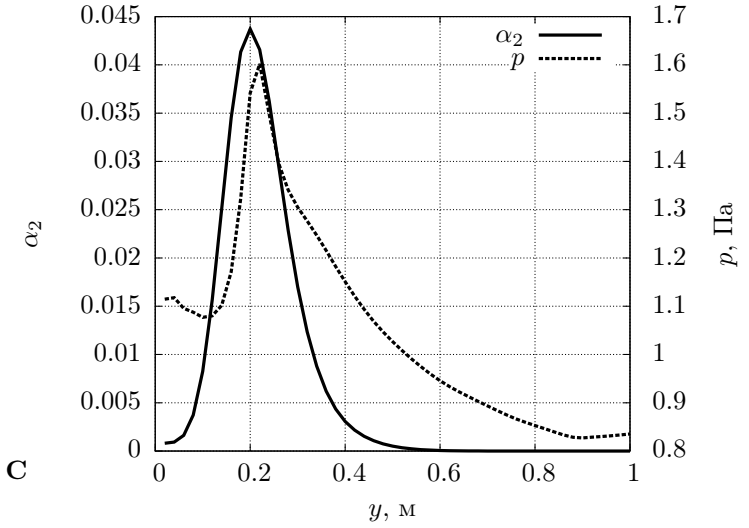


Рис. 4. Распределение концентрации дисперсной фазы α_2 и давления p в моменты времени $t = 30$ с (**A**), $t = 60$ с (**B**) и $t = 90$ с (**C**);

концентрацию в нижележащих слоях и ускоряя этот процесс. В итоге, в некоторый момент времени, приблизительно посреди расчетной области образуется небольшой слой с концентрацией, почти на порядок превосходящей начальную, как это показано на рис. 4 (B). Этот слой состоит из бóльшей части ранее находящейся выше него дисперсной фазы.

На рис. 4 (C) показан заключительный этап осаждения дисперсной фазы. Образовавшийся на предыдущем этапе слой высокой концентрации начинает с заметной скоростью оседать, собирая по пути и те дисперсные частицы, которые находятся ниже. Этот процесс происходит достаточно быстро и в ходе него наблюдается «размывание» концентрации дисперсной среды, вызванное, по-видимому, волнами давления, наиболее энергичными вблизи их источника — нижней границы области.

Из полученных результатов можно сделать вывод, что процесс осаждения состоит из двух, сравнимых по длительности, этапов. На первом этапе происходит постепенное выведение дисперсной среды из положения равновесия под действием проходящих волн давления. После того, как в результате перераспределения концентрации дисперсной среды образовалась область с достаточно заметно повышенной концентрацией, начинается второй этап — собственно осаждение дисперсной среды.

Как было показано выше, второй этап процесса вначале приводит к образованию слоя с высокой концентрацией дисперсных частиц, после чего данный слой достаточно быстро опускается вниз, вбирая в себя все встреченные дисперсные частицы.

Описанная двухэтапность процесса осаждения, определяемого механизмом коагуляции дисперсных частиц при росте концентрации можно наблюдать и на рис. 5 где показана кривая изменения средней концентрации дисперсной фазы $\bar{\alpha}_2$. На этой кривой можно видеть два перегиба: слабо выраженный в момент времени $t \approx 40$ с и резкий при $t \approx 80$ с. При этом первый перегиб как раз и разделяет этап начального перераспределения концентрации и этап последующего осаждения. Второй перегиб образуется, когда в процессе осаждения область повышенной концентрации достигает нижней границы, и дисперсная среда начинает активно выводится из расчетной области.

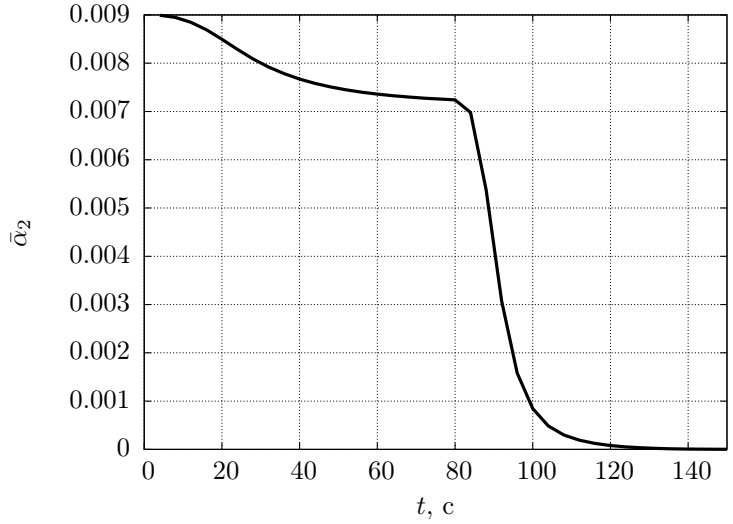


Рис. 5. Изменение средней концентрации дисперсной фазы $\bar{\alpha}_2$ в расчетной области с течением времени

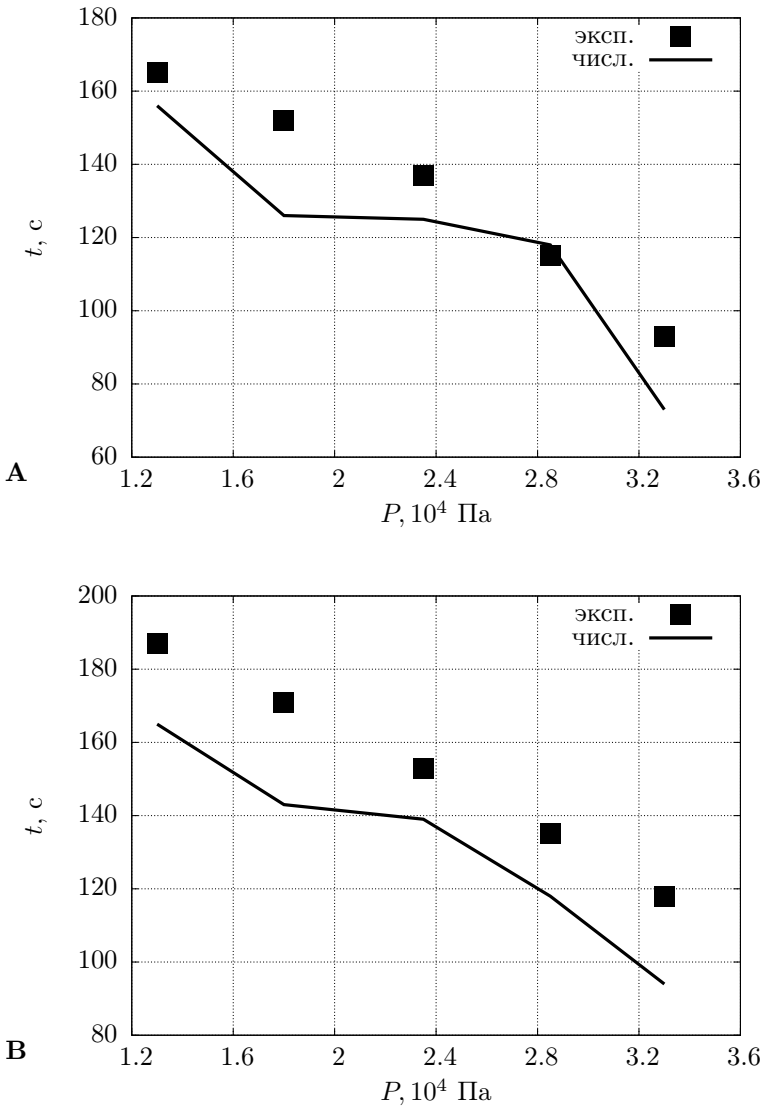


Рис. 6. Время осаждения дисперсной фазы в зависимости от амплитуды волны давления при различных частотах в эксперименте [2] (квадраты) и полученные из расчета (линия): на графике (**A**) $\omega = 158$ Гц; на графике (**B**) — $\omega = 160.66$ Гц

4.2. Сравнение с экспериментом

На рис. 6 приведены результаты сравнения данных, полученных из эксперимента [2] с расчетами, в которых основной упор делался на эмпирическое получение используемого в модели коэффициента коагуляции ξ.

На приведенных графиках сравниваются времена осаждения дисперсной фазы в зависимости от частоты и амплитуды волны давления, генерируемой на нижней границе области.

Соответствие с экспериментальными данными достигается при следующих параметрах вычислительного эксперимента:

- исходный размер дисперсных частицы $a_0 = 0.83 \cdot 10^{-6}$ м, их плотность $\rho_2^0 = 557$ кг/м^3 и начальная концентрация $\alpha = 0.073$;
- плотность и вязкость газовой фазы: $\rho_1^0 = 2$ кг/м^3, $\mu = 1.97 \cdot 10^{-4}$ Пз;
- размер расчетной области: длина 1 м, диаметр 0.03 м);

- коэффициент коагуляции $\xi = 4.7$.

Наблюдаемое расхождение между вычисленными и экспериментальными параметрами может быть объяснено в том числе и используемыми в модели упрощающими предположениями. В частности, предположение о том, что отдельная частица дисперсной среды всегда представляет собой сферу, независимо от того, сколько исходных частиц ее образовали. При таком подходе сохраняется возможность использования силы Стокса для вычисления межфазного взаимодействия на всех этапах, однако, тем самым модель занижает влияние несущей фазы на дисперсную. По-видимому, это одна из причин, по которой скорость осаждения дисперсной среды в расчетах оказывается выше, чем в эксперименте.

5. Высокопроизводительные вычисления

Использование трехмерной математической модели приводит значительному увеличению требований к используемым аппаратным ресурсам. По этой причине расчеты проводились на суперкомпьютере Уфимского государственного авиационного технического университета. Нам было выделено для расчетов четыре двухсокетных вычислительных узла с 4-х ядерными процессорами Intel Quad Xeon 5300 и 8 ГБ оперативной памяти на один вычислительный узел. Среда передачи данных — Infiniband.

В результате распараллеливания программного кода с использованием технологии MPI было получено ускорение близкое к линейному, как это видно из рис. 7. Столь высокое ускорение обусловлено достаточно удобной явной численной схемой модели, позволяющей минимизировать значительно минимизировать обмен теневыми гранями для больших расчетных областей.

Наблюдаемые отклонения от линейного ускорения можно объяснить двумя причинами. Некоторое «излишнее» ускорение при использовании 4 процессов определяется архитектурными особенностями вычислительной системы и тем, что программа в этом случае обрабатывается в пределах одного вычислительного узла. Наблюдаемое постепенное уменьшение ускорения при использовании большого количества процессов можно объяснить ростом отношения объема пересылок к количеству операций, выполняемых в рамках одного процесса.

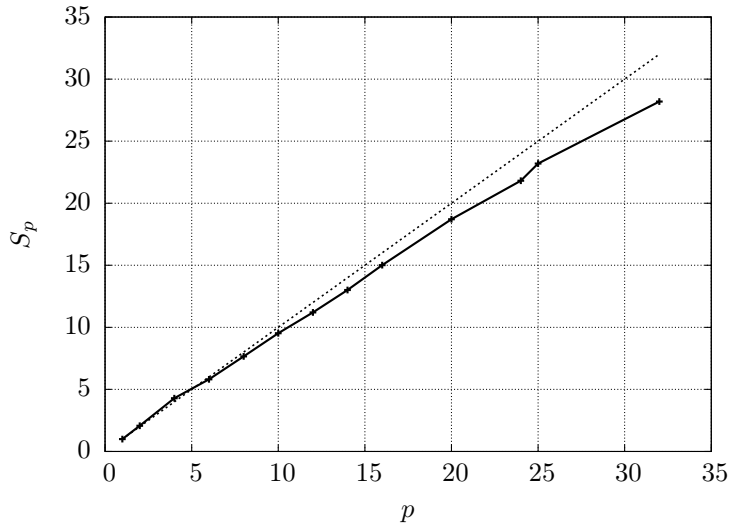


Рис. 7. Зависимость ускорения вычислительного процесса от числа используемых ядер; пунктиром показана линия линейного роста ускорения

6. Заключение

В представленной работе показано, что известный из экспериментальных работ процесс осаждения дисперсной фазы под воздействием волн давления, может быть описан в рамках модели конкуренции силы тяжести и силы межфазного взаимодействия, когда возможна коагуляция частиц дисперсной среды.

Следует особо отметить, что даже в рамках чрезвычайно простой модели оказалось возможно получить достаточно хорошее соответствие с результатами экспериментов.

Дальнейшая работа должна предусматривать развитие модели путем включения в систему уравнений некоторого кинетического соотношения, описывающего процессы коагуляции и диссоциации дисперсных частиц. Также необходимо учесть, что после коагуляции частицы становятся несферическими, поэтому выражение для сил межфазного взаимодействия не должно основываться только на силе Стокса.

Список литературы

1. Губайдуллин Д.А., Зарипов Р.Г., Галиуллин Р.Г. Экспериментальное исследование коагуляции аэрозоля в трубе вблизи субгармонического резонанса // Теплофизика высоких температур. 2004. Т. 42, № 5. С. 788–795.

2. Ткаченко Л.А., Зарипов Р.Г. Особенности нелинейных колебаний аэрозоля в закрытой трубе в безударно-волновом режиме // Вестник нижегородского университета им. Н.И. Лобачевского. 2011. Т. 3, № 4. С. 1171–1173.

3. Нигматулин Р.И. Динамика многофазных сред. Ч I. М.: Наука, 1987. 464 с.

4. Фукс Н.А. Механика аэрозолей. М.: Наука, 1995. 352 с.

5. Михайленко К.И., Везиров Р.Р., Ахатов И.Ш., Урманчеев С.Ф. Численное моделирование течения мелкодисперсного катализатора в канале лифт-реактора // Нефтепереработка и нефтехимия. 1997. № 12. С. 17–20.

6. Белоцерковский О.М., Давыдов Ю.М. Метод крупных частиц в газовой динамике. М.: Наука, 1982. 392 с.

Молекулярно-динамическое моделирование метастабильных фазовых состояний. Термодинамические свойства леннард-джонсовской системы*

С.П. Проценко, В.Г. Байдаков, З.Р. Козлова

Федеральное государственное бюджетное учреждение науки Институт теплофизики Уральского отделения РАН

Методом молекулярной динамики рассчитаны давление, внутренняя энергия и изохорная теплоемкость для 209 состояний леннард-джонсовской системы в интервале температур $0.35 \le k_B T/\varepsilon \le 2.0$ и плотностей $0.001 \le \rho\sigma^3 \le 1.2$. Полученный массив данных, наряду со стабильными состояниями, включает и однородные метастабильные (пересыщенный пар, перегретая и переохлажденная жидкость, перегретый кристалл). По данным моделирования построены термическое и калорическое уравнения состояния. Аппроксимированы спинодали пересыщенного пара и перегретой жидкости. В стабильной области полученные данные сопоставляются с результатами предшествующих работ. Для достижения требуемой точности рассчитываемых свойств усреднение микроскопических аналогов функций динамических переменных проводилось по $5 \cdot 10^5$ шагам интегрирования уравнений движения при моделировании конденсированных состояний и по $(1\text{-}4) \cdot 10^6$ шагам при исследовании газовой фазы. В однопроцессорном режиме моделирование одного состояния жидкости и кристалла составляло от 300 до 900 минут. Моделирование на системе из 64 процессоров при эффективности алгоритма ≈ 0.5 сокращало время расчета примерно в 30 раз.

1. Введение

Молекулярные системы, состоящие из большого числа частиц, могут находиться в нескольких фазовых состояниях. В определенной области температуры и давления фазы равновесно сосуществуют. При изменении состояния вдоль линии, пересекающей кривую равновесия фаз, вещество может остаться однородным, то есть сохранить свойства исходной фазы. Такие состояния не являются полностью устойчивыми. Они представляют метастабильные фазы вещества. Быстрые фазовые переходы в энергонапряженных процессах сопровождаются значительным отклонением системы от условия равновесия фаз. При этом одна из фаз оказывается в метастабильном состоянии. Чем интенсивнее процесс, тем больше отступление от положения равновесия, обеспечивающее необходимую "движущую силу" превращения. В этом смысле метастабильные состояния не только возможны, но и неизбежны.

Информация о поведении вещества в метастабильных состояниях важна для разработки эффективных и безопасных технологических процессов в химии и энергетике, при получении, хранении и транспортировке криогенных жидкостей, создании новых конструкционных материалов, для описания кинетики фазовых и полиморфных переходов, кинетики разрушения материалов при высоких скоростях деформации. Актуальными представляются исследования термодинамических и транспортных свойств в метастабильных состояниях, границ достижимого пересыщения метастабильных фаз, кинетики и молекулярных механизмов кристаллизации, плавления и кавитации, определение параметров фазового равновесия и поверхностного натяжения, механизмов зарождения и роста зародышей новых фаз.

Один из подходов к решению этих задач – натурный эксперимент. Экспериментальные исследования свойств и процессов в сильно метастабильных системах чрезвычайно затруднены

* Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-2-1049). Статья рекомендована к публикации программным комитетом международной научной конференции "Параллельные вычислительные технологии 2013.

малым временем их существования. Новые возможности здесь открывают методы компьютерного моделирования (методы Монте–Карло и молекулярной динамики). Они позволяет не только рассчитать свойства метастабильных фаз в экспериментально достижимых условиях, но и провести исследования фазовой метастабильности при больших пересыщениях вблизи границы существенной неустойчивости - спинодали, а также исследовать кинетику фазовых превращений. Компьютерные модели позволяют реализовать степени метастабильности, значительно превышающие достигаемые в реальных экспериментах. Последнее обстоятельство обусловлено относительной малостью моделируемых систем. С другой стороны, с развитием многопроцессорных систем появилась возможность работать с моделями, содержащими десятки и сотни миллионов взаимодействующих частиц. Это позволяет приблизить результаты моделирования к свойствам макроскопических объектов, учесть корреляции на больших межчастичных расстояниях, присущие системам с сильной метастабильностью, описать свойства границ раздела равновесно сосуществующих фаз в плоском пределе. Решение всех указанных задач невозможно без реализации параллельных процессов на высокопроизводительных вычислительных системах. Новые перспективы перехода к исследованию систем больших масштабов открываются в связи с увеличением производительности вычислителей за счет оснащения GPU.

2. Модель

Модели систем взаимодействующих частиц, основанные на описании взаимодействий потенциалом Леннард-Джонса (ЛД), традиционно широко используются для изучения простых флюидов. Потенциал ЛД учитывает наиболее существенные особенности поведения простого вещества не только в однородных состояниях (газ, жидкость, кристалл), но и пригоден для описания фазовых переходов и фазовых равновесий. Это свойство предопределяет его широкое использование для получения информации о свойствах простых веществ методами компьютерного моделирования с целью развития и совершенствования теорий жидкого состояния.

В данной работе моделирование проведено методом равновесной молекулярной динамики (МД) в *NVE* ансамбле, где N - число частиц, V - объем, E - внутренняя энергия системы. Все расчеты выполнены в кубической ячейке с периодическими граничными условиями при N равном 2048 и 4000. Выбранные размеры систем обеспечивали возможность глубокого проникновения в область метастабильных состояний и их исследование за характерные времена компьютерных экспериментов. Частицы взаимодействовали посредством потенциала Леннард-Джонса, обрезанного при $r = r_c$

$$\phi = \begin{cases} 4\varepsilon\left[\left(\dfrac{\sigma}{r}\right)^{12} - \left(\dfrac{\sigma}{r}\right)^{6}\right], & r \le r_c \\ 0, & r > r_c \end{cases}, \qquad (1)$$

где ε и σ - характеристические параметры энергии и расстояния.

Радиус обрезания потенциала выбирался равным $r_c = 6.78\sigma$ при плотностях $\rho^* = \rho\sigma^3 < 0.82$ и был равен половине ребра ячейки для системы из 2048 частиц при больших плотностях, но не меньше чем 5.975σ ($\rho^* = \rho\sigma^3 = 1.2$). Выбор радиуса обрезания $r_c^* \ge 5.975$ обеспечил: во-первых, пренебрежимо малый скачек сил при $r^* = r_c^*$, во-вторых, максимально возможный в масштабах исследованных систем учет корреляций на больших межчастичных расстояниях, принципиально важеый при моделировании метастабильных и околокритических состояний.

Далее все величины представлены в приведенных единицах. Приведенные параметры обозначены знаком (*): расстояние $r^* = r/\varepsilon$, температура $T^* = k_B T/\varepsilon$, плотность $\rho^* = \rho\sigma^3$, потенциальная энергия $u^* = u/\varepsilon$, внутренняя энергия $e^* = e/\varepsilon$, давление $p^* = p\sigma^3/\varepsilon$, изохорная теплоемкость $c_v^* = c_v/k_B$, где k_B – постоянная Больцмана, m масса частицы. Для интегрирования уравнений движения частиц использовался алгоритм Верле [1]. Шаг интегрирования по времени $\Delta t^* = \Delta t\sqrt{\varepsilon/m}/\sigma = 0.0046$ при температурах $T^* > 0.35$. При $T^* = 0.35$ $\Delta t^* = 0.0023$.

Расчеты проведены с применением пакета параллельного молекулярно-динамического моделирования LAMMPS [2] на вычислительных системах Института математики и механики УрО РАН и Межведомственного суперкомпьютерного центра РАН. В пакет LAMMPS встроен объект для расчета изохорной теплоемкости.

Для оценки качества программы использованы следующие показатели: ускорение $S = t_1/t_{Np}$, где t_{Np} — время исполнения распараллеленной программы на Np процессорах, t_1 — время исполнения программы на одном процессоре; эффективность $E = S/N$p, определяющая среднюю долю времени выполнения параллельного алгоритма, в течение которого процессоры реально используются для решения задачи. Результаты расчетов ускорения и эффективности в интервале Np от 1 до 256 для системы из 4000 частиц в случаях моделирования фаз с плотностями, отличающимися на 2 порядка (жидкость, $\rho^* = 0.75$), газ, $\rho^* = 0.0075$), показаны на **Рис. 1.**

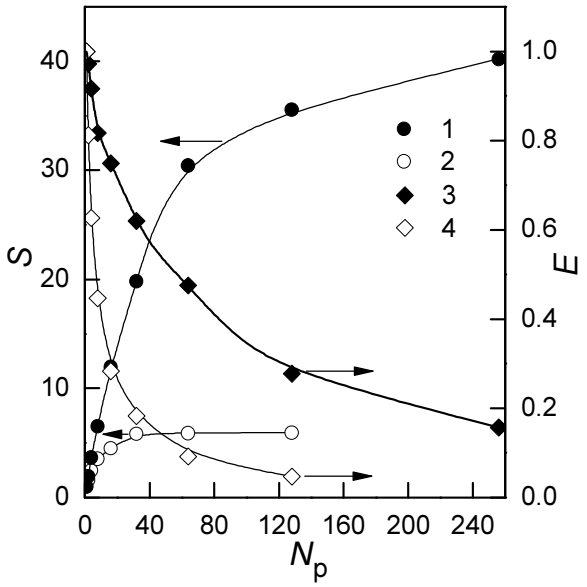


Рис. 1. Зависимость ускорения (1, 2) и эффективности (3, 4) от количества процессоров на задачу и числовой плотности моделируемого состояния. Открытые значки отвечают плотности $\rho^* = 0.75$, закрытые - $\rho^* = 0.0075$

Резкое различие показателей ускорения и эффективности при моделировании систем разной плотности связано с различной загрузкой процессоров. При использованном в модели радиусе обрезания потенциала межчастичного взаимодействия в системе с плотностью $\rho^* = 0.75$ среднее число соседей каждой частицы составляло 605 атомов, а при $\rho^* = 0.0075$ уменьшалось до 6. В соответствии с результатами вычислительного эксперимента, представленными на **Рис. 1**, для моделирований конденсированных состояний использовалось 64 процессора, что обеспечивало ускорение равное 30 при эффективности ≈ 0.5 и продолжительности расчета до 40 минут. Моделирование газовой фазы проводилось не более чем на 32 процессорах с ускорением до 5 и эффективностью ≈ 0.18.

3. Методика моделирования метастабильных состояний

Расчеты проводились по изотермам в интервале температур $0.1-2.0$ и плотностей $0.001-1.2$. В газе начальные состояния отвечали приведенной плотности 0.001, в ГЦК кристалле плотности 1.2. В жидкости на изотермах $T^* = 0.7-2.0$ расчеты начинались из стабильной области состояний. При заданной температуре последовательный переход к следующим состояниям осуществлялся линейным масштабированием ребер ячейки (сжатием, растяжением) и координат частиц конечной конфигурации предыдущего состояния. По мере роста степени метастабильности шаг по плотности уменьшался. Процедура последовательного увеличения (уменьшения) плотности при постоянной температуре обеспечивала глубокие заходы за линии

равновесия фаз без потери однородности системы. Резкое изменение состояния приводило к возникновению локальной неоднородности и преждевременному фазовому переходу. Время уравновешивания варьировалось от 100 до 250 тысяч шагов интегрирования уравнений движения и зависело от термодинамического состояния системы.

При температурах $T^* < 0.7$, все состояния жидкой фазы метастабильны. Большинство из них относится к области отрицательных давлений. В качестве начальной конфигурации частиц для этих температур использовалась конфигурация, полученная при $T^* = 0.7$ и $\rho^* = 0.875$. Переход к требуемой температуре осуществлялся последовательным изохорическим охлаждением. Масштабированием размеров ячейки и межчастичных расстояний создавались состояния с меньшей и большей плотностью.

Расчеты проводились до плотностей, при которых однородные состояния разрушались в результате спонтанного образования и последующего роста зародыша новой фазы [3, 4]. Распад метастабильной фазы фиксировался по скачку давления (см. **Рис. 2**). В этот момент расчеты всех термодинамических параметров прекращались. Если до распада метастабильного состояния набранной статистики было недостаточно, то генерировалась новая МД траектория и расчеты повторялись. Расчет свойств проводился до тех пор, пока время жизни метастабильного состояния не оказывалось достаточным для оценки термодинамических и кинетических параметров с требуемой точностью.

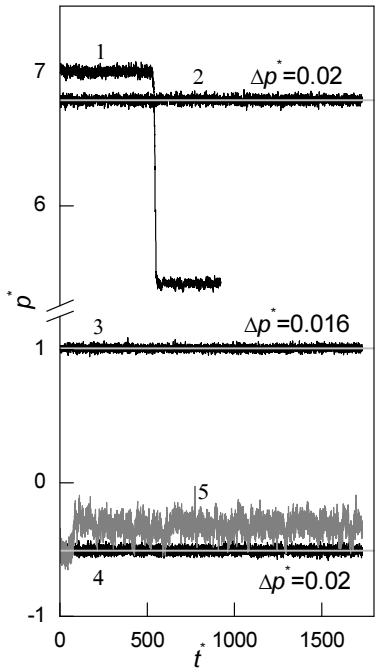


Рис. 2. Зависимость давления от времени при T^* = 0.85: (1) - $\rho^* = 1.015$, переход жидкость-кристалл; (2) - $\rho^* = 1.01$, переохлажденная жидкость; (3) - $\rho^* = 0.85$, стабильная жидкость; (4) - $\rho^* = 0.69$, перегретая жидкость; (5) - $\rho^* = 0.67$, переход жидкость-пар. Прямыми горизонтальными линиями показаны средние по времени значения давления, Δp^* - стандартное отклонение давления в однородных состояниях

Метастабильная фаза допускает наличие в ней кластеров новой фазы, размеры которых не превышают критического. Несмотря на присутствие кластеров докритических размеров, уровень флуктуаций температуры, давления, потенциальной энергии и других термодинамических свойств в метастабильном состоянии незначительно отличается от их значений в области стабильных состояний. Появление зародыша закритического размера сразу сопровождается его необратимым ростом. Такие события легко фиксируется по скачкообразному изменению термодинамических свойств системы. Наиболее чувствительным свойством к образованию закритических зародышей является давление. Характерное время разрушения метастабильного состояния с образованием двухфазной микрогетерогенной системы составляет всего несколько тысяч шагов интегрирования уравнений движения частиц.

На **Рис. 2** представлена зависимость давления от времени для $T^* = 0.85$ при пяти значениях плотности. Рисунок демонстрирует постоянство (с точностью до флуктуаций) давления, как в состоянии стабильной жидкости (кривая 3), так и в метастабильных состояниях жидкой фазы вблизи границ достижимого перегрева (кривая 4) и переохлаждения (кривая 2). Стандартные отклонения давления от средних значений в состояниях стабильной, перегретой и переохлажденной жидкости практически совпадают. Появлению и последующему росту как критического кристаллика (кривая 1), так и критического пузырька пара (кривая 5) отвечают скачки давления.

Поскольку амплитуда таких скачков в несколько раз превышает уровень флуктуаций в однофазных состояниях, моменты разрушения метастабильных состояний однозначно идентифицируются. Подобный контроль «однофазности» входил в стандартную процедуру наших компьютерных экспериментов и проводился во всех исследованных состояниях. Это позволило при расчете термодинамических и транспортных свойств исключать двухфазные микрогетерогенные состояния. Использованный подход не противоречит результатам более детального анализа структуры систем до момента распада метастабильного состояния, основанного на выявлении кластеров новой фазы, определении их размеров и построении функции распределения кластеров по размерам. Этот метод был использован нами при исследовании кинетики зародышеобразования в метастабильных системах [3, 4].

4. Результаты расчетов термодинамических свойств

В молекулярно-динамических экспериментах рассчитаны кинетическая, потенциальная и внутренняя энергии, температура, давление и изохорная теплоемкость в стабильных (газ, жидкость, кристалл) и метастабильных (пересыщенный газ, перегретая и перохлажденная жидкость, перегретый кристалл) состояниях леннард-джонсовской системы.

Кинетическая энергия на частицу равна

$$e_k^* = \frac{1}{2N}\sum_{i=1}^{N} v_i^{*2}\,, \tag{2}$$

где v_i^* - скорость i частицы.

Потенциальная энергия на частицу есть

$$u^* = \frac{1}{N}\sum_{i=1}^{N-1}\sum_{\substack{j=2 \\ j>i,\ r_{ij}^*\le r_c^*}}^{N} \phi\left(r_{ij}^*\right) - \frac{8\pi\rho^*}{3r_c^{*3}}\ . \tag{3}$$

Внутренняя энергия на частицу равна

$$e^* = e_k^* + u^*\,. \tag{4}$$

Температура рассчитывается согласно

$$T^* = 2e_k^*/3\,. \tag{5}$$

Давление определяется как

$$p^* = \frac{\rho^*}{3}\left(2e_k^* - \frac{1}{N}\sum_{i=1}^{N-1}\sum_{\substack{j=2 \\ j>i,\ r_{ij}^*\le r_c^*}}^{N} r_{ij}^*\frac{\partial\phi\left(r_{ij}^*\right)}{\partial r_{ij}^*}\right) - \frac{16\pi\rho^{*2}}{3r_c^{*3}}\,. \tag{6}$$

Изохорная теплоемкость c_v^*, рассчитывалась через среднеквадратичные флуктуации кинетической энергии [5]

$$c_v^* = c_v / k_{\text{B}} = \frac{3}{2}\left(1 - \frac{3}{2}N\frac{< e_k^{*2} > - < e_k^* >^2}{< e_k^* >^2}\right)^{-1} . \qquad (7)$$

Вклады в давление и потенциальную энергию частиц, находящихся на расстояниях $r^* > r_c^*$, учитывались поправками, которые вводились в приближении отсутствия корреляции во взаимодействии частиц на таких расстояниях. Вклад отталкивательного члена потенциала взаимодействия в поправках не учитывался. Для изохорной теплоемкости поправка на дальнодействие не вводилась. При плотностях, близких к критической плотности, вклад вторых слагаемых уравнений (3) и (6) в давление составляет ≈ 4%, в потенциальную энергию 2.8%, а при максимальной плотности $\rho^* = 1.2$ – 0.3% и 1.2%, соответственно.

Время выхода системы на равновесие и время, по которому определялись средние значения термодинамических свойств, зависело от термодинамического состояния системы (стабильное, метастабильное, околокритическое), плотности, температуры, рассчитываемого свойства. В стабильных состояниях и в состояниях, удаленных от критической точки, для определения средних значений рассчитываемых величин с погрешностью ≈1% достаточно усреднения по десяткам тысяч состояний. В области сильной метастабильности такая точность достигается только увеличением интервала усреднения. Особенно критичной к интервалу усреднения была изохорная теплоемкость c_v^*. В состояниях, близких к границам достижимых пересыщений, регистрировались выбросы частичных средних значений, полученных по интервалам времени $5\cdot10^3\Delta t$. Результаты для c_v^* с погрешностью ≈1% во всем исследованном интервале плотностей и температур получены усреднением по $(0.1-1)\cdot10^6$ микросостояний. В ряде случаев получить надежные значения c_v^* при температурах ниже 0.5 не удалось вследствие распада метастабильной жидкости за характерные времена компьютерных экспериментов. С учетом этих обстоятельств, длина интервала уравновешивания выбиралась (100–250)·10^3 временных шагов, средние значения определялись по интервалам $5\cdot10^5\Delta t^*$ для жидкости и $1\cdot10^6\Delta t^*$ для газа. При исследовании разреженных систем особое внимание необходимо уделять накоплению достаточной статистики столкновений. Поэтому все термодинамические свойства получены усреднением не менее чем по 10^6 состояний. Дополнительные пробные расчеты длиной $2\cdot10^6$ и $4\cdot10^6$ шагов показали совпадение средних значений любых термодинамических свойств. Таким образом, стандартной траектории длиной в 10^6 шагов достаточно для оценки свойств в газовой фазе с погрешностью не хуже чем в плотной жидкости.

Поскольку компьютерные эксперименты проводились в условиях постоянства *N*, *V*, *E*, температура не являлась фиксированным параметром. Поэтому при каждом значении плотности расчеты проводились для двух температур, близких к заданной. Значения *p*, *e*, c_v, соответствующие требуемой температуре, определялись линейной интерполяцией на температурных интервалах, не превышающих 0.03.

Погрешности средних значений *p*, *u*, c_v рассчитывались посредством разбиения полных наборов мгновенных значений указанных величин на блоки длиной 5·10^3 значений. Частичные средние по блокам формировали массивы независимых измерений, необходимых для вычисления среднеквадратичных отклонений.

Результаты расчета давления, внутренней энергии и изохорной теплоемкости по изотермам представлены на **Рис. 3-7**. Среднеквадратичные погрешности рассчитанных свойств укладываются в размеры значков на соответствующих графиках. Полученные данные по давлению и внутренней энергии в стабильных и метастабильных состояниях леннард-джонсовской системы использованы для построения локальных термических и калорических уравнений состояния

$$p^* = \sum_{j=0}^{n}\sum_{i=0}^{m} a_{ij}\rho^{*j}T^{*i} , \qquad (8)$$

$$e^* = \sum_{j=0}^{n}\sum_{i=0}^{m} a_{ij}\rho^{*j}T^{*i}\,. \qquad (9)$$

Уравнение (8) позволяет оценить положения ветвей спинодали - границы существенной неустойчивости метастабильной фазы, определяемой условием

$$\left(\frac{\partial p^*}{\partial \rho^*}\right)_{T^*} = 0\,. \qquad (10)$$

Коэффициенты уравнений (8), а также максимальные значения показателей степеней *n* и *m* определялись методом регрессионного анализа.

Для газовой фазы значения коэффициентов равны $a^{\mathrm{g}}_{02} = $-13.9024, $a^{\mathrm{g}}_{03} = $-91.6712, $a^{\mathrm{g}}_{04} = 525.696$, $a^{\mathrm{g}}_{05} = $-255.649, $a^{\mathrm{g}}_{11} = 0.994318$, $a^{\mathrm{g}}_{12} = 16.0688$, $a^{\mathrm{g}}_{13} = 151.379$, $a^{\mathrm{g}}_{14} = $-854.9177, $a^{\mathrm{g}}_{15} = 291.499$, $a^{\mathrm{g}}_{21} = 0.005776$, $a^{\mathrm{g}}_{22} = $-9.29945, $a^{\mathrm{g}}_{23} = 13.9024$. Для жидкой фазы: $a^{l}_{01} = 12.727$, $a^{l}_{02} = -31.7465$, $a^{l}_{05} = 14.9614$, $a^{l}_{12} = -18.1082$, $a^{l}_{13} = 75.0892$, $a^{l}_{14} = -61.2481$, $a^{l}_{15} = 19.1732$, $a^{l}_{21} = 1.80406$, $a^{l}_{22} = -5.71191$, $a^{l}_{32} = 0.64104$. Для кристаллической фазы имеем $a^{\mathrm{cr}}_{00} = $-69.8834, $a^{\mathrm{cr}}_{01} = 286.813$, $a^{\mathrm{cr}}_{02} = $-412.723, $a^{\mathrm{cr}}_{03} = 191.151$, $a^{\mathrm{cr}}_{10} = 9.4696$, $a^{\mathrm{cr}}_{20} = $-31.8944, $a^{\mathrm{cr}}_{21} = 47.2784$, $a^{\mathrm{cr}}_{22} = $-17.2507, $a^{\mathrm{cr}}_{30} = 71.9482$, $a^{\mathrm{cr}}_{31} = $-170.891, $a^{\mathrm{cr}}_{32} = 135.637$, $a^{\mathrm{cr}}_{33} = $-36.0205.

Для сопоставления результатов наших расчетов с данными других авторов мы использовали уравнение состояния Kolafa and Nesbeda [8]. Данное уравнение построено относительно свободной энергии Гельмгольца и позволяет описывать как термические, так и калорические свойства леннард-джонсовского флюида. Область действия уравнения по температуре охватывает интервал от $T^* = 0.68$ до 10.

Уравнение состояния Kolafa and Nesbeda [8] качественно правильно воспроизводит поведение изотерм давления газа и жидкости как в стабильной, так и в метастабильной областях (см. **Рис. 3, 4**). Хорошее количественное согласие по давлению имеет место для газовых ветвей изотерм, исключая участки, прилегающие к спинодали. Параметры спинодали газа, рассчитанные из уравнения состояния Kolafa and Nesbeda [8] и локальных уравнений состояния (6), существенно расходятся. Так по плотности на спинодали расхождения составляют $\pm(3-8)\%$. Аналогично со стороны жидкой фазы. Здесь вблизи спинодали расхождения по плотности составляют $\pm(1-3)\%$. Погрешность описания наших данных уравнением состояния [8] также растет по мере приближения к нижней температурной границе уравнения (T^*=0.68) и с заходом в область переохлажденных состояний жидкой фазы. При $T^* = 0.4$ расхождение по давлению $\Delta p^* = p^*_{\mathrm{MD}} - p^*_{\mathrm{calc}}$ с уравнением состояния [8] достигает -0.3, т.е. примерно $(15-20)\%$. В области переохлажденных состояний жидкости эта величина достигает -0.6 при $T^* = 2$.

Для внутренней энергии наибольше расхождение с уравнением состояния работы [8] также наблюдается на метастабильных участках изотерм и при T^*<0.68. Причем, в отличие от термического уравнения состояния, здесь более существенные расхождения имеют место не столько в абсолютных значениях внутренней энергии, сколько в величине производной $\left(\partial e/\partial\rho\right)_T$. Причиной этих расхождений является то, что при построении уравнения состояния [8] в набор данных не были включены результаты расчетов в метастабильных состояниях леннард-джонсовской системы.

Результаты расчетов изохорной теплоемкости представлены на **Рис. 7**. В отличие от механической, термическая устойчивость флюидной фазы в районе фазового перехода ведет себя более сложным образом (см. **Рис. 7**). По мере захода в область метастабильных состояний механическая устойчивость жидкости и пара понижается (производная $\left(\partial p/\partial\rho\right)_T$ уменьшается). Из **Рис. 7** следует, что если изохорная теплоемкость газовой фазы монотонно возрастает с рос-

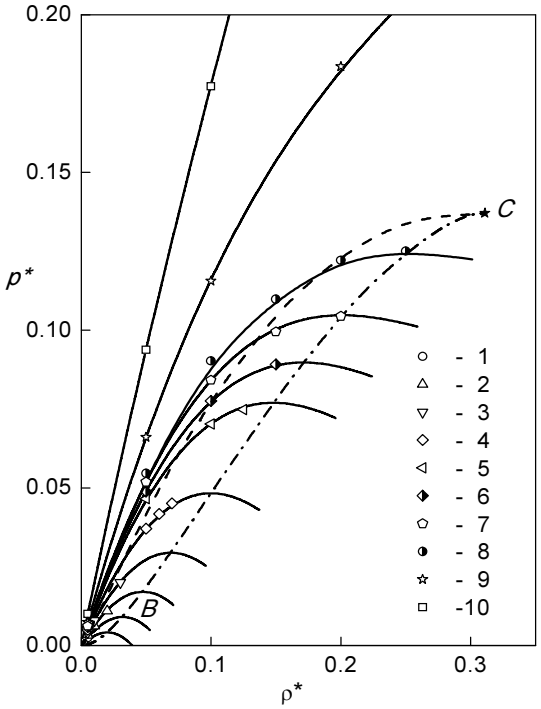


Рис. 3. Изотермы давления (газ): 1 - T^* = 0.55, 2 - 0.7, 3 - 0.85, 4 - 1.0, 5 - 1.15, 6 - 1.2, 7 - 1.25, 8 - 1.3, 9 - 1.5, 10 - 2.0. Сплошные линии – расчет по уравнению состояния (8) для газовой фазы. Штриховая линия - линия фазового равновесия газ-жидкость [6], *C* – критическая точка, *CB* – спинодаль газа

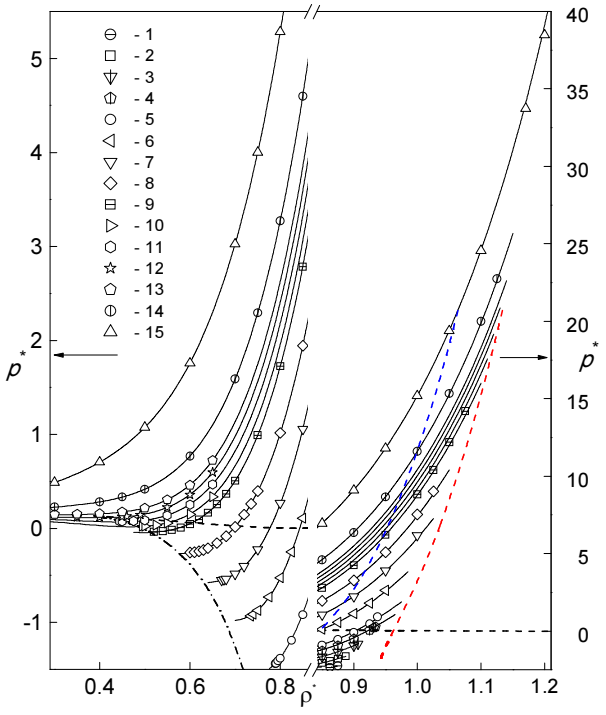


Рис. 4. Изотермы давления (жидкость): 1 – T^* = 0.35 , 2 – 0.4, 3 – 0.45, 4 – 0.5, 5 – 0.55, 6 – 0.7, 7 – 0.85, 8 – 1.0, 9 – 1.15, 10 – 1.2, 11 – 1.25, 12 – 1.3, 13 – 1.35, 14 – 1.5, 15 – 2.0. Сплошные линии – расчет по уравнению состояния (8) для жидкой фазы, штриховые линии: линии фазового равновесия жидкость-газ [6] и жидкость-кристалл [7], штрих-пунктирная линия: спинодаль перегретой жидкости

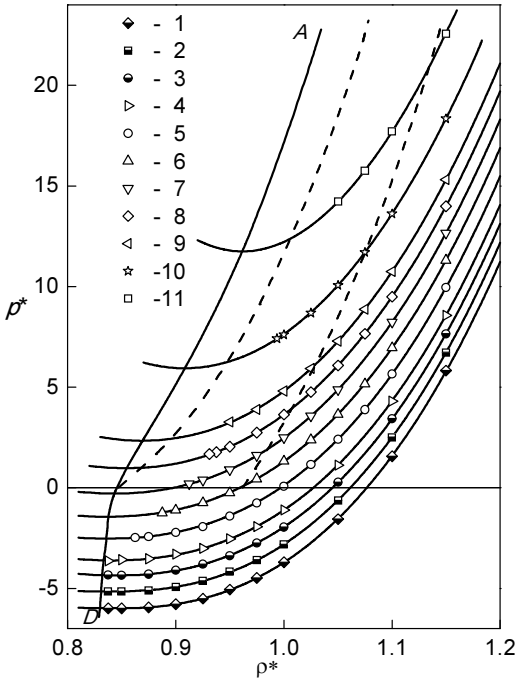


Рис. 5. Изотермы давления (кристалл): 1 - T^* = 0.1, 2 - 0.2, 3 - 0.3, 4 - 0.4, 5 - 0.55, 6 -0.7, 7 - 0.85, 8 - 1.0, 9 - 1.15, 10 - 1.5, 11 - 2.0. Штриховая линия - линия фазового равновесия жидкость-кристалл [7], *AD* – спинодаль кристалла

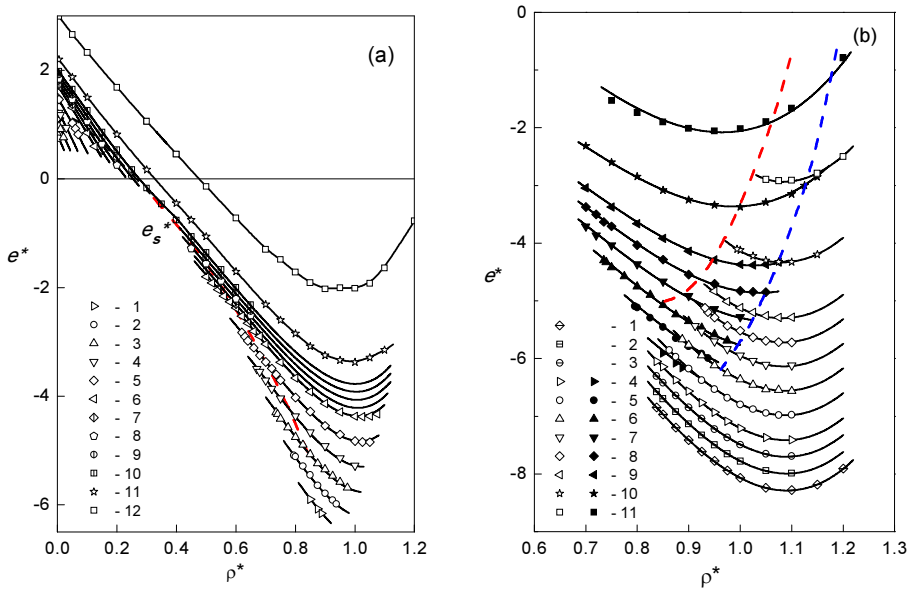


Рис. 6. Изотермы внутренней энергии газа и жидкости (а): 1 - T^* = 0.4, 2 - 0.55, 3 - 0.7, 4 - 0.85, 5 – 1.0, 6 - 1.15, 7 – 1.2, 8 - 1.25, 9 - 1.3, 10 - 1.35, 11 – 1.5, 12 - 2.0 и жидкости (темные точки) и кристалла (светлые точки) (b): 1 - T^* = 0. 1, 2 - 0.2, 3 - 0.3, 4 - 0.4, 5 - 0.55, 6 -0.7, 7 - 0.85, 8 - 1.0, 9 - 1.15, 10 - 1.5, 11 - 2.0. Сплошные линии – результаты расчета по уравнениям состояния (9), штриховые линии - линии фазового равновесия (а) жидкость-газ [6], (а) жидкость-кристалл [7]

том пересыщения $(T = \text{const})$, то теплоемкость жидкости по мере приближения к линии насыщения и заходом в метастабильную область проходит через минимум и далее возрастает при движении к спинодали.

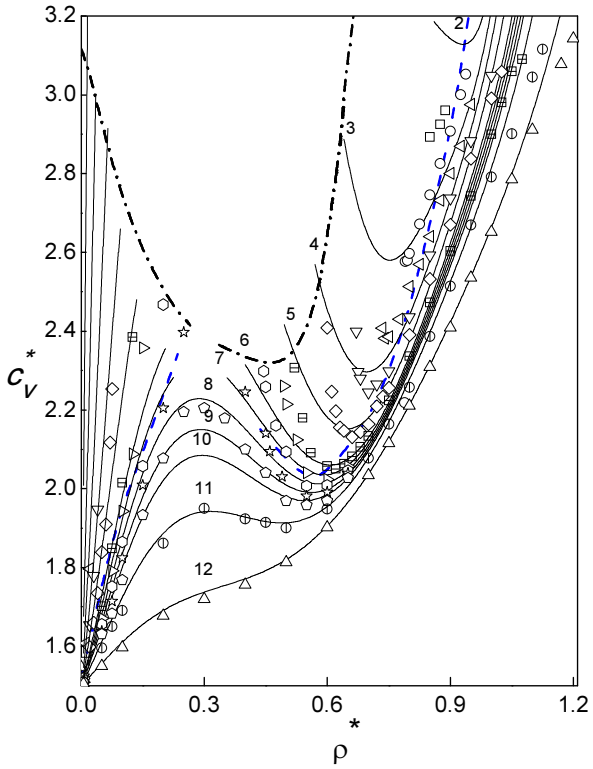


Рис. 7. Изотермы изохорной теплоемкости c_v^*: 1 – $T^* = 0.4$, 2 – 0.55, 3 – 0.7, 4 – 0.85, 5 – 1.0, 6 – 1.15, 7 – 1.2, 8 – 1.25, 9 – 1.3, 10 – 1.35, 11 – 1.5, 12 – 2.0. Штриховые линии – бинодаль [6], штрих-пунктирные – спинодаль

5. Заключение

Леннард-джонсовский флюид является “пробным камнем” для теоретических моделей простых жидкостей. Это предопределяет актуальность изучения его термодинамических, кинетических и структурных характеристик в компьютерных экспериментах методами Монте-Карло и молекулярной динамики. Особый интерес в поведении молекулярной системы представляют точки фазовых переходов. Положение данных точек на термодинамической поверхности состояний определяется силами межмолекулярного взаимодействия. Универсальность микроскопического поведения различных веществ при сжатии от идеально-газового состояния с превращениями сначала в жидкость, а затем в кристалл, отражает универсальность взаимодействия частиц при изменении расстояния между ними.

Нами методом молекулярной динамики рассчитаны давление, внутренняя энергия газа, жидкости и кристалла и изохорная теплоемкость в области фазового перехода жидкость-газ. Как любой переход первого рода, фазовый переход жидкость-газ сопровождается фазовой метастабильностью, а именно перегревом жидкости и переохлаждением газовой фазы. Так как вероятность появления в метастабильной системе за заданный промежуток времени зародыша новой фазы прямо пропорциональна ее объему, то малые размеры молекулярно-динамической системы позволяют осуществить глубокий заход в метастабильную область и рассчитать здесь термодинамические и другие свойства. В серии компьютерных экспериментов получены данные о термодинамическом потенциале, его первой и второй производной для 209 состояний, из которых 101 относятся к области метастабильных состояний. Самая низкая температура, при которой удалось удержать метастабильную жидкую фазу без нарушения ее однородности, составила $T^* = 0.35$. При данной температуре жидкость находится в растянутом состоянии при отрицательном давлении $p^* \cong -2.3$.

Характер плотностной зависимости давления на изотермах подобен предсказываемому уравнением состояния ван-дер-Ваальса. Однако вторая производная термодинамического по-

тенциала (изохорная теплоемкость) свидетельствует о принципиальном отличии в термодинамическом поведении ван-дер-ваальсовского и леннард-джонсовского флюидов. Изохоры давления ван-дер-ваальсовского флюида прямолинейны, а изохорная теплоемкость не зависит от плотности, в то время как изохоры леннард-джонсовского флюида имеют кривизну, которая меняет свой знак по мере приближения к линии фазового перехода и заходом в метастабильную область. О последнем убедительно свидетельствуют изотермы изохорной теплоемкости, которые, как видно из **Рис. 7**, содержат минимумы, максимумы и точки перегиба. Такой характер поведения изохорной теплоемкости в районе фазового перехода жидкость-газ качественно и количественно согласуется с результатами прямых измерений в аргоне.

Имеющиеся в литературе уравнения состояния леннард-джонсовского флюида, построенные по результатам компьютерных экспериментов, качественно правильно передают зависимость давления от плотности и температуры не только в стабильной, но и в метастабильной области. Однако количественные расхождения с результатами наших расчетов увеличиваются по мере роста пересыщения метастабильной фазы и приближения к спинодали, а также при понижении температуры, так как во всех предшествующих работах нижний температурный предел был ограничен температурой тройной точки. Более существенные рассогласования эмпирических уравнений состояния и наших данных имеют место для изохорной теплоемкости. Здесь расхождения достигают 50% и более. Тем не менее, среди эмпирических уравнений состояния леннард-джонсовского флюида можно выделить уравнения состояния [8, 9], которые наиболее лучшим образом согласуются с нашими данными по первым и вторым производным термодинамического потенциала.

Полученные в ходе компьютерного эксперимента термодинамические свойства леннард-джонсовского флюида использованы для составления локальных уравнений состояния. Такие уравнение, описывающее стабильные и метастабильные области, необходимы для определения положения спинодали, описания кинетики нуклеации, проверки микроскопических теорий фазовой метастабильности.

Литература

1. Verlet L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules, Phys. Rev. 1967. Vol. 159, No. 1. P. 98-103.
2. http://lammps.sandia.gov
3. Protsenko S.P., Baidakov V.G., Teterin A.S, Zhdanov E.R. Computer simulation of nucleation in a gas-saturated liquid, J. Chem. Phys. 2007. Vol. 126, No. 9. P. 094502(14).
4. Baidakov V.G., Tipeev A.O., Bobrov K.S., Ionov G.V. Crystal nucleation rate isotherms in Lennard-Jones liquids, J. Chem. Phys. 2010. Vol. 132, No. 23. P. 234505 (9).
5. Lebowitz J.L., Percus J.K., Verlet L. Ensemble Dependence of Fluctuations with Application to Machine Computations, Phys. Rev. 1967. Vol. 153, No. 1. P. 250-254.
6. Baidakov V.G., Protsenko S.P., Kozlova Z.R., Chernykh G.G. Metastable extension of the liquid–vapor phase equilibrium curve and surface tension, J. Chem. Phys. 2007. Vol. 126, No. 21. P. 214505(9).
7. Baidakov V.G., Protsenko S.P. Singular Point of a System of Lennard-Jones Particles at Negative Pressures, Phys. Rev. Lett. 2005. Vol. 95, No. 1. P. 015701(4).
8. J. Kolafa, I. Nesbeda, The Lennard-Jones fluid: an accurate analytic and theoretically-based equation of state, Fluid Phase Equilibria. 1994. Vol. 100. P. 1-34.
9. Mecke M., Müller A., Winkelmann J., Vrabec J., Fischer J., Span R., and Wagner W. An Accurate Van der Waals-Type Equation of State for the Lennard-Jones Fluid International Journal of Thermophysics. 1996. Vol. 17. P. 391-404.

Математическое моделирование условий формирования заморов в мелководных водоемах на многопроцессорной вычислительной системе

А.И. Сухинов, А.В. Никитина, А.Е. Чистяков, И.С. Семенов

Таганрогский кампус Южного федерального университета

Предложена математическая модель взаимодействия планктона и популяции промысловой рыбы пеленгас, учитывающая движение водного потока, микротурбулентную диффузию, пространственно-неоднородное распределение солености и температуры в мелководных водоемах – Азовское море и Таганрогский залив. Устойчивость полученного численного решения задачи позволила проводить вычислительные эксперименты на многопроцессорной вычислительной системе в широком диапазоне значений управляющих параметров. Результаты показали, что с помощью ихтиологического моделирования можно исследовать условия формирования заморов в мелководных водоемах, а также оказывать положительное влияние на функционирование их экологической системы.

1. Введение

В ходе экспедиционных исследований сотрудниками кафедры высшей математики ТТИ ЮФУ в 2000 – 2012 годах было обнаружено, что на одной трети площади центрально-восточной части Азовского моря, которая воспроизводит около 70 процентов биомассы всего моря, регистрируются обширные зоны кислородного голодания (зоны аноксии), что приводит к гибели всего живого (заморным явлениям) [1]. Улучшение кислородного режима в мелководном водоеме возможно с помощью использования метода его биологической очистки путем зарыбления Азовского моря промысловой рыбой - детритофагом – пеленгас. Эта дальневосточная кефаль проявляет высокую пищевую пластичность и является очень перспективным и ценным видом рыб в условиях мелководных водоемов как биологический мелиоратор донных органических отложений, запасы которых в большинстве водоемов неограниченны и не используются никем.

Управление качеством вод мелководных водоемов, а также изучение условий формирования заморов, возникающих в результате антропогенной эвтрофикации, можно осуществлять с помощью методов математического моделирования.

2. Трехмерная математическая модель взаимодействия планктона и промысловой рыбы пеленгас

Рассмотрим нелинейную пространственно-неоднородную 3D модель взаимодействия планктона и популяции промысловой рыбы пеленгас: “рыба – фитопланктон – зоопланктон – питательные вещества – детрит”, которая описывается системой дифференциальных уравнений в частных производных в области G, представляющей собой замкнутый бассейн, ограниченный невозмущенной поверхностью водоёма Σ_0, дном $\Sigma_H = \Sigma_H(x,y)$ и цилиндрической поверхностью, для интервала $0 < t \le T_0$. $\Sigma = \Sigma_0 \cup \Sigma_H \cup \sigma$ – кусочно-гладкая граница области G [2], представленной на **Рис.1**:

$$\frac{\partial X}{\partial t} + div\left(\vec{U}X\right) = \mu_X \Delta X + \frac{\partial}{\partial z}\left(\nu_X \frac{\partial X}{\partial z}\right) + \gamma_X \alpha_S XS - \delta_X XZ - \varepsilon_X X - \sigma_X XP,$$

$$\frac{\partial Z}{\partial t}+div\left(\vec{U}Z\right)=\mu_Z\Delta Z+\frac{\partial}{\partial z}\left(\nu_Z\frac{\partial Z}{\partial z}\right)+\gamma_Z\delta_X XZ-\varepsilon_Z Z-\delta_Z ZP,$$

$$\frac{\partial S}{\partial t}+div\left(\vec{U}S\right)=\mu_S\Delta S+\frac{\partial}{\partial z}\left(\nu_S\frac{\partial S}{\partial z}\right)+\gamma_S\varepsilon_D D-\alpha_S XS+B\left(S_P-S\right)+f, \quad (1)$$

$$\frac{\partial D}{\partial t}+div\left(\vec{U}D\right)=\mu_D\Delta D+\frac{\partial}{\partial z}\left(\nu_D\frac{\partial D}{\partial z}\right)+\varepsilon_X X+\varepsilon_Z Z-\varepsilon_D D-\beta_D DP,$$

$$\frac{\partial P}{\partial t}+div\left(\vec{U}_P P\right)=\mu_P\Delta P+\frac{\partial}{\partial z}\left(\nu_P\frac{\partial P}{\partial z}\right)+\gamma_P\beta_D DP-\varepsilon_P P+\xi_P\sigma_X XP-\delta_P P,$$

$$\vec{u}_P=k_D gradD+k_Z gradZ+k_X gradX.$$

В системе (1) приняты следующие обозначения: X,Z,S,D,P – концентрации фитопланктона (Coscinodiscus), зоопланктона (Copepoda), биогенного вещества (азота), детрита, пеленгаса; α_S – коэффициент потребления биогенного вещества фитопланктоном; $\gamma_X,\gamma_Z,\gamma_P$ – передаточные коэффициенты трофических функций; γ_S – доля питательного вещества, находящегося в биомассе фитопланктона; $\varepsilon_Z,\varepsilon_P$ – коэффициенты элиминации (смертности) Z,P соответственно; ε_X – коэффициент, учитывающий смертность и метаболизм X; δ_X – убыль фитопланктона за счет выедания зоопланктоном; δ_Z – убыль зоопланктона за счет выедания рыбами (пеленгасом); δ_P – убыль пеленгаса за счет выедания рыбами и вылова; S_P – предельно возможная концентрация биогенного вещества; $f=f\left(t,x,y,z\right)$ – функция источника загрязнения; B – удельная скорость поступления загрязняющего вещества; ε_D – коэффициент разложения детрита; β_D – скорость потребления органических остатков пеленгасом; σ_X – коэффициент убыли фитопланктона в результате потребления его пеленгасом; ξ_P – передаточный коэффициент роста концентрации пеленгаса за счет фитопланктона; μ_i – диффузионные коэффициенты в горизонтальном направлении субстанций $i\in\left\{P,X,Z,S\right\}$ соответственно; ν_i – диффузионные коэффициенты в вертикальном направлении для $i=D,X,Z,S,P$ соответственно; Δ – двумерный оператор Лапласа; $\vec{u}$ - поле скоростей водного потока; $\vec{U}=\vec{u}+\vec{u}_{0i}$ – скорость конвективного переноса вещества; $\vec{U}_P=\vec{u}+\vec{u}_P$ - скорость конвективного переноса пеленгаса; $\vec{u}_P$ - скорость движения рыбы относительно воды, k_D,k_Z,k_X - коэффициенты таксиса, $\vec{u}_{0i}$ – скорость осаждения i-й субстанции; $i\in\left\{X,Z,S,D\right\}$.

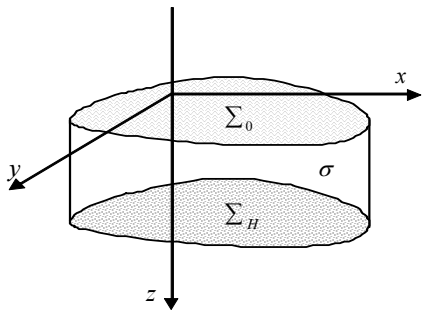


Рис. 1. Схема расчетной области $\bar{G}$

Пусть n – вектор внешней нормали к поверхности, U_n – нормальная по отношению к $\sum$ составляющая вектора скорости водного потока.

Зададим начальные условия:

$$X\left(x,y,z,0\right)=X_0\left(x,y,z\right),Z\left(x,y,z,0\right)=Z_0\left(x,y,z\right),S\left(x,y,z,0\right)=S_0\left(x,y,z\right),$$

$$D\left(x,y,z,0\right)=D_0\left(x,y,z\right),P\left(x,y,z,0\right)=P_0\left(x,y,z\right),\left(x,y,z\right)\in\bar{G},t=0. \quad (2)$$

Граничные условия (условие Неймана на границах, образованных береговой линией области, и третьего рода для X, Z, S, D, P на открытых участках водоема) для системы (1) будут иметь вид:

$$
\begin{aligned}
&X = Z = S = D = P = 0 \quad на \quad \sigma, \ если \ u_n < 0; \\
&\frac{\partial X}{\partial n} = \frac{\partial Z}{\partial n} = \frac{\partial S}{\partial n} = \frac{\partial D}{\partial n} = \frac{\partial P}{\partial n} = 0 \quad на \quad \sigma, \ если \ u_n \geq 0; \\
&\frac{\partial X}{\partial z} = \frac{\partial Z}{\partial z} = \frac{\partial S}{\partial z} = \frac{\partial D}{\partial z} = \frac{\partial P}{\partial z} = 0 \quad на \quad \sum_0; \\
&\frac{\partial X}{\partial z} = -\varepsilon_1 X, \frac{\partial S}{\partial z} = -\varepsilon_2 S, \frac{\partial Z}{\partial z} = -\varepsilon_3 Z, \frac{\partial D}{\partial z} = -\varepsilon_4 D, \frac{\partial P}{\partial z} = -\varepsilon_5 P \ на \ \sum_H,
\end{aligned}
\tag{3}
$$

где $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4, \varepsilon_5$ – неотрицательные постоянные; ε_1, $\varepsilon_3, \varepsilon_5$ – учитывают опускание водорослей, зоопланктона и пеленгаса на дно и их затопление; $\varepsilon_2, \varepsilon_4$ – учитывают поглощение биогенного вещества и детрита донными отложениями.

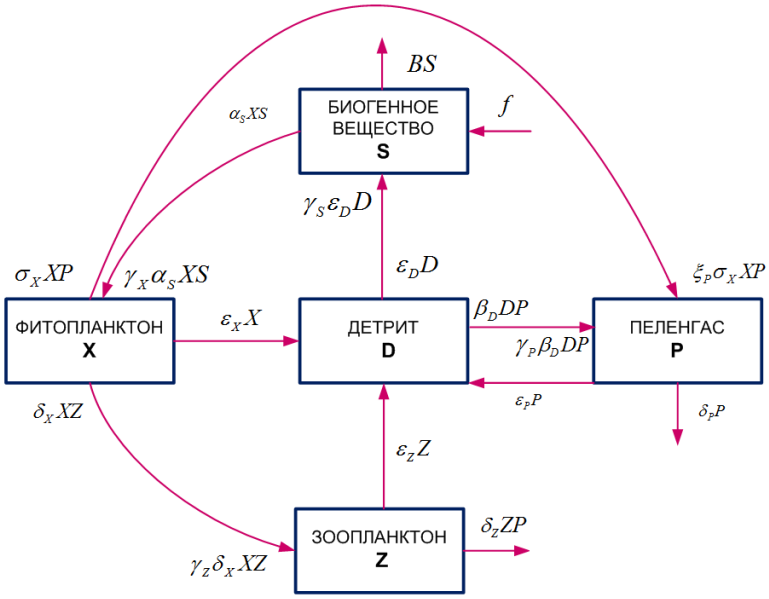


Рис. 2. Схема $X - Z - S - D - P$ модели

Схема модели “рыба – фитопланктон – зоопланктон – питательные вещества – детрит” представлена на **Рис. 2**.

3. Математическая модель гидродинамики

Входными данными сконструированной выше модели взаимодействия планктона и популяции промысловой рыбы пеленгас, является поле вектора скорости водного потока, что требует в свою очередь построения математической модели движения водной среды. Исходными уравнениями гидродинамики мелководных водоемов являются [3,4]:

– уравнения движения (Навье – Стокса):

$$
\begin{aligned}
&u'_t + uu'_x + vu'_y + wu'_z = -\frac{1}{\rho} p'_x + \left(\mu u'_x\right)'_x + \left(\mu u'_y\right)'_y + \left(\nu u'_z\right)'_z + 2\Omega(v\sin\theta - w\cos\theta), \\
&v'_t + uv'_x + vv'_y + wv'_z = -\frac{1}{\rho} p'_y + \left(\mu v'_x\right)'_x + \left(\mu v'_y\right)'_y + \left(\nu v'_z\right)'_z - 2\Omega u\sin\theta, \\
&w'_t + uw'_x + vw'_y + ww'_z = -\frac{1}{\rho} p'_z + \left(\mu w'_x\right)'_x + \left(\mu w'_y\right)'_y + \left(\nu w'_z\right)'_z + 2\Omega u\cos\theta;
\end{aligned}
\tag{4}
$$

– уравнение неразрывности в случае переменной плотности запишется следующим образом:

$$\rho_t' + (\rho u)_x' + (\rho v)_y' + (\rho w)_z' = 0 . \quad (5)$$

где $\overline{u} = \{u, v, w\}$ – компоненты вектора скорости, p – превышение давления над гидростатическим давлением невозмущенной жидкости, ρ – плотность, Ω – угловая скорость вращения земли, θ – угол между вектором угловой скорости и вертикалью, μ,ν – горизонтальная и вертикальная составляющая коэффициента турбулентного обмена.

Система уравнений (4) – (5) рассматривается при следующих граничных условиях:

– на входе (устье рек Дон и Кубань) –

$u(x,y,z,t) = u(t), \quad v(x,y,z,t) = v(t), \quad p_n'(x,y,z,t) = 0, \quad \overline{u}_n'(x,y,z,t) = 0 ,$

– боковая граница (берег и дно) –

$\rho_v \mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho_v \mu(v')_n(x,y,z,t) = -\tau_y(t),\ \overline{u}_n(x,y,z,t) = 0,\ p_n'(x,y,z,t) = 0 ,$

– верхняя граница –

$$\rho\mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho\mu(v')_n(x,y,z,t) = -\tau_y(t), \quad (6)$$

$w(x,y,t) = -\omega - p_t' / \rho g,\ p_n'(x,y,t) = 0 ,$

– на выходе (Керченский пролив) –

$p_n'(x,y,z,t) = 0, \quad \overline{u}_n'(x,y,z,t) = 0 ,$

где ω – интенсивность испарения жидкости, τ_x, τ_y – составляющие тангенциального напряжения (закон Ван-Дорна), ρ_v – плотность взвеси.

Составляющие тангенциального напряжения для свободной поверхности:

$\tau_x = \rho_a C_p\left(|\vec{w}|\right) w_x |\vec{w}|,\ \tau_y = \rho_a C_p\left(|\vec{w}|\right) w_y |\vec{w}|,$

где $\vec{w}$ – вектор скорости ветра относительно воды, ρ_a – плотность атмосферы,

$C_p(x) = \begin{cases} 0.0088,\ x < 6,6\ м/с \\ 0.0026,\ x \geq 6,6\ м/с \end{cases}$ — безразмерный коэффициент.

Составляющие тангенциального напряжения для дна, с учетом введенных обозначений, могут быть записаны следующим образом:

$\tau_x = \rho C_p\left(|\overline{u}|\right) u |\overline{u}|,\ \tau_y = \rho C_p\left(|\overline{u}|\right) v |\overline{u}| .$

На основе приведенной ниже аппроксимации рассчитывается коэффициент вертикального турбулентного обмена, неоднородный по глубине, на основании измеренных пульсаций скоростей [5]:

$$\nu = C_s^2 d^2 \frac{1}{2} \sqrt{\left(\frac{\partial \overline{U}}{\partial z}\right)^2 + \left(\frac{\partial \overline{V}}{\partial z}\right)^2} , \quad (7)$$

где $\overline{U}, \overline{V}$ – осредненные по времени пульсации горизонтальных компонент скорости, d – характерный масштаб сетки, C_s – безразмерная эмпирическая константа, значение которой обычно определяется на основе расчета процесса затухания однородной изотропной турбулентности.

Для решения поставленной задачи использован метод сеток [6]. Ячейки представляют собой параллелепипеды, они могут быть заполненными, частично заполненными или пустыми [14], на основе данной методики аппроксимируется сложная геометрия рельефа дна и береговой линии. Аппроксимация уравнений по временной переменной выполнена на основе схем расщепления по физическим процессам [7] при этом использованы схемы с весами [8,15].

Погрешность аппроксимации математической модели взаимодействия планктона и популяции промысловой рыбы пеленгас, учитывающей гидродинамические процессы, равна $O\left(\tau + \|h\|^2\right)$, где $\|h\| = \sqrt{h_x^2 + h_y^2 + h_z^2}$. Доказано сохранение потока на дискретном уровне для разработанной гидродинамической модельи. Достаточное условие устойчивости и монотонности разработанной модели определяется на основе принципа максимума [6] при ограничениях на шаг по пространственным переменным:

$h_x < |2\mu / u|$, $h_y < |2\mu / v|$, $h_z < |2\nu / w|$

или $\mathrm{Re} \le 2N$, где $\mathrm{Re} = |\overline{u}| \cdot l / \mu$ – числа Рейнольдса, l – характерный размер области.

4. Метод решения сеточных уравнений

Полученные сеточные уравнения можно записать в матричном виде:

$$Ax = f, \tag{8}$$

где A – линейный, положительно определенный оператор ($A > 0$). Для нахождения решения задачи (8) будем использовать неявный итерационный процесс [8]:

$$B\frac{x^{m+1} - x^m}{\tau_{m+1}} + Ax^m = f. \tag{9}$$

В уравнении (9) m – номер итерации, $\tau > 0$ –итерационный параметр, а B – некоторый обратимый оператор, который называется предобуславливателем или стабилизатором. Обращение оператора B в (9) должно быть существенно проще, чем непосредственное обращение исходного оператора A в (8). При построении B исходили из аддитивного представления оператора A_0 - симметричной части оператора А:

$$A_0 = R_1 + R_2, \;\; R_1 = R_2^*, \tag{10}$$

где $A = A_0 + A_1, \;\; A_0 = A_0^*, \;\; A_1 = -A_1^*$.

Оператор предобуславливатель запишется в следующем виде:

$$B = (D + \omega R_1)D^{-1}(D + \omega R_2), \; D = D^* > 0, \; \omega > 0, \tag{11}$$

где D – некоторый оператор.

Соотношения (10) – (11) задают модифицированный попеременно-треугольный метод (МПТМ) решения задачи, если определены операторы R_1, R_2, указаны способы определения параметров τ_{m+1}, ω и оператора D.

Алгоритм адаптивного модифицированного попеременно-треугольного метода минимальных поправок для расчета сеточных уравнений с несамосопряженным оператором имеет вид [9,10]:

$$r^m = Ax^m - f, \;\; B(\omega_m)w^m = r^m, \;\; \tilde{\omega}_m = \sqrt{\frac{\left(Dw^m, w^m\right)}{\left(D^{-1}R_2w^m, R_2w^m\right)}}, \tag{12}$$

$$s_m^{\,2} = 1 - \frac{\left(A_0w^m, w^m\right)^2}{\left(B^{-1}A_0w^m, A_0w^m\right)\left(Bw^m, w^m\right)}, \;\; k_m = \frac{\left(B^{-1}A_1w^m, A_1w^m\right)}{\left(B^{-1}A_0w^m, A_0w^m\right)}, \;\; \theta_m = \frac{1 - \sqrt{\frac{s_m^{\,2}k_m}{(1+k_m)}}}{1 + k_m\left(1 - s_m^{\,2}\right)},$$

$$\tau_{m+1} = \theta_m \frac{\left(A_0w^m, w^m\right)}{\left(B^{-1}A_0w^m, A_0w^m\right)}, \;\; x^{m+1} = x^m - \tau_{m+1}w^m, \;\; \omega_{m+1} = \tilde{\omega}_m,$$

где r^m – вектор невязки, w^m – вектор поправки, в качестве оператора D используется диагональная часть оператора А.

Оценка скорости сходимости запишется в виде:

$$\rho \le \frac{\nu^* - 1}{\nu^* + 1}, \;\; \nu^* = \nu\left(\sqrt{1+k} + \sqrt{k}\right)^2, \;\; k = \frac{\left(B^{-1}A_1w^m, A_1w^m\right)}{\left(B^{-1}A_0w^m, A_0w^m\right)}.$$

где ν - число обусловленности матрицы $C_0, C_0 = B^{-1/2}A_0B^{-1/2}$.

Условием остановки работы предложенного алгоритма является: $\left\|r^m\right\|_C < \varepsilon$, где ε - заданная погрешность.

5. Параллельный вариант метода решения сеточных уравнений

Пиковая производительность МВС составляет 18.8 TFlops. МВС включает в себя 8 компьютерных стоек. Вычислительное поле многопроцессорной вычислительной системы (МВС) ТТИ ЮФУ построено на базе инфраструктуры HP BladeSystem c-class с интегрированными коммуникационными модулями, системами электропитания и охлаждения. В качестве вычислительных узлов используется 128 однотипных 16-ядерных Blade-серверов HP ProLiant BL685c, каждый из которых оснащен четырьмя 4-ядерными процессорами AMD Opteron 8356 2.3GHz и оперативной памятью в объеме 32ГБ. Общее количество вычислительных ядер в комплексе – 2048, суммарный объем оперативной памяти – 4 ТВ. Для управления МВС используется 3 управляющих сервера HP ProLiant DL385G5. Для задач резервного копирования используется библиотека MSL4048.

Параллельная реализация метода решения сеточных уравнений выполнена на основе метода декомпозиции области по двум координатным направлениям [11]. После разбиения исходной расчетной области на части каждый процессор получает свою расчетную область, как показано на **Рис.3,** при этом смежные области перекрываются двумя слоями узлов по направлению, перпендикулярному плоскости разбиения.

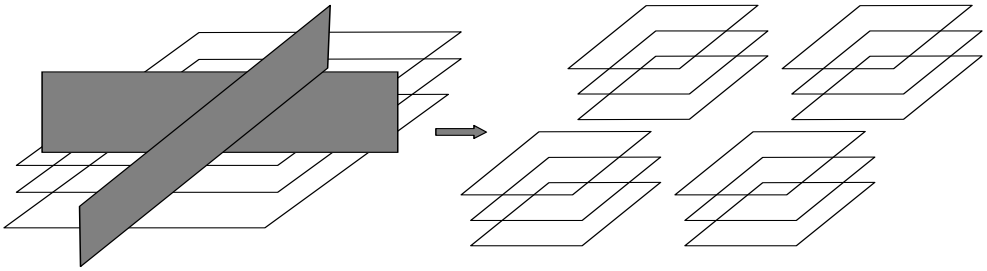


Рис. 3. Декомпозиция области

После того как каждый процессор получит информацию для своей части области, рассчитывается вектор невязки и его равномерная норма. Затем каждый процессор определяет максимальный по модулю элемент вектора невязки и передает его значение всем оставшимся вычислителям. Теперь для вычисления равномерной нормы вектора невязки достаточно на каждом процессоре найти максимальный элемент.

Рассмотрим параллельный алгоритм расчета вектора поправки:

$$(D+\omega_m R_1)D^{-1}(D+\omega_m R_2)w^m = r^m,$$

где R_1 – нижне-треугольная матрица, а R_2 – верхне-треугольная матрица. Для вычисления вектора поправки нужно последовательно решить два уравнения:

$$(D+\omega_m R_1)y^m = r^m,\ (D+\omega_m R_2)w^m = Dy^m.$$

Вначале вычисляется вектор y^m, при этом расчет начинается в левом нижнем углу. Затем из правого верхнего угла начинается вычисление вектора поправки w^m. Схема расчета вектора y^m изображена на **Рис. 4** (показана передача элементов после расчета двух слоев первым процессором).

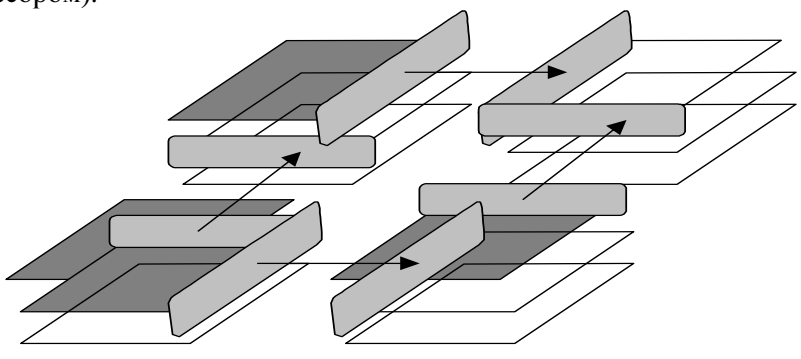


Рис. 4. Схема для расчета вектора y^m

На первом шаге вычислений первый процессор обрабатывает верхний слой, затем осуществляется передача перекрывающихся элементов смежным процессорам. На следующем шаге первый процессор обрабатывает второй слой, а его соседи – первый. Передача элементов после расчета двух слоев первым процессором показана на рис 4. В схеме для расчета вектора y^m только первый процессор не требует дополнительной информации и может независимо от других процессоров вести обработку своей части области, остальные процессоры ждут результатов от предыдущего процессора, пока он не передаст вычисленные значения сеточных функций, для узлов сетки, располагающихся в предшествующих позициях данной строки. Процесс продолжается до тех пор, пока не будут рассчитаны все слои. Аналогичным образом можно решить СЛАУ с верхне-треугольной матрицей для расчета вектора поправки. Далее вычисляются скалярные произведения (11), и выполняется переход на следующий итерационный слой.

Результаты расчета ускорения и эффективности в зависимости от количества процессоров для параллельного варианта адаптивного попеременно-треугольного метода приведены в таблице 1. Расчеты производились на сетке размерами 351x251x46 узлов.

Таблица 1. Зависимость ускорения и эффективности от количества процессоров

Количество процессоров	Время, с.	Ускорение	Эффективность
1	7,490639	1	1
2	4,151767	1,804	0,902
4	2,549591	2,938	0,734
8	1,450203	5,165	0,646
16	0,882420	8,489	0,531
32	0,458085	16,351	0,511
64	0,265781	28,192	0,44
128	0,171535	43,668	0,341

6. Описание программного комплекса

Разработанное экспериментальное программное обеспечение на базе многопроцессорной вычислительной системы предназначено для математического моделирования возможных сценариев развития экосистем мелководных водоемов на примере Азовского моря. Программный комплекс «Azov3d» предназначен для построения турбулентных потоков несжимаемого поля скоростей водной среды на сетках с высокой разрешающей способностью. Данная программа используется для расчета трехмерного вектора скорости течения водной среды в акватории Азовского моря, учитывает такие физические параметры как: сила Кориолиса, турбулентный обмен, сложная геометрия дна и береговой линии, испарение, стоки рек, сгонно-нагонные явления, ветровые течения и трение о дно, и обеспечивает выполнение следующих функций:

– расчет поля скорости без учета давления;

– расчет гидростатического давления (используется в качестве начального приближения для гидродинамического давления);

– расчет гидродинамического давления;

– расчет трехмерного поля скорости.

К выходным параметрам относятся: шаги по пространственным координатам, шаги по пространственным координатам, погрешность вычисления сеточных уравнений, размеры расчетной сетки, временной интервал, интенсивность испарения, начальные распределения компонент вектора скорости движения водной среды и давления.

Разработанный комплекс программ допускает внедрение новых расчетных функций, в частности, в данный комплекс были встроены программные блоки, предназначенные для

вычисления трехмерных распределений концентраций планктона и популяции промысловой рыбы пеленгас.

Поля скоростей водного потока, рассчитанные на основе математической модели (4) – (6), относится к входным данным для модели гидробиологических процессов (1) – (3). Выходные файлы содержат информацию о преобразованиях, произошедших с различными химико-биологическими параметрами в результате работы модели (1) – (3).

7. Результаты численных экспериментов

С помощью численной реализации гидростатической модели (4) – (6) проводились исследования причин возникновения заморов в мелководных водоемах, таких как Азовское море и Таганрогский залив. Было установлено, что при наличии замкнутого вихревого движения среды значительное количество органических веществ не покидает пределы этой структуры и, опускаясь на дно, образует органический осадок, что приводит к появлению участков анаэробного заражения. На **Рис. 5** видно наличие вихревой структуры течения в восточной части Азовского моря, в данном районе вода богата органическими примесями, источниками которых являются реки Дон и Кубань.

С помощью численной реализации гидробиологической пространственно-неоднородной модели вида (1) – (3) изучались возможности уменьшения площадей заморных зон в мелководном водоеме, а значит и биологической очистки вод, за счет зарыбления его биологическим донным мелиоратором промыслового значения - пеленгас. Предполагалось, что интегральное воздействие сезонно изменяющихся внешних факторов отражается на величине коэффициентов роста и продуцирования популяций планктона и рыб.

В соответствии с направленностью работы при разработке модельных сценариев учитывались особенности размножения, стадийного развития биологических организмов и различия в длительности их репродукционных циклов. Поэтому сдвиги по времени между изменениями биомасс фитопланктона, кормового зоопланктона и пеленгаса соответствуют реальным процессам, происходящим в мелководном водоеме. Учёт различий в удельной скорости увеличения биомассы организмов разных трофических уровней осложняется ещё и тем, что физиологические процессы чувствительны к изменениям внешних условий.

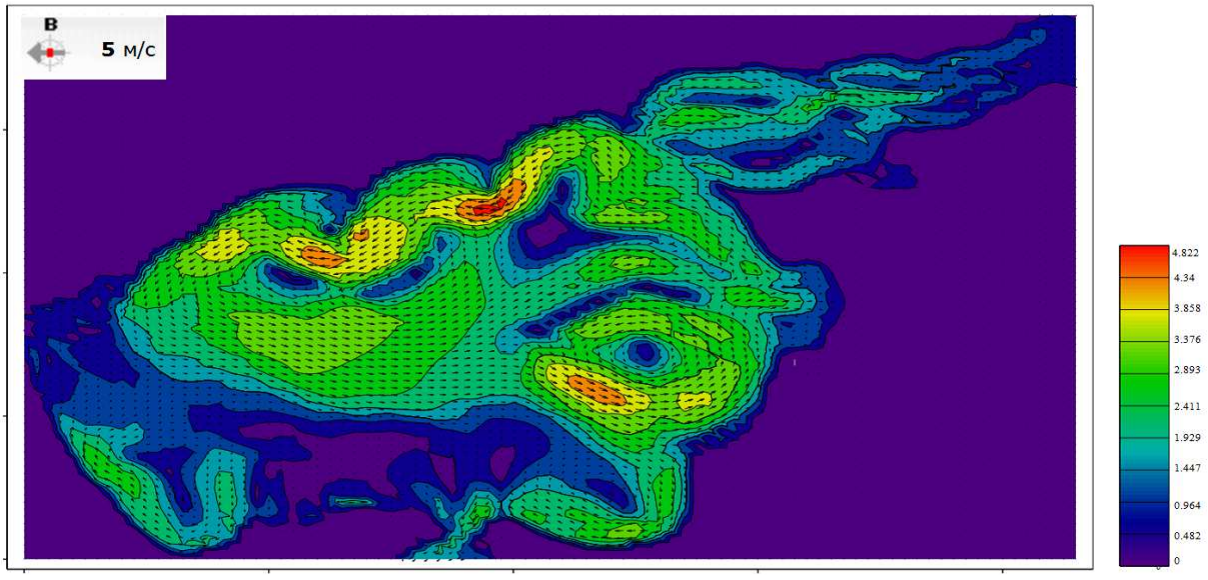


Рис.5. Поле вектора скорости движения водной среды при восточном ветре 5 м/с (баротропные течения)

Численные эксперименты на основе выше описанной математической модели гидродинамических процессов производились на сетке размерами 351x251x46 узлов для Азовского моря с шагом между узлами: по горизонтали – 1 км; по вертикали – 50 см; по времени – 1000

сек. На основе экспедиционных данных и литературных источников проведена калибровка и верификация описанных в работе моделей, подобраны оптимальные значения параметров, в них входящих. Из проведенных исследований на устойчивость используемых разностных схем можно сделать вывод о том, что на основе разработанного программного комплекса можно проводить вычислительные эксперименты прогностической направленности в широком диапазоне задаваемых параметров.

При ихтиологическом моделировании и разработке всевозможных сценариев биологической очистки вод мелководных водоемов учитывались экологические особенности пеленгаса - типично эврибионтного вида, способного жить в очень разнообразных по экологическим условиям водоемах. Эта рыба достаточно эвригалинная и живет в водоемах с разными показателями солености. Кроме того, она эвритермная (т. е., приспособлена к обитанию в водоемах с разным температурным режимом), мирная и стайная, быстро растет и нагуливается на относительно малых глубинах, преимущественно на дне прибрежных зон водоемов, насыщенных различными органическими остатками и соединениями. Однако для размножения (на нерест) пеленгас мигрирует в более глубокие места с повышенной соленостью воды.

Лимитирующим фактором, который определяет численность популяции промысловой рыбы пеленгас, может являться ограниченность мест, пригодных для зимовки рыбы.

Было выполнено численное моделирование для трехмерной задачи взаимодействия планктона и популяции промысловой рыбы пеленгас: “рыба – фитопланктон – зоопланктон – питательные вещества – детрит” в областях сложной формы (мелководные водоемы: Азовское море и Таганрогский залив) при различных начальных условиях и направлениях ветра.

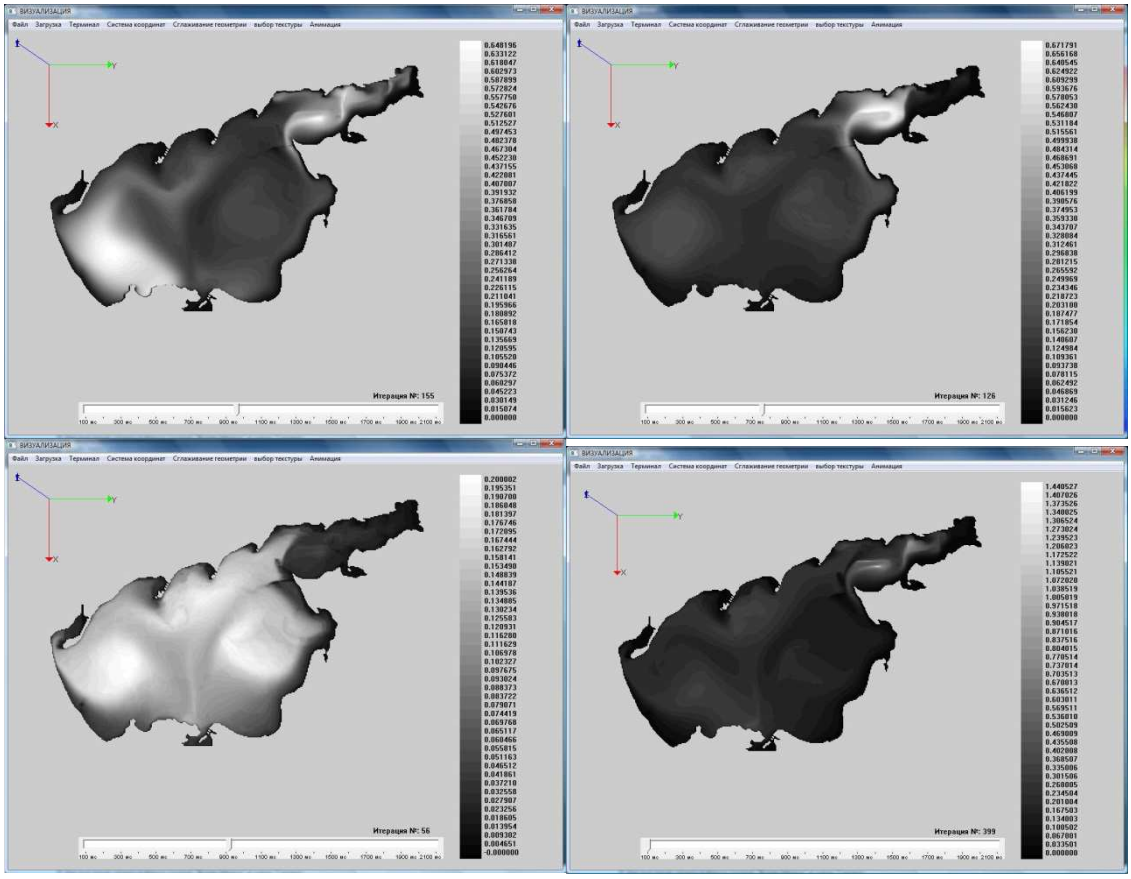


Рис.6. Распределение концентраций детрита, временной интервал T=56, 126, 155, 400 дней

В работе были построены диаграммы распределения концентраций субстанций $X, Z, D, P, S,$ которые характеризуются определенной периодичностью в пространстве и во времени. Для реальной экосистемы (Азовское море и Таганрогский залив) возникающие устойчивые диссипативные структуры соответствуют устойчивой "пятнистости" распреде-

ления популяций планктона по акватории водоема, что является одной из качественных, не нашедшей четкого объяснения в современной гидробиологии, особенностей.

На **Рис. 6 и 7** представлены результаты моделирования возможных сценариев развития экосистемы Азовского моря (изменения концентрации рыбной популяции пеленгас) для северного направления ветра. Белым цветом выделено максимальное значение концентраций пеленгаса и детрита. С помощью представленных результатов численного эксперимента можно исследовать возможный сценарий зарыбления акватории Азовского моря пеленгасом, участвующем в донной мелиорации мест скопления детрита. При значении временного интервала, начиная со 126 дней, согласно представленным результатам, наблюдается уменьшение концентрации детрита, а значит и концентрации донных отложений в центрально-восточной части Азовского моря, что, в конечном счете, приведет к уменьшению площадей заморных зон и улучшению качества вод в данном водоеме.

С помощью разработанного программного комплекса можно исследовать вопросы акклиматизации пеленгаса, в новом для него по экологическому режиму, мелководном водоеме, оценить специфику условий водного тела, что позволит избежать непредвиденных отрицательных технологических влияний и планировать увеличение производства этой рыбы.

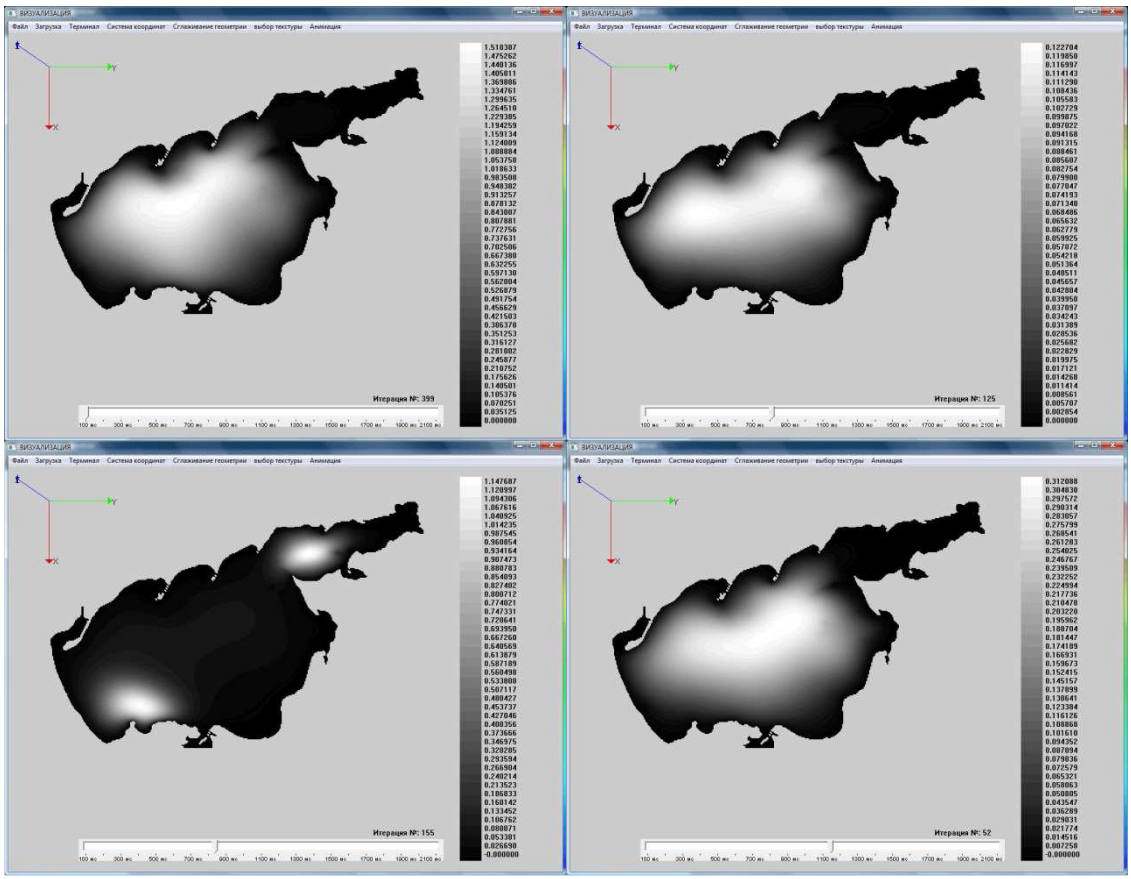


Рис.7. Распределение концентраций пеленгаса, временной интервал T=56, 126, 155, 400 дней

Анализ полученных результатов позволил сделать конкретные выводы о свойствах математической модели детрит - пеленгас и возможностях управления качеством вод мелководных водоемов, таких как Азовское море, с помощью методов математического моделирования.

Рассмотренная картина, демонстрирующая происходящие в мелководном водоеме процессы, является идеальной, так как описывает лишь регулярное периодическое изменение среды обитания и не отражает ее случайных флуктуаций, которые, однако, не изменяют поведение системы принципиально, а лишь "зашумляют" динамику. Это затрудняет выявление периодичностей.

Наблюдаемая в природе закономерность динамики численности рыбных популяций есть результат сложного наложения периодических вариаций условий внешней среды на внутренние периодичности, порождаемые биотическим взаимодействием.

Умеренная внешняя модуляция несколько изменяет внутренний цикл системы, делая его период кратным внешнему.

Заключение

С помощью экспедиционных исследований проведена первичная верификация модели экосистемы Азовского моря [12]. Реализована задача моделирования и прогноза состояния водной экосистемы Азовского моря в условиях антропогенного воздействия и всестороннего изучения уникального водного объекта, который в силу мелководности в большей степени подвержен антропогенному влиянию [13]. Создан исследовательско-прогнозный комплекс, объединяющий математические модели и базы данных, с помощью которого изучены условия, при которых возникают заморы в мелководных водоемах. Разработаны всевозможные сценарии зарыбления мелководных водоемов детритофагом пеленгас с целью возможного уменьшения площадей заморных зон, возникающих в мелководных водоемах.

Отличительными особенностями разрабатываемых алгоритмов, реализующих поставленные гидробиологические модельные задачи, являются: высокая производительность, достоверность и точность получаемых результатов. Высокая производительность достигается за счет использования эффективных численных методов решения сеточных уравнений, ориентированных для применения на параллельных вычислительных системах в реальном и ускоренном масштабах времени. Достоверность достигается за счет учета определяющих физических факторов, таких как: сила Кориолиса, турбулентный обмен, сложная геометрия дна и береговой линии, испарение, стоки рек, динамическое перестроение расчетной области, ветровые напряжения и трение о дно, а также за счет учета отклонения значения поля давления от гидростатического приближения. Точность достигается применением подробных расчетных сеток, учитывающих степень «заполненности» расчетных ячеек [14], а также отсутствием неконсервативных диссипативных слагаемых и нефизичных источников (стоков), возникающих в результате конечно-разностных аппроксимаций. Также в работе показана эффективность алгоритма адаптивного попеременно - треугольного итерационного метода и его параллельной реализации, выполненной на основе декомпозиции области по двум пространственным направлениям, применительно к решению задач гидродинамики мелководных водоемов при достаточно большом количестве вычислителей.

Литература

1. Якушев Е.В., Сухинов А.И. и др. Комплексные океанологические исследования Азовского моря в 28-м рейсе научно-исследовательского судна «Акванавт» // Океанология. 2003. Т. 43, №1. С.44–53.
2. Сухинов А.И., Никитина А.В., Чистяков А.Е. Моделирование сценария биологической реабилитации Азовского моря // Математическое моделирование. 2012. Т.24, №9. С. 3–21.
3. Сухинов А.И., Чистяков А.Е., Алексеенко Е.В. Численная реализация трехмерной модели гидродинамики для мелководных водоемов на супервычислительной системе// Математическое моделирование. 2011. Т. 23, № 3. С.3-21.
4. Сухинов А.И., Чистяков А.Е. Параллельная реализация трехмерной модели гидродинамики мелководных водоемов на супервычислительной системе// Вычислительные методы и программирование: Новые вычислительные технологии. 2012. Т.13. С. 290–297.
5. Белоцерковский О. М. Турбулентность: новые подходы - М.: Наука. 2003. 286 с.
6. Самарский А.А. Теория разностных схем. М. Наука. 1989. 616 с.

7. Белоцерковский О. М., Гущин В. А., Щенников В. В. Метод расщепления в применении к решению задач динамики вязкой несжимаемой жидкости// Ж. вычисл. матем. и матем. физ. 1975. С. 197–207.

8. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М. Наука, 1978. 592 с.

9. Коновалов А.Н. К теории попеременно - треугольного итерационного метода// Сибирский математический журнал. 2002. 43:3. С. 552–572.

10. Сухинов А.И., Чистяков А.Е. Адаптивный модифицированный попеременно-треугольный итерационный метод для решения сеточных уравнений с несамосопряженным оператором// Математическое моделирование. 2012. Т.24, №1. С. 3–20.

11. Чистяков А.Е. Теоретические оценки ускорения и эффективности параллельной реализации ПТМ скорейшего спуска// Известия ЮФУ. Технические науки. 2010. №6(107). С 237–249.

12. Никитина А.В. «Численное решение задачи динамики токсичных водорослей в Таганрогском заливе». // Известия ЮФУ. Технические науки. 2010. №6(107). С 113 - 116.

13. Никитина А.В. Модели биологической кинетики, стабилизирующие экологическую систему таганрогского залива // Известия ЮФУ. Технические науки. 2009. № 8 (97). С. 130–134.

14. Сухинов А.И., Чистяков А.Е., Тимофеева Е.Ф., Шишеня А.В. Математическая модель расчета прибрежных волновых процессов// Математическое моделирование. 2012. Т.24, №8. С. 32–44.

15. Сухинов А.И., Чистяков А.Е., Бондаренко Ю.С. Оценка погрешности решения уравнения диффузии на основе схем с весами// Известия ЮФУ. Технические науки.– 2011. №8 (121). – С 6-13.

Моделирование прямых и обратных задач диффузии-конвекции на многопроцессорных системах для прогноза и ретроспективного анализа водных экосистем

А.И. Сухинов, Д.В. Лапин, А.Е. Чистяков

Южный федеральный университет

В работе рассмотрены двумерные обратные задачи диффузии-конвекции, необходимость оперативного решения которых возникает при ретроспективном анализе техногенных и природных экологических катастроф. Описан численный алгоритм решения обратных задач, основанный на методе квазиобращения и последующего итерационного уточнения начального условия. Приводится описание параллельных алгоритмов и теоретические оценки ускорения эффективности и масштабируемости.

1. Введение

При ретроспективном анализе техногенных и природных катастроф и их воздействия на экологические системы, возникает необходимость решать обратные эволюционные задачи распространения и переноса вещества, для определения места и параметров воздействия на экосистему, времени происхождения выброса. Причем, возникающие в процессе решения сеточные уравнения, часто имеют слишком большую размерность, для решения на одном вычислительном узле. Более того, если ретроспективный анализ производится с целью принять решения для выполнения действий по минимизации ущерба экосистеме, время выполнения анализа играет ключевую роль. Этим и определяется необходимость разработки параллельных алгоритмов для решения обратных задач диффузии-конвекции.

2. Обратная эволюционная задача транспорта вещества

Будем рассматривать некорректную эволюционную задачу с обратным временем получаемую из соответствующей прямой задачи заменой t на $-t$ (т.е. переходом к обратному времени)

$$c'_t - uc'_x - vc'_y = -\left(\mu c'_x\right)'_x - \left(\mu c'_y\right)'_y. \qquad (1)$$

где $U = \{u, v\}$ – компоненты вектора скорости, μ – горизонтальная составляющая коэффициента турбулентного обмена.

Уравнение (1) рассматривается при следующих граничных

$$c'_n(x, y, t)\big|_{(x,y)\in\gamma} = 0 \qquad (2)$$

и начальных условиях:

$$c(x, y, t)\big|_{t=0} = c_0(x, y). \qquad (3)$$

Для приближенного решения некорректной задачи (1) - (3), будем использовать следующее уравнение

$$c'_t - uc'_x - vc'_y = -\left(\mu c'_x\right)'_x - \left(\mu c'_y\right)'_y - \alpha\left(\mu c'''_{xxx}\right)'_x - a\left(\mu c'''_{yyy}\right)'_y. \qquad (4)$$

Данный подход был предложен в работе [1]. Уравнение (4) может быть записано в следующем виде:

$$c'_t - uc'_x - vc'_y = \left(\mu\left(-c - \alpha c''_{xx}\right)'_x\right)'_x + \left(\mu\left(-c - \alpha c''_{yy}\right)'_y\right)'_y. \qquad (5)$$

Для удобства последующей дискретизации уравнение (5) запишем в виде следующей системы уравнений:

$$c'_t - uc'_x - vc'_y = \left(\mu\left(s_x\right)'_x\right)'_x + \left(\mu\left(s_y\right)'_y\right)'_y, \quad (6)$$

$$s_x = -c - \alpha c''_{xx}. \quad (7)$$

$$s_y = -c - \alpha c''_{yy}. \quad (8)$$

Выбор оптимального значения параметра регуляризации α определялся серией численных экспериментов, с использованием последовательности $\alpha_k = \alpha_0 q^k$, $q > 0$ [2], для каждого значения α оценивалось в норме сеточного пространства L2h (ωh), отклонение финального решения цепочки задач - обратная-прямая задача и начального распределения для обратной задачи на сетке $\omega_h = \omega_x \times \omega_y$ (см. ниже).

2.1 Дискретизация обратной задачи транспорта вещества

Расчетная область вписана в прямоугольник. Для численной реализации дискретной математической модели поставленной задачи вводится равномерная сетка $\omega = \omega_t \times \omega_x \times \omega_y$:

$\omega_t = \{t^n = nh_t, 0 \le n \le N_t - 1, l_t = h_t N_t\}$, $\omega_x = \{x_i = ih_x, 0 \le i \le N_x - 1, l_x = h_x N_x\}$,

$\omega_y = \{y_j = jh_y, 0 \le j \le N_y - 1, l_y = h_y N_y\}$,

где n, i, j – индексы по временной координате и пространственным координатным направлениям Ox, Oy соответственно, h_t, h_x, h_y – шаги по временной координате и пространственным координатным направлениям Ox, Oy соответственно, N_t, N_x, N_y – количество узлов по временной координате и пространственным координатным направлениям Ox, Oy соответственно, l_t, l_x, l_y – длина расчетной области по временной координате и пространственным координатным направлениям Ox, Oy соответственно.

Дискретные аналоги операторов конвективного и диффузионного переноса в случае граничных условий в форме Неймана могут быть записаны в следующем виде:

$$uc'_x \quad \left(q_1\right)_{i,j} u_{i+1/2,j} \frac{c_{i+1,j} - c_{i,j}}{2h_x} + \left(q_2\right)_{i,j} u_{i-1/2,j} \frac{c_{i,j} - c_{i-1,j}}{2h_x},$$

$$\left(\mu c'_x\right)'_x \quad \left(q_1\right)_{i,j} \mu_{i+1/2,j} \frac{c_{i+1,j} - c_{i,j}}{h_x^2} - \left(q_2\right)_{i,j} \mu_{i-1/2,j} \frac{c_{i,j} - c_{i-1,j}}{h_x^2},$$

где q_m - коэффициенты заполненности контрольных областей [2].

Дискретный аналог уравнения (6) с учетом приведенных выше аппроксимаций запишется в следующем виде:

$$\left(q_0\right)_{i,j} \frac{c_{i,j}^{n+1} - c_{i,j}^n}{h_t} - \left(q_1\right)_{i,j} u_{i+1/2,j} \frac{c_{i+1,j}^{n+\sigma_1} - c_{i,j}^{n+\sigma_1}}{2h_x} - \left(q_2\right)_{i,j} u_{i-1/2,j} \frac{c_{i,j}^{n+\sigma_1} - c_{i-1,j}^{n+\sigma_1}}{2h_x} - \quad (9)$$

$$-\left(q_3\right)_{i,j} v_{i,j+1/2} \frac{c_{i,j+1}^{n+\sigma_2} - c_{i,j}^{n+\sigma_2}}{2h_y} - \left(q_4\right)_{i,j} v_{i,j-1/2} \frac{c_{i,j}^{n+\sigma_2} - c_{i,j-1}^{n+\sigma_1}}{2h_y} = \left(q_1\right)_{i,j} \mu_{i+1/2,j} \frac{\left(s_x\right)_{i+1,j}^{n+\sigma_1} - \left(s_x\right)_{i,j}^{n+\sigma_1}}{h_x^2} -$$

$$-\left(q_2\right)_{i,j} \mu_{i-1/2,j} \frac{\left(s_x\right)_{i,j}^{n+\sigma_1} - \left(\overline{s}_x\right)_{i-1,j}^{n+\sigma_1}}{h_x^2} + \left(q_3\right)_{i,j} \mu_{i,j+1/2} \frac{\left(\overline{s}_y\right)_{i,j+1}^{n+\sigma_2} - \left(\overline{s}_y\right)_{i,j}^{n+\sigma_2}}{h_y^2} - \left(q_4\right)_{i,j} \mu_{i,j-1/2} \frac{\left(\overline{s}_y\right)_{i,j}^{n+\sigma_2} - \left(\overline{s}_y\right)_{i,j-1}^{n+\sigma_2}}{h_y^2}$$

,

где $c_{i,j}^{n+\sigma} = \sigma c_{i,j}^{n+1} + \left(1-\sigma\right) c_{i,j}^n$, $\sigma \in [0,1]$ - вес схемы.

Дискретные аналоги уравнений (7) – (8) с запишутся в следующем виде:

$$\left(q_0\right)_{i,j} \left(s_x\right)_{i,j} = -\left(q_0\right)_{i,j} c_{i,j} - \alpha\left(\left(q_1\right)_{i,j} \frac{c_{i+1,j} - c_{i,j}}{h_x^2} - \left(q_2\right)_{i,j} \frac{c_{i,j} - c_{i-1,j}}{h_x^2}\right), \quad (10)$$

$$(q_0)_{i,j}(s_y)_{i,j} = -(q_0)_{i,j} c_{i,j} - \alpha\left((q_3)_{i,j} \frac{c_{i,j+1} - c_{i,j}}{h_y^2} - (q_4)_{i,j} \frac{c_{i,j} - c_{i,j-1}}{h_y^2} \right). \quad (11)$$

Для численного решения уравнения (5) с соответствующими начальными и граничными условиями используется схема расщепления по геометрическим направлениям – локально-одномерная при этом для четных шагов веса схемы равны $\sigma_1 = 1, \sigma_2 = 0$, для нечетных – $\sigma_1 = 0, \sigma_2 = 1$.

Прямое использование разностной схемы для численного решения уравнения (5) связано с численным решением сеточного эллиптического уравнения четвертого порядка, что достаточно сложно с точки зрения вычислительной трудоемкости. Поэтому целесообразно выполнить переход к аддитивной схеме расщепления по пространственным переменным [3] вида

$$\frac{y_{n+1} - y_n}{\tau} + \sum_{\beta=1}^{2} \left(E + \alpha\sigma\tau\Lambda_\beta^2 \right)^{-1} \left(\alpha\Lambda_\beta^2 - \Lambda_\beta \right) y_n = 0$$

Помимо уменьшения вычислительной сложности, переход к усреднено-аддитивной схеме позволяет организовать параллельные вычисления, т.к. в таком случае требуется решать серию локально-одномерных, независимых по данным задач. Из этих же соображений для решения прямых задач также выбрана аддитивно-усредненная схема.

3. Математическая модель гидродинамики

Входными данными сконструированной выше обратной эволюционной задачи распространения и переноса вещества, является поле вектора скорости, что требует, в свою очередь, построения математической модели движения водной среды. Исходными уравнениями гидродинамики мелководных водоемов являются[5-7]:

– уравнения движения (Навье – Стокса):

$$u'_t + uu'_x + vu'_y + wu'_z = -\frac{1}{\rho} p'_x + \left(\mu u'_x\right)'_x + \left(\mu u'_y\right)'_y + \left(\nu u'_z\right)'_z + 2\Omega(v\sin\theta - w\cos\theta),$$

$$v'_t + uv'_x + vv'_y + wv'_z = -\frac{1}{\rho} p'_y + \left(\mu v'_x\right)'_x + \left(\mu v'_y\right)'_y + \left(\nu v'_z\right)'_z - 2\Omega u\sin\theta, \quad (12)$$

$$w'_t + uw'_x + vw'_y + ww'_z = -\frac{1}{\rho} p'_z + \left(\mu w'_x\right)'_x + \left(\mu w'_y\right)'_y + \left(\nu w'_z\right)'_z + 2\Omega u\cos\theta;$$

– уравнение неразрывности в случае переменной плотности запишется следующим образом:

$$\rho'_t + \left(\rho u\right)'_x + \left(\rho v\right)'_y + \left(\rho w\right)'_z = 0. \quad (13)$$

где $U = \{u, v, w\}$ – компоненты вектора скорости, p – превышение давления над гидростатическим давлением невозмущенной жидкости, ρ – плотность, Ω – угловая скорость вращения земли, θ – угол между вектором угловой скорости и вертикалью, μ,ν – горизонтальная и вертикальная составляющая коэффициента турбулентного обмена.

Система уравнений (12) – (13) рассматривается при следующих граничных условиях:

– на входе (устье рек Дон и Кубань) –

$$u(x,y,z,t) = u(t), \quad v(x,y,z,t) = v(t), \quad p'_n(x,y,z,t) = 0, \quad \overline{u}'_n(x,y,z,t) = 0,$$

– боковая граница (берег и дно) –

$$\rho_v \mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho_v \mu(v')_n(x,y,z,t) = -\tau_y(t),\ \overline{u}_n(x,y,z,t) = 0,\ p'_n(x,y,z,t) = 0,$$

– верхняя граница –

$$\rho\mu(u')_n(x,y,z,t) = -\tau_x(t),\ \rho\mu(v')_n(x,y,z,t) = -\tau_y(t), \quad (14)$$

$$w(x,y,t) = -\omega - p'_t / \rho g,\ p'_n(x,y,t) = 0,$$

– на выходе (Керченский пролив) –

$$p'_n(x,y,z,t) = 0, \quad \overline{u}'_n(x,y,z,t) = 0,$$

где ω - интенсивность испарения жидкости, τ_x, τ_y – составляющие тангенциального напряжения (закон Ван-Дорна), ρ_v – плотность взвеси.

Составляющие тангенциального напряжения для свободной поверхности:

$\tau_x = \rho_a C_p\left(|\vec{w}|\right) w_x |\vec{w}|$, $\tau_y = \rho_a C_p\left(|\vec{w}|\right) w_y |\vec{w}|$,

где $\vec{w}$ – вектор скорости ветра относительно воды, ρ_a – плотность атмосферы,

$C_p(x) = \begin{cases} 0.0088, & x < 6,6\ м/с \\ 0.0026, & x \geq 6,6\ м/с \end{cases}$ — безразмерный коэффициент.

Составляющие тангенциального напряжения для дна, с учетом введенных обозначений, могут быть записаны следующим образом:

$\tau_x = \rho C_p\left(|\overline{u}|\right) u |\overline{u}|$, $\tau_y = \rho C_p\left(|\overline{u}|\right) v |\overline{u}|$.

Рассмотренная ниже аппроксимация позволяет на основании измеренных (или вычисленных) пульсаций скоростей получить коэффициент вертикального турбулентного обмена, неоднородный по глубине[9], в соответствии с формулой:

$$\nu = C_s^2 \Delta^2 \frac{1}{2} \sqrt{\left(\frac{\partial \overline{U}}{\partial z}\right)^2 + \left(\frac{\partial \overline{V}}{\partial z}\right)^2}\ , \qquad (15)$$

где $\overline{U}, \overline{V}$ осредненные по времени пульсации горизонтальных компонент скорости, Δ - характерный масштаб сетки, C_s - безразмерная эмпирическая константа, значение которой обычно определяется на основе расчета процесса затухания однородной изотропной турбулентности.

3.1 Построение и исследование дискретной модели

Расчетная область вписана в параллелепипед. Для численной реализации дискретной математической модели поставленной задачи гидродинамики вводится равномерная сетка:

$\overline{w}_h = \{ t^n = n\tau, x_i = ih_x, y_j = jh_y, z_k = kh_z;\ n = \overline{0..N_t}, i = \overline{0..N_x}, j = \overline{0..N_y}, k = \overline{0..N_z};$

$N_t\tau = T, N_x h_x = l_x, N_y h_y = l_y, N_z h_z = l_z \}$,

где τ – шаг по времени, h_x, h_y h_z – шаги по пространству, N_t – количество временных слоев, T – верхняя граница по временной координате, N_x, N_y N_z – количество узлов по пространственным координатам, l_x, l_y l_z – границы параллелепипеда в направлении осей Ox, Oy и Oz соответственно.

Для решения задачи гидродинамики использовался метод поправки к давлению [7,8]. Вариант данного метода в случае переменной плотности примет вид:

$$\begin{aligned}
&\frac{\tilde{u}-u}{\tau} + u\overline{u}'_x + v\overline{u}'_y + w\overline{u}'_z = \left(\mu\overline{u}'_x\right)'_x + \left(\mu\overline{u}'_y\right)'_y + \left(\nu\overline{u}'_z\right)'_z + 2\Omega(v\sin\theta - w\cos\theta)\,,\\
&\frac{\tilde{v}-v}{\tau} + u\overline{v}'_x + v\overline{v}'_y + w\overline{v}'_z = \left(\mu\overline{v}'_x\right)'_x + \left(\mu\overline{v}'_y\right)'_y + \left(\nu\overline{v}'_z\right)'_z - 2\Omega u\sin\theta\,,\\
&\frac{\tilde{w}-w}{\tau} + u\overline{w}'_x + v\overline{w}'_y + w\overline{w}'_z = \left(\mu\overline{w}'_x\right)'_x + \left(\mu\overline{w}'_y\right)'_y + \left(\nu\overline{w}'_z\right)'_z + 2\Omega u\cos\theta\,, \qquad (16)\\
&p''_{xx} + p''_{yy} + p''_{zz} = \frac{\hat{\rho}-\rho}{\tau^2} + \frac{\left(\hat{\rho}\tilde{u}\right)'_x}{\tau} + \frac{\left(\hat{\rho}\tilde{v}\right)'_y}{\tau} + \frac{\left(\hat{\rho}\tilde{w}\right)'_z}{\tau}\,,\\
&\frac{\hat{u}-\tilde{u}}{\tau} = -\frac{1}{\rho}\hat{p}'_x\,,\ \frac{\hat{v}-\tilde{v}}{\tau} = -\frac{1}{\rho}\hat{p}'_y\,,\ \frac{\hat{w}-\tilde{w}}{\tau} = -\frac{1}{\rho}\hat{p}'_z\,,
\end{aligned}$$

где $\{u, v, w\}$ – компоненты вектора скорости, $\{\hat{u}, \hat{v}, \hat{w}\}$, $\{\tilde{u}, \tilde{v}, \tilde{w}\}$ – компоненты полей вектора скорости на «новом» и промежуточном временных слоях соответственно, $\overline{u} = (\tilde{u} + u)/2$, $\hat{\rho}$ и

ρ - распределение плотности водной среды на новом и предыдущем временных слоях соответственно.

При построении дискретных математических моделей гидродинамики также учитывалась «заполненность» [4] контрольных ячеек, что позволяет повысить реальную точность решения в случае сложной геометрии исследуемой области за счет улучшения аппроксимации границы.

Через $o_{i,j,k}$ обозначена «заполненность» ячейки (i,j,k). Степень «заполненности» ячейки определяется давлением столба жидкости внутри данной ячейки. Если среднее давление в узлах, которые относятся к вершинам рассматриваемой ячейки, больше давления столба жидкости внутри ячейки, то ячейка считается заполненной полностью $\left(o_{i,j,k}=1\right)$. В общем случае «заполненность» ячеек можно вычислить по следующей формуле:

$$o_{i,j,k}=\frac{P_{i,j,k}+P_{i-1,j,k}+P_{i,j-1,k}+P_{i-1,j-1,k}}{4\rho g h_z}, \tag{17}$$

где $P=p+\rho gz$ – давление.

Погрешность аппроксимации математической модели равна $\mathrm{O}\left(\tau+\|h\|^2\right)$, где $\|h\|=\sqrt{h_x^2+h_y^2+h_z^2}$. Доказано сохранение потока на дискретном уровне разработанной гидродинамической модели, а также отсутствие неконсервативных диссипативных слагаемых, полученных в результате дискретизации системы уравнений. Достаточное условие устойчивости и монотонности разработанной модели определяется на основе принципа максимума [10] при ограничениях на шаг по пространственным координатам:

$h_x<\left|2\mu/u\right|$, $h_y<\left|2\mu/v\right|$, $h_z<\left|2\nu/w\right|$ или $\mathrm{Re}\le 2N$,

где $\mathrm{Re}=\left|V\right|\cdot l/\mu$ – числа Рейнольдса, l – характерный размер области, $N=\max\left\{N_x,N_y,N_z\right\}$.

Дискретные аналоги системы уравнений (16) решаются адаптивным модифицированным попеременно – треугольным методом вариационного типа[11-13].

4. Параллельные алгоритмы

4.1 Параллельная реализация для систем с общей памятью

Параллельная реализация основана на том факте, что при выборе аддитивно-усредненной схемы расщепления по пространственным переменным, получаемые локально-одномерные задачи не зависят по данным и могут решаться одновременно. В рассматриваемом двухмерном случае необходимо выполнить два шага расщепления.

Так как аддитивно-усредненная схема выбрана и для прямой и для обратной задачи, то принцип параллельного выполнения одинаков для обоих задач, разница заключается лишь в том, что использование аддитивно-усредненной схемы для прямой задачи приводит к необходимости решать трехдиагональные СЛАУ, а для обратной задачи — пятидиагональные СЛАУ. Такие СЛАУ решаются методом, соответственно, трех- и пятидиагональной прогонки [3].

Время, необходимое для решения таких СЛАУ на системе с общей памятью [14], имеющей k вычислителей можно оценить, соответственно, как:

$\frac{\left(7\cdot \mathrm{N}\cdot \mathrm{M}+7\cdot \mathrm{M}\cdot \mathrm{N}\right)t_a}{k}$ и $\frac{\left(23\cdot \mathrm{N}\cdot \mathrm{M}+23\cdot \mathrm{M}\cdot \mathrm{N}\right)t_a}{k}$, где t_a - время выполнения одной арифметической операции, N и M - размеры вычислительной области.

После выполнения всех (K) шагов по времени, для каждой из задач, выполняется расчет невязки, требующий времени $5\cdot N\cdot M\cdot t_a$ для выполнения арифметических операций.

4.2 Параллельная реализация для систем с распределенной памятью

Параллельная реализация для систем с распределенной памятью отличается необходимостью выполнять транспонирование матрицы на решающем поле, **Рис. 1**.

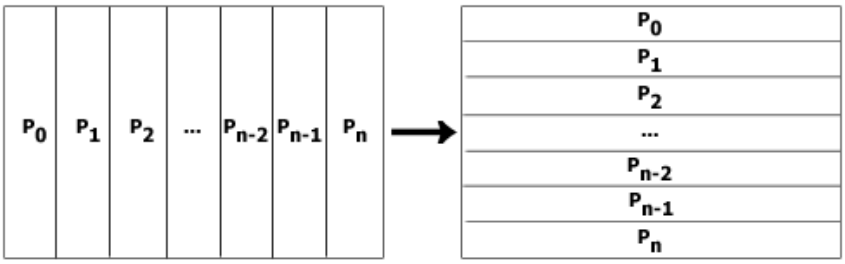


Рис. 1. Распределение неизвестных по узлам, транспонирование

При этом между шагами расщепления по пространственным переменным необходимо передать $N \cdot M \cdot (1 - 1/p)$ чисел с плавающей запятой двойной точности, т.е. $N \cdot M \cdot (1 - 1/p) \cdot 8$ байт. Здесь p - количество вычислителей в системе с распределенной памятью. Этот объем данных передается за p операций передач. Таким образом, время затрачиваемое на передачу этих данных составляет $p \cdot t_l + N \cdot M \cdot (1 - 1/p) \cdot 8 \cdot t_{1b}$. Здесь t_l - латентность (время необходимое для инициализации передачи), а t_{1b} - время, необходимое на передачу одного байта информации.

Для перехода к следующему шагу по времени для прямой задачи затрачивается время $\left((7 \cdot \mathrm{N} \cdot \mathrm{M} + 7 \cdot \mathrm{M} \cdot \mathrm{N}) t_a\right)/p$ на арифметические операции и $2\left(p \cdot t_l + N \cdot M \cdot (1 - 1/p) \cdot 8 \cdot t_{1b}\right)$ на операции обмена, для обратной задачи время обмена такое же, а время выполнения арифметических операций составляет $\left((23 \cdot \mathrm{N} \cdot \mathrm{M} + 23 \cdot \mathrm{M} \cdot \mathrm{N}) t_a\right)/p$. После выполнения всех (K) шагов по времени, для каждой из задач, выполняется расчет невязки, требующий времени $5 \cdot N \cdot M \cdot t_a$ для выполнения арифметических операций и $t_l + p \cdot 8 \cdot t_{1b}$ на выполнения операций обмена.

4.3 Параллельная реализация для гибридных систем

Параллельная реализация для гибридных систем фактически совмещает подходы использованные в реализациях для систем с общей и распределенной памятью. А именно, данные распределяются между узлами вычислительной системы как для системы с распределенной памятью, имеют место те же самые обмены, как для системы с распределенной памятью. При этом на каждом узле, которой является системой с общей памятью, все выделенные ему локально-одномерные задачи решаются параллельно, как на системе с общей памятью.

Таким образом, используя полученные ранее оценки для времени параллельного выполнения на системах с общей и распределенной памятью, выражение для ускорения при выполнении на гибридной вычислительной системе может быть записано в следующем виде:

$$S_p = \frac{60 \cdot L \cdot t_a \cdot K + 5 \cdot L \cdot t_a}{5 \cdot L \cdot t_a + K \cdot \left(4 \cdot p \cdot t_l + L \cdot (1 - 1/p) \cdot 8 \cdot t_{1b} + 60 \cdot L \cdot t_a / (k \cdot p)\right) + t_l + p \cdot 8 \cdot t_{1b}},$$

где $L = M \cdot N$ - характерный размер задачи.

Тогда эффективность параллельного выполнения на гибридной вычислительной системе:

$$E_p = \frac{S_p}{k \cdot p}$$

Масштабируемость предложенного параллельного алгоритма может выполняться до тех пор, пока $\min(M, N) \geq k \cdot p$.

Подставим в полученные оценки данные имеющейся в ЮФУ гибридной вычислительной системы. Вычислительная система ЮФУ обладает следующими характеристиками:

$t_l = 1.82 \cdot 10^{-6}\, c$, $t_{1b} = 8.16 \cdot 10^{-13}\, c$, $t_a = 1.09 \cdot 10^{-10}\, c$.

Подставляя эти данные, построим график для теоретической оценки ускорения (рис. 2 сплошная кривая), который достаточно хорошо согласуются с экспериментально полученными данными ускорения для гибридной супервычислительной системы ЮФУ (пунктирная кривая).

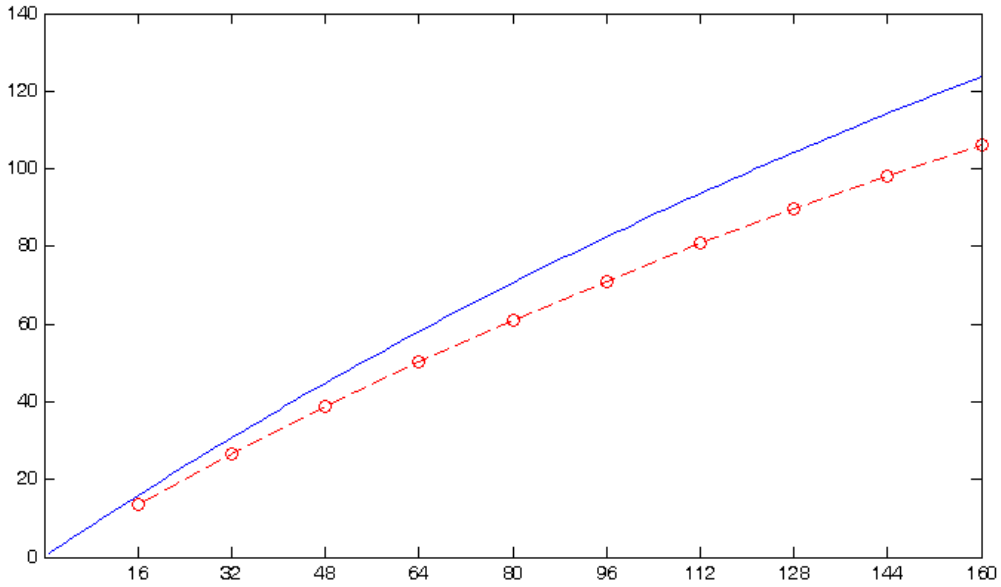


Рис. 2. Зависимости ускорения от числа процессоров. Гладкая кривая – теоретическая зависимость, ломаная – экспериментальная

5. Результаты численных экспериментов

Численные эксперименты выполнялись на супервычислительной системе ЮФУ, представляющей собой гибридную вычислительную систему, состоящую из 128 узлов, объединенных сетью Infiniband 4x DDR ConnectX. Каждый вычислительный узел, в свою очередь, оборудован четырьмя 4-х ядерными процессорами AMD Opteron 8356 2.3GHz и 32-мя гигабайтами оперативной памяти. Используемые в статье оценки производительности сетевой инфраструктуры вычислительной системы ЮФУ получены в результате выполнения тестирования в пакете Pallas MPI benchmark.

Поля скоростей водного потока, рассчитанные на основе математической модели (12)–(14), относятся к входным данным для модели транспорта вещества (1)–(2). На основе разработанных алгоритмов был построен комплекс программ, предназначенный для моделирования возможных сценариев развития экосистемы Азовского моря. На основе разработанного комплекса программ был поставлен численный эксперимент, результаты которого приведены на **Рис. 3** – **Рис. 5**. При численном решении задачи в начальный момент времени поле концентрации загрязняющих веществ распределено равномерно в некоторой части водоема, данная область показана на **Рис. 3**. После чего была решена задача транспорта веществ, результаты расчетов приведены на **Рис. 4**. Палитрой показано значение концентрации вещества.

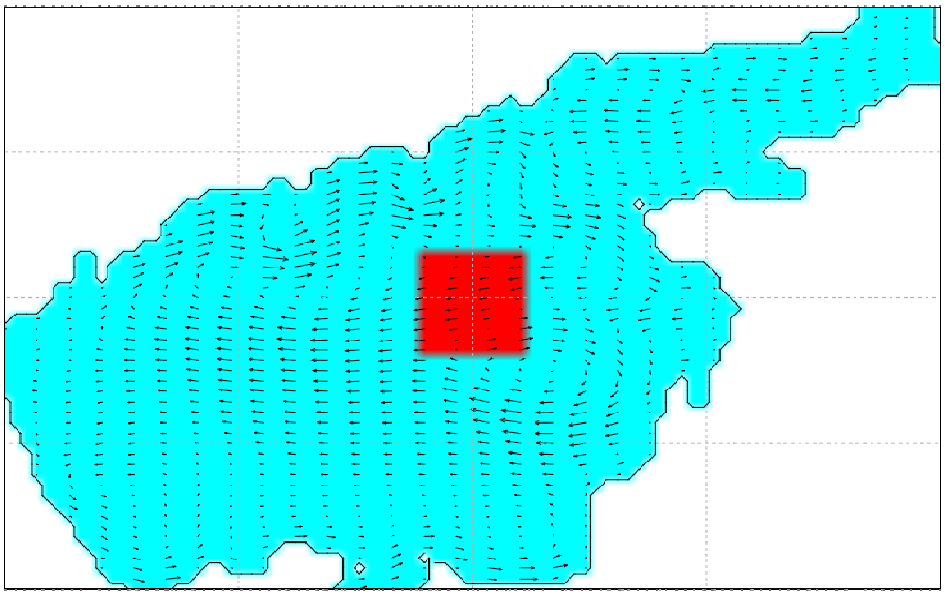


Рис. 3. Исходное распределение концентрации загрязняющих веществ (начальное распределения для прямой задачи)

Максимальное значение концентрации загрязняющих веществ для решенной прямой задачи (**Рис. 5**) составляет 45% от исходного значения концентрации в точке выброса (**Рис. 4**). Максимальное значение концентрации загрязняющих веществ для обратной задачи (**Рис. 6**) составляет 65%.

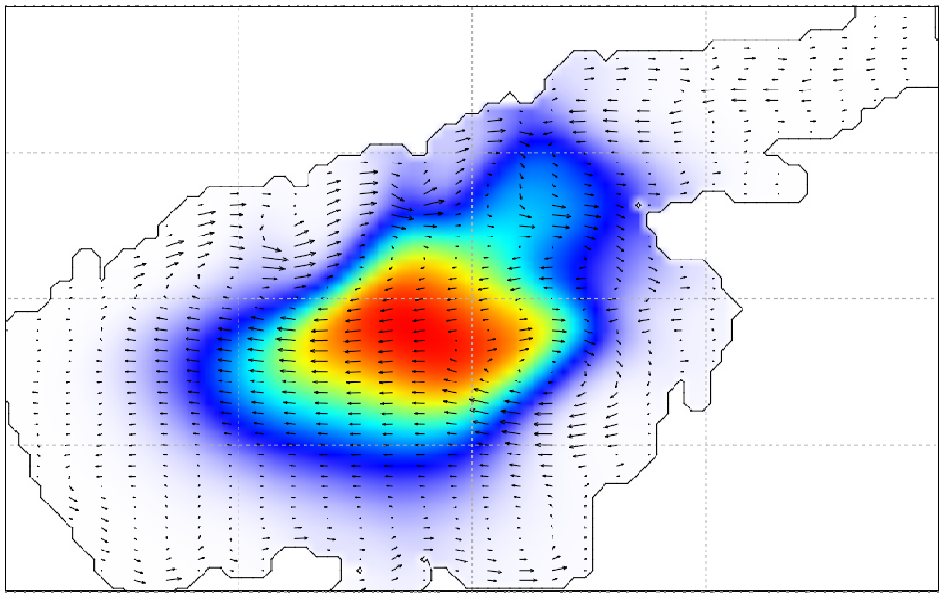


Рис. 4. Распределение концентрации загрязняющих веществ через заданный интервал времени (решение прямой задачи)

Поле концентрации загрязняющих веществ, приведенные на **Рис. 4**, относится к входным данным для обратной задачи транспорта вещества. Результаты расчета обратной задачи приведены на **Рис. 5**.

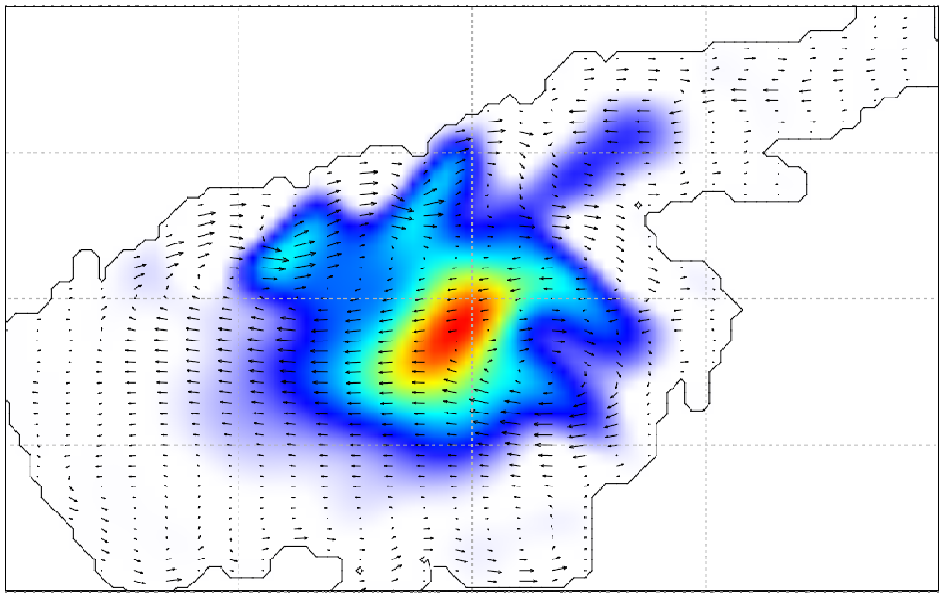


Рис. 5. Восстановленное распределение концентрации загрязняющих веществ (решение обратной задачи)

Ниже приведены результаты численных экспериментов по моделированию течений в Азовском море (**Рис. 6**). Палитрой показана интенсивность течения.

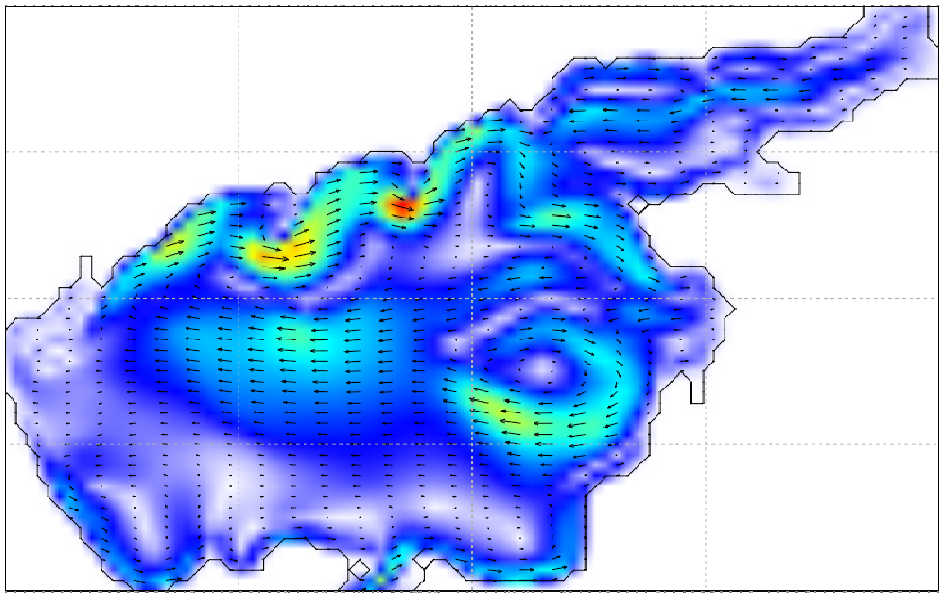


Рис. 6. Поле вектора скорости движения водной среды при западном ветре 5 м/с (баротропные течения)

Из **Рис. 3**- **Рис. 5** видно, что при решении обратной задачи транспорта вещества (**Рис. 5**) исходное поле концентрации (**Рис. 3**) восстанавливается частично, однако, несмотря на это, предложенная математическая модель позволяет достаточно точно восстановить сценарий распространения загрязняющих веществ (экологической катастрофы). Из приведенных результатов расчетов также видно, что при решении обратной задачи происходит локализации области, где предположительно, изначально было распределено загрязняющее вещество (локализация предположительной области выброса загрязняющих веществ). Следует отметить, что точность восстановления исходной концентрации загрязняющих веществ зависит от структуры течения и расчетного временного интервала, а также значения регуляризирующего параметра.

6. Заключение

Отличительными особенностями разрабатываемых алгоритмов являются: высокая производительность, достоверность и приемлемая точность получаемых результатов. Высокая производительность достигается за счет использования эффективных численных методов решения сеточных уравнений, ориентированных для применения на параллельных вычислительных системах в реальном и ускоренном масштабах времени. Достоверность достигается за счет учета определяющих физических факторов, таких как: сила Кориолиса, турбулентный обмен, сложная геометрия дна и береговой линии, испарение, стоки рек, динамическое перестроение расчетной области, ветровые напряжения и трение о дно, а также за счет учета отклонения значения поля давления от гидростатического приближения. Точность достигается применением подробных расчетных сеток, учитывающих степень «заполненности» расчетных ячеек, а также отсутствием неконсервативных и диссипативных слагаемых и нефизичных источников (стоков), в дискретных аппроксимациях исходных задач.

Литература

1. Самарский А. А., Вабищевич П. Н., Васильев В. И., “Итерационное решение ретроспективной обратной задачи теплопроводности”, Матем. моделирование, 9:5 (1997), 119–127
2. Тихонов А.Н., Арсенин В.Я., Методы решения некорректных задач — М.: Наука, 1979.
3. Самарский А.А., Вабищевич П.Н., Численные методы решения обратных задач математической физики — М.: Эдиториал УРСС, 2004
4. Сухинов А.И., Чистяков А.Е., Тимофеева Е.Ф., Шишеня А.В. Математическая модель расчета прибрежных волновых процессов// Математическое моделирование. – 2012. – Т.24, №8, – С. 32–44.
5. Сухинов А.И., Никитина А.В., Чистяков А.Е. Моделирование сценария биологической реабилитации Азовского моря // Математическое моделирование. – 2012. – Т.24, №9, – С. 3–21.
6. Сухинов А.И., Чистяков А.Е. Параллельная реализация трехмерной модели гидродинамики мелководных водоемов на супервычислительной системе// Вычислительные методы и программирование: Новые вычислительные технологии. – 2012. – Т.13. – С. 290–297.
7. Сухинов А.И., Чистяков А.Е., Алексеенко Е.В. Численная реализация трехмерной модели гидродинамики для мелководных водоемов на супервычислительной системе". Математическое моделирование. 2011. Т. 23, № 3. – С.3-21.
8. Белоцерковский О. М., Гущин В. А., Щенников В. В.. Метод расщепления в применении к решению задач динамики вязкой несжимаемой жидкости. Ж. вычисл. матем. и матем. физ., 15:1 (1975), 197–207.
9. Белоцерковский О. М. Турбулентность: новые подходы - М.: Наука, 2003
10. Самарский А.А. Теория разностных схем. М. Наука, 1989.
11. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М. Наука, 1978.
12. Коновалов А.Н. К теории попеременно - треугольного итерационного метода// Сибирский математический журнал, 2002, 43:3, с. 552-572.
13. Сухинов А.И., Чистяков А.Е. Адаптивный модифицированный попеременно-треугольный итерационный метод для решения сеточных уравнений с несамосопряженным оператором// Математическое моделирование. – 2012. – Т.24, №1, – С. 3–21.
14. Воеводин В. В., Воеводин Вл. В., Параллельные вычисления — СПб.: БХВ-Петербург, 2004, сс. 134-154.

18-я редакция списка Top50 самых мощных компьютеров России: ожидания и перспективы

А.С. Антонов, Д.А. Никитенко, С.И. Соболев

Научно-исследовательский вычислительный центр МГУ имени М.В. Ломоносова

Список Top50 наиболее мощных компьютеров России и СНГ ведется с 2004 г. В преддверии выхода новой 18-й редакции списка в статье кратко описаны некоторые тенденции, ожидания и перспективы развития суперкомпьютерного потенциала России. 18-я редакция списка Top50 будет объявлена в первый день работы конференции «Параллельные вычислительные технологии (ПаВТ) 2013».

Чтобы помочь правильно сориентироваться в мире высокопроизводительных вычислительных систем и иметь возможность оперативно отслеживать тенденции развития данной области, Межведомственный суперкомпьютерный центр РАН и Научно-исследовательский вычислительный центр МГУ имени М.В. Ломоносова в мае 2004 года начали совместный проект по формированию списка 50 наиболее мощных компьютеров России и стран СНГ [1-6].

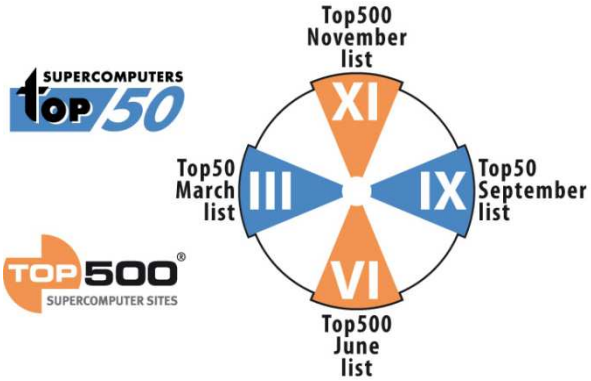


Рис. 1. Распределение времени анонсирования рейтингов Top50 и Top500.

В список включаются 50 вычислительных систем, установленных на территории СНГ и показавших к моменту выхода списка наибольшую производительность на тесте Linpack. Текущая версия списка, архив редакций, статистика, дополнительная информация по вошедшим в рейтинг суперкомпьютерным системам, заявочная форма для участия в очередной редакции списка – все это доступно на web-сайте Top50: http://top50.supercomputers.ru.

Список обновляется и публикуется два раза в год – в начале осени и в начале весны. Очередная редакция списка объявляется в первый день работы крупнейших из проводящихся в России международных научных конференций серии «Научный сервис в сети Интернет» и «Параллельные вычислительные технологии (ПаВТ)». Такое время публикации рейтинга выбрано не случайно. Смещение на квартал относительно времени публикации мирового списка Top500 (http://top500.org), представляемого традиционно в ходе крупнейшей европейской суперкомпьютерной конференции серии «International Supercomputing Conference (ISC)», проводящейся в начале лета в Германии, и крупнейшей мировой суперкомпьютерной конференции серии «Supercomputing Conference (SC)» (ноябрь, США), дает возможность иметь четыре относительно равномерно распределенные «контрольные точки» (рис. 1) для оценки суперкомпьютерного потенциала России и стран СНГ. Разумеется, это относится только к наиболее выдающимся системам списка Top50, которые могут войти в мировой рейтинг Top500.

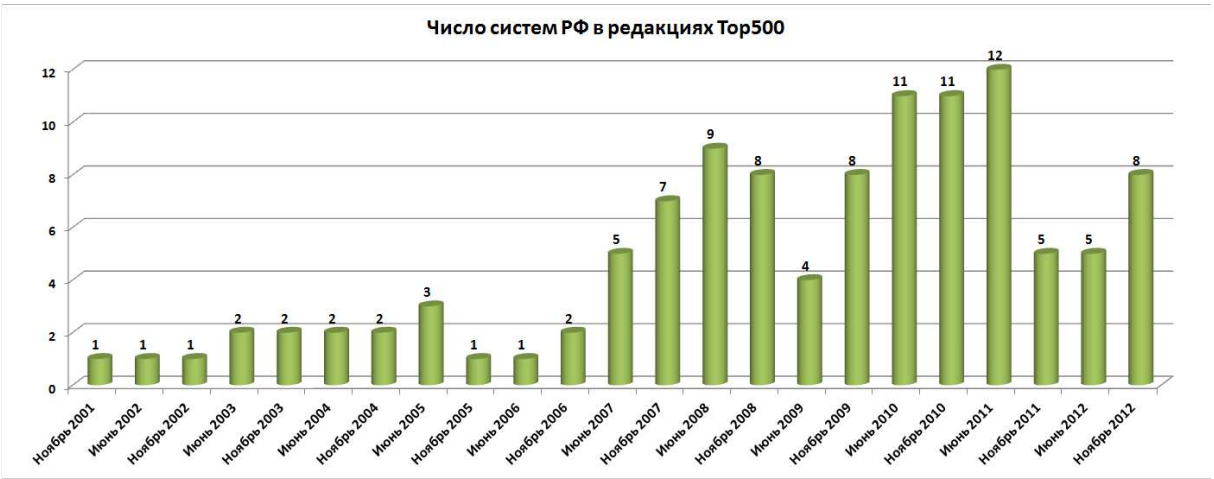


Рис. 2. Число российских систем в редакциях рейтинга Top500.

Число таких систем разнится из года в год. Хотя общая тенденция свидетельствует скорее о росте числа российских систем в международном рейтинге (рис. 2), «экзафлопсная гонка» в странах США, Азии и Европы вносит в этот рост свои существенные коррективы.

Наиболее часто имеют место обновления и изменения конфигурации суперкомпьютеров высокого уровня производительности. Это обусловлено как постоянно растущей потребностью в вычислительных ресурсах в крупных суперкомпьютерных центрах, обслуживающих большое число рабочих групп, так и тем, что зачастую крупные системы вводятся в эксплуатацию поэтапно, постепенно наращивая вычислительную мощность до своего расчетного максимума.

Рис. 3. Географическое распределение систем, вошедших в редакцию 18.09.2012 Top50.

Наибольшая концентрация вычислительных ресурсов, безусловно, наблюдается в Москве. Действительно, флагманская система «Ломоносов», системы Курчатовского института и МСЦ РАН вносят существенный вклад не только в верхушку, но и во весь список в целом. Одновременно выделяются и региональные центры, среди которых в первую очередь хотелось бы отметить Нижний Новгород, Томск и Челябинск. Именно там, в Нижегородском, Южно-Уральском и Томском государственном университете, сформированы опорные научно-образовательные центры национальной Системы НОЦ «Суперкомпьютерные технологии», созданной в ходе реализации проекта комиссии Президента РФ по модернизации и технологическому развитию экономики России «Создание системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения» [7-8].

Рис. 4. Число обновленных и новых систем по редакциям списка Top50.

Порог вхождения в редакцию от 18 сентября 2012 г. составил 11 TFlops на тесте Linpack. При этом в диапазоне 10-15 TFlops находится 11 систем, что вряд ли позволит существенно подняться порогу вхождения в грядущей редакции, принимая во внимание последние темпы обновления списка (рис.4).

Традиционно прием заявок на участие в рейтинге Top50 завершается примерно за месяц до его публикации – 1 марта и 1 сентября для весенней и осенней редакции соответственно. Все поданные данные должны быть проверены, потому статью с обзором невозможно опубликовать в рамках трудов той же конференции. В связи с этим статистика и анализ текущей редакции традиционно представляется в виде доклада на стендовой секции.

Литература

1. Антонов А.С., Никитенко Д.А., Соболев С.И. Рейтинг Топ50: тенденции // Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции (Новосибирск, 26-30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. С. 733.
2. Никитенко Д.А. Рейтинг Top50 как индикатор развития области HPC // Труды X международной конференции «Высокопроизводительные параллельные вычисления на кластерных системах (HPC-2010)». Пермь, 2010. Т. 2. С. 144-148.
3. Никитенко Д.А. Самые производительные и самые «зеленые» системы к концу 2010-го // Суперкомпьютеры. 2010. № 4. С. 58-62.
4. Никитенко Д.А. Рейтинг Top500 – что нового? // Суперкомпьютеры. 2010. № 3. С. 56–59.
5. Никитенко Д.А. Top50 – рейтинг наиболее мощных суперкомпьютеров СНГ // Суперкомпьютеры. 2010. № 2. С. 76-80.
6. Никитенко Д.А. Топ 50. Рейтинг наиболее производительных вычислительных систем СНГ // Численные методы, параллельные вычисления и информационные технологии: Сборник научных трудов / Под ред. Вл.В. Воеводина и Е.Е. Тыртышникова. М.: Изд-во МГУ, 2008. С. 173-182.
7. Воеводин Вл.В., Гергель В.П., Соколинский Л.Б., Демкин В.П., Попова Н.Н., Бухановский А.В. Развитие системы суперкомпьютерного образования в России: текущие результаты и перспективы // Вестник Нижегородского университета. 2012. № 4. С.203-209.
8. Антонов А.С., Артемьева И.Л., Бухановский А.В., Воеводин Вл.В., Гергель В.П., Демкин В.П., Коньков К.А., Крукиер Л.А., Попова Н.Н., Соколинский Л.Б., Сухинов А.И. Проект «Суперкомпьютерное образование»: 2012 год // Научный сервис в сети Интернет: поиск новых решений: Труды Всероссийской научной конференции (17-22 сентября 2012 г., г. Новороссийск). М.: Изд-во МГУ, 2012. С. 4-8.

Клеточно-автоматная модель динамики популяций трех видов организмов озера Байкал.*

И.В. Афанасьев

Федеральное государственное бюджетное учреждение науки Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (ИВМиМГ СО РАН)

В работе приведена композиционная клеточно-автоматная модель динамики численности трёх видов организмов: макрогектопуса, малой и большой голомянки. Каждый из видов разделён на возрастные группы. Между группами определены демографические отношения и отношения хищник-жертва. Модель позволяет учитывать перемещение особей по области моделирования, сезонность и влияние загрязнений. Проведён ряд вычислительных экспериментов для различных значений мощностей загрязнений, показывающих, что модель в результате колебаний численности приходит к устойчивому колебательному процессу с периодом в 1 год, а амплитуда колебаний и среднегодовые значения зависят от мощности загрязнения области.

1. Введение

Самоорганизация – процесс пространственно-временного упорядочения в системе за счёт согласованного взаимодействия элементов её составляющих [1]. Примерами самоорганизующихся процессов служат разделение фаз, лазер, гомеостаз и др.

Процессы самоорганизации проще исследовать с применением компьютерного моделирования. Исследования клеточно-автоматных (КА) моделей для процессов самоорганизации были проведены в работах Wolfram [2], Chua [3], Ванага [4]. Предложены КА-модели для некоторых реакций Белоусова-Жаботинского [5]. Исследованы КА-модели для процесса «разделения фаз» [6, 7].

КА – набор конечных автоматов, называемых *клетками*. Для каждой клетки определено *множество соседних клеток*. Для КА-моделей определяется *оператор перехода* – правило изменения состояний клеток, зависящее от состояний соседних клеток. Для моделирования сложных явлений используются композиционные КА, предложенные в работе [6]. Основная идея параллельной композиции нескольких КА – одновременное функционирование всех КА, причём их операторы перехода зависят не только от состояний соседних клеток, но и от состояний клеток других КА. В работе [8] предложена и исследована композиционная КА-модель абстрактной задачи хищник-жертва для двух групп организмов.

Длительное время динамика популяций исследовалась с помощью систем дифференциальных уравнений [9, 10]. При этом принимаются следующие ограничения:

1. Параметры популяций усреднены по области моделирования (mean-field гипотеза)
2. Число взаимодействующих организмов не превосходит 3.

В работе [11] предложена и исследована модель восьми групп организмов, основанная на системе дифференциальных уравнений. Таким образом, с помощью численного моделирования было снято второе ограничение.

В данной статье предложена композиционная КА-модель динамики численности 8 групп организмов озера Байкал, снимающая и первое ограничение. Модель учитывает пространственное распределение организмов по области моделирования, сезонность и неравномерное воздействие внешних факторов среды. Данные о популяции организмов взяты из статьи [11].

Первая часть работы посвящена постановке задачи. Во второй части дана математическая интерпретация КА-модели. В заключительной части приводятся способ распараллеливания и результаты вычислительных экспериментов для нескольких вариантов загрязнения озера.

* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 11-01-00567а, а также по Программе Президиума РАН (проект № 15-9, 2012)

2. Постановка задачи

Среди рыб озера Байкал по биомассе лидирует голомянка. Её основной корм – макрогетопус. Аналогично [11] в статье рассмотрены 3 вида организмов: макрогектопус (*macrohectopus*), малая (*comephorus dybovski*) и большая (*comephorus bakalensis*) голомянки. Каждый из видов разделён на возрастные группы (в скобках указаны используемые далее сокращения: буква – вид $\{m,d,b\}$, цифра – возрастная группа $\{1,2,3\}$):

- Макрогектопус – неполовозрелые (m_1), половозрелые (m_2),
- Малая голомянка – однолетки (d_1), неполовозрелые (d_2), половозрелые (d_3),
- Большая голомянка – однолетки (b_1), неполовозрелые (b_2), половозрелые (b_3).

Между группами определены взаимоотношения хищник-жертва и демографические взаимоотношения. Хищниками являются особи неполовозрелых и половозрелых голомянок возрастных групп. Жертвами являются макрогектопус и однолетки голомянок. Все хищники питаются всеми жертвами с разными коэффициентами предпочтений. Демографические отношения (кто в кого вырастает, и кто кого порождает) отображены на рис 1.

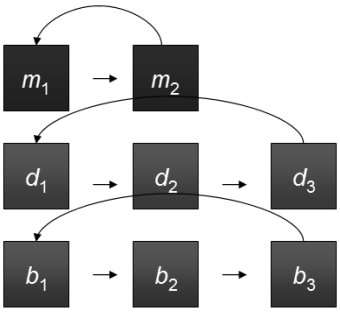


Рис. 1. Демографические межгрупповые отношения

3. Композиционная КА-модель

Следующие предположения использованы при построении КА-модели:

1. Влияние других видов организмов предполагается постоянным. Например, макрогектопусу всегда хватает еды, а нерпа поедает строго одинаковый процент голомянок в год.
2. Все параметры особей (скорость, размер, диета) усреднены по группам.
3. Размножаются только особи последней возрастной группы

КА-модель динамики популяций организмов озера Байкал:

$$\mathrm{N} = \langle \Sigma, M, F, \rho \rangle, \text{ где}$$

Σ – алфавит состояний,
M – множество имён клеток,
F – глобальный оператор перехода,
ρ - режим функционирования

Модель – параллельная композиция восьми КА, каждый из которых предназначен для моделирования численности конкретной группы организмов. Множество имён клеток M разбито на 8 подмножеств M_α^i для каждой группы организмов. Каждое из подмножеств может быть представлено квадратной сеткой с набором ячеек Q.

$$M = M_m^1 \cup M_m^2 \cup M_d^1 \cup M_d^2 \cup M_d^3 \cup M_b^1 \cup M_b^2 \cup M_b^3$$

Предполагается, что существует биекция $\mathrm{Q} \rightarrow M_\alpha^i$.

Клеткой называется элемент множества $M \times \Sigma$. Подмножество Ω множества $M \times \Sigma$ называется *клеточным массивом* тогда и только тогда, когда $|\mathrm{M}| = |\Omega|$ и в Ω нет ячеек с одинаковыми

именами. *Состояния клетки* – элементы множества Σ – целые числа – обозначают модельную плотность числа организмов в клетке. *Близнецами* назовём клетки, имена которых соответствуют одной и той же ячейке сетки Q. Ближайшими соседями назовём *клетки*, имена которых соответствуют соседним ячейкам квадратной сетки Q.

Конечный набор $S(m) = \langle (\varphi_1(m), n_1), \ldots, (\varphi_k(m), n_k) \rangle$ называется *локальной конфигурацией*, где n_i из Σ, $\varphi_i(m)$: $M \rightarrow M$. Клетки с именами $\varphi_1(m) \ldots \varphi_k(m)$ называются *соседями*.

В общем случае, *локальный оператор перехода f*:

$$f : \{S(m)\} \rightarrow \{S(m)\}$$

Результат применения f к клетке (m, n) – замена локальной конфигурации $S(m)$ на локальную конфигурацию $f(S(m))$. *Итерация* или *применение глобального оператора F* – применение локального оператора f ко всем клеткам. *Эволюция* – последовательный процесс применения глобального оператора.

В данной работе использованы два глобальных оператора:

- F_1 – для моделирования перемещения организмов,
- F_2 – для моделирования процессов поедания, вымирания и роста.

Известны два основных режима функционирования КА: *синхронный* и *асинхронный*. Синхронный режим предполагает, что сначала вычисляются новые состояния клеток согласно локальному оператору перехода, а затем все клетки одновременно изменяют свои старые состояния на новые. В асинхронном режиме случайно выбирается клетка, вычисляется её новое состояние и сразу заменяется старое состояние клетки на новое [12].

Предложенная КА-модель использует чуть более сложный режим, комбинирующий синхронный и асинхронный подходы. Сначала асинхронно применяется F_1, затем синхронно применяется F_2.

3.1 Оператор перемещения

Пусть f_d и F_d – локальный и глобальный операторы целочисленной диффузии, предложенной в статье [13].

$$f_d : \{S_1(m)\} \rightarrow \{S_1(m)\}, \text{ где}$$

$S_1(m)$ - набор клеток - ближайших соседей клетки с именем m, включая саму клетку с именем m.

Применение f_d к клетке с именем m выполняется в соответствии с алгоритмом:

1. Пусть $m_1 \ldots m_k$ – имена клеток – ближайших соседей клетки m. $k \leq 4$.
2. Случайно равновероятно выбирается m_i.
3. Пусть n and n_i – состояния клеток с именами m и m_i соответственно.
4. Новые состояния n' и n_i' вычисляются по формуле:

$$n' = n - [\delta \cdot n] + [\delta \cdot n_i]$$
$$n'_i = n_i + [\delta \cdot n] - [\delta \cdot n_i]$$

δ – коэффициент целочисленной диффузии, $0 \leq \delta \leq 1$.

Глобальный оператор перехода F_d применяется в асинхронном режиме.

Определим глобальный оператор F_1. Пусть a – физический размер квадратной ячейки озера. Пусть v_{cr} – крейсерская скорость организмов вида α возраста i. Пусть Δt – физическое время, соответствующее одной глобальной итерации КА. Максимальное число ячеек, пройденное организмом за время Δt, может быть вычислено как:

$$K = \frac{v_{cr} \cdot \Delta t}{a} \qquad (1)$$

Действие глобального оператора перемещения F_1 на множестве M_α^i эквивалентно K-кратному применению глобального оператора диффузии F_d.

$$F_1 = (F_d)^K$$

3.2 Оператор изменения численности

Пусть f_2 – локальный оператор изменения численности:

$$f_2 : \{S_2(m)\} \to \{S_2(m)\}, \text{ где}$$

$S_2(m)$ – набор клеток-близнецов (рис 2).

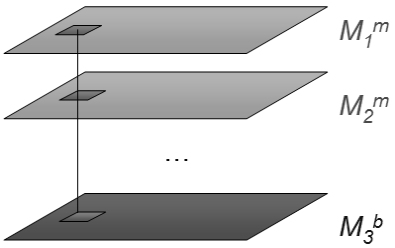


Рис. 2. Локальная конфигурация $S_2(m)$ – набор клеток-близнецов – клеток, имена которых соответствуют одной ячейке квадратной сетки.

Режим глобального оператора изменения численности F_2 синхронный.

Изменение состояния клетки после применения оператора f_2 вычисляется по формуле:

$$n = \left(\rho_\alpha^i n_j - \lambda_\alpha^i n_i - \theta_\alpha^i n_i\right)\Delta t, \text{ где}$$

j – возрастная группа особей, порождающих особей возраста i,

Δt – физическое время, соответствующее одной итерации КА,

$\rho_\alpha^i n_j$ – приток в группу за счёт рождаемости или старения предыдущей группы,

$\lambda_\alpha^i n_i$ – отток из группы за счёт смертности,

$\theta_\alpha^i n_i$ – отток из группы за счёт старения.

В модель введена зависимость рождаемости от сезонов. Пик рождаемости у малых голомянок приходится на весну, у больших голомянок – на осень [14]. В КА-модели коэффициенты рождаемости ρ_d^l ρ_b^l умножаются на функции $season_d(t)$ и $season_b(t)$, графики которых приведены на рис 3.

$$season_b(t) : R \to R \qquad season_d(t) : R \to R$$

$$season_b(t+1) = season_b(t) \qquad season_d(t+1) = season_d(t)$$

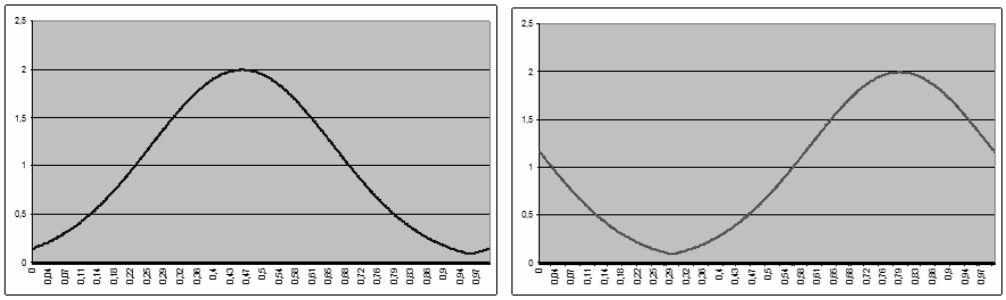


Рис. 3. Графики функций $season_d(t)$ (слева, м.голомянка) и $season_b(t)$ (справа, б.голомянка). Ось абсцисс – время года (0 соответствует 1 января, 1 соответствует 31 декабря).

Также в КА-модели учитывается влияние вероятного загрязнения на смертность рыб. Пусть $poll(m)$ – карта загрязнений – функция, ставящая в соответствие имени клетки положительное действительное число, характеризующее степень загрязнения. Коэффициенты смертности в КА-модели вычисляются по формуле:

$$\lambda_\alpha^i{}' = \left(1 + poll(m)\right)\lambda_\alpha^i, \qquad (2)$$

где λ_α^i – коэффициент смертности рыб из [11]

4. Реализация и распараллеливание

В вычислительных экспериментах размер области моделирования $|M_\alpha^i| = 4\,744\,494$ ячеек. Площадь Байкала – 31 722 км^2. Δt – два дня. Соответственно, получим значения чисел K из формулы (1):

Таблица 1. Значения чисел K для различных групп.

возраст/вид	макрогектопус, m	малая голомянка, d	большая голомянка, b
1	2	2	2
2	4	3	3
3		4	6

Для проведения вычислительных экспериментов разработана программа на языке C++ с использованием библиотек Qt и OpenGL. Параллелизация вычислений проводилась с помощью технологии OpenMP. Расчёты проведены на Intel Core i7-2600 4 cores × 3.4 GHz, 8Gb RAM. Наиболее дорогая по времени операция – моделирование перемещения. В таблице 2 приведена стоимость вычисления одной итерации алгоритма. Для проведения одного вычислительного эксперимента необходимо около 5000 итераций.

Таблица 2. Время в секундах на одну итерацию в однопоточной программе

	F_1	F_2	всего на итерацию
время	16 сек	1 сек	17 сек

Временная сложность выполнения оператора F_2 – $O(|M|)$, оператора F_1 – $O(|M|^2)$ (с увеличением числа $|M|$ линейно уменьшается размер клетки a, следовательно, линейно растёт число K). Поэтому смысл есть только в распараллеливании процедуры вычисления F_1.

Был выбран следующий способ распараллеливания:

1. 1-ый поток вычисляет действие F_1 для M_m^1 и M_d^1 и M_b^1.
2. 2-ой поток вычисляет действие F_1 для M_m^2 и M_d^2.
3. 3-ий поток вычисляет действие F_1 для M_b^2 и M_d^3.
4. 4-ый поток вычисляет действие F_1 для M_b^3

Предложенный алгоритм прост в реализации, но не масштабируем на число потоков большее четырёх. Получены следующие результаты:

Таблица 3. Результат распараллеливания процедуры вычисления F_1

F_1 1 thread	F_1 4 threads	эффективность распараллеливания
16 сек	4,5 сек	89%

5. Вычислительные эксперименты

Начальное состояние – равномерное распределение особей по области моделирования, совпадающее по значению с устойчивым состоянием численности, взятому из [11]. Была выбрана карта загрязнений, показанная на рис 4. Функция $poll(m)$ из формулы (2) представляет собой плотность стандартного нормального распределения с центром в точке m_0 в южной части озера Байкал, умноженную на константу.

Рис. 4. Карта загрязнений. Синий цвет – нет загрязнения. Красный цвет – есть загрязнение.

В вычислительных экспериментах варьировалось значение $poll(m_0)$. Для каждого эксперимента отслеживалась численность особей в клетке вблизи эпицентра загрязнения. Получившиеся графики численности организмов приведены на рис 5,6.

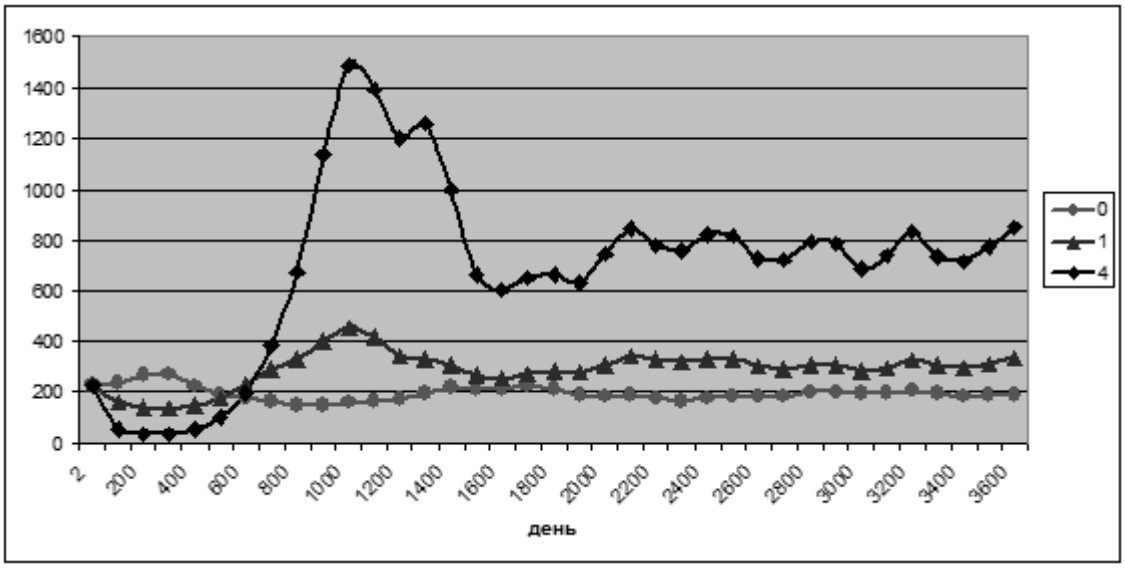


Рис. 5. Динамика численности макрогектопуса в точке вблизи эпицентра загрязнения при различных $poll(m_0)$ (ось абсцисс – время в днях, ось ординат – модельная плотность)

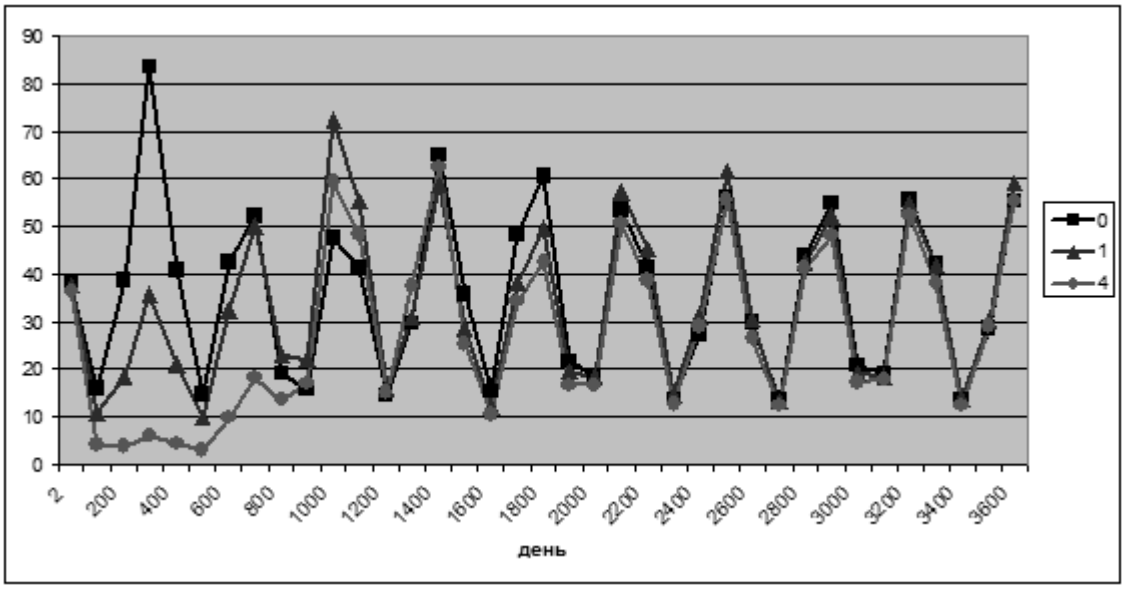
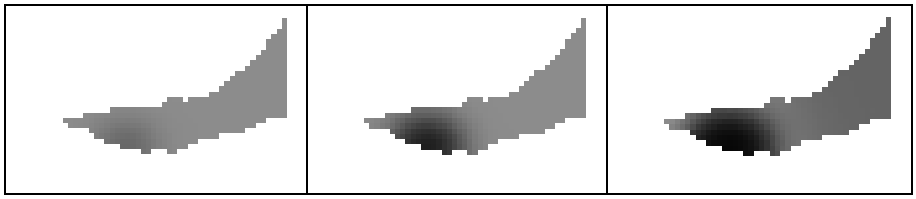


Рис. 6. Динамика численности малой голомянки в точке вблизи эпицентра загрязнения при различных $poll(m_0)$ (ось абсцисс – время в днях, ось ординат – модельная плотность)

Динамика численности большой голомянки сходна с динамикой численности малой голомянки с разницей во времени пика рождаемости. Во всех экспериментах система приходит к устойчивым годовым колебаниям.

Некоторые глобальные состояния половозрелой голомянки в южной части озера показаны на рис 7. Как видно из рисунка, влияние загрязнения не распространяется за его территорию.



t=20	t=100	t=5000

Рис. 7. Глобальные состояния в южной части озера половозрелой малой голомянки для случая $poll(m_0) = 4$. Более тёмный цвет означает меньшую численность рыб.

6. Заключение

Предложена КА-модель динамики популяции организмов озера Байкал, позволяющая учитывать особенности пространственного распределения организмов, сезонность и неравномерное влияние окружающей среды.

В будущем предполагается учесть течения в Байкале в операторе моделирования перемещения организмов и неравномерность распределения макрогектопуса по котловинам Байкала.

Литература

1. Хакен Г. Синергетика. М.: Мир, 1980. 406 с.
2. Wolfram S. A new kind of science. USA: Wolfram Media Inc., 2002. 1197p
3. Chua L. O. CNN: A Paradigm for Complexity. Berkely: World Scientific Series on Nonlinear Science. University of California, 1998. 320~p
4. Ванаг В. К. Диссипативные структуры в реакционно-диффузионных системах. Ижевск: ИКИ , 2008. 300 с.
5. Madore B., Freedman W. Computer simulation of the Belousov-Zhabotinski reaction // Science. 1983. Т. 222. С. 615-618
6. Bandman O. L. Cellular Automata composition techniques for spatial dynamics simulating. Simulating Complex Systems by Cellular Automata (A.G.Hoekstra et al. eds) Berlin: Springer. 2010, pp 81-115.
7. Афанасьев И. В. Исследование эволюции клеточных автоматов, моделирующих процесс «разделения фаз» на треугольной сетке // Прикладная дискретная математика. 2010. №4. С. 79–90.
8. Bandman O. L. A method for construction of cellular automata simulating pattern formation processes // Theoretical background of applied discrete mathematics. 2010 №4. pp.91–99.
9. Свирежев Ю. М., Логофет Д. О. Устойчивость биологических сообществ. Москва: Наука, 1978. 352 с.
10. Базыкин А. Д. нелинейная динамика взаимодействующих популяций. Ижевск: ИКИ. 2003. 368 с.
11. Зоркальцев В. И., Казазаева А. В., Мокрый И. В. Модель взаимодействия трёх пелагических видов организмов озера Байкал // Современные технологии. Системный анализ. Моделирование. Иркутский государственный университет путей сообщения. 2008. №1. С. 182–193.
12. Бандман О. Л. Клеточно-автоматные модели пространственной динамики // Системная информатика. 2006. №10. С. 58–113.
13. Medvedev Y. G. Multi-particle Cellular Automata Models For Diffusion Simulation. Methods and tools of parallel programming multicomputers. 2011. V. 6083/2011. p. 204-211.
14. Дзюба Е. В., Тереза Е. П., Помазкова Г. И. и др. Связь сезонной динамики зоопланктона, питания рыб и их зараженности паразитами в пелагиали озера Байкал // Теория, методы и инструменты принятия решений в живых, социальных и технических системах: Материалы к 19-му заседанию междунар. постоянно действующего семинара «Гомеостатика живых, природных, технических и социальных систем». Иркутск, 2001 С. 90–95.

Разработка кинетических моделей реакций синтеза ароматических и гетероциклических соединений на основе многоядерных вычислительных систем*

И.В. Ахметов

Институт нефтехимии и катализа РАН

Обратные задачи химической кинетики относятся к таким физико-химическим задачам, которые предполагают значительный объем вычислений. В настоящее время активно внедряются вычислительные системы с многоядерными процессорами, преимуществами которых являются доступность и легкость использования, что расширяет возможности их применения в научных исследованиях. В данной работе предложен параллельный метод решения обратных кинетических задач на многоядерных систем.

1. Введение

Каталитические реакции синтеза ароматических и гетероциклических соединений, таких как N-бензилиденбензиламин и метиловый эфир 5-ацетил-2-пирролкарбоновой кислоты обладают широким спектром применения. N-бензилиденбензиламин известен как индикатор количественного определения литийорганических соединений титриметрическим методом и является исходным соединением для синтеза ряда гетероциклов. Метиловый эфир 5-ацетил-2-пирролкарбоновой кислоты представляет большой интерес для получения порфиринов и лекарственных препаратов [1].

Для изучения механизмов вышеуказанных реакций необходимо построить кинетические модели, решение обратных кинетических задач для которых усложнено тем обстоятельством, что данный этап разработки кинетической модели является наиболее трудоемким и требует больших временных затрат.

Применение параллельных вычислений становится все более востребованным методом математической обработки экспериментальных данных в связи с увеличивающейся сложностью получения детальной информации о химических реакциях.

Обратные задачи химической кинетики относятся к таким физико-химическим задачам, которые предполагают значительный объем вычислений. Использование высокопроизводительных вычислительных систем принципиально изменило возможности анализа сложных химических процессов: стал доступным детальный анализ достаточно сложных кинетических моделей с большим количеством экспериментальной информации; во много раз сократилось время построения кинетических моделей; повысилась точность решений.

В настоящее время предложены решения обратных кинетических задач с применением параллельных вычислений на кластерных системах и графических процессорах. Активно внедряются вычислительные машины с многоядерными процессорами, преимуществами которых являются доступность и легкость использования, что расширяет возможности их применения в научных исследованиях.

В данной работе предложен метод поиска кинетических параметров, использующий технологию параллельных вычислений на многоядерных системах, для построения кинетических моделей химических реакций металлокомплексного катализа с целью сокращения сроков изучения и освоения новых каталитических процессов.

* Работа выполнена при финансовой поддержке Министерства образования и науки РФ (госконтракт № 02.740.11.0631) и поддержана грантом РФФИ 12-07-00324

2. Кинетические модели реакций синтеза ароматических и гетероциклических соединений

Для понимания физико-химической сущности каталитической реакции, последующего математического моделирования каталитического процесса и определения условий его промышленной реализации необходима, прежде всего, разработка его кинетической и математической моделей [2].

Основополагающей, базисной основой моделирования каталитических процессов являются, прежде всего, детальные исследования физико-химической сущности химических реакций, поскольку полученные в натурных и уточненные в вычислительных экспериментах количественные характеристики позволят построить кинетические модели, которые станут надежной основой последующих разработок.

Кинетическая модель процесса представляет собой совокупность элементарных стадий, реакций и уравнений, характеризующих зависимость скорости химического превращения от параметров реакции: давления, температуры, концентраций реагентов и др. Подобные зависимости определяются на основе экспериментальных данных, полученных в области изменения параметров реакции, охватывающей практические ведения процесса в промышленных условиях. Построенная таким образом кинетическая условия модель является первым уровнем модели каталитического реактора и базисом для решения статических и динамических задач, возникающих при разработках технологических процессов.

Разработка кинетических моделей, приведенная в данной работе, основана на экспериментальных данных по синтезу ароматических соединений (на примере бензилиденбензиламина) и гетероциклических соединений (на примере метилового эфира 5-ацетил-2-пирролкарбоновой кислоты), полученных в лаборатории химии углеводородов ИНК РАН. В ходе экспериментов разработан новый метод введения в соединения пиррольного ряда карбоксильной группы по оригинальной методике, основанной на взаимодействии пирролов с системой CCl_4-CH_3OH-катализатор.

На основе анализа экспериментальных данных и результатов их математической обработки [3] предложены следующие схемы химических превращений в реакциях синтеза ароматических и гетероциклических соединений под действием металлокомплексных катализаторов и соответствующие им кинетические уравнения (1)-(2):

1. $C_1 + C_2 \rightarrow C_3 + C_4$	$W_1=k_1C_1C_2$
2. $C_3 \rightarrow C_5 + C_6$	$W_2=k_2C_3$
3. $C_5 + C_1 \rightarrow C_7 + C_8$	$W_3=k_3C_5C_1$ (1)
4. $C_8 + C_6 \rightarrow C_9$	$W_4=k_4C_8C_6$

C_i – концентрации компонентов, моль/л: $C_1=C_7H_9N$ – бензиламин, $C_2=CCl_4$ – четыреххлористый углерод, $C_3=C_7H_8NCl$ – хлорбензиламин, $C_4=CHCl_3$ – хлороформ, $C_5=C_7H_7N$ – 1-фенилметанимин, $C_6=HCl$ – хлористый водород, $C_7=C_{14}H_{13}N$ – бензилиденбензиламин, $C_8=NH_3$ – аммиак, $C_9=NH_4Cl$ – хлористый аммоний; k_j – кинетическая константа скорости j-ой реакции, $л \cdot моль^{-1} \cdot ч^{-1}$ (j=1, 3, 4), $ч^{-1}$ (j=2).

1. $C_1 + C_2 \rightarrow C_3 + C_4$	$W_1=k_1C_1C_2$
2. $C_3 + C_5 \rightarrow C_6 + C_4$	$W_2=k_2C_3C_5$
3. $C_6 + C_5 \rightarrow C_7 + C_4$	$W_3=k_3C_5C_6$ (2)
4. $C_7 + C_5 \rightarrow C_8 + C_4$	$W_4=k_4C_5C_7$
5. $C_8 \rightarrow C_9 + C_{10}$	$W_5=k_5C_8$

где C_i – концентрации компонентов, моль/л: $C_1=C_6H_7NO$ – 2-ацетилпиррол, $C_2=CCl_4$ – четыреххлористый углерод, $C_3=C_7H_6NOCl_3$ – 2-ацетил-5-трихлорметилпиррол, $C_4=HCl$ – соляная кислота, $C_5=CH_4O$ – метанол, $C_6=C_8H_9Cl_2NO_2$ – 2-ацетил-5-дихлор(метокси)метанпиррол, $C_7=C_9H_{12}ClNO_3$ – 2-ацетил-5-диметокси(хлор)метанпиррол, $C_8=C_{10}H_{15}NO_4$ – 2-ацетил-5-

триметоксиметанпиррол, $C_9=C_8H_9NO_3$ – метиловый эфир 5-ацетил-2-пирролкарбоновой кислоты, $C_{10}=CH_3OCH_3$ – формальдегид; k_j – кинетическая константа скорости j-ой реакции, $л \cdot моль^{-1} \cdot ч^{-1}$ (j=1, 2, 3, 4), $ч^{-1}$ (j=5).

В (1) и (2) W_j – скорость j-ой реакции, моль/(л·ч).

Кинетические уравнения схемы превращений (1)–(2) проанализированы в рамках закона действующих масс. Корректным описанием лабораторного реактора с мешалкой является модель идеального смешения [4].

Суммарный баланс изотермического реактора идеального смешения для i-того компонента в элементе объема реактора (ΔV) для варианта, когда мольная плотность газа или жидкости (или суммарная концентрация C, кмоль/м^3) изменяется во времени, т.е. когда реакция протекает с изменением числа молей, определяется за счет изменения:

– было в момент времени (t): $C_i(t)\cdot\Delta V=C(t)\cdot X_i(t)\cdot\Delta V$

– стало в момент времени (t+Δt): $C_i(t+\Delta t)\cdot\Delta V=C(t+\Delta t)\cdot X_i(t+\Delta t)\cdot\Delta V$

– изменилось за время (Δt) за счет химических реакций: $\left(\sum_{j=1}^{J}\nu_{ij}W_j\right)\cdot\Delta V\cdot\Delta t$,

где $X_i=C_i/C$ – концентрация i-того компонента в мольных долях.

Тогда, материальный баланс изотермического реактора идеального смешения описывается уравнением:

$$C(t)\cdot X_i(t)\cdot\Delta V - C(t+\Delta t)\cdot X_i(t+\Delta t)\cdot\Delta V + \left(\sum_{j=1}^{J}\nu_{ij}W_j\right)\cdot\Delta V\cdot\Delta t = 0$$

или

$$C(t)\cdot X_i(t) - C(t+\Delta t)\cdot X_i(t+\Delta t) + \left(\sum_{j=1}^{J}\nu_{ij}W_j\right)\Delta t = 0.$$

В пределе, при Δt, стремящемся к нулю, получим систему дифференциальных уравнений:

$$\frac{d(Cx_i)}{dt}=\sum_{j=1}^{J}\nu_{ij}W_j \qquad (3)$$

с начальными условиями – t = 0: $X_i = X_i^o$, $C=C_o$.

Система уравнений (3) замыкается условием нормировки по компонентам реакционной среды: $\sum_{i=1}^{I}X_i = 1$.

При кинетических исследованиях экспериментальные данные получают, как правило, в изотермических условиях при постоянном давлении. Для газофазных реакций мольная плотность газа (C_o) может быть рассчитана из уравнения состояния, например, уравнения Менделеева-Клапейрона для идеальных газов: $PV=\frac{m}{M}RT=nRT$ или $P=C_oRT$, $C_o=P/(RT)$, где $C_o=\frac{m}{MV}=\frac{n}{V}$.

Начальная мольная плотность жидкости (C_o) постоянна при любых температурах.

Тогда, разделив (3) на C_o, получим систему дифференциальных уравнений:

$$\frac{d(\overline{N}x_i)}{dt}=\sum_{j=1}^{J}\nu_{ij}\omega_j \qquad (4)$$

где $\overline{N}=C/C_o$ – относительное изменение числа молей реакционной среды, $\omega_j=W_j/C_o$ – приведенные скорости химических реакций.

Суммируя уравнения (4) с учетом условия нормировки, получим:

$$\frac{d\overline{N}}{dt} = F_N = \sum_{j=1}^{J} \omega_j \sum_{i=1}^{I} \nu_{ij} \quad (5)$$

Продифференцировав уравнения (5), получим:

$$x_i \frac{d\overline{N}}{dt} + N\frac{dx_i}{dt} = F_i = \sum_{j=1}^{J} \nu_{ij}\omega_j \quad (6)$$

Умножим (5) на X_i и вычтем из (6). Тогда, с учетом правой части уравнения (5), получим систему дифференциальных уравнений материального баланса периодического реактора идеального смешения (7)-(8):

$$\frac{d\overline{N}}{dt} = F_N, \; F_N = \frac{1}{V_o}\sum_{j=1}^{J} \delta_j \omega_j, \; \delta_j = \sum_{i=1}^{I} \nu_{ij} \quad (7)$$

$$\frac{dX_i}{dt} = \frac{F_i - X_i F_N}{\overline{N}} \quad (8)$$

с начальными условиями: при t=0 – $X_i = X_i^o$, $\overline{N} = 1$,

где $\overline{N} = C/C_o$ – относительное изменение числа молей реакционной смеси; C и C_o – мольная плотность и ее начальное значение, моль/л; $X_i=C_i/C$ – концентрации компонентов, мольные доли; V_o – объем реакционного пространства, л; $\omega_j=W_j/C_o$ – приведенные скорости реакций, $ч^{-1}$; J – число стадий химического превращения; I – количество компонентов.

Правые части систем уравнений (7)-(8) для реакций синтеза ароматических и гетероциклических соединений имеют следующий вид:

– Синтез бензилиденбензиламина реакцией бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$

$F_1 = -\omega_1 - \omega_3$; $F_2 = -\omega_1$; $F_3 = \omega_1 - \omega_2$; $F_4 = \omega_1$; $F_5 = \omega_2 - \omega_3$; $F_6 = \omega_2 - \omega_4$; $F_7 = \omega_1$; $F_8 = \omega_3 - \omega_4$; $F_9 = \omega_4$; $F_n = \omega_2 - \omega_4$.

– Синтез метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

$F_1 = -\omega_1$; $F_2 = -\omega_1$; $F_3 = \omega_1 - \omega_2$; $F_4 = \omega_1 + \omega_2 + \omega_3 + \omega_4$; $F_5 = \omega_2 - \omega_3 - \omega_4$; $F_6 = \omega_2 - \omega_3$; $F_7 = \omega_1 - \omega_4$; $F_8 = \omega_4 - \omega_5$; $F_9 = \omega_5$; $F_{10} = \omega_5$; $F_n = \omega_5$.

3. Метод решения обратных кинетических задач с использованием многоядерных систем

Расчет кинетических параметров химической реакции в присутствии металлокомплексного катализатора – длительный процесс. Для ускорения решения обратной задачи химической кинетики была разработана программа на языке C++ с технологией многопоточного программирования OpenMP [5].

Созданный программный комплекс реализует расчеты с помощью многопоточности, в которой мастер-поток создает набор управляемых потоков и распределяет задачу между ними. Вычисления выполняются параллельно в системе с несколькими ядрами.

Описание задач, выполняемых параллельно несколькими потоками, и данных, используемых в этих задачах, осуществляется с помощью специальных директив языка программирования [6].

Количество создаваемых потоков необязательно совпадает с количеством ядер. Это регулируется при помощи вызова специальной библиотеки, а также переменных окружения.

При решении обратной кинетической задачи генерируется набор из n исходных констант скоростей стадий, основанный на методах регрессионного анализа. Существует 2^n вариантов изменения набора констант в сторону увеличения или уменьшения на шаги Δ_i, равные заданному проценту от предыдущих значений для всех кинетических констант, т.е.: $K^m=K^{m-1}(1+\Delta_i/100)$,

где K^m и K^{m-1} – значения новых констант и констант, полученных на предыдущем шаге расчетов.

В табл.1 и табл.2 приведены все возможные варианты изменения набора констант для синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$ при n=4 и синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты при n=5.

Набор с наиболее удовлетворяющими решениями выбирается как исходный набор констант скоростей стадий. Для нового набора констант выполняют такие же операции, что и для предыдущего.

Процесс поиска оптимальных кинетических параметров останавливают при достижении заданной точности.

В качестве критерия отклонения расчетных и экспериментальных данных выбран следующий функционал:

$$EE = \sum_{i=1}^{N} \sum_{j=1}^{M} \left| X_{ij}^{p} - X_{ij}^{э} \right| \rightarrow \min, \qquad (9)$$

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; M – количество веществ реакции.

Таблица 1. Вариации четырех констант для реакции синтеза бензилиденбензиламина

№	K_1	K_2	K_3	K_4	№	K_1	K_2	K_3	K_4
1	+	+	+	+	9	+	–	+	+
2	+	+	+	–	10	+	–	–	+
3	+	+	–	–	11	+	–	+	–
4	+	–	–	–	12	+	+	–	+
5	–	+	+	+	13	–	+	–	+
6	–	–	+	+	14	–	+	+	–
7	–	–	–	+	15	–	+	–	–
8	–	–	–	–	16	–	–	+	–

Таблица 2. Вариации пяти констант для реакции синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

№	K_1	K_2	K_3	K_4	K_5	№	K_1	K_2	K_3	K_4	K_5
1	+	+	+	+	+	17	–	+	+	+	+
2	+	+	+	+	–	18	–	+	+	+	–
3	+	+	+	–	+	19	–	+	+	–	+
4	+	+	+	–	–	20	–	+	+	–	–
5	+	+	–	+	+	21	–	+	–	+	+
6	+	+	–	+	–	22	–	+	–	+	–
7	+	+	–	–	+	23	–	+	–	–	+
8	+	+	–	–	–	24	–	+	–	–	–
9	+	–	+	+	+	25	–	–	+	+	+
10	+	–	+	+	–	26	–	–	+	+	–
11	+	–	+	–	+	27	–	–	+	–	+
12	+	–	+	–	–	28	–	–	+	–	–
13	+	–	–	+	+	29	–	–	–	+	+
14	+	–	–	+	–	30	–	–	–	+	–
15	+	–	–	–	+	31	–	–	–	–	+
16	+	–	–	–	–	32	–	–	–	–	–

Для всех комбинаций набора констант (см. табл.1 и 2) решается прямая кинетическая задача: уравнения (7)-(8). Схема метода решения прямых задач с учетом распараллеливания на все доступные ядра приведена на рис. 1.

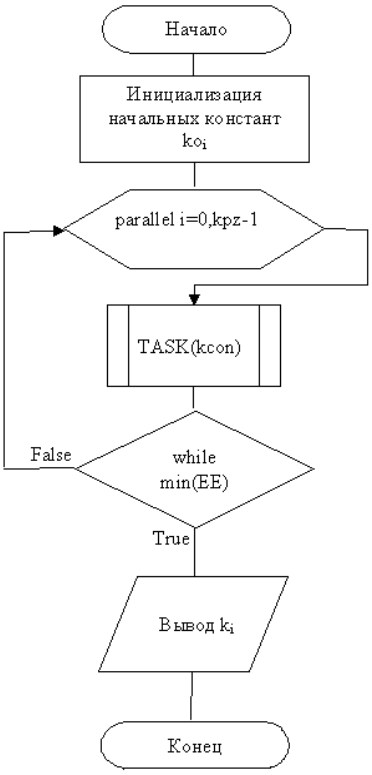


Рис. 1. Алгоритм метода решения обратной кинетической задачи на основе многоядерных систем

При распараллеливании алгоритма решения обратной кинетической задачи время расчетов на 2 ядрах для достижения заданной точности решения сокращается на 35-40%, а на 4 ядрах – на 50-55%.

Зависимости времени расчетов прямых кинетических задач для двух исследованных реакций представлены на рис.2 и рис.3.

Рис. 2. Зависимость времени расчетов прямых кинетических задач на многоядерных системах для синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$

Рис. 3. Зависимость времени расчетов прямых кинетических задач на многоядерных системах для синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

Для решения обратных кинетических задач разработан программный комплекс KinMP, тестирование которого проведено на 4-ядерном компьютере ИНК РАН с процессором AMD Phenom II X4 940 Black 3.0 ГГц.

При увеличении в ходе расчетов количества ядер процессора от 1 до 4 произведено измерение времени выполнения программы при решении обратных кинетических задач для реакций синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$ (рис.4) и синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты (рис.5).

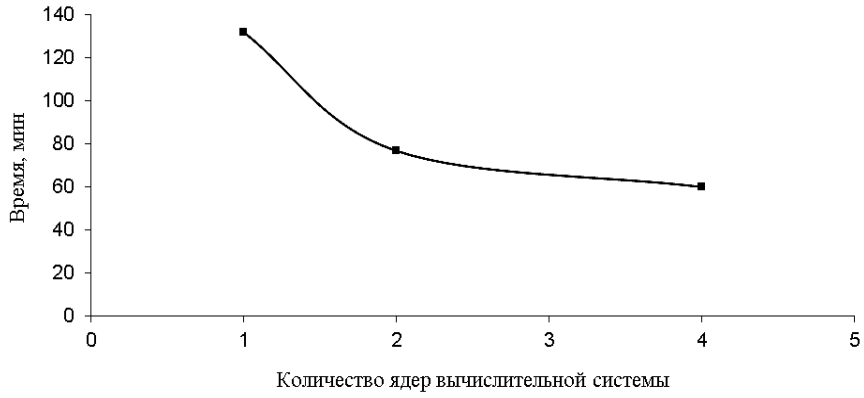


Рис. 4. Зависимость времени выполнения программы от количества ядер для реакции синтеза бензилиденбензиламина

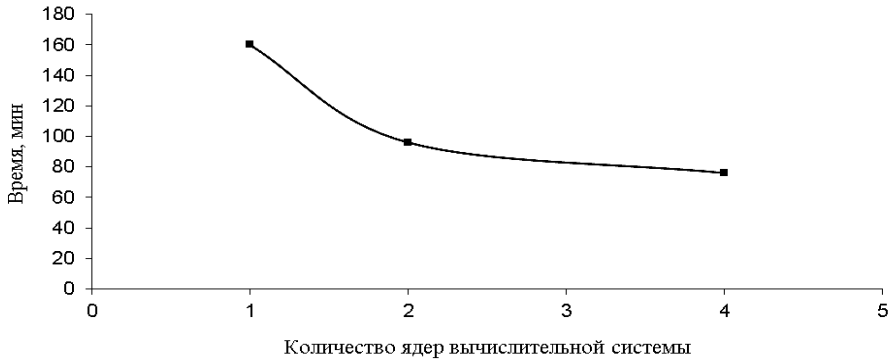


Рис. 5. Зависимость времени выполнения программы от количества ядер для реакции синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

Рассчитано получаемое ускорение и эффективность выполнения программы для реакций синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$ (см. рис.6, рис.8) и метилового эфира 5-ацетил-2-пирролкарбоновой кислоты (см. рис.7, рис.9).

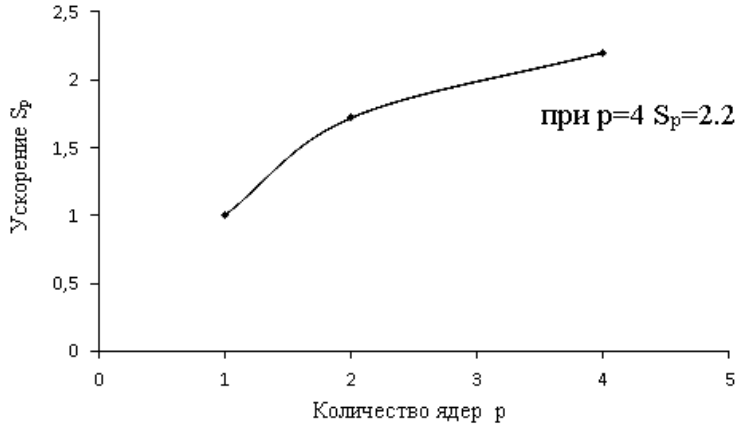


Рис. 6. Зависимость ускорения выполнения программы от количества ядер для синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$

Рис. 7. Зависимость ускорения выполнения программы от количества ядер для синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

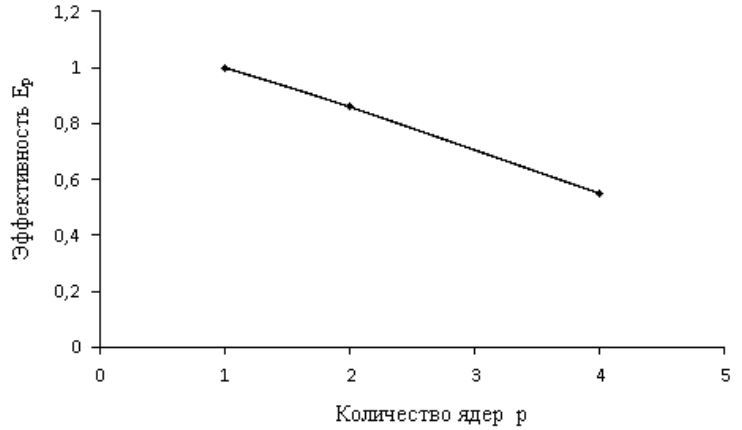


Рис. 8. Зависимость эффективности выполнения программы от количества ядер для синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$

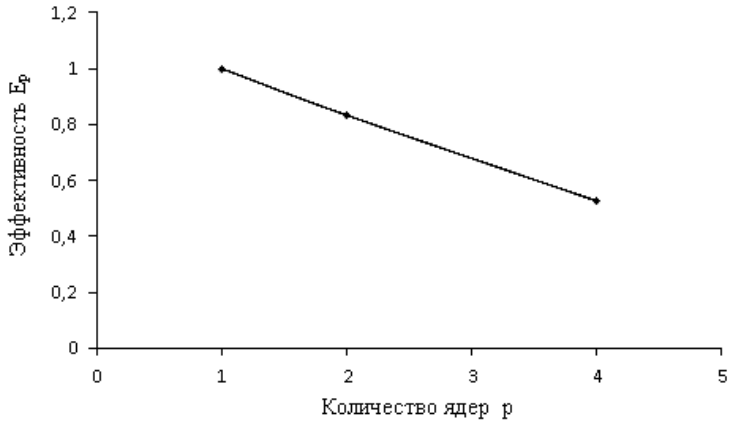


Рис. 9. Зависимость эффективности выполнения программы от количества ядер для синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

При расчете кинетических параметров и использовании четырех ядер для реакции синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$ получено ускорение в 2.2 раза, а для синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты – 2.1.

4. Результаты вычислительных экспериментов

Численные значения найденных констант скоростей стадий и энергий активации для синтеза бензилиденбензиламина и синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты (МЭАПКК) приведены соответственно в табл.3 и табл.4. В реакции получения метилового эфира 5-ацетил-2-пирролкарбоновой кислоты лимитирующей оказалась совокупность стадий 2, 3 и 4. Наиболее энергоемкой является первая стадия – алкилирование 2-ацетилпиррола.

Таблица 3. Кинетические параметры для синтеза бензилиденбензиламина

Кинетические константы при температуре 23°C, $ч^{-1}$		E_i, ккал/моль
K_1	1.5×10^{-2}	10.6
K_2	4.7	7.7
K_3	13.4	1.6
K_4	0.6	0.4

Таблица 4. Кинетические параметры для синтеза метилового эфира 5-ацетил-2-пирролкарбоновой кислоты

Кинетические константы при температуре 105°C, $ч^{-1}$		E_i, ккал/моль
K_1	4.9×10^{-2}	23.8
K_2	4.3×10^{-3}	3.6
K_3	4.1×10^{-2}	4.4
K_4	7×10^{-4}	9.7
K_5	6.4	3.1

5. Заключение

Таким образом, разработан алгоритм использования многоядерных вычислительных систем для решения обратных задач химической кинетики. Метод реализован в виде программного комплекса, который включает в себя базу данных кинетических исследований, последова-

тельные и параллельные алгоритмы решения систем обыкновенных дифференциальных уравнений, реализованные на одноядерных и многоядерных вычислительных системах.

Разработана информационно-аналитическая система, успешное применение которой при построении кинетических моделей реакций синтеза ароматических и гетероциклических соединений показало универсальность предлагаемого системного подхода решения обратных кинетических задач.

Система позволяет пользователям достаточно легкую адаптацию при разработке кинетических моделей различных реакций за счет формирования новых блоков в базе экспериментального данных, выбора или добавления новых методов обработки данных, построения математических моделей исследуемых объектов разной сложности.

5.1 Основные результаты и выводы

1. Предложен новый метод параллельных вычислений на многоядерных системах при построении кинетических моделей химических реакций металлокомплексного катализа.

При решении обратной кинетической задачи генерируется набор из n исходных констант скоростей стадий, основанный на методах регрессионного анализа.

2. На основе разработанного подхода построены кинетические модели для реакций: синтеза бензилиденбензиламина взаимодействием бензиламина с четыреххлористым углеродом под действием $FeCl_3 \cdot 6H_2O$ и метилового эфира 5-ацетил-2-прирролкарбоновой кислоты. Показано, что использование четырехъядерной системы позволяет в 2,1-2,2 раза ускорить вычислительный процесс.

Литература

1. Хуснутдинов Р.И., Байгузина А.Р., Аминов Р.И. Синтез N-бензилиденбензиламина из бензиламина при действии железосодержащих катализаторов в CCl_4 // Журнал органической химии. 2012. С. 1063-1065.
2. Слинько М.Г. Моделирование химических реакторов. Новосибирск: Наука, 1968. 96 с.
3. Хуснутдинов Р.И., Байгузина А.Р., Мукминов Р.Р., Ахметов И.В., Губайдуллин И.М., Спивак С.И., Джемилев У.М. Новый метод синтеза эфиров 2-пирролкарбоновой и 2,5-пирролдикаробоновой кислот реакцией пирролов с CCl_4 и алифатическими спиртами под действием Fe-содержащих катализаторов // Журнал органической химии. 2010. С.1054-1060.
4. Губайдуллин И.М. Информационно-аналитическая система решения многопараметрических обратных задач химической кинетики // Дисс. … доктора физико-математических наук. Уфа, 2012. 243 с.
5. Гергель В.П. Теория и практика параллельных вычислений. М.: БИНОМ, 2010. 423 с.
6. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб: БХВ-Петербург, 2002. 608 с.

Клеточно-автоматное моделирование процесса просачивания жидкости через пористый материал *

О.Л. Бандман

Институт вычислительной математики и математической геофизики СО РАН

В статье излагаются результаты исследования клеточно-автоматной модели просачивания жидкости (или газа) через трехмерную пористую среду с заданной морфологией. Клеточно-автоматная модель представлена в виде композиции двух клеточных автоматов, моделирующих 1) конвекцию под действием внешнего давления или гравитации, и 2) диффузию, выравнивающую плотность вещества в порах. Тестирование модели проводилось на примере просачивания воды через трехмерный образец почвы [1] размером 2 см^3 при линейном размере клетки 15 мкм и разных типах взаимодействий влаги со стенками пор. Параллельная версия реализована на кластере ССКЦ СО РАН.

1. Введение

В связи с развитием новых технологий производства и использования пористых материалов возникают новые требования к компьютерному моделированию процессов в них. Рассмотрение пористых материалов как сплошных сред, характеризуемых определенным коэффициентом пористости, теперь не удовлетворяет ни разработчиков новых композитных материалов, ни исследователей плодородия почвы. Во многих случаях необходимо учитывать внутренние свойства материала, среди которых главные - морфология среды и характер взаимодействия стенок пор с проходящим через них газом или жидкостью. Поскольку вычислительные мощности современных суперкомпьютеров позволяют представлять морфологию материала на микро-уровне, то становится возможным имитировать прохождение газа или влаги через все извилины и преграды в толще материала [1, 2]. Использовать для этого традиционные математические модели, основанные на дифференциальных уравнениях в частных производных, невозможно из-за трудно описываемых непрерывными функциями границ пор. Поэтому появились попытки применить дискретные модели типа "решеточный газ" (Lattice-Gas) [3] и "решеточную модель Больтцмана" (Lattice-Boltzman) [4], а также клеточные автоматы [5, 6], которые "не боятся сложных граничных условий" и допускают эффективную параллельную реализацию. Модели, основанные на принципах решеточного газа, имитируют стационарные ламинарные потоки. Их можно использовать в случаях, когда жидкость или газ проходит сквозь пластину из пористого материала, как это, например, происходит в полимерных мембранах водородных энергетических элементов [1]. Нестационарные процессы, такие как проникание в материал, наполнение его влагой или, наоборот, высыхание, решеточные модели не берут. Кроме того, трехмерная их реализация имеет очень высокую сложность и трудна для распараллеливания [7]. Здесь необходима более гибкая имитация движения частиц и их взаимодействий со стенками пор.

В статье предлагается простая версия вероятностного клеточного автомата, минимальная конфигурация которой имитирует три вида движений абстрактных частиц: конвекцию под действием внешней силы, диффузию (растекание) и взаимодействие со стенками. Эту модель можно считать аналогичной диффузионно-конвекционной модели пористой среды, основанной на системе дифференциальных уравнений конвекция-диффузия [8], которая, однако, работает только в прямых пористых каналах. При всей своей простоте предлагае-

*Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 11-01-00567а, а также по Программе Президиума РАН (проект № 15-9,2012).

[1]Томографическое представление образца почвы любезно предоставлено проф.В.Корнелисом, prof. Wim Cornelis, SoPHy, Department Soil Management, Ghent University, Coupure links 653, 9000 Gent, Belgium.

мая модель обладает большой гибкостью. Модификации функций переходов КА позволяют моделировать высыхание материала, а также учитывать свойства взаимодействий влаги со стенками пор, например, набухание стенок в гидрофильных пористых материалах. Возможности модели иллюстрируются результатами моделирования процесса увлажнения и высыхания почвы.

Статья состоит из Ведения, четырех разделов и Заключения. Во Введении обоснована актуальность проведенного исследования. Во втором разделе даны необходимые определения и формальное представление предлагаемой модели. В третьем разделе приводится алгоритм моделирования. Четвертый и пятый разделы посвящены результатам моделирования процессов увлажнения и высыхания образца почвы в последовательной и параллельной версиях, соответственно. В Заключении обсуждается возможности совершенствования и применения модели.

2. Формальное представление клеточно-автоматной модели

КА — это множество одинаковых простых вычислителей, которые в модели представлены парами (u, x), называемыми *клетками*, где $u \in A$ – *состояние клетки* из алфавита A, $\mathbf{x} \in X$ – имя клетки, часто задаваемое вектором $\mathbf{x} = (i, j, k)$ из конечного множества координат d-мерного дискретного пространства X. В пространстве X определены подмножества, называемые *шаблонами*,

$$T(\mathbf{x}) = \{\mathbf{x}, \mathbf{x} + \mathbf{a}_1, \ldots, \mathbf{x} + \mathbf{a}_{n-1}\}, \tag{1}$$

где $\mathbf{a}_j$ вектор смещения координат $\mathbf{x}$, $n = |T(\mathbf{x})|$. Клетки с именами из $T(\mathbf{x})$ образуют *локальную конфигурацию*

$$S(\mathbf{x}) = \{(u_0, \mathbf{x}), (u_1, \mathbf{x} + \mathbf{a}_1), \ldots, (u_{n-1}, \mathbf{x} + \mathbf{a}_{n-1})\}. \tag{2}$$

Множество клеток $\Omega = \{(u_i, \mathbf{x}_i) | u \in A, \mathbf{x}_4 \in X, \mathbf{x}_i \neq \mathbf{x}_j\}$ называется *клеточным массивом*, а перечень состояний клеток $\Omega_A = (u_1, u_2, \ldots, u_{|X|})$ – *глобальным состоянием* КА.

Функционирование КА задается *локальным оператором* и режимом его применения к клеткам из Ω. Локальный оператор

$$\Theta(\mathbf{x}) = \Phi(\Theta_1(\mathbf{x}), \ldots, \Theta_n(\mathbf{x})) \tag{3}$$

может быть композицией более простых локальных операторов, которые, в свою очередь, являются композицией подстановок $\theta(\mathbf{x})$ [9]. Подстановки выражаются через локальные конфигурации следующим образом.

$$\theta(\mathbf{x}) : S(\mathbf{x}) \to S'(\mathbf{x}), \tag{4}$$

где $|S(\mathbf{x})| \geq |S'(\mathbf{x})|$, т.е. $T'(\mathbf{x}) \subseteq T(\mathbf{x})$ причем первые $m' = |T'(\mathbf{x})|$ клеток в локальной конфигурации составляют *базу* подстановки, а остальные $(m - m')$ клеток играют роль *контекста*.

Подстановка применима к клетке $(u, \mathbf{x}) \in \Omega$, если $S(\mathbf{x}) \in \Omega$, причем клетка с переменным состоянием $(u, \mathbf{x})$ считается принадлежащей Ω, если область значений u включена в алфавит A. Применение подстановки $\theta(\mathbf{x})$ сводится к замене состояний базовых клеток $(u_j, \mathbf{x} + \mathbf{a}_j) \in S(\mathbf{x})$ на значения

$$u'_j = f_j(u_1, \ldots, u_n), \quad n = |S(\mathbf{x})|, \quad j = 0, \ldots, |S'(\mathbf{x}|, \tag{5}$$

где $f_j(u_1, \ldots, u_n)$ – *функция перехода*. Контекстные клетки не меняют своих состояний при применении подстановки (4).

В локальном операторе обычно используются следующие способы композиции подстановок: суперпозиция, случайный выбор одной из подстановок, применение подстановок в случайном порядке [9].

Применение $\Theta(\mathbf{x})$ ко всем $\mathbf{x} \in X$ называется *глобальным оператором* и обозначается $\Theta(X)$. Применение $\Theta(X)$ изменяет глобальное состояние $\Omega(t)$ на новое $\Omega(t+1)$. Такое изменение составляет *итерацию*. Итерация может выполняться разными способами, которые называются *режимами функционирования КА*. Основными из них являются: синхронный и асинхронный.

При *синхронном режиме* на каждой t-й итерации выполняется следующее:

1) для всех $(u, \mathbf{x}) \in \Omega(t)$ вычисляются новые состояния $u'(\mathbf{x})$ путем применения к ним функции перехода (4);

2) во всех клетках $(u, \mathbf{x}) \in \Omega(t)$ производится замена состояний $u(\mathbf{x})$ на новые $u'(x)$;

3) $\Omega(t) \to \Omega(t+1)$.

Локальные операторы синхронных КА ограничены тем, что их подстановки (4) должны иметь одноклеточную базовую локальную конфигурацию,

$$|S'_i(\mathbf{x})| = 1 \quad \forall \theta_i \in \Theta(\mathbf{x}), \tag{6}$$

что следует из условий корректности вычислений [12]:

$$T_k(\mathbf{x}) \cap T_m(\mathbf{y}) = \emptyset, \quad \forall \mathbf{x}, \mathbf{y} \in X, \quad \forall k, m \in \{1, \ldots, l\}, \quad l = |\Theta(\mathbf{x})|. \tag{7}$$

Асинхронный режим предполагает следующий порядок применений локального оператора:

1) с вероятностью $p = 1/|X|$ выбирается клетка $(u, \mathbf{x}) \in \Omega$;

2) к выбранной клетке применяется локальный оператор $\Theta(\mathbf{x})$, и состояния клеток базовых локальных конфигураций $(u', \mathbf{x}) \in S'(\mathbf{x})$ немедленно меняются на новые значения;

3) условно принимается, что $|X|$ повторений пп.1 и 2 составляет одну итерацию, такое соглашение удобно для сравнения синхронного и асинхронного режимов и соответствует понятию одного шага *кинетического метода Монте-Карло* [10].

Поскольку в асинхронных КА локальный оператор применяется последовательно к выбранным клеткам, условие (7) всегда выполняется, т.е. вычисление эволюции КА всегда корректно. Это значит, что не может произойти потери данных (в одну клетку одновременно записаться два разных значения). Проблема корректности возникает только тогда, когда алгоритм асинхронного КА реализуется параллельно на нескольких процессорах [11, 13].

Кроме основных режимов возможны их сочетания, а также любой другой порядок применения подстановок, если он соответствует моделируемому явлению и удовлетворяет условию (6).

Режим функционирования КА является его существенным параметром, т.е. если два КА различаются только режимами функционирования, то они являются представлениями двух разных КА. Чтобы определить однозначно КА надо задать четыре параметра: A, X, Θ, ρ, где $\rho \in \{\alpha, \sigma\}$, α обозначает асинхронный режим, σ — синхронный, а КА $\aleph_\alpha$ и $\aleph_\sigma$ обозначают асинхронный и синхронный КА, соответственно.

Задание КА в общем случае не определяет характера его поведения. Эволюции одного и того же КА с разными исходными массивами Ω_0 могут кардинально различаться. Ярким примером является всем известный КА “Игра жизнь”, который при разных начальных массивах порождает самые разные эволюции. Таким образом, КА–модель определяется пятью понятиями,

$$\mathbf{A} = \langle A, X, \Theta, \rho, \Omega(0) \rangle. \tag{8}$$

3. КА-модель процесса увлажнения пористого материала

В предлагаемой далее КА-модели просачивания жидкости сквозь пористый материал символы в (8) имеют следующие значения.

Алфавит состояний клеток $A = \{0, 1, 2, 3\}$, причем 0 интерпретируется как пустое пространство поры, 1 — как твердый материал стенки поры, 2 – как частица влаги, 3 — разбухшее от влаги твердое вещество, характерное для гидрофильных материалов.

Дискретное пространство X — прямоугольная решетка

$$X = \{(i,j,k) : i = 0,\ldots,I; j = 0,\ldots,J; k = 0,\ldots,K\}. \tag{9}$$

Оператор Θ является суперпозицией [14] двух глобальных операторов:

$$\Theta(X) = \Theta_C(\Theta_D(X)), \tag{10}$$

$\Theta_C(X)$ — *конвекционный* глобальный оператор, и $\Theta_D(X)$ — *диффузионный* глобальный оператор. Суперпозиция глобальных операторов предполагает применение локальнного оператора $\Theta_D(i,j,k)$ на каждой итерации t ко всем $(i,j,k) \in X$, а затем применение локального оператора $\Theta_C(i,j,k))$ ко всем клеткам результирующего массива (рис.1).

Конвекционный локальный оператор $\Theta_C(i,j,k)$ имитирует движение абстрактных частиц вдоль направления внешней силы. Например, если сила направлена вдоль координаты k, то его действие состоит из K последовательных шагов. На каждом шаге подстановка $\theta_C(i,j,k)$ применяется синхронно ко всем клеткам массива с координатой k, начиная от $k = K-1$ до $k = 0$.

$$\theta_C(i,j,k) : \{(1,(i,j,k))(2,(i,j,k-1))\} \xrightarrow{p_c} \{(2,(i,j,k)(1,(i,j,k-1))\} \quad \forall (i,j,k) \in X. \tag{11}$$

Вероятность p_c принимается равной или близкой к 1. так как это самое быстрое действие.

Диффузионный локальный оператор $\Theta_D(i,j,k)$ имитирует разлив жидкости, т.е. процесс выравнивания плотности жидкости, который совместно с конвекцией, приводит к выравниванию свободной поверхности жидкости в кавернах, порах, а также на внешней (верхней) поверхности. Таким образом, в каждой $k-$й плоскости массива используется двумерный вариант асинхронной *наивной диффузии* [15]. Ее локальный оператор содержит одну подстановку, основанную на 5-точечной локальной конфигурации:

$$\begin{aligned} S(i,j,k) &= \{(u_0,(i,j,k)),(u_1,(i-1,j,k)),(u_2,(i,j+1,k)),(u_3,(i+1,j,k)),(u_4,(i,j-1,k))\} \\ &= \{(u_l(i,j,k)_l) : l = 0,\ldots,4\}, \end{aligned} \tag{12}$$

выполняя обмен состояниями между центральной клеткой и одним, выбираемым равновероятно соседом, у которого $u_l \neq 0$ (не твердое вещество).

$$\theta_D(i,j,k) : \{(u_0,(i,j,k)_0)(u_l,(i,j,k)_l)\} \xrightarrow{p} \{(u_l,(i,j,k)_0)(u_0,(i,j,k)_l\}, \quad l = 1,2,3,4; \tag{13}$$

при условии, что $u_l, u_0 \in \{1,2\}$. Если стенки пор гладкие, то вероятность выбирается равной $p = 1/(1-m)$, где m — число твердых клеток в $S(i,j,k)$. Если стенки не гладкие, то вероятность применения (13) выбирается более низкой.

Поскольку диффузия — процесс более сложный чем конвекция, то чтобы согласовать скорости конвекционной итерации с диффузионной, глобальный оператор диффузии $\Theta_D(X)$ должен содержать несколько итераций применения $\theta_D(i,j,k)$ ко всем $(i,j,k) \in X$ (рис.1), т.е.

$$\Theta_D(X) = (\theta_D(X))^n. \tag{14}$$

Величина n зависит от свойств вещества и конкретных условий. В рамках модельных величин ее можно подобрать в процессе тестового моделирования, определив путем визуального наблюдения на мониторе, или включив в программу моделирования проверку, при какой величине n свободная поверхность воды остается абсолютно гладкой.

Операторы конвекции и диффузии составляют базовую часть модели, которая допускает разные расширения. Наиболее полезными для случая увлажнения почвы являются учет процесса испарения и учет гидрофильных включений.

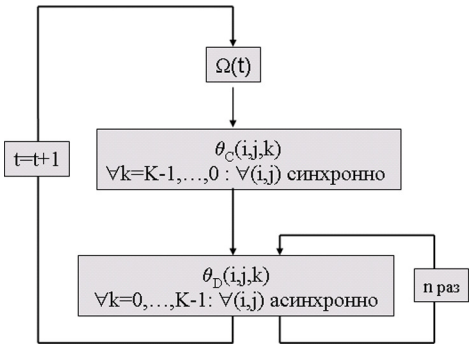


Рис. 1. Алгоритм моделирования процесса просачивания влаги через пористую среду

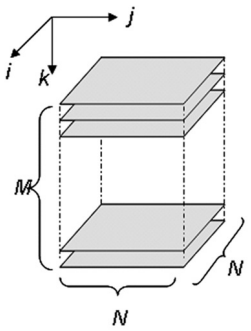


Рис. 2. Структура трехмерного клеточного массива.

Локальный оператор испарения моделирует превращение частицы жидкости в пар (пустое пространство):

$$\theta_E(i,j,k) : \{(2,(i,j,k))(1,(i,j,k-l))\} \xrightarrow{p_E} \{(1,(i,j,k))(1,(i,j,k-1)\}, \quad \forall (i,j,k) \in X. \quad (15)$$

Подстановка применяется синхронно ко всем клеткам массива с координатой k, начиная от $k = 0$ до $k = K - 1$ на каждом шаге.

Локальный оператор гидрофильности моделирует разбухание стенки поры, которая становится препятствием на пути влаги.

$$\theta_H(i,j,k) : \{(1,(i,j,k)_0)(2,(i,j,k)_l)\} \xrightarrow{p_h} \{(1,(i,j,k)_0)(3,(i,j,k)_l\} \quad l = 1,2,3,4; \quad (16)$$

Режим применения (16) такой же, как для (15). Состояние разбухшей клетки $u = 3$ в процессе увлажнения материала ведет себя так же, как твердая клетка, но в процессе высыхания может снова принять состояние $u = 2$. Для описания процесса высыхания разбухшей стенки следует применить подстановку (16), в которой состояния $u = 3$ и $u = 2$ следует поменять местами и поставить нужное значение вероятности.

Исходный клеточный массив $\Omega(0)$ — трехмерный булев массив размером $I \times J \times K$ (рис.2), в котором клетки $(0,(i,j,k))$ имитируют твердое вещество, а клетки $(1,(i,j,k))$ — пустое пространство поры. Обычно $\Omega(0)$ задается текстовым файлом, который является оцифрованным представлением морфологии пористой среды. Получение этого файла — самостоятельная проблема. Если речь идет о конкретном материале, образцы которого имеются у исследователя, то его оцифрованное представление может быть получено с помощью томографа. Если в распоряжении исследователя (например, разработчика новых материалов или устройств с их использованием) нет реальных образцов, то оцифрованное представление приходится синтезировать по заданным характеристикам. Существует несколько методов такого синтеза (в [16] имеется довольно полный их обзор). В частности, известен метод синтеза пористого массива с помощью КА [1].

Основной особенностью предлагаемой трехмерной КА-модели является то, что его конвекционная и диффузионная составляющие разделены в пространстве таким образом, что конвекционная составляющая в нем одномерная, а диффузионная — двумерная. По сути, здесь применен прием "разделяй и властвуй", который резко упрощает программирование и распараллеливание алгоритма моделирования. Оценить, насколько такое упрощение уменьшает точность моделирования, практически невозможно, поскольку сам объект "не точен": морфология пористых сред разнится от образца к образцу и часто зависит от ряда внешних факторов, как-то влажность, температура и др.

Исследование вычислительных характеристик предложенной КА-модели выполнялось в два этапа. На первом этапе проводились вычислительные эксперименты в соответствии с последовательной версией алгоритма (рис.1). На втором этапе исследовались возможности и характнристики параллельной версии.

4. Испытание последовательной версии алгоритма моделирования

Экспериментальное моделирование производилось на примере процесса увлажнения и высыхания почвы. Испытанию подвергалось оцифрованное томографическое представление образца почвы размером $10,304 \times 10,304 \times 21,88892$ мм^3 любезно присланное проф. В.Корнелли (Университет г. Гент, Бельгия) в виде 1480 текстовых файлов, каждый размером 700×700 байт (рис.2). Таким образом, линейный размер клетки $h = 14,72$ микрон, и мощность клеточного массива $|X| = 725.2 \cdot 10^6$. Морфология исследуемого образца почвы показана в разрезе (j=400) на рис.3. Последовательная версия алгоритма тестировалась на компьютере Intel Core-i7 (2,66 ГГц). Исследовался фрагмент клеточного пространства размером $200 \times 200 \times 500$ клеток, обозначенный красным прямоугльником на рис.3. Время одной итерации $\Delta t = 0,4$ сек.

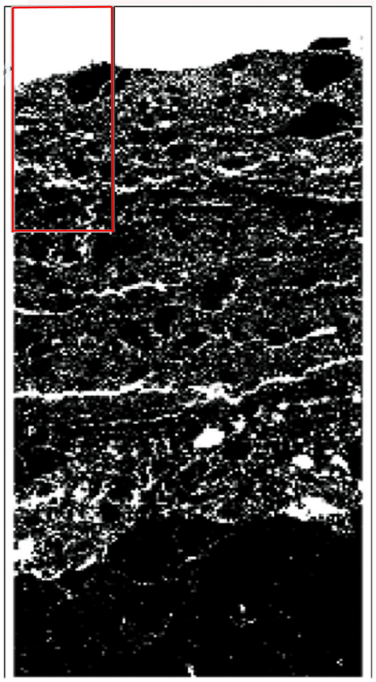


Рис. 3. Вид исследуемого образца в разрезе $j = 400$. В левом верхнем углу обозначен размер тестируемого фрагмента

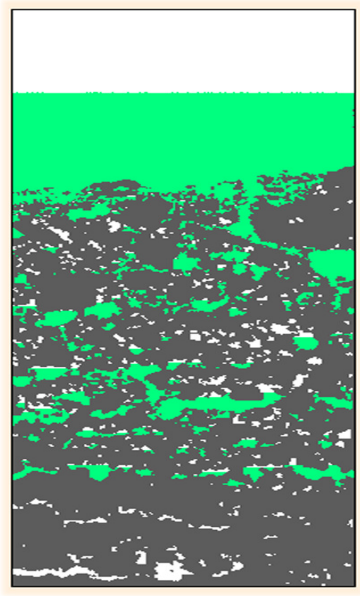


Рис. 4. Тестируемый фрагмент клеточного массива в разрезе ($j = 50$) после длительного увлажнения

Реализация последовательного алгоритма выполнялась для отладки самой модели, т.е. получения оценочных значений скорости проникания влаги в почву (рис.4), а также подбора диапазонов значений тех ее параметров, которые заранее неизвестны, и могут быть подобраны только путем вычислительного экспериментирования. Такими параметрами в исследуемой модели являются следующие величины.

1) *Коэффициент* n в (14), представляющий соотношение интенсивностей конвекции и диффузии. Он определяется путем выполнения многих реализаций алгоритма (рис.1) с разными n до тех пор, пока свободная поверхность воды не станет полностью гладкой на каждой итерации. Коэффициент n является инвариантом КА-модели конкретного явления [17], и для новой модели должен быть снова экспериментально определен. На рис. 5 показаны выводимые при моделировании изображения разреза $j = 50$ исследуемого фрагмента при $n = 5$ и при $n = 20$. Хорошо видно, что при $n = 5$ поверхность воды не успевает полностью сгладиться, а при $n = 20$ условие достаточно хорошего растекания выпонено. Этот параметр оказывает существенное влияние на время моделирования. Поэтому следует стремиться к тому, чтобы его величина была как можно меньше. Но, с другой стороны, не следует брать его слишком малым, так как это может привести к искажению результата моделирования. Аналитически рассчитать значение n невозможно, так как оно зависит от морфологии пористого материала. Поэтому, единственно приемлемым способом его определения является подбор путем моделирования.

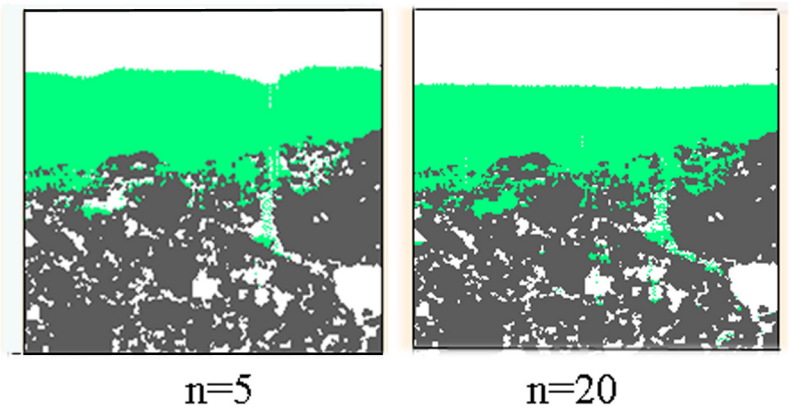


Рис. 5. Вид верхней части фрагмента образца в разрезе при $T = 1000$ при разных коэффициентах n.

2) *Возможный диапазон значений вероятности применения оператора высыхания* при моделировании процесса увлажнения с учетом испарения влаги. Хотя скорость высыхания, наверное, может быть оценена на основе физических соображений, рассчитать модельную вероятность для конкретной модели не представляется возможным. Поэтому её также необходимо определять путем проведения вычислительных экспериментов. Результаты вычислений показаны на рис.6 в виде зависимости количества влаги в порах от времени при увлажнении при разных интенсивностях испарения. Интенсивность испарения выражается в значении вероятности p_d. Из рис. 6 видно, что даже при $p_d = 0,01$ процесс увлажнения быстро заканчивается.

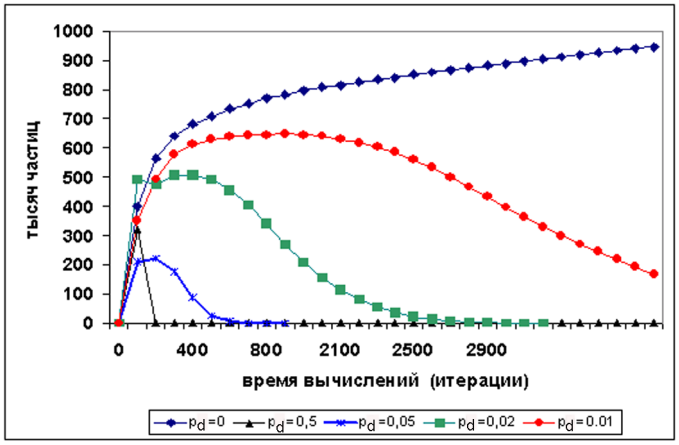


Рис. 6. Зависимость количества влаги, проникшей в почву, от времени при разных значениях интенсивности испарения

5. Результаты параллельной реализации алгоритма моделирования

Предлагаемый здесь алгоритм имеет архитектуру, которая предопределяет способ размещения обрабатываемых данных по процессорам для параллельной реализации. Конвекция, направленная сверху вниз по оси k, задает способ разрезания области вычислений плоскостями параллельными этой оси так, чтобы весь объем клеточного автомата был разделен на n частей: *доменов*. Каждый домен обрабатывается своим процессом. Межпроцессные

обмены происходят через плоскости раздела по горизонтальным направлениям. Известная проблема распараллеливания асинхронных взаимодействий, состоящая в том, что передача новых значений состояний граничных клеток должна производиться немедленно после изменения их состояний, чтобы не нарушить условия корректности (7), решается путем преобразования асинхронного режима в блочно-синхронный [11].

Процедура такого преобразования состоит в следующем.

1) Каждая k-ая плоскость клеточного массива подвергается разбиению на $|T| = 9$ непересекающихся подмножеств таким образом, чтобы согласно условию (7), соседства входящих в них клеток не пересекались.

2) На каждой итерации диффузионный оператор Θ_D применяется 9 раз подряд (за 9 стадий), каждый раз к одному из подмножеств.

3) Обмен граничными состояниями между доменами производится после каждой стадии синхронно.

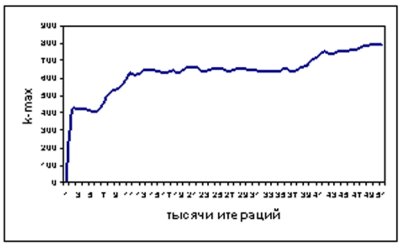


Рис. 7. Глубина проникания влаги от времени при отсутствии испарения и гидрофильных включений.

Реализация параллельного алгоритма с блочно-синхронным режимом выполнена на кластере НКС-30Т Сибирского суперкомпьютерного центра с использованием библиотеки MPI. Клеточный массив размером $700 \times 700 \times 1480$ разделен на 14 доменов, каждый размером $350 \times 100 \times 1480$, размещенных на двух восьмиядерных узлах Nehalem E5540 (2.53 GHz) кластера.

Были приняты следующие начальные условия: в почве поры пусты, на поверхности находится определенное количество воды. Граничные условия по осям i и j периодические. Программа состоит из трех частей:

1) считывание исходного цифрового представления образца исследуемого материала и разделение его на домены;

2) выполнение $T = 50000$ итераций оператора Θ (10) с выводом через каждые 1000 итераций достигнутой влагой глубины (рис.7) без учета испарения и гидрофильности;

3) выполнение начальных $T = 100$ и $T = 1000$ итераций проникания влаги в почву с учетом испарения и гидрофильности (рис.8).

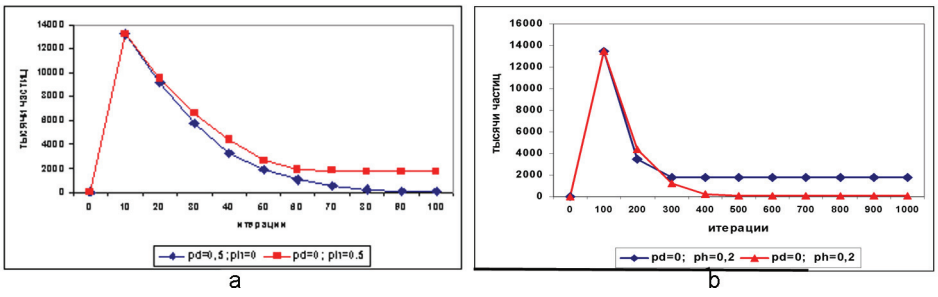


Рис. 8. Результаты моделирования увлажнения образца почвы при разных интенсивностях гидрофильности и испарения: **a**) в самом начале процесса до $T = 100$ итераций, и **b**) до $T = 1000$ итераций

В таблице 1 даны максимальные глубины проникания влаги для всех исследуемых случаев.

Анализ результатов показывает, что

1) скорость проникания влаги в почву резко уменьшается со временем (рис.7),

2) испарение влаги как с поверхности, так и из слоев почвы существенно влияет на

Таблица 1. Максимальная глубина проникания влаги в почву при разных интенсивностях испарения и гидрофильности

$T = 50000$	$T = 1000$		$T = 100$	
$p_d = 0$,$p_h = 0$	$p_d = 0,2$,$p_h = 0$	$p_d = 0$,$p_h = 0,2$	$p_d = 0,5$,$p_h = 0$	$p_d = 0$,$p_h = 0.5$
798	218	175	206	173

возможность увлажнения,

3) наличие в почве гидрофильных включений может не допустить влагу на нужную глубину.

6. Заключение

Предложенная конвекционно-диффузионная КА-модель прохождения жидкости через пористый материал отличается тем, что в ней конвекционная и диффузионная составляющие разделены как по времени (на каждой итерации выполняется их суперпозиция), так и по пространственным направлениям. Последнее обстоятельство существенно упрощает распараллеливание вычислений, поскольку позволяет свести асинхронные обмены данными к двумерному случаю. Вероятностный характер представления движений влаги в порах позволяет учитывать разную природу взаимодействия влаги со стенками: прилипание частиц влаги к стенкам и даже разбухание стенок от длительного соседства с водой. Результаты тестирования на примере увлажнения и высыхания почвы, заданной ее томографическим представлением, показали качественно соответствие с тем, что должно быть в натуре. Для получения количественных характеритик не хватает коэффициента масштабирования времени, т.е. реального времени, соответствующего одной модельной итерации. Этот коэффициент зависит от материала и может быть получен только путем сравнения результатов моделирования с натурными испытаниями этой же самой почвы.

Литература

1. О.Л.Бандман. Клеточно-автоматный метод исследования свойств пористых сред // Сиб.ЖВМ. 2010. -Том 13, № 1. С. 1–13.

2. L.M.Keller, et al., 3D geometry and topology of pore pathways in Opalinus clay: Implications for mass transport //Applied Clay Science. 2011. Vol. 52. P. 85–95.

3. Rothman B.H., Zaleski S. Lattice-Gas Cellular Automata. Simple Models of Complex Hydrodynamics. London: Cambridge Univ. Press, 1997. 480 p.

4. Hidemitsu Hayashi. Lattice Boltzmann Method and its Application to Flow Analysis in Porous Media //*R&D* Review of Toyota CRDL. 2007. Vol. 38, N 1. P. 17–25.

5. Complex Systems by Cellular Automata. Understanding complex Systems (A.G.Hoekstra et al., eds). Berlin: Springer, 2010. 450 p.

6. О.Л.Бандман. Клеточно-автоматные модели пространственной динамики // Системная информатика: Сб.научн. тр. / ИСИ СО РАН. Новосибирск: СО РАН, 2006. Вып. 10. С. 58–16.

7. Frish U., Hasslacher B., Pomeau Y. Lattice-gas Automata for Navier-Stokes equation // Physical Review Letter, 1986. Vol. 56. P. 1505-1508.

8. M.Sahimi. Flow phenomena in rocks: from continuum models to fractals, percolation, cellular automata, and simulation annealing // Review in Modern Physics, 1993. Vol. 65, N 4. P. 1393–1534.

9. Domain Specific Language and Translator for Cellular Automata Models of Physico-Chemical Processes // Proceedings of PaCT-2011, Lecture Notes in Computer Science 6873, 2011. Berlin: Springer, P. 172–177.

10. Jansen A.P.J. An Introduction to Monte-Carlo Simulation of Surface Reactions `arXiv:cond-mat/0303028v1[stst-mech]`. 2003. 51 p.

11. Bandman O. Parallel Simulation of Asynchronous Cellular Automata Evolution // Proceedings of ACRI-2006, Lecture Notes in Computer Science 4173, 2006. Berlin: Springer. P. 41–48.

12. Ачасова С.М., Бандман О.Л. Корректность параллельных прцессов. Новосибирск: Наука. 1999. 280 с.

13. Калгин К.В. Параллельная реализация асинхронных клеточных автоматов на 32-ядерной вычислительной системе. // Сиб. журн. вычисл. матем. 2012, № 1. С. 55–63.

14. О.Л.Бандман. Методы композиции клеточных автоматов для моделирования пространственной динамики. Вестник Томского госуниверситета. № 9(1). С.183–193.

15. Toffolli T., and Margolus N. Cellular Automata Machines: A new Environment for Modeling. USA: MIT Press, 1987. 297 p.

16. A.Ramirez, D.E.Jaramillo. Porous media generated by using an immiscible Lattice-Gas model // Computational Material Science. Elsevier (in press)

17. О.Л.Бандман. Инварианты клеточно-автоматных моделей реакционно-диффузионных процессов // Прикладная дискретная математика, 2012. N 3. С. 55–64.

Численное моделирование резонансного возбуждения колебаний плазмы, нагреваемой электронным пучком*

Е.А. Берендеев[1], А.А. Ефимова[2]

Новосибирский государственный университет[1],
Институт вычислительной математики и математической геофизики СО РАН[2]

Рассматривается задача взаимодействия электронного пучка с плазмой. Физический механизм взаимодействия плазмы с релятивистским электронным пучком включает в себя резонансное возбуждение колебаний плазмы, возникновение модуляции плотности плазмы с последующим рассеянием электронов в области с повышенной плотности. Для моделирования этих эффектов использовался метод частиц-в-ячейках (PIC-метод). Задача является достаточно ресурсоемкой, поэтому для ее решения реализован параллельный алгоритм, расчеты проводились на различных СуперЭВМ.

1. Введение

Одной из важнейших задач физики плазмы является нагрев высокотемпературной плазмы в термоядерных установках. В настоящей работе на основе численного моделирования исследуются процессы установления и нелинейной эволюции квазистационарной плазменной турбулентности, возбуждаемой мощным электронным пучком в установках УТС. При этом наибольший интерес представляют параметры пучка и плазмы, которые характерны для экспериментов по нагреву плазмы в открытой ловушке ГОЛ-3 (ИЯФ СО РАН) [1]. Установка ГОЛ-3 состоит из многопробочной термоядерной ловушки открытого типа с плотной плазмой, которая по своим параметрам является субтермоядерной, и генератора сильноточного релятивистского электронного пучка (РЭП), используемого для нагрева плазмы. Одним из важных достижений последних лет в физике открытых ловушек стало обнаружение подавления продольной электронной теплопроводности на торцы установки в процессе инжекции РЭП.

На данном этапе работы над задачей создан алгоритм и программа, позволяющая моделировать эффекты теплопроводности в плазме. Рассматривалось приближение бесстолкновительной плазмы, которая описывается системой уравнений Власова-Максвелла. Для моделирования задачи использовался метод частиц-в-ячейках (PIC-метод). Для тестирования программы рассматривалась задача о двухпотоковой неустойчивости. Известно, что электронный пучок, распространяющийся в плотной плазме, неустойчив по отношению к продольной модуляции плотности. Для нахождения гармоники с максимальным инкрементом нарастания напряженности электрического поля проводился дисперсионный анализ. Рассматривалась трехмерная полная гидродинамическая постановка задачи в предположении, что движение происходит вдоль оси x. Численное моделирование показало хорошее соответствие результатов с полученным аналитическим решением.

2. Постановка задачи

Полноценное исследование физических процессов в плазме может быть проведено только при комплексном подходе, сочетающем как экспериментальные исследования, так и исследования вычислительными методами, адекватно описывающими эти процессы. Для того чтобы избежать упрощений и получить качественно правильную физическую картину, необходимо построить максимально полную математическую модель. Общепринято, что хорошей исходной моделью полностью ионизованной бесстолкновительной плазмы является система уравнений, состоящая из кинетических уравнений Власова [2] для функций распределения ионов и электронов и уравнений Максвелла с самосогласованными электромагнитными полями:

* Работа выполнена при поддержке грантов РФФИ № 12-07-00065, № 11-01-00249 и интеграционного проекта СО РАН № 130.

$$\frac{\partial f_k}{\partial t} + \left(\vec{v}, \vec{\nabla}\right) f_k + q_k \left(\vec{E} + \frac{1}{c}\left[\vec{v} \times \vec{H}\right]\right) \frac{\partial f_k}{\partial \vec{p}} = 0 \quad (1.1)$$

$$rot\vec{H} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t} = 0 \quad (1.2)$$

$$rot\vec{E} = -\frac{1}{c}\frac{\partial \vec{H}}{\partial t} \quad (1.3)$$

$$div\vec{E} = 4\pi\rho \quad (1.4)$$

$$div\vec{H} = 0 \quad (1.5)$$

$$\vec{j} = \sum_k q_k \int \vec{v} f_k(\vec{p}, \vec{r}, t) d\vec{p} \quad (1.6)$$

$$\rho = \sum_k q_k \int f_k(\vec{p}, \vec{r}, t) d\vec{p} \quad (1.7).$$

Здесь индексом k обозначается сорт частиц (ионы и электроны плазмы, электроны пучка); $f_k(\vec{p}, \vec{r}, t)$ - функция распределения частиц сорта k; q_k - заряд; $\vec{j}$ - плотность тока; ρ - плотность пространственного заряда; $\vec{E}$ - напряжённость электрического поля; $\vec{H}$ - напряжённость магнитного поля.

Уравнение (1.1) является бесстолкновительным кинетическим уравнением Власова, уравнения (1.2-1.5) образуют систему уравнений Максвелла, уравнения (1.6), (1.7) определяют плотности тока и заряда через функции распределения частиц. Предполагаем, что все величины зависят от пространственных декартовых координат (x; y). Расчетная область имеет форму прямоугольника: ($0 \le x \le L_x$; $0 \le y \le L_y$), направление инжекции пучка параллельно оси x. Граничные условия периодические. Налагались условия однородности начальной плотности электронов, ионов и электронов пучка. Система уравнений (1.1 – 1.7) является самосогласованной интегро-дифференциальной системой уравнений. Наиболее хорошо для решения подобных систем уравнений себя зарекомендовал метод частиц-в-ячейках [2]. При использовании этого метода плазма моделируется набором дискретных частиц, траектории движения которых являются характеристиками уравнения (1.1). Таким образом, решаемая система уравнений состоит из уравнений Максвелла и релятивистских уравнений движения для макрочастиц.

3. Решение основных уравнений

Решение уравнения Власова производится в лагранжевых координатах – характеристики этого уравнения описывают движение модельных частиц:

$$\frac{d\vec{p}}{dt} = \frac{q_k}{m_k}\left(\vec{E} + \frac{1}{c}\left[\vec{v} \times \vec{H}\right]\right), \frac{d\vec{r}}{dt} = \vec{v} \quad (2.1)$$

Здесь m_k - масса частицы сорта k.

В этом случае используется следующая схема с перешагиванием:

$$\frac{p_i^{m+1/2} - p_i^{m-1/2}}{\tau} = \frac{q_k}{m_k}\left(E_i^m + \frac{1}{c}\left[\frac{v_i^{m+1/2} + v_i^{m-1/2}}{2}, H_i^m\right]\right) \quad (2.2)$$

Уравнения Максвелла решаются в эйлеровых переменных. Необходимые для их решения плотности заряда и тока определяются по скоростям и координатам отдельных частиц:

$$j(r,t) = \sum_i q_i v_i(t) R(r, r_i(t)) \quad (2.3)$$

$$\rho(r,t) = \sum_i q_i R(r, r_i(t)) \quad (2.4)$$

Здесь q_i – заряд частицы с номером i; функция $R(r, r_i(t))$ - (функция ядра) характеризует форму, размер частицы и распределение в ней заряда.

В настоящей работе плотности заряда и плотности тока вычисляются по формулам, предложенным Вилласенором и Бунеманом [3]. При таком подходе разностный аналог уравнения (1.4) выполняется автоматически.

Для нахождения электрических и магнитных полей используется схема, в которой напряженность электрического и магнитного полей вычисляются на сетках, смещенных относительно друг друга по времени и пространству [3]:

$$\frac{H^{m+1/2} - H^{m-1/2}}{\tau} = -c\, rot_h E^m \quad (2.5)$$

$$\frac{E^{m+1} - E^m}{\tau} = -4\pi j^{m+\frac{1}{2}} + c\, rot_h H^{m+\frac{1}{2}} \quad (2.6)$$

Таким образом, схема решения задачи на одном шаге разбивается на два этапа. На первом (лагранжевом) этапе по схеме (2.2) вычисляются скорости и координаты частиц. Здесь же определяются компоненты плотности тока $j^{m+1/2}$ и плотности заряда ρ^{m+1}. На втором (эйлеровом) этапе по решаются уравнения Максвелла, т.е. определяются значения $H^{m+1/2}$ и E^{m+1} и в узлах сетки. Значения электрических и магнитных полей, действующих на каждую частицу, вычисляются с помощью билинейной интерполяции.

4. Параллельная реализация алгоритма

В настоящей работе используется смешанная эйлерово-лагранжевая декомпозиция. Область делится на несколько подобластей вдоль одного измерения. На рисунке 1 приведена иллюстрация используемого метода - с каждой подобластью связана группа процессоров и частицы в каждой подобласти разделены между всеми процессорами группы. Каждая группа решает уравнения Максвелла только в своей подобласти. В этом случае происходит обмен граничными значениями полей между группами, также группы должны обмениваться частицами, перелетевшими в соответствующую подобласть. Внутри группы происходит обмен значениями плотности тока (как и в первом варианте распараллеливания).

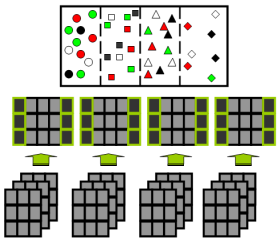


Рис. 1. Декомпозиция области и межпроцессорные коммуникации

Как было показано в работе [4], использование эйлерово-лагранжевой декомпозиции может дать существенное ускорение расчёта по сравнению с другими реализациями – это связано с уменьшением объёма пересылаемых данных и более эффективным использованием кэш-памяти процессорных ядер. В то же время из-за декомпозиции области может возникать дисбаланс по количеству частиц на разных процессоров, что приводит к неравномерной загрузке вычислительных ядер. Использование одномерной декомпозиции отчасти помогает решить эту проблему.

Программа реализована на языке Fortran-90 с использованием MPI. Расчеты проводились на следующих вычислительных системах:

- Суперкомпьютер «Ломоносов» Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова, г. Москва. Процессоры Intel Xeon 5570 2932 МГц, Cache 8 Mb;
- Суперкомпьютер «НКС-30Т» (НКС-G6) Сибирского Суперкомпьютерного Центра ИВМиМГ СО РАН, г. Новосибирск. Процессоры Intel Xeon E5540 2530 МГц, Cache 8 Mb.

4.1. Масштабируемость

Проведено исследование масштабируемости параллельного алгоритма на суперкомпьютере «Ломоносов». В таблице 1 представлено время расчёта одного шага (в секундах) при использовании различного количества процессорных ядер, а также полученное при этом ускорение. В связи с большим объёмом требуемой оперативной памяти, масштабируемость рассматривается относительно 256 процессорных ядер. Использование меньшего количества процессорных ядер не представляет собой интереса с точки зрения реальных задач. Параметры задачи:

- Число узлов 1024x1024, декомпозиция области вдоль направления Y на 64 подобласти
- Число частиц в ячейке 2500 (Всего 2 621 440 000 частиц)

Общее количество используемых процессорных ядер	Число процессорных ядер на подобласть	Время расчёта одного шага, сек.	Ускорение по сравнению с 256 процессорными ядрами
256	4	2,737	1
512	8	1,448	1,89
1024	16	0,763	3,58

Таб. 1. Время счёта одного шага (в секундах) для различного числа процессоров и полученное ускорение.

5. Вычислительный эксперимент

5.1. Расчёт инкремента амплитуды поля

Для тестирования рассматривалась задача о двухпотоковой неустойчивости. Расчетная область имеет форму прямоугольника. Ионы образуют однородный неподвижный фон. В расчетной области находятся два пучка электронов. Предполагается, что оба пучка занимают все пространство области. Граничные условия периодические, т.е. если частица вылетает за пределы области, то она вносится снова в область с другой стороны границы. Начальное распределение электронов пространственно однородно и представляет собой суперпозицию двух встречных максвелловских потоков, т.е.

$$f_0(v) = a_1\, exp\left[-\frac{(\vec{v}-\vec{v}_0)^2}{2\sigma_1^2}\right] + a_2\, exp\left[-\frac{(\vec{v}+\vec{v}_0)^2}{2\sigma_2^2}\right].$$

С течением времени пучки начинают взаимодействовать и постепенно переходят в один пучок электронов с максвелловской функцией распределения частиц по скоростям. В процессе этого взаимодействия возникает, так называемая, двухпотоковая неустойчивость. На рис.2 приведены фазовые плоскости в начальный момент времени (рис.2а) и в момент времени, соответствующий нарастанию неустойчивости (рис.2б):

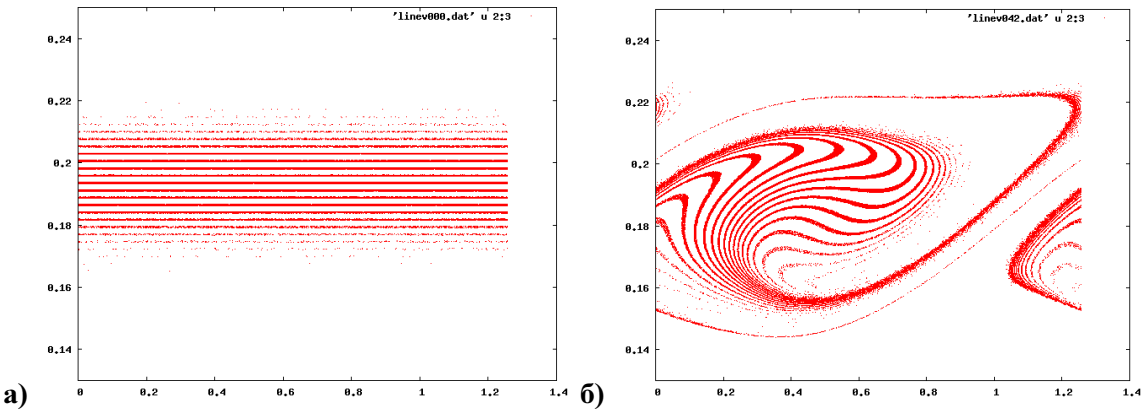


Рис. 2. Фазовые плоскости в а) начальный момент времени, б) момент времени, соответствующий нарастанию неустойчивости

На рис.3 представлен график изменения напряженности электрического поля в зависимости от времени для сетки 102x4:

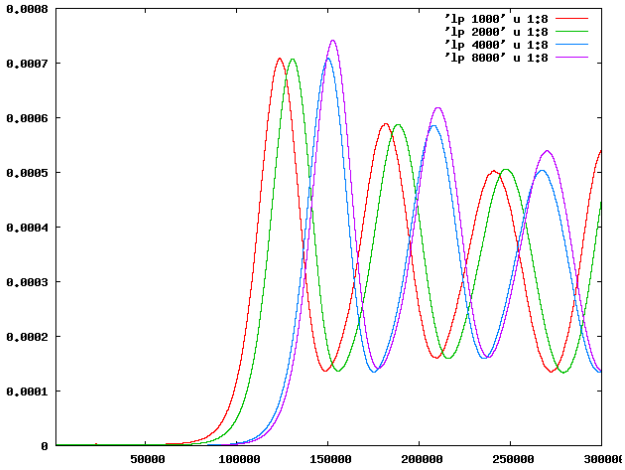


Рис. 3. Напряженность электрического поля в зависимости от времени (время в 0.001 плазменного периода) для различного числа частиц в ячейке (lp)

Из графиков видно, что на некотором временном участке происходит нарастание амплитуды электрического поля. Напряженность электрического поля в этом случае описывается экспоненциальной функцией. Из рисунка 3 видно, что при увеличении числа частиц в ячейке этот временной участок сдвигается вправо по шкале времени. Было найдено численное и аналитическое значение инкремента нарастания амплитуды поля.

Для определения аналитического значения инкремента проводился дисперсионный анализ задачи в полной гидродинамической постановке. Для этой системы уравнений выполняем линеаризацию, т.е. представляем каждую функцию в виде: $f = f^0 + f^*$, где f - одна из функций $n, v_x, v_y, v_z, n_b, v_{bx}, v_{by}, v_{bz}, E_x, E_y, E_z, B_x, B_y, B_z$, f^0- начальное значение функции, f^* - отклонение.

Предполагаем, что любую функцию f^* можно представить в виде: $f^* = \tilde{f}\exp(-iwt + ikx)$,где $\tilde{f}$ – амплитуда волны, w- частота колебаний, k- волновое число. Искомый инкремент роста амплитуды поля определяется как $\gamma = \mathrm{Im}(w) > 0$ (рис.4). Из рисунка видим, что искомое значение γ получается при $k = 0.69$, которое соответствует локальному максимуму функции γ.

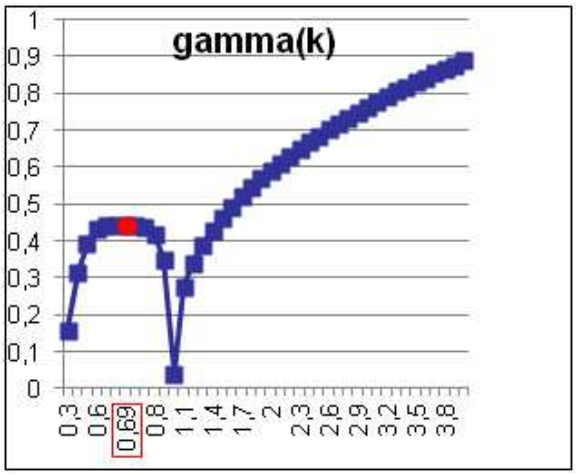


Рис. 4. Инкремент роста амплитуды поля.

5.2. Модуляция плотности

Рассматривается следующая задача:

в области, имеющей форму прямоугольника, находится полностью ионизованная плазма, состоящая из электронов и ионов. Дополнительно в область вводится пучок заряженных частиц – набор электронов, движущихся в одном направлении с достаточно большой относительно остальных частиц скоростью; предполагаются, что пучок уже полностью вошёл в моделируемую область. Модельные электроны пучка имеют меньшую массу, нежели модельные электроны плазмы (отношение их масс равно отношению плотности плазмы и плотности пучка [5]). Все частицы (как плазмы, так и пучка) распределены по области равномерно. Условия на границах области выбраны периодические.

Характерные значения: температура плазмы 500 эВ, отношение плотности пучка к плотности плазмы $2 \cdot 10^{-3}$. Размер области 1024 микрометра по каждому направлению. Сетка 1024x1024 узла, общее число модельных частиц 5 242 880 000. Расчёты проводились на суперкомпьютере «Ломоносов» с использованием до 8192 процессорных ядер. Программа реализована на языке Fortran-90 с использованием MPI.

Была исследована эволюция плотности заряда электронного пучка с развитием плазменной турбулентности. В начальный момент времени заряд электронов пучка распределен по области равномерно. При взаимодействии с плазмой в результате резонансных колебаний образуются участки модуляции плотности – от -0,0008 до -0,0037 (при начальной плотности заряда -0,002), т.е. модуляция составляет более 200% (рис. 5).

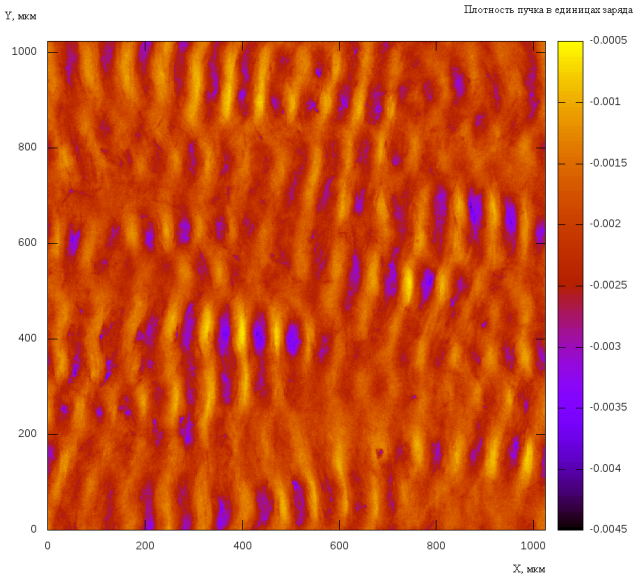


Рис. 5. Плотность заряда электронного пучка в единицах заряда электрона.

6. Заключение

В работе показано, что используемая модель позволяет моделировать эффекты теплопроводности в плазме. Рассматриваемая задача достаточно ресурсоемка, но используемый масштабируемый параллельный алгоритм, обеспечивающий равномерную загрузку вычислительных ядер, позволяет выполнять сложные расчеты. В частности, для задачи взаимодействия релятивистского электронного пучка с плазмой удалось воспроизвести эффект возникновения модуляций плотности плазмы и резонансных колебаний плазмы. Как предполагается, именно за счет этих механизмов происходит подавления продольной электронной теплопроводности на торцы установки в процессе инжекции РЭП.

Литература

1. Бурдаков А.В., Поступаев В.В. Особенности переноса тепла при пучковом нагреве плазмы в экспериментах на установке ГОЛ-3. Новосибирск,1992 (Препринт ИЯФ СО РАН; № 9).
2. Березин Ю.А., Вшивков В.А. Метод частиц в динамике разреженной плазмы // Новосибирск: "Наука", 1980.
3. Langdon A.B, Lasinski B.F. Electromagnetic and relativistic plasma simulation models // Meth. Comput. Phys. Vol. 16, 1976, P.327-366.
4. Берендеев Е.А., Ефимова А.А. Реализация эффективных параллельных вычислений при моделировании больших задач физики плазмы методом частиц в ячейках // Сборник трудов Международной научной конференции «Параллельные вычислительные технологии 2012» Новосибирск: 2012. С.380-385.
5. Вшивков В.А., Снытников А.В. Вычисление температуры при моделировании высокотемпературной плазмы методом частиц-в-ячейках на суперЭВМ // Научный вестник НГТУ. Новосибирск: 2010, № 3(40), С.61-68.

Сбор и визуализация данных о ресурсах, используемых распределенной задачей*

А.Ю. Берсенёв

Институт математики и механики им. Н.Н. Красовского УрО РАН

В данной статье пойдет речь о решении для сбора и визуализации данных, спроектированном в Институте математики и механики им. Н.Н. Красовского УрО РАН (ИММ УрО РАН), которое успешно используется на кластере «Уран».

1. Введение

Как сделать кластер производительнее? Кажется, что ответ на этот вопрос очевиден: закупить новые узлы, добавить памяти, или, скажем, модернизировать систему хранения или сеть. Но приведет ли, например, модернизация системы хранения к реальному уменьшению времени счета большинства задач?

Как правило, у задачи в каждый момент времени есть узкое место – ресурс, который она потребляет на 100%. Это тот ресурс, который имеет смысл оптимизировать, поскольку оптимизация других ресурсов не позволит обойти узкое место и заметно сократить суммарное время счета.

Для поиска узких мест можно собирать некие данные с вычислительных узлов – данные об использовании ресурсов. Так как просмотр этих данных в «сыром» виде малоинформативен, то появляется необходимость в системе анализа и визуализации данных.

В данной статье пойдет речь о решении для сбора и визуализации данных, спроектированном в Институте математики и механики им. Н.Н. Красовского УрО РАН (ИММ УрО РАН), которое успешно используется на кластере «Уран».

2. Принципы построения и архитектура

Система спроектирована с учетом следующих принципов:

1. Использование в качестве компонентов системы только продуктов с открытыми лицензиями;
2. Простота внутреннего устройства, ведущая к простоте развертывания и использования;
3. Минимальные накладные расходы на сбор и хранение данных;
4. Возможность изменения набора собираемых данных;
5. Максимальная безопасность.

Система логически разделена на две составляющие: система сбора данных и система обработки и визуализации этих данных. Такое разделение обеспечивает выполнение второго принципа. Для развертывания системы нужно установить rpm-пакет системы сбора данных и скопировать несколько файлов системы визуализации.

2.1 Система сбора данных

В качестве агентов для сбора данных используется программа collectl. Она выбрана т. к. имеет расширяемую архитектуру и распространяется под лицензией GPL v2. Программа выполняет одну простую задачу – периодически считывает данные со счетчиков в ядре ОС

* Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-1-1034).

(например, загрузку ядер cpu, размер занятой памяти и текущую скорость передачи данных), и записывает их в текстовый файл, находящийся на сетевом диске, доступном со всех узлов.

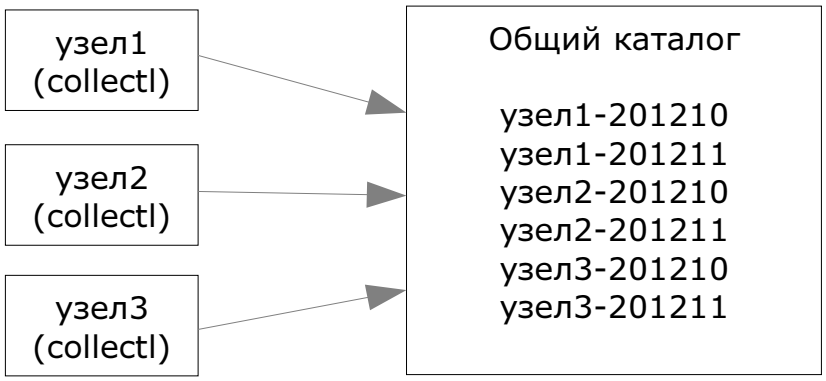


Рис. 1. Схема работы системы сбора данных

Эта программа была доработана под конкретную задачу:

1. Была добавлена возможность записывать данные в файл, имя которого формируется из имени узла, текущего года и месяца. Например, в декабре 2012 года, данные с узла «узел1» попадут в файл узел1-201212. Такая организация файлов позволяет легко делать выборки как по узлам, так и по временным интервалам, а также позволяет организовать удобную систему архивации старых файлов.
2. Для уменьшения размеров выходного файла был изменен его формат. Удаление из выдачи малоинформативных счетчиков уменьшило размер файла в 4 раза.
3. Для уменьшения влияния на запущенные процессы была добавлена буферизация вывода. Модернизированная версия collectl «накапливает» данные за час, а в конце часа записывает их в файл за одну операцию. Таким образом, обеспечивается выполнение третьего принципа.
4. Исправлено несколько ошибок при считывании данных с GPU.

При сборе статистики был выбран интервал в одну минуту, что позволяет сохранить данные о 300 узлах за год в наборе файлов общим размером около 20 гигабайт.

Для упрощения установки доработанной программы был подготовлен стандартный rpm-пакет.

2.2 Система обработки и визуализации данных

Для обработки собранных с узлов данных в ИММ УрО РАН был разработан двухкомпонентный анализатор-визуализатор. Анализатор получает от системы сбора данных информацию о загрузке ресурсов на узлах, а от системы распределения ресурсов кластера – информацию о том, в какое время и на каких узлах была запущена каждая задача. В процессе работы анализатор формирует вспомогательные таблицы использования ресурсов конкретными задачами и подготавливает данные в формате json для системы визуализации.

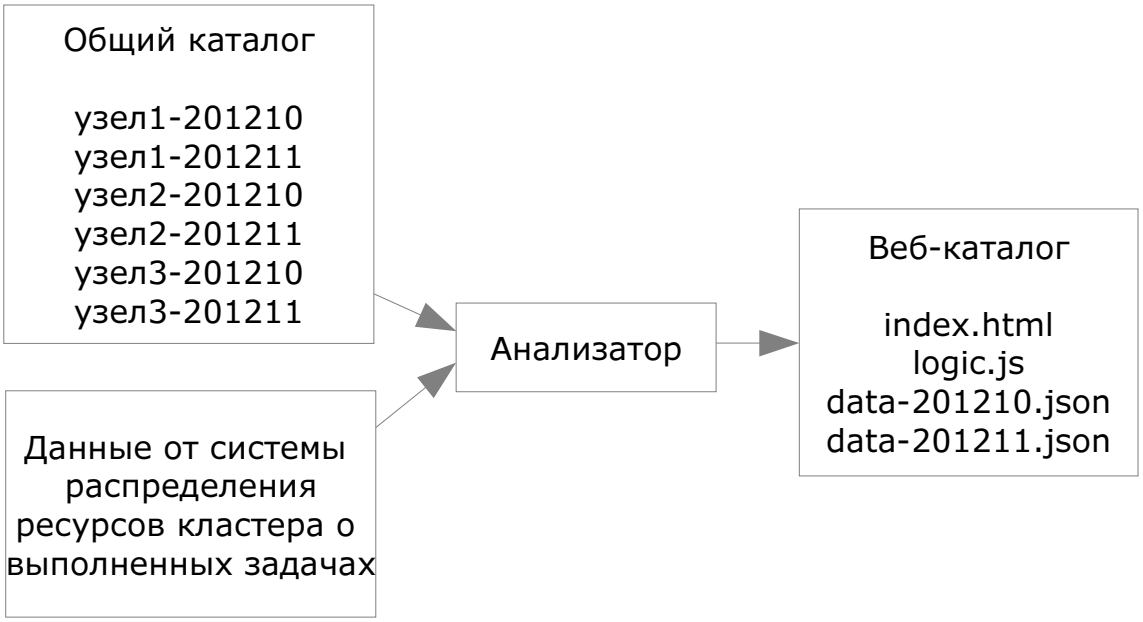


Рис. 2. Схема работы системы визуализации данных

Анализатор написан на языке Python [1]. Использование высокопроизводительного интерпретатора языка Python – PyPy – позволило в три раза ускорить работу анализатора по сравнению с запуском его в среде популярного интерпретатора CPython. Анализ реальных данных кластера «Уран» за три месяца на узле с 8 ГБ памяти и процессором 3 ГГц длится около 10 минут. Узким местом при анализе данных оказался жесткий диск, что удалось выявить с помощью самого анализатора.

Визуализатор написан на JavaScript [2, 3] и является веб-интерфейсом анализатора. Из соображений *безопасности* и предсказуемости нагрузки на сервер технологии динамической генерации страниц на стороне сервера не используются – веб-сервер отдает только статические файлы.

Вся визуализация данных происходит на клиенте с помощью генерации страницы средствами JavaScript. Для загрузки данных, ранее подготовленных анализатором в формате json, используется технология ajax.

Визуализатор позволяет просмотреть информацию о счетных задачах с разной степенью детализации. На основном экране задачи группируются по пользователям, и по каждой задаче формируется краткий ее обзор, например:

Рис. 3. Пример задачи, требовательной к памяти

В данном примере показано, что задача с именем lda_dmft.v.7.1 выполнялась на 32 узлах 20 часов; в пиковый момент на одном из узлов задача занимала 30 ГБ оперативной памяти; в среднем использовалось 19% возможностей CPU; было передано по сети всеми узлами в сумме 2.45 ГБ информации; сделано 5400 запросов ввода-вывода.

Можно сделать вывод, что задача требовательна к памяти – она использует практически всю доступную память. Запуск этой задачи на узлах с большим объемом памяти может оказаться для нее очень полезным.

При нажатии мышью на область описания задачи, можно узнать более подробные сведения по узлам:

NODE	RAM	CPU USER								CPU SYSTEM								IB	ETH	NFS READ	NFS WRITE
umt10	13.6	0	100	0	100	99	0	0	0	0	0	0	0	0	0	0	0	0.02	0	0.284	0.029
umt100	14.9	0	0	0	52	52	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.182	0.004
umt102	13.7	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.163	0
umt103	14.1	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.168	0
umt104	13.6	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.163	0
umt105	13.7	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.169	0
umt106	13.6	0	100	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.178	0.001
umt107	13.6	100	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.162	0
umt11	13.6	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0.185	0.001
umt112	13.6	0	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.165	0.001
umt117	15.2	100	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.16	0
umt118	14	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.161	0
umt119	14	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	0.165	0

Рис. 4. Подробные сведения по узлам

Видно, что задача использует два-три ядра на 100% и не использует остальные. Возможно, имеет смысл сильнее «распараллелить» задачу.

Еще несколько примеров.

Обычные счетные задачи без обмена данными между узлами:

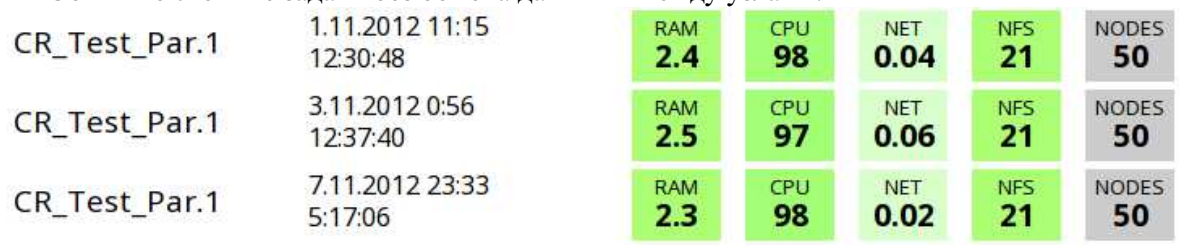


Рис. 5. Пример счетных задач

Задачи, использующие только одно ядро из доступных восьми:

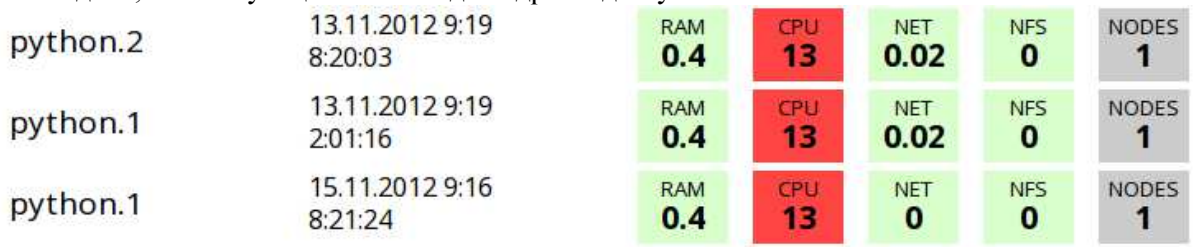


Рис. 6. Пример задач, использующих только одно ядро

Задача с интенсивным дисковым вводом-выводом:

Рис. 7. Пример задачи с интенсивным дисковым вводом-выводом

Задачи с интенсивным сетевым вводом-выводом:

		RAM	CPU	NET	NFS	NODES
rungms_imm.1	6.11.2012 17:22 3:49:41	1	40	179	2	8
rungms_imm.2	6.11.2012 18:23 1:00:01	0.8	41	79	3	8
rungms_imm.1	7.11.2012 4:54 2:16:41	1	41	132	4.2	8

Рис. 8. Пример задачи с интенсивным сетевым вводом-выводом

Восемь узлов второй задачи обменялись 79ГБ данных за час.

По каждой задаче доступны графики, показывающие, как менялось потребление ресурсов в динамике:

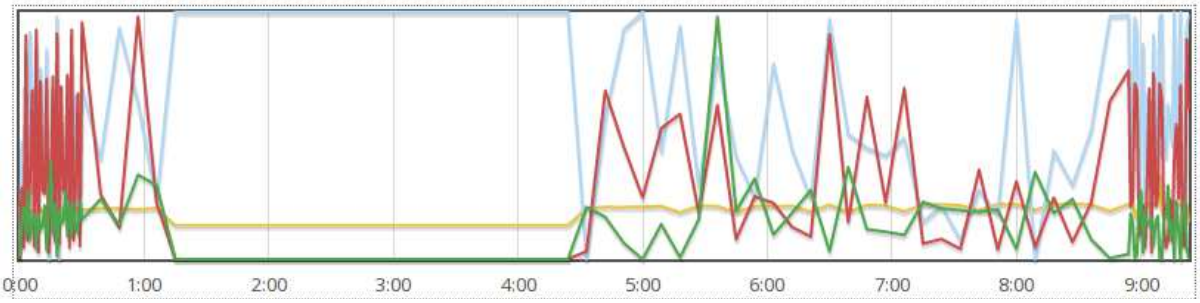


Рис. 9. Пример визуализации потребления ресурсов задачей

Бледно голубым цветом показано потребление cpu, зеленым — дисковый ввод-вывод, желто-оранжевым – потребление памяти и красным – сетевая активность.

Здесь ясно виден «счетный» участок – начиная с первого часа. Длительность этапа можно уменьшить, повысив быстродействие или количество cpu.

В среднем за месяц на кластере «Уран» запускается около 3 000 задач. Для того, чтобы посмотреть в списке только проблемные задачи, в интерфейсе предусмотрен механизм фильтрации по потребляемым ресурсам:

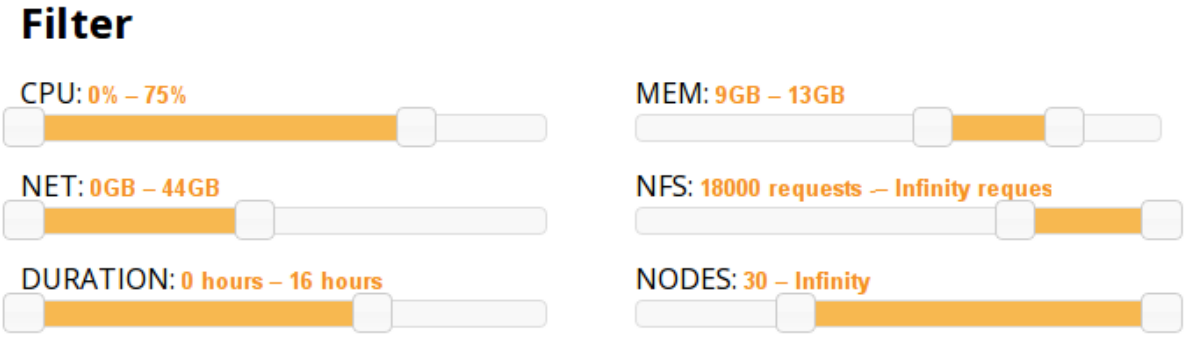


Рис. 10. Интерфейс механизма фильтрации

3. Выводы

В статье были приведены конкретные примеры того, как сбор и визуализация данных о распределенных задачах помогает определить направления дальнейшего развития кластера. Описанная система проста в установке и в использовании. Написанные в ИММ УрО РАН части доступны под лицензией GPL v2. Примеры кода доступны по адресу: https://alexbers.com/stat/src/, а визуализатор в действии – по адресу: https://alexbers.com/stat/.

Литература

1. Lutz M. Learning Python, 4th Edition. O'Reilly, 2009.
2. Bibeault B., Katz Y. jQuery in Action, Second Edition. Manning, 2010.
3. Sawyer D. M. JavaScript & jQuery: The Missing Manual. O'Reilly, 2011.
4. А.В. Адинец, П.А. Брызгалов, Вад. В. Воеводин, С.А. Жуматий, Д.А. Никитенко, К.С. Стефанов. JOB DIGEST – ПОДХОД К ИССЛЕДОВАНИЮ ДИНАМИЧЕСКИХ СВОЙСТВ ЗАДАЧ НА СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМАХ//Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (17-22 сентября 2012 г., г. Новороссийск). - М.: Изд-во МГУ, 2012. стр.9-15.

Исследование методов размещения и организации распределенного доступа к данным облачного хранилища системы дистанционного обучения*

И.П. Болодурина, Д.И. Парфёнов

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Оренбургский государственный университет"

В рамках представленного исследования построена модель хранения и организации распределенного доступа к данным с использованием облачной платформы мультимедийных образовательных ресурсов, развернутых в системе дистанционного обучения. При этом основной задачей исследования является разработка алгоритмов и методов управления производительностью и оптимизация использования программных и аппаратных ресурсов.

1. Введение

В настоящее время обучение все больше связано с использованием информационных технологий, что в свою очередь требует разработки и активного применения мультимедийных ресурсов. Наиболее широкое распространение эти ресурсы получили среди сервисов, используемых при организации обучения с применением дистанционных образовательных технологий. Лекции, семинары, форумы, а также интерактивное вещание образовательного контента активно применяются в процессе обучения. При этом инфраструктура таких решений должна обладать не только значительными аппаратными ресурсами для обеспечения необходимого качества предоставляемых услуг, но и масштабируемой и гибкой архитектурой, способной за короткое время произвести реконфигурацию ресурсов с учетом востребованности сервиса. Выполнение этого условия позволит эффективно распределять ресурсы между пользователями, что в свою очередь снизит затраты на обслуживание инфраструктуры.

Наиболее перспективными, в этом плане, являются технологии облачных вычислений. Они позволяют унифицировать доступ не только к конечным данным, но и к ресурсу в целом, что является очень важным при построении высоконагруженных приложений. Кроме того, стоит отметить несколько ключевых характеристик, показывающих перспективность внедрения облачных систем для организации массовых сервисов:

- эластичность – потребитель ресурсов самостоятельно может определять и изменять вычислительные потребности, такие как серверное время, скорость доступа и обработки данных, требуемый объем размещаемых данных;
- объединение ресурсов – возможность консолидировать несколько сервисов, используя одну и туже аппаратную базу, управляя распределением вычислительных мощностей между потребителями в условиях динамической востребованности ресурсов.

Такой подход к организации ресурсов может быть использован для организации современных образовательных мультимедийных сервисов, применяемых как компоненты систем дистанционного обучения, таких как:

- цифровое телевидение (TV over IP - TVoIP);
- видео по запросу (Video on Demand - VoD);
- интернет-трансляции;
- вебинары;
- веб-конференции.

Однако, несмотря на универсальность решения, повсеместное внедрение сетевых мультимедийных технологий в образовательный процесс остается весьма затруднительным. Прежде

* Представленная работа выполнена при поддержке Минобрнауки РФ в рамках ФЦП «Научные и научно-педагогические кадры инновационной России», проекты № 14.B37.21.0621 и № 14.132.21.1801.

всего, это связанно с построением самой инфраструктуры, обеспечивающей работу сервисов. Нами проведено исследование востребованности аппаратных ресурсов Оренбургского государственного университета, в ходе которого установлено, что:

- нагрузка на ключевые ресурсы носит периодический и неравномерный характер;
- одновременно происходят обращения к нескольким типам ресурсов;
- интенсивность обращения к каждому ресурсу может изменяться в зависимости от внешних условий;
- ввиду отсутствия распределения нагрузки между ресурсами при пиковой нагрузке оборудование не всегда позволяет обслужить все запросы;
- до 90% нагрузки предопределены, поскольку для доступа к ресурсам используется предварительная регистрация.

Кроме того, стоит отметить, что 80% ресурсов востребованы лишь в 20% времени. Так же нами установлено, что единой точкой агрегации трафика выступает система хранения данных (СХД), обеспечивающая обработку потока запросов, поступивших от потребителей мультимедийных образовательных услуг. Следовательно, эффективность работы всего мультимедийного сервиса, а так же качество предоставляемых им услуг напрямую зависит от производительности хранилища данных. Поэтому основной задачей представленного в настоящей статье исследования является разработка алгоритмов и методов управления производительностью системы хранения данных.

2. Постановка задачи и алгоритм решения

В рамках разработки консолидированного облачного сервиса для системы дистанционного обучения (СДО) нами проведен анализ мультимедийных ресурсов, задействованных при ее эксплуатации. Применение дистанционных образовательных технологий предполагает доступ к учебным материалам и образовательному контенту из любой точки, поэтому все сервисы построены как веб-приложения. Такой подход позволяет легко масштабировать решения в зависимости от потребностей пользователей, что весьма эффективно при использовании облачной инфраструктуры. Для исследования механизмов оптимизации аппаратных ресурсов нами разработана уровневая модель на основе базовых высоконагруженных подсистем:

- контроля знаний (уровень 1);
- предоставления учебных материалов (электронная библиотека) (уровень 2);
- трансляции и публикации видео и аудио материалов (видеопортал) (уровень 3).

Каждый из выделенных уровней подсистем СДО, предъявляет собственные требования к прикладному программному обеспечению, оборудованию и качеству обслуживания (QOS), что позволило провести моделирование с использованием многокритериальных показателей и как следствие создать базу знаний для управления распределением поступающей нагрузки.

Для повышения надежности и улучшения качества предоставляемых сетевых мультимедийных услуг требуется внедрение эффективных методов обеспечения распределения нагрузки аппаратно-программных ресурсов. Проанализировав интенсивность использования каждого из компонентов в системы дистанционного обучения, нами получен рейтинг востребованности ключевых ресурсов:

- канал связи;
- система хранения данных;
- система управления базами данных.

Для представленных в рейтинге ресурсов могут быть применены методы, позволяющие оптимизировать и повысить эффективность обслуживания запросов, поступающих от пользователей. При этом следует учитывать индивидуальные характеристики выбранного ресурса и алгоритмы его работы для обеспечения необходимого качества предоставляемого сервиса. В рамках данной статьи нами предложены методы распределения нагрузки и оптимизации использования ресурсов дискового пространства высоконагруженных хранилищ данных, используемых как компоненты облачной системы.

Как уже отмечалось ранее, ключевым элементом мультимедийных сетевых приложений, от которого зависит производительность и эффективность, является система хранения данных. Организация доступа и размещение данных имеют свою специфику, при этом не всегда воз-

можно использовать стандартные подходы, применяемые в большинстве систем хранения. Ключевым отличием хранилищ мультимедийных данных является неоднородность размещаемой информации (текстовые, аудио или видео данные), и как следствие разные подходы к организации доступа к ней. Помимо методов доступа к данным существенным является интенсивность обращения к тем или иным элементам, которая может быть получена с использованием внутрисистемных алгоритмов идентификации пользователей, что в свою очередь позволяет оценить востребованность и спрогнозировать нагрузку на устройства системы хранения. В связи с этим важным аспектом управления ресурсами системы, при значительном увеличении количества одновременных запросов, является грамотная организация процесса размещения и распределение элементов данных по устройствам [2,3].

Отличительной характеристикой облачных хранилищ является реконфигурируемость их структуры в зависимости от потребляемых ресурсов. Это в свою очередь позволяет внедрять алгоритмы оптимизации в плане размещения данных внутри дискового пространства, а также управлять изменением количества используемых системой устройств. При этом процесс оптимизации размещения не должен приводить к снижению качества обслуживания клиентов СХД, для чего в алгоритмах необходимо учитывать пропускную способность сети и максимальный объем данных, который можно передавать в один момент времени [4]. Кроме того необходимо учитывать текущую загрузку самих устройств, а также их расположение относительно друг друга и клиентов, подключаемых к ним.

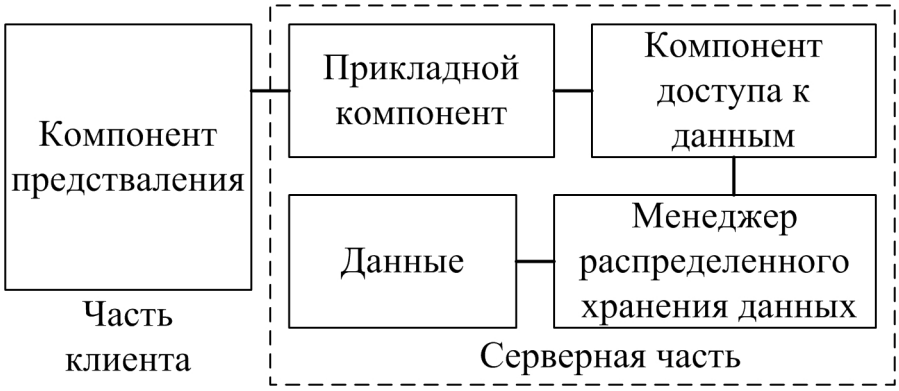


Рис. 1. Модель доступа к мультимедийным данным

Проведя анализ модели доступа к данным (см. **Рис. 1**) с учетом специфики мультимедийного сервиса, а также используя статистику поступления заявок в систему на основе типичной модели поведения клиента (пользователя), нами разработан алгоритм, позволяющий оптимизировать механизм доступа к данным. В зависимости от характера запроса выделены следующие условия работы алгоритма:

- доступ к динамическим (потоковым) данным (аудио и видео);
- доступ к статическим данным (текст, документы, архивы).

Для оптимизации механизмов доступа к данным необходимо построить общую модель требований пользователей к системе хранения. Пусть R=(U,M,Q), где $U = \{u_1,u_2,\ldots\}$ – множество пользователей; $M = \{m_1,m_2,\ldots\}$ – множество уникальных элементов данных, размещаемых на устройствах хранения. При этом минимальной единицей данных m_i будем считать файл, имеющий обязательное свойство h – размер.

Для обеспечения безопасного хранения данных и балансировки нагрузки между устройствами хранения определим функцию распределение элементов данных, для этого введем множество $M_c=\{m_1^{j_1}, m_1^{j_2}, m_1^{j_3}, \ldots, m_2^{j_1}, m_2^{j_2}, m_2^{j_3}, \ldots\}$, где $m_i^{j_k}$ - k-я копия элемента размещаемых данных (m_i) на j_k-м устройстве хранения, при условии $k\geq 3$ (не менее трех копий минимальной единицы хранения на различных устройствах).

Тогда функция распределения элементов данных по устройствам хранения принимает вид $P:M_c \rightarrow D$.

Исходя из изложенного выше, запишем требование пользователя к элементам данных. $Q:U \rightarrow X \subseteq M_c$, где X – множество данных запрошенных множеством пользователей U. Тогда хранилище данных можно записать в виде кортежа $S=(M_c,D,P,L,C,R,G)$, где

$D=\{d_1,d_2,\ldots\}$ – множество устройств хранения;

$L=\{l_1,l_2,\ldots\}$ – множество значений характеризующее загрузку каждого устройства хранения (количество одновременных обращений пользователей к конкретному устройству);

$C=\{c_1,c_2,\ldots\}$ – множество значений, характеризующее объем каждого из устройств в хранилище;

$G\in N$ – натуральный коэффициент, характеризующий географический (топологический) приоритет использования хранилища.

Как правило, для крупных облачных структур используются консолидированные хранилища, состоящие из ферм, объединяющих несколько хранилищ в единый массив. Представим его как $S_{farm}=\{S_1,S_2,\ldots\}$.

Так как характеристики требований пользователей меняются во времени, преобразуем кортеж требований $R(t)=(U,M_c,Q(t))$. Тогда $Q(t):U \rightarrow X \subseteq M_c$ – требования пользователя к элементам данных, меняющиеся во времени. Так как кроме активности пользователя изменяются свойства хранилища, запишем кортеж хранилища в зависимости от времени $S(t)=(M_c(t),D(t),P(t),L(t),C,R(t),G)$, где

$D(t)=\{d_1,d_2,\ldots\}$ множество устройств хранения, меняющихся во времени, таких что $\forall t$, $D(t)>0$;

$P(t):M_c \rightarrow D$ – функция распределения элементов данных по устройства хранения, меняющаяся во времени.

При этом для оптимизации затрат на аппаратные ресурсы и сокращения одновременно используемых устройств введем кортеж отношений $S_{cloud}(t)=\{S(t),D(t),D_{use}(t)\}$, где $\forall t$, $D_{use}(t) \subseteq D(t)$ множество устройств хранения используемых в масштабируемом хранилище S в момент времени t. Кроме того, при масштабировании хранилища и миграции данных должно выполняться условие $\forall t$, i,j $i\neq j \Rightarrow D_i(t)\cap D_j(t)=0$, т.е. при миграции данных хранилища не должны использовать одни и те же устройства. Это позволит как гарантировать скорость обработки информации, так и обеспечить приемлемое время реконфигурации.

Таким образом, для минимизации количества одновременно используемых устройств хранения, в рамках одного масштабируемого хранилища, и максимизации количества обработанных запросов пользователей в единицу времени введем целевую функцию вида:

$$\sum_{i=1}^{N} P_i(t) \rightarrow \min;$$

$$\sum_{i=1}^{N} L_i P_i(t) R_i(t) \rightarrow \max.$$

Как правило, для распределения нагрузки и повышения эффективности работы масштабируемых хранилищ, помимо дублирования и перемещения данных между устройствами так же применяют систему кеш-областей (массивы устройств, обеспечивающих возможность быстрой обработки операций чтения/записи), построенных с использованием твердотельных SSD накопителей или больших объемов оперативной памяти [5]. Однако, алгоритмы и методы использования таких ресурсов недостаточно эффективны. Чаще всего устройства кеш-области заполняются наиболее востребованными данными, при этом не учитывается модель поведения пользователя. Как правило, при обращении к мультимедийному сервису клиент отправляет последовательно несколько запросов для получения данных. В рамках мультимедийного образовательного сервиса можно предсказать набор запрашиваемых данных и порядок их получения, что позволяет построить прогноз и осуществить резервирование вычислительных ресурсов для решения поставленной задачи.

Используя вышесказанное, представим алгоритм, позволяющий оптимизировать доступ пользователя к мультимейдийным данным.

Шаг 1. Получение входных параметров.

При регистрации нового запроса, выделяются узлы (устройства хранения, D), содержащие необходимые данные и анализируется их загрузка (L) и географический приоритет относительно клиента (G). Определяется тип (статические, динамические) и рейтинг востребованности запрошенных данных, составленный на основе статистики обращений.

Шаг 2. Обработка запроса.

Для статических данных, используя полученные на шаге 1 показатели (G,L), определяется оптимальный узел.

Для динамических данных осуществляется поиск необходимого элемента данных в кеш-области. Если он не найден, то производится процедура кеширования данных с оптимального узла, полученного с использованием показателей (G,L).

Далее, используя алгоритм поиска связей, учитывающий рейтинг востребованности ресурсов, формируется перечень элементов, которые могут быть запрошены клиентом в ближайшее время. Кроме того для эффективной работы алгоритма, также осуществляется поиск наименее нагруженных узлов системы, содержащих необходимые данные, что в свою очередь позволяет частично изолировать процесс кеширования от основных операций, производимых системой. Используя полученные данные, производится процедура кеширования. При этом количество элементов зависит от востребованности начальных данных и общей нагрузки на систему.

Шаг 3. Передача данных.

Запрошенные в текущий момент времени данные направляются пользователю из выбранного источника.

Шаг 4. Постобработка результатов.

По окончанию работы алгоритма в базе данных хранимых ресурсов обновляется рейтинг востребованности использованных в обработке элементов.

3. Результаты вычислительных эксперементов

Предложенный алгоритм и модель требований положена в основу разработанного механизма балансировки нагрузки и распределения элементов данных по устройствам хранения. Разработанный симулятор модели системы хранения данных позволяет проводить исследование по установлению зависимости производительности от распределения данных в хранилище. Для оценки эффективности разработанного алгоритма нами проведено моделирование работы системы с различными параметрами. Одной из часто применяемых моделей является начальное размещение данных в единственном экземпляре на минимальном количестве устройств. При этом для всех устройств, в процессе исследования, установлено ограничение по максимальной производительности L. Эффективность работы алгоритма оценивается путем сравнения длины очереди из общего количества заявок, одновременно находящихся в системе, и числа отброшенных заявок. В результате моделирования выявлена значительная перегруженность узлов, что говорит о неэффективном распределении нагрузки (см. **Рис. 2**).

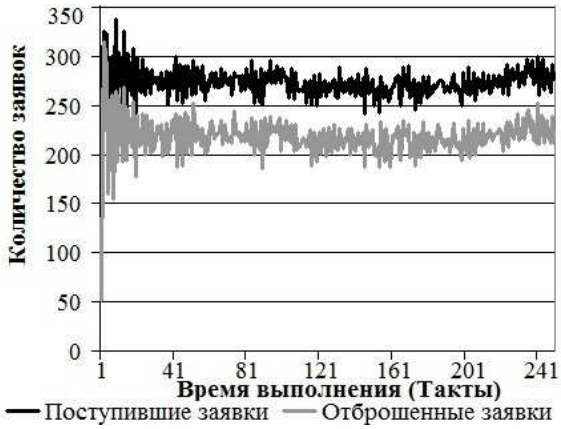


Рис. 2. Диаграмма обслуживания заявок при единственном экземпляре данных

При работе высоконагруженных систем часто применяют метод дублирования (резервирования) данных для обеспечения целостности и организации работы распределенных хранилищ. При этом количество копий каждого из элементов размещаемых может отличаться, что может вызвать дисбаланс потребления ресурсов. Тем не менее проведенное исследование с использованием симулятора СХД показало, что увеличение количества копий элементов размещаемых данных дает прирост производительности (количество одновременно обслуженных заявок) в среднем в 1,5 раза (см. **Рис. 3**).

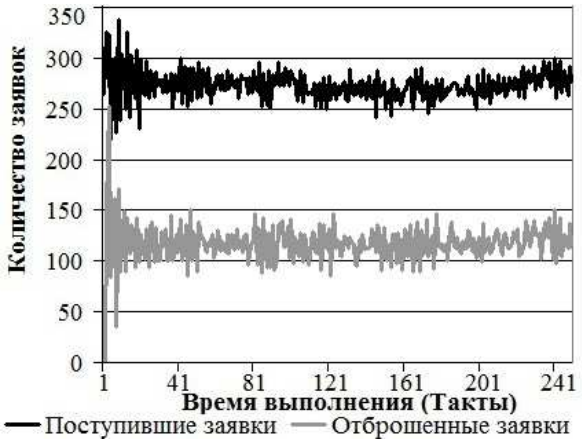


Рис. 3. Диаграмма обслуживания заявок при использовании минимального количества устройств с несколькими экземплярами данных

Однако, при использовании такого подхода к размещению данных отмечены следующие недостатки:

- постоянное масштабирование элементов данных и занимаемых ими устройств физически не всегда возможно;

- определенная часть устройств испытывает перегрузку, другая же остается ненагруженной, что говорит о неэффективности использования ресурсов хранилища.

Так же, такой подход противоречит одной из целей решаемой нами задачи – сокращение используемых ресурсов. В ходе моделирования нагрузки на хранилище нами получена закономерность, влияющая на производительность всей системы. Устройства, хранящие большие по объему (h) данные, остаются под нагрузкой дольше, чем устройства, на которых размещены данные малого размера. Большинство промышленных хранилищ анализируют только суммарный объем, используемого пространства на устройстве при выборе места первичного размещения элемента данных, что в последствии создает проблемы в ходе миграции и резервного копирования элементов на другие устройства. Для оптимизации размещения элементов в симулятор системы хранения данных нами добавлен балансировщик, производящий качественную оценку размещенных данных на устройстве по критерию объема дискового пространства занимаемого элементами данных, как отношение количества больших и малых файлов. При этом относительность оценки большого и малого размера файла производится как в отношении анализируемого устройства, так и в рамках всей системы хранения в целом. Проведя моделирование с использованием полученных показателей на минимальном количестве устройств, получено дополнительное увеличение производительности на 5-7%(в зависимости от входного потока данных и суммарного объема размещаемых данных) при условии хранения нескольких копий данных (см **Рис. 4**).

Рис. 4. Диаграмма обслуживания заявок при использовании минимального количества устройств с интеллектуальным размещением данных

Стоит отметить, что применение перечисленных методов может быть дополнено предложенным в данной статье алгоритмом интеллектуального кеширования. Для оценки эффективности результатов его работы, в качестве балансировщика нагрузки СХД, нами проведено комплексное моделирование с использованием всех описанных выше методов. В результате получено дополнительное увеличение производительности на 4-9% (в зависимости от начального метода размещения данных) (см. **Рис. 5**).

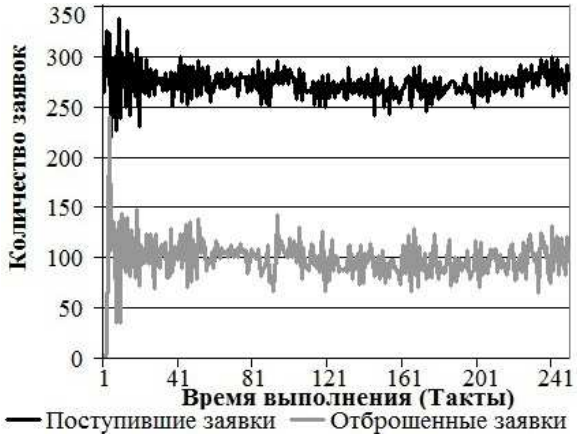


Рис. 5. Диаграмма обслуживания заявок при использовании минимального количества устройств с интеллектуальным размещением и кешированием данных

Заключение

Таким образом, оценивая общий результат работы алгоритма балансировки нагрузки и размещения данных при организации доступа к образовательным мультимедиа ресурсам можно получить прирост производительности от 5 до 15% по сравнению со стандартными средствами, что является весьма эффективным при большой интенсивности запросов. Кроме того, полученная целевая функция и построенная модель могут применяться для дальнейшего исследования эффективности использования аппаратных и программных ресурсов с целью повышения качества предоставления услуг в распределенных информационных системах дистанционного обучения.

Литература

1. Болодурина И.П., Решетников В.Н., Парфёнов Д.И. Распределение ресурсов в информационной системе дистанционной поддержки образовательного процесса Программные продукты и системы. -Тверь: НТП «Фактор», 2012. -3:-стр. 151-155.
2. Петров Д.Л. Оптимальный алгоритм миграции данных в масштабируемых облачных хранилищах // Управление большими системами. Вып. 30, 2010. -С.180-197.
3. Петров Д.Л. Динамическая модель масштабируемого облачного хранилища данных // Известия ЛЭТИ, #4, 2010. -С. 17-21
4. Проблема адекватной оценки производительности веб-серверов в корпоративных сетях на предприятиях ЦБП / О.В. Гусев, А.В. Жуков, В.В. Поляков, С.В.Поляков // Материалы 6-й научно-технической конференции «Новые информационной технологии в ЦБП и энергетике».- Петрозаводск, 2004. – С. 84-87
5. Жуков А.В. Некоторые модели оптимального управления входным потоком заявок в интранет-системах. // Материалы 6-й научно-технической конференции «Новые информационной технологии в ЦБП и энергетике».- Петрозаводск, 2004. – С. 87-90.
6. Бойченко И. В. Управление ресурсами в сервис-ориентированных системах типа «приложение как сервис» / И.В. Бойченко, С.В. Корытников // Доклады Томского государственного университета систем управления и радиоэлектроники Вып. 1-2, 2010. -С. 156-160.

Использование гибридных вычислительных узлов на базе GPU TESLA C2075 при проведении расчетов в области вычислительной химии и молекулярной динамики

А.В. Волохов[1], В.М. Волохов[1], Д.А. Варламов[1,2],
А.В. Пивушков[1], Г.А. Покатович[1], А.И. Прохоров[1]

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

На примере прикладных пакетов в области квантовой химии и молекулярной динамики, реализованных на базе гибридных вычислительных узлов в ИПХФ РАН (с применением GPU Nvidia Tesla C2075), изучена возможность применения гибридных вычислений для повышения эффективности расчетов в области вычислительной химии, в том числе на ресурсных узлах различных грид-сред.

1. Введение

Развитие «гибридных» (CPU+GPU) вычислительных технологий достигло точки, когда множество существующих приложений в различных областях реализуются с их использованием и работают значительно быстрее, чем на обычных многоядерных системах. В настоящее время резко интенсифицировался перевод программного кода прикладных пакетов на параллельные технологии с использованием технологий программирования CUDA™ и аналогичных. При этом программирование обновленного или создаваемого заново «гибридного» кода для квантово-химических и молекулярно-динамических пакетов занимает одно из ведущих позиций в этом направлении. В течение последних лет существенно упростилось программирование соответствующей модели параллельных вычислений с использованием среды программирования CUDA™, которая обеспечивает набор абстракций, позволяющих выражать как параллелизм данных, так и параллелизм задач. Это позволяет проводить высокоинтенсивные вычисления с применением встраиваемых в расчетные узлы кластеров высокопроизводительных графических процессоров (Nvidia – Tesla и Kepler, AMD – ATI Radeon), что значительно повышает эффективность использования параллельных методов расчетов и снижает требования к расчетным центрам и операционную стоимость расчетов, особенно при использовании большого числа расчетных узлов. Применение методов параллелизации с использованием GPU устройств для прикладных пакетов в области квантовой химии и молекулярной динамики в большинстве случаев ведет к снижению времени расчетов от десятков процентов до нескольких раз, либо (как вариант) дает возможность существенного повышения точности или детальности расчетов.

В 2012 году авторами были изучены предпосылки широкого использования новых программных вариантов прикладных квантово-химических пакетов (ППП), основанных на гибридных вычислениях, в интересах вычислительного центра ИПХФ и в перспективе – для применения в качестве вычислительных грид-сервисов (на базе выделенного грид-ресурса). Было проведено тестирование и оптимизация имеющихся вариантов ППП с последующей адаптацией их для проведения расчетов в грид-средах в рамках грид-центра ИПХФ РАН. В настоящее время разрабатываются методики запуска грид-заданий, ориентированных на использование ресурсов с поддержкой CUDA™ технологий, в том числе на доступных в рамках грид-полигонов суперкомпьютерных установках.

2. Использование ППП вычислительной химии с применением «гибридных» вычислений в ИПХФ РАН

2.1 Оценки повышения эффективности проведения расчетов с использованием «гибридных» расчетных узлов

Существует большой спектр различных источников (основанных на оценках собственно разработчиков, тестеров, конечных пользователей), демонстрирующих эффективность «гибридных» методов вычислений для проблемно-ориентированных ППП в области вычислительной химии и молекулярной динамики. Предлагаемые к внедрению в практику грид-вычислений гибридные GPU расчеты представлены совместным использованием CPU и GPU в гетерогенной модели вычислений. Стандартная часть приложения выполняется на CPU, а более требовательная к вычислениям часть, допускающая параллельное проведение расчетов, обрабатывается с GPU ускорением. С точки зрения пользователя приложение работает быстрее, так как оно использует высокую производительность и параллельность GPU для повышения производительности. В настоящее время наиболее мощные GPU устройства способны обеспечить производительность, оцениваемую как первые терафлопсы при проведении вычислений с плавающей запятой благодаря новой архитектуре массивно-параллельных вычислений CUDA. Архитектура CUDA состоит из сотен процессорных ядер, которые работают в единой массивно-параллельной связке, чтобы разом справиться с набором данных в приложении.

Разработчиками прикладных пакетов показано, что использование массивно-параллельных GPU архитектур NVIDIA и Radeon для большинства ППП позволяет получать превосходные результаты при работе с приложениями в сфере квантовой химии и молекулярной динамики. Различные источники дают варьирующие оценки выигрыша во времени расчетов и усиления коэффициентов масштабируемости и эффективности использования расчетных узлов. Это определяется: а) качеством программного кода, переделанного или созданного заново для использования GPU; б) различными количественными вариациями соотношений CPU vs. GPU – зависит от типов задач и во многом определяет эффект вычислений; в) используемыми в прикладных задачах вычислительными методами (например, CPU-ориентированный метод в редких случаях может давать даже замедление при использовании его на GPU устройствах).

Для единичного узла выигрыш во времени расчетов составляет (приведены оценки только для ППП, уже реализованных авторами в ИПХФ в качестве грид-сервисов и в дальнейшем планируемых к рекомпиляции или установке GPU-ориентированных бинарных версий на грид-ресурсы ИПХФ): **GAMESS/FireFly** (метод самосогласованного поля) – до 50-60 раз, **Gaussian** (расчет потенциала Кулона) – 12-15 раз, **VASP** – 3-6 раз, **NWChem** – 3-8 раз, **GROMACS** (метод суммы Эвальда) – от 2 до 5 раз, **LAMMPS** (потенциалы Леннард-Джонса, Гей-Берне) – в 6 раз, **NAMD** (расчет валентно-невалентных взаимодействий) – от 2 до 7 раз. Приведены как оценки авторов на основе проведенных расчетов реальных квантово-химических задач, так и оценки, скомпилированные в обзоре прикладных пакетов на сайте Nvidia Corp. (http://www.nvidia.ru/object/computational_chemistry_ru.html). Это пакеты, для которых массивно-параллельные GPU-ориентированные алгоритмы реализованы лишь частично. Для сравнения – для прикладных пакетов, которые создавались именно для CUDA-ориентированных вычислений (например, TeraChem и PetaChem - http://www.petachem.com или ‘Schrodinger Core Hopping’) выигрыш по времени оценивается существенно выше и может составлять от 50 до 5000 раз (сведения приведены разработчиками пакетов).

2.2 Опытный полигон ИПХФ РАН по оценке перспективности «гибридных» вычислений для проблемно-ориентированных грид-сервисов

В 2012 году в ИПХФ РАН были начаты методические и практические исследования возможности использования новых GPU-оптимизированных программных вариантов различных прикладных квантово-химических пакетов как на локальных расчетных узлах ИПХФ (включая кластеры), так и в рамках грид-сайтов различных грид-полигонов.

На средства программы Президиума РАН была произведена закупка 2-х высокопроизводительных вычислительных узлов, оптимизированных для задач вычислительной химии, со следующей конфигурацией:

1 узел – 12 вычислительных ядер [2x6 3.46GHz Intel® Xeon® X5675], 48 Гб RAM, 3 Тб HDD/SSD, 2 GPU Nvidia Tesla C2075, 3 сети Gigabit Ethernet, пиковая производительность узла – до 1,3 Тф для вычислений двойной точности (2.3 Тф для вычислений ординарной точности).

Таким образом, ожидаемая пиковая производительность пула составляет около 2,6 Тф. После тестирования физических узлов было установлено и сконфигурировано системное ПО узлов (ОС CentOS 6.3) с установкой драйверов Nvidia (версия 304.54 с последующим обновлением до 310.19). Были установлены все необходимые программные (gcc и gfortran95 – 4.4.6, CUDA SDK 5.0.35 и 4.2.9 (для совместимости ряда ППП), специализированные библиотеки *BLAS, Lapack, FFT*, скомпилированные с поддержкой CUDA технологий), сетевые и кластерные средства. После установки ПО была протестирована работа узлов в качестве гибридных вычислителей на внутренних тестах пакета CUDA и независимых тестах (типа cuda_memtest), далее узлы были объединены в единый пул с поддержкой GPU технологий. После тестирования пула в локальном режиме проведена его интеграция в ресурсный грид-центр ИПХФ РАН с последующим представлением в качестве грид-ресурса (выделенная очередь заданий) в рамках доступных российских грид-полигонов (ГридННС и пилотная российская грид-сеть для высокопроизводительных вычислений).

Для отработки методики GPU вычислений в качестве первоочередного был установлен ППП NAMD версии 2.9, откомпилированный в версии «Linux-x86_64-multicore-CUDA», который был запущен для тестирования и оценки эффективности. На ряде тестовых задач ППП NAMD была проведена предварительная оценка повышения эффективности расчетов, которая показала выигрыш во времени от 2 до 4-4,5 раз на мицеллах средней размерности (время расчета до 1-2 суток). Были проведены первичные испытания различных соотношений CPU/GPU (от 1:1 до 2:4), варьирование разных используемых вычислительных методов (определяется конфигурационным файлом NAMD), проверена возможность передачи GPU-ориентированной версии NAMD входных грид-заданий через существующие шлюзы ГридННС и пилотной российской грид-сети. Полная оценка эффективности GPU ускорения для ППП NAMD возможна при расчете высокомолекулярных соединений с числом атомов 10,000 и более (желательно более 10^6 атомов), где высокий параллелизм расчетов хорошо оправдывает накладные расходы на связь CPU-GPU и затраты на организацию внутренних циклов большинства *ab initio* квантово-химических методов.

В настоящее время (декабрь 2012 г.) проводится инсталляция и тестирование на расчетном пуле GPU-ориентированных версий квантово-химических пакетов AbInit, NWChem, PWScf, Orca и др. Инсталляция проводится (при возможности) в двух вариантах: заранее откомпилированные разработчиками бинарные варианты ППП и (при предоставлении исходных кодов) откомпилированные авторами варианты с учетом оптимизации под конкретные конфигурации расчетных узлов. Оценки повышения эффективности использования указанных ППП будут приведены в докладе на конференции ПАВТ-2013.

Также начата разработка методики запуска грид-заданий (для промежуточного ПО Globus Tools 4 и 5 и основанных на нем грид-сред), ориентированных на поиск и использование грид-ресурсов, поддерживающих использование CUDA технологий. В настоящее время существующие системы управления грид-заданиями (брокеры ресурсов, pilot-менеджеры и т.п.) не поддерживают впрямую в файлах описания грид-заданий целеуказания на использование именно гибридных вычислительных узлов, что не позволяет пользователю, как в ручном, так и тем более в автоматическом режиме выбирать более эффективные «гибридные» ресурсы для запуска ППП. Однако, в дальнейшем ситуация, как ожидается, будет исправлена разработчиками грид-сред (как и в случае использования новейших мультисопроцессоров типа Intel Phi).

3. Заключение

Использование «гибридных» массивно-параллельных GPU архитектур (NVIDIA, AMD) для большинства квантово-химических и молекулярно-динамических ППП позволяет существенно улучшить эффективность использования единичных расчетных узлов и основанных на их ис-

пользовании пулов ресурсов, а в перспективе – и ресурсных грид-центров на их основе. Введение в массовую практику использования GPU-ориентированных версий прикладных пакетов в области вычислительной химии позволяет существенно расширить общий круг расчетных задач, особенно – для интенсивно использующих параллелизацию квантово-химических расчетов (в первую очередь *ab initio*). При этом также становится возможным значительно повысить точность и детальность проводимых с помощью ППП расчетов. Настоятельно стоит вопрос о решении проблемы идентификации пулов «гибридных» вычислительных узлов в качестве специфичных грид-ресурсов с возможностью их целенаправленного выбора. В целом, применение «гибридных» вычислений (в том числе в рамках грид-полигонов) позволит значительно интенсифицировать расчеты с применением параллельных прикладных квантово-химических пакетов на ресурсных узлах грид-инфраструктуры в рамках виртуальных организаций, базирующихся на разнородных вычислительных ресурсах.

Достижение экзафлопсной производительности в задачах глобальной оптимизации*

В.П. Гергель, К.А. Баркалов

Нижегородский государственный университет им. Н.И. Лобачевского

Эффективное использование огромных вычислительных возможностей экзафлопсных систем представляет собой глобальную проблему «вызова» всему спектру вычислительных наук. Условиями для успешного достижения экзафлопсной производительности являются: значительная вычислительная трудоемкость решаемых задач, высокий запас параллелизма вычислений, низкая интенсивность информационная зависимость вычислений, устойчивость вычислений к аппаратным сбоям вычислителей. В работе показывается, что все перечисленные условия достижимы для задач глобальной оптимизации.

1. Проблема организации эффективных экзафлопсных вычислений

Колоссальный вычислительный потенциал современных суперкомпьютерных систем позволяет приступить к решению многих сложнейших научно-прикладных проблем, анализ которых ранее находился за гранью возможного. Динамика развития производительности суперкомпьютеров опережает все самые смелые ожидания специалистов. Так, в 2008 г. был преодолен петафлопсный (10^{15} операций с плавающей запятой в сек.) рубеж скорости вычислений, в 2018-2020 гг. ожидается достижение экзафлопсного (10^{18}) уровня производительности. По оценкам специалистов, подобные суперкомпьютерные системы с рекордными вычислительными показателями будут существенно многопроцессорными (до миллиарда вычислительных ядер), гибридными с разными типами вычислительных устройств с многоуровневой структурой организации вычислений (распределенные вычислительные устройства – вычислительные узлы с общей разделяемой памятью – многоядерные процессорные элементы – ускорители вычислений) [1]. Эффективное использование огромных вычислительных возможностей экзафлопсных систем представляет собой глобальную проблему «вызова» всему спектру вычислительных наук. Условиями для успешного преодоления этой проблемы являются:

- значительная **вычислительная трудоемкость** решаемых задач (свыше 10^{18} операций);
- высокий запас параллелизма (**масштабируемость**) вычислений (вплоть до использования 10^{9} процессоров);
- низкая интенсивность информационных взаимодействий (**локальность**) параллельно выполняемых вычислений;
- **устойчивость** вычислений к аппаратным сбоям вычислителей, которые неизбежно будут происходить при столь больших количествах вычислительных элементов.

В работе показывается, что все перечисленные условия достижимы для задач глобальной оптимизации.

2. Оценка вычислительной трудоемкости задач глобальной оптимизации

Прежде всего, следует отметить, что задачи оптимизации имеют важное прикладное значение и чрезвычайно широкую область приложений – практически каждая задача проектирования новых устройств, изделий, систем включает в себя этап выбора оптимальных вариантов. В наиболее сложных ситуациях проблема выбора формулируется как задача глобальной оптимизации, когда критерии оптимальности могут быть многоэкстремальными, т.е. иметь множество

* Исследование выполнено при поддержке РФФИ, грант № 11-01-00682-а, а также Министерства образования и науки Российской Федерации, соглашение № 14.B37.21.0393

локально-оптимальных решений, среди которых необходимо выделить наилучшие варианты. Допустимость многоэкстремальности существенно повышает сложность решения оптимизационной задачи, ибо, если для подтверждения локального минимума достаточно исследования локальной окрестности, то глобальный минимум является интегральной характеристикой решаемой оптимизационной задачи и требует исследования всей области глобального поиска. Как результат, поиск глобального оптимума сводится к построению некоторого покрытия (сетки) в области вариации параметров. Поэтому на сложность решения задач рассматриваемого класса решающее влияние оказывает размерность: они подвержены "проклятию размерности", состоящему в экспоненциальном росте вычислительных затрат при увеличении размерности. Так, если вычисление одного значения оптимизируемой функции требует m операций, а количество узлов сетки при решении одномерной задачи равно k, то общая оценка общего объема вычислений для N-мерной задачи глобальной оптимизации имеет вид

$$T = mk^N.$$

Приняв, например, $m=10^3$, $k=10$, $N=20$, общий объем вычислений получается равным $T=10^{23}$, что требует порядка суток вычислений на суперкомпьютере экзафлопского уровня производительности.

Снижение объема вычислений может быть достигнуто при построении существенно неравномерных покрытии областей глобального поиска – эти сетки должны быть достаточно плотными в окрестности глобального оптимума и более разреженными вдали от искомого решения. Построение таких оптимальных покрытий обеспечивается при повышении сложности самих численных методов глобального поиска.

Применение неравномерных покрытий позволяет повысить размерность решаемых задач глобальной оптимизации в 2-3 раза, что является критичным в приложениях. Получаемые оценки размерности (25-50) позволяют охватить большинство научно-прикладных задач, поскольку, как правило, количество варьируемых параметров, по которым проявляется многоэкстремальность оптимизируемых критериев, ограничено.

3. Основные принципы организации вычислений в задачах глобальной оптимизации

3.1 Класс решаемых задач

Задача многомерной многоэкстремальной оптимизации может быть определена как проблема поиска наименьшего значения действительной функции $\varphi(y)$ [1]

$$\varphi(y^*) = \min\{\varphi(y): y \in D\},\ D = \{y \in R^N: a_i \le y_i \le b_i,\ 1 \le i \le N\}, \quad (1)$$

где a, $b \in R^N$ есть заданные векторы.

Численное решение задачи (1) сводится к построению оценки $y_\varepsilon^* \in D$, отвечающей тому или иному понятию близости к точке y^* на основе конечного числа k значений функционалов задачи, вычисленных в точках области D.

Относительно класса рассматриваемых задач предполагается выполнение двух важных принципиальных требований:

- процедуры проведения *испытаний* (вычисления значений целевой функции) является *вычислительно-трудоемкими*,
- функция $\varphi(y)$ должна удовлетворять *условию Липшица* (ограниченность изменения значений функции при ограниченной вариации параметров).

Следует отметить, что именно выполнимость условия Липшица позволяет строить численные оценки искомого глобального решения по конечному набору вычисленных значений функции $\varphi(y)$.

[1] Для уменьшения сложности изложения материала задача глобальной оптимизации в данной работе рассматривается при отсутствии функциональных ограничений.

Отметим также, что все далее излагаемые результаты получены в рамках информационно-статистической теории многоэкстремальной оптимизации [2, 3], являющейся одним из наиболее активно развиваемых направлений исследований в области глобального поиска.

3.2 Редукция размерности

Для снижения сложности алгоритмов глобальной оптимизации, формирующих эффективные покрытия области поиска, в теории и практике многоэкстремальной оптимизации широко используются различные схемы редукции размерности, которые позволяют свести решение многомерных оптимизационных задач к семейству задач одномерной оптимизации. Редукция размерности позволяет существенно снизить сложность разрабатываемых алгоритмов глобального поиска. Кроме того, данный подход позволяет задействовать весь имеющийся аппарат одномерной многоэкстремальной оптимизации для построения эффективных многомерных методов глобального поиска.

Один из наиболее общих методов редукции размерности состоит в применении разверток единичного отрезка вещественной оси на гиперкуб. Роль таких разверток играют непрерывные однозначные отображения типа кривой Пеано, называемые также кривыми, заполняющими пространство.

Использование развертки Пеано $y(x)$, однозначно отображающей единичный отрезок вещественной оси на единичный гиперкуб, позволяет свести многомерную задачу минимизации в области D к одномерной задаче условной минимизации на отрезке $[0,1]$

$$\varphi(y(x^*))=\min\{\varphi(y(x)):x\in[0,1]\}. \qquad (2)$$

Вопросы численного построения отображений типа кривой Пеано и соответствующая теория подробно рассмотрены в [2, 3].

3.3 Общая схема алгоритмов глобального поиска

Успехи в разработке одномерных алгоритмов глобального поиска в рамках информационно-статистической теории глобального поиска позволили получить чрезвычайно значимый результат – к формулированию общей схемы представления алгоритмов, в рамках которой с единых позиций можно провести анализ свойств методов и даже ставить задачу построения алгоритмов с заданными свойствами.

Данная общая схема, получившая наименование *схемой характеристической представимости алгоритмов глобального поиска* (см., например, [2]), состоит в следующем.

Алгоритм глобального поиска называется *характеристическим*, если, начиная с некоторого шага поиска $k_0\geq 1$, выбор точки x^{k+1} очередной итерации (*решающее правило алгоритма*) включает выполнение следующего набора действий:

1. Упорядочить поисковую информацию в порядке возрастания значений точек ранее выполненных итераций (новый порядок расположения данных в поисковой информации отражается при помощи нижнего индекса):

$$x_1 \leq x_2 \leq \ldots \leq x_k ,$$

2. Вычислить для каждого интервала $(x_{i-1}, x_i), 1 < i \leq k$, величину $R(i)$, называемую далее *характеристикой* интервала,

3. Определить интервал (x_{t-1}, x_t), которому соответствует максимальная характеристика $R(t)$, т.е.

$$R(t) = \max\{R(i) : 1 < i \leq k\},$$

4. Вычислить точку x^{k+1} очередной итерации глобального поиска в интервале с максимальной характеристикой в соответствии с некоторым правилом

$$x^{k+1} = s(t) \in (x_{t-1}, x_t) .$$

4. Параллельные вычисления для поиска глобально-оптимальных решений

4.1 Основной принцип организации параллельных вычислений

Прежде всего следует отметить, что в случае многомерной многоэкстремальной оптимизации многие широко используемые подходы для распараллеливания или не обеспечивают должного эффекта или вообще не применимы. В частности, один из основных способов организации параллельных вычислений состоит в разделении области решения задачи между процессорами и параллельного решения задачи в этих подобластях. В случае решения оптимизационных задач такой подход обладают низкой эффективностью, поскольку при разделении области поиска только небольшая часть процессоров будет обладать подобластью с искомым глобальным минимумом, все же остальные процессоры будут работать в подобластях, в которых отсутствует глобальный минимум исходной задачи.

Плохо применим в данном случае и подход, связанный с распараллеливанием непосредственно алгоритма решения задачи, ибо в силу сформулированных предположений о классе рассматриваемых задач основной объем вычислений в процессе глобального поиска занимают расчеты, связанные с вычислением значений функционалов оптимизационной задачи. В этом плане, целесообразным представляется распараллеливание процедур расчета значений функционалов, однако такой подход не обладает общностью и требует проведения действий по распараллеливанию для каждой оптимизационной задачи в отдельности.

С учетом выполненного обсуждения, в качестве основного принципа построения параллельных методов многоэкстремальной оптимизации в диссертации будет применяться схема параллельных вычислений, в которой будет обеспечиваться возможность одновременного (параллельного) расчета значений функционалов оптимизационной задачи в нескольких разных точках области поиска (см., например, [3, 4]). Такой подход характеризуется *эффективностью* (распараллеливается именно та часть вычислительного процесса, в котором выполняется основной объем вычислений) и *общностью*, поскольку применим для всех вычислительно-трудоемких задач многоэкстремальной оптимизации.

4.2 Параллельные вычисления для систем с распределенной памятью

Новый подход для организации параллельных вычислений в задачах глобальной оптимизации состоит в использование для редукции размерности множества отображений

$$Y_L(x)=\{y^1(x),\ldots,\ y^L(x)\}$$

вместо применения единственной кривой Пеано $y(x)$ [3, 4]. Использование множества отображений $Y_L(x)$ приводит к формированию соответствующего множества одномерных многоэкстремальных задач

$$\min\{\ \varphi(y^l(x)):x\in[0,1]\ \},\ 1\le l\le L. \qquad (3)$$

Каждая задача из данного набора может решаться независимо, при этом любое вычисленное значение $z=\varphi(y^i(x'))$ функции $\varphi(y)$ в i-й задаче может интерпретироваться как вычисление значения $z=\varphi(y^s(x''))$ для любой другой s-й задачи без повторных трудоемких вычислений функции $\varphi(y)$. Подобное информационное единство позволяет решать исходную задачу (1) путем параллельного решения L задач вида (4) на наборе отрезков [0,1]. Каждая одномерная задача решается на отдельном процессоре. Результаты вычислений в точке x^k, полученные конкретным процессором для решаемой им задачи, интерпретируются как результаты вычислений во всех остальных задачах (в соответствующих точках $x^{k1},\ldots,x^{kL}$). При таком подходе вычисления в точке $x^k\in[0,1]$, осуществляемое в s-й задаче, состоит в последовательности действий:

1. Определить образ $y^k=y^s(x^k)$ при соответствии $y^s(x)$.
2. Проинформировать остальные процессоры о начале проведения испытания в точке y^k (*блокирование* точки y^k).
3. Вычислить значение оптимизируемой функции $z^k=\varphi(y^s(x^k))$.
4. Определить прообразы $x^{kl}\in[0,1]$, $1\le l\le L$, точки y^k, и интерпретировать значение функции $\varphi(y)$, вычисленное в точке $y^k\in D$, как одновременные вычисления в L точках

$$x^{k1},\ldots,\ x^{kL},$$

с одинаковыми результатами

$$\varphi(y^1(x^{k1}))=\ldots=\ \varphi(y^L(x^{kL}))=z^k.$$

5. Проинформировать остальные процессоры о результатах испытания в точке *yk*.

Каждый процессор при таком подходе должен свою копию программных средств, реализующих вычисление функций задачи, и решающее правило алгоритма. Для организации взаимодействия на каждом процессоре создается L очередей, в которые процессоры помещают информацию о выполненных итерациях (точки очередных итераций и значения оптимизируемой функции).

Рассмотренный подход позволяет использовать для решения задач глобальной оптимизации порядка нескольких тысяч процессоров – ограничением возможного количества используемых вычислительных элементов является трудоемкость информационного взаимодействия при организации обменов результатов итераций глобального поиска между процессорами.

4.3 Параллельные вычисления для систем с общей разделяемой памятью

Ранее рассмотренная схема применима и для систем с общей разделяемой памятью, тем более что, что в этом случае будут отсутствовать затраты на обмен результатов итераций глобального поиска между процессорами. Вместе с этим, для организации параллельных вычислений может быть применена и другая схема, при которой параллельно может решаться и отдельные задачи семейства (3). Для обоснования подхода можно отметить, что используемые в алгоритмах характеристики интервалов (см. выше общую схему методов глобального поиска) могут рассматриваться как некоторые меры важности интервалов на предмет содержания в интервалах искомого решения оптимизационной задачи. Следуя данному пониманию, для организации параллельных вычислений после выбора точки вычисления значения функции для первого процессора в точном соответствии с последовательным алгоритмом для второго процессора точку очередной итерации целесообразно выбирать из следующего по важности интервала (т.е. из интервала со следующей по порядку максимальной характеристикой) и т.д.

На данной основе может быть предложена общая схема *параллельных асинхронных характеристических методов*, определяемая следующим образом [5].

Выбор точек очередных итераций глобального поиска начиная с некоторого шага поиска $k_0 \geq 1$ включает выполнение следующего набора действий:

1) При отсутствии свободных процессоров алгоритм находится в состоянии ожидания. Если закончили проведение испытаний $1 \leq q = q(k) \leq p(k)$ процессоров, то устанавливается номер итерации $k = k + q$, и осуществляется переход к следующему правилу общей схемы.

2) Упорядочить поисковую информацию в порядке возрастания значений точек ранее выполненных итераций (новый порядок расположения данных в поисковой информации отражается при помощи нижнего индекса):

$$0 = x_0 < x_1 < \ldots < x_{\tau-1} < x_k = 1\,.$$

3) Вычислить для каждого интервала $(x_{i-1}, x_i), 1 < i \leq k$, величину $R(i)$, называемую далее *характеристикой* интервала,.

4) Упорядочить характеристики $R(i), 1 \leq i \leq \tau,$ в порядке убывания:

$$R(t_1) \geq R(t_2) \geq \ldots \geq R(t_{k-1}) \geq R(t_k) \qquad (4)$$

5) Выбрать в последовательности (4) q наибольших характеристик с номерами $t_j, 1 \leq j \leq q,$ и в интервалах, соответствующих этим характеристикам, выполнить испытания в точках

$$\bar{x}^j = d(t_j) \in (x_{t_j-1}, x_{t_j}),\ 1 \leq j \leq q.$$

Как можно заметить, рассмотренная общая схема практически полностью соответствует схеме характеристической представимости алгоритмов глобального поиска.

Отметим также, рассмотренная схема асинхронных параллельных вычислений полностью применима и для случая, когда вычислительные узлы с общей разделяемой памятью содержат ускорители вычислений вида современных графических процессоров. Отличие состоит лишь в

том, что определяемые точки итераций глобального поиска должны передаваться для использования ускорителям вычислений.

Предполагая наличие количество вычислительных элементов (ядер) на общей памяти порядка 100, использование параллельных алгоритмов в рамках рассмотренных схем параллельного глобального поиска, позволяет довести возможное число одновременно используемых вычислительных устройств суперкомпьютерных систем экзафлопсного уровня производительности до нескольких сотен тысяч.

Литература

1. International Exascale Software Project. – http://www.exascale.org
2. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. М.: Наука, 1978.
3. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht, 2000.
4. Стронгин Р.Г., Гергель В.П., Баркалов К.А. Параллельные методы решения задач глобальной оптимизации // Известия высших учебных заведений. Приборостроение. 2009. Т. 52. № 10. С. 25–33.
5. Стронгин Р.Г., Гергель В.П., Гришагин В.А., Баркалов К.А. Параллельные вычисления в задачах глобальной оптимизации. – М.: МГУ, 2012.

Исследование масштабируемости параллельных алгоритмов методом агентно-ориентированного моделирования*

Б.М. Глинский, А.С. Родионов, М.А. Марченко, Д.А. Караваев,
Д.И. Подкорытов, Д.В. Винс

Федеральное государственное бюджетное учреждение науки
Институт вычислительной математики и математической геофизики
Сибирского отделения Российской академии наук

В докладе исследуется возможность масштабирования алгоритмов распределенного статистического моделирования и сеточных методов на большое число (сотни тысяч и миллионы) вычислительных ядер предполагаемого экзафлопсного суперкомпьютера. Актуальность предмета исследования обосновывается оценкой вычислительной эффективности этих алгоритмов в свете ожидаемого появления к 2020 году суперкомпьютеров экзафлопсного уровня производительности. Рассматриваются особенности поведения вычислительных алгоритмов при их реализации на кластерах Сибирского суперкомпьютерного центра (ССКЦ) и оценка их масштабируемости на предполагаемых архитектурах MPP и гибридной, проведённая с помощью мультиагентной системы моделирования AGNES (AGent NEtwork Simulator).

1. Введение

Детальный анализ масштабируемости параллельных алгоритмов в сложных системах большой размерности, к которым, в частности, относятся будущие суперкомпьютеры экзафлопсного уровня производительности, невозможен применением одних лишь математических моделей. Повышение параллелизма является одной из основных тенденций развития суперЭВМ.

Так, например, недавний лидер мирового списка Top 500 Sequoia это суперкомпьютер семейства IBM BlueGene/Q построен на шестнадцатиядерных процессорах Power BQC, всего 1572864 вычислительных ядер, производительность на Linpack 16.32 PFLOPS. Для эффективного использования таких вычислительных ресурсов необходимо разрабатывать прикладные алгоритмы и программы с высокой степенью параллелизма. Проблемы прикладного программного обеспечения будут обостряться с переходом на суперЭВМ экзафлопной производительности и уже сейчас необходимо разрабатывать алгоритмы и программы с высокой степенью параллелизма и уметь оценивать их поведение на десятках и сотнях миллионов вычислительных ядер.

Сложный характер алгоритмов и стохастическая природа входящих потоков данных и задач делает необходимым проведение имитационного моделирования таких систем. С другой стороны, размерность детализированных моделей такова (сотни тысяч и миллионы одновременно присутствующих в модели объектов), что использование традиционных систем имитационного моделирования становится невозможным как для описания моделей, так и их исполнения.

Системы имитационного моделирования, в сравнении с универсальными языками программирования, дают ряд преимуществ. Они обеспечивают исследователя естественной средой для создания имитационных моделей и предоставляют такие возможности, как генерирование случайных чисел с заданным распределением вероятности, продвижение модельного времени, обработка списков текущих и будущих событий и др.; имеют встроенные механизмы представления параллельных процессов. Имитационные модели, созданные с помощью систем моделирования, как правило, проще модифицировать и использовать.

* *Контактная информация: 8(383)3306279 www.sscc.ru, www2.sscc.ru
Настоящая работа проводилась при финансовой поддержке грантов РФФИ 12-01-00034, 12-01-00727; МИП № 39 СО РАН, МИП № 47 СО РАН, МИП № 126 СО РАН, МИП № 130 СО РАН.

Отметим несколько важных моментов, связанных с исследованием масштабируемости алгоритмов:

- Исследование свойств масштабируемости параллельных алгоритмов является важной задачей при оценке эффективности их реализации, как на настоящих, так и будущих суперкомпьютерах пета- и экзафлопсного уровня.
- Данная проблема выходит за уровень технологических задач и требует научно-исследовательского подхода к ее решению.
- Вычислительные алгоритмы, как правило, являются более консервативными по сравнению с развитием средств вычислительной техники.
- Оценить поведение алгоритмов можно, путем реализации их на имитационной модели, содержащей тысячи и миллионы вычислительных ядер.
- При исследовании масштабируемости абсолютные значения оцениваемого времени исполнения программ не имеют значения, важно лишь относительное его изменение при наращивании ресурсов моделируемой вычислительной системы.
- Имитационная модель позволит выявить узкие места в алгоритмах, понять, как нужно модифицировать алгоритм, какие параметры необходимо настраивать при его масштабировании на большое количество ядер.

Задача моделирования масштабируемых алгоритмов не является новой, ею занимаются многие группы исследователей во всём мире. Из зарубежных выделим исследования, проводимые в США (Университет Урбана-Шампань, Иллинойс). Одним из основных проектов этого коллектива является проект BigSim (http://charm.cs.uiuc.edu/research/bigsim, руководитель проекта Kale Laxmikant). Проект направлен на создание имитационного окружения, позволяющего разработку, тестирование и настройку посредством моделирования ЭВМ будущих поколений, одновременно позволяя разработчикам ЭВМ улучшать их проектные решения с учётом специального набора приложений [1].

Из отечественных выделим исследования, проводимые в Институте системного программирования РАН (г. Москва) под руководством академика В.П. Иванникова [2,3]. Этим коллективом разработана модель параллельной программы, которая может эффективно интерпретироваться на инструментальном компьютере, обеспечивая возможность достаточно точного предсказания времени реального выполнения параллельной программы на заданном параллельном вычислительном комплексе. Модель разработана для параллельных программ с явным обменом сообщениями, написанных на языке Java с обращениями к библиотеке MPI, и включена в состав среды ParJava. Модель получается преобразованием дерева управления программы, которое для Java-программ может быть построено путем модификации абстрактного синтаксического дерева. Для моделирования коммуникационных функций используется модель LogGP, что позволяет учитывать специфику распределенной вычислительной системы. Предсказание времени счёта отдельных участков параллельной программы производится с учётом затрат, связанных с управлением MPI, т.е. производится корректировка модельных часов с учётом средней доли процессорного времени, которую занимает нить RTS (Run Time System). Таким образом, проект ParJava, с одной стороны, позволяет решать широкий круг задач по оценке эффективности исполнения параллельных программ на перспективных вычислительных системах, но, с другой стороны, привязан к конкретному языку программирования, что существенно сужает его возможности. Стоит отметить, что оба рассмотренных проекта не учитывают, по крайней мере, явно, вопросы отказоустойчивости при исполнении больших программ, в то время как использование в вычислениях одновременно десятков и сотен тысяч, а для отдельных задач и миллионов вычислительных ядер не может их не поставить.

В данной работе с использованием системы имитационного моделирования AGNES (AGent NEtwork Simulator), разработанной в ИВМиМГ СО РАН рассматриваются особенности масштабирования двух типов алгоритмов: распределенного статистического моделирования и численного моделирования 3D сейсмических полей при их отображении на архитектуры экзафлопсного класса.

Моделирование исполнения параллельных алгоритмов для анализа асштабируемости проводилось как на кластере с MPP-архитектурой с пиковой производительностью 30 TFlops, так и на гибридном кластере ССКЦ, который состоит из 40 вычислительных узлов HP SL390s G7. Каждый узел содержит: два 6-ядерных CPU Xeon X5670 (2.93GHz); 96 ГБ оперативной памяти;

три карты NVIDIA Tesla M2090. Каждая карта содержит GPU с 512 ядрами и 6 ГБ оперативной памяти. Суммарно гибридный кластер содержит 80 процессоров (480 ядер) CPU и 120 процессоров (61440 ядер) GPU. Пиковая производительность – 85 TFlops, на тесте Linpak – 38 TFlops.

2. Агентно-ориентированная система имитационного моделирования AGNES

Пакет AGNES (AGent NEtwork Simulator) базируется на Java Agent Development Framework (JADE) [4]. JADE – это мощный инструмент для создания мультиагентных систем на JAVA, и он состоит из 3-х частей: среда исполнения агентов; библиотека базовых классов, необходимых для разработки агентной системы; набор утилит, позволяющих наблюдать и администрировать МАС (мультиагентная система). Для моделирования больших вычислений важно, что JADE – это FIPA-совместимая, распределенная агентная платформа, которая может использовать один или несколько компьютеров (узлов сети), на каждом из которых должна работать только одна виртуальная JAVA машина.

AGNES использует преимущества, предоставляемые JADE, и расширяет мультиагентную систему до системы моделирования. AGNES состоит из двух типов агентов [5,6]:

- управляющие агенты (УА), которые создают среду моделирования;
- функциональные агенты (ФА), которые образуют модель, работающую в среде моделирования.

Основная задача функциональных агентов – имитация работы исследуемой системы. Основные задачи управляющих агентов: инициализация и запуск модели; сбор и хранение информации о ходе моделирования; синхронизация модельного времени; перераспределение нагрузки между вычислителями, участвующих в моделировании; взаимодействие с пользователем (вывод отчетов и возможность влиять на ход моделирования); обеспечение отказоустойчивости, восстановление модели; балансировка нагрузки.

Для обеспечения отказоустойчивости AGNES реализуется несколько механизмов: отсутствие централизованного хранения данных для восстановления; хранение информации располагается частями на разных агентах среды моделирования, и эта информация хранится с избытком, для гарантии её восстановления; динамическое изменение хранилищ информации во время работы среды моделирования, т.е. в ходе моделирования хранилище данных о конкретном агенте изменяется, и эта обязанность переходит от одного агента к другому.

Таким образом, основные принципы улучшения отказоустойчивости среды моделирования: децентрализации хранилищ и избыточность информации. Эффективность средств обеспечения отказоустойчивости проверена на экспериментах с физическими нарушениями аппаратной среды (локальной сети персональных ЭВМ) при реализации исполнения модели сенсорной сети.

Приложение AGNES – это распределенная мультиагентная сеть (МАС), называемая платформой. Платформа AGNES состоит из системы контейнеров, распределенных в сети. Обычно на каждом хосте находится по одному контейнеру (но при необходимости их может быть несколько). Агенты существуют внутри контейнеров.

Достоинства пакета AGNES: отказоустойчивость; сбалансированное распределение нагрузки; наличие проблемно-ориентированных библиотек агентов; возможность динамического изменения модели в ходе эксперимента.

Мультиагентный подход органично подходит для задачи имитации вычислений. В качестве атомарной, независимой частицы в модели вычислений выбран вычислительный узел и исполняемый на нем код алгоритма. Каждый функциональный агент эмулирует поведение вычислительного узла кластера, и программу вычислений, работающую на этом узле. Вычисления представляются в виде набора примитивных операций (вычисление на ядре; запись/чтение данных в память; парный обмен данными; синхронизация данных между вычислителями) и временных характеристик каждой операции.

3. Модель экзафлопсной суперЭВМ и отображение параллельной программы на ее архитектуру

Архитектура ЭВМ экзафлопсной производительности в настоящее время не определена. Есть некоторые соображения, однако какой она будет, вероятнее всего, это мы узнаем только ближе к 2018 году. Однако, в некоторых работах зарубежных и отечественных авторов высказаны предположения, что это будут гибридные архитектуры, например в работе [7] дано обоснование необходимости применения гибридных архитектур для достижения экзафлопсной производительности. В своей работе мы предполагаем экстенсивное развитие инструментального вычислительного кластера НКС-30Т+GPU ЦКП ССКЦ СО РАН (многократное увеличение количества существующих ядер, в составе этого кластера, как было указано выше, есть и однородные структуры и гибридные). Тем самым делается оценка производительности «снизу», поскольку естественно ожидать повышения характеристик ядер и интерконнекторов ЭВМ экзафлопсной производительности по сравнению с существующими.

Архитектура вычислительной системы как таковая явно в модели не присутствует, поскольку значение имеют лишь задержки на вычисления и обмены. Соответственно этому, задержки либо (как в случае представленных экспериментов) замеряются в ходе проведения реальных расчётов на конкретных ВС, либо задаются как функция от вложения графа задачи и графа размещения данных в моделируемую ВС. При этом функция вычисления задержки на передачу данных может быть достаточно сложна. Делается допущение, что данные передаются по кратчайшим путям. Возможные дополнительные задержки моделируются добавлением случайной составляющей.

Модель программы представляется взвешенным графом переходов между блоками программы с указанием параллельных ветвей. Временные задержки в блоках определяются на основе измерений, производимых в тестовых прогонах реальных программ на НКС-30Т+GPU. Прогоны реальных программ на конфигурациях с более чем 30 000 ядер позволяют надеяться на учёт в измеренных задержках эффектов от системной составляющей.

Далее рассмотрим применение системы AGNES для исследования масштабируемости распределенного статистического моделирования и численного моделирования распространения сейсмических полей в 3D неоднородных упругих средах.

4. Имитационное моделирование метода Монте-Карло.

С целью изучения возможности масштабирования распределенного статистического моделирования на большое число вычислительных ядер производилось имитационное моделирование работы экзафлопсного суперкомпьютера при решении задачи динамики разреженного газа по методу прямого статистического моделирования (ПСМ). Это типичная задача статистического моделирования, требующая всей вычислительной мощи многопроцессорного суперкомпьютера, т.е. требующие моделирования экстремально большого количества независимых реализаций [8]. К числу таких проблем относятся задачи моделирования с использованием ПСМ течений разреженного газа с учетом химических реакций, задачи переноса излучения и теории дисперсных систем.

Общая схема вычислений по методу Монте-Карло (см. рис.1):

Шаг 1: Подготовка к моделированию независимых реализаций на группах ядер.
Шаг 2: Моделирование реализаций, вычисление выборочных средних для группы.
Шаг 3: Сбор и осреднение данных.

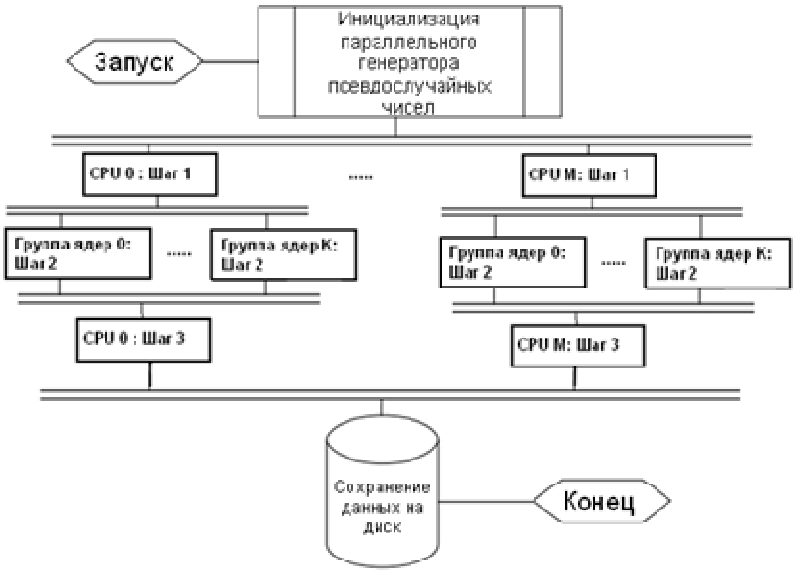


Рис. 1. Схема параллельных вычислений методов Монте-Карло.

Имитационное моделирование проводилось с использованием мультиагентной системы AGNES. Для имитации вычислений методов Монте-Карло созданы два класса функциональных агентов:

- *DataAgregator*: ядро-сборщик, собирает информацию о вычислениях, обрабатывает и агрегирует её. Возможно иерархическое построение сборщиков, которые на нижнем уровне обрабатывают данные непосредственно вычислителей, а затем передают их вышестоящему агенту *DataAgregator*. На вершине этой пирамиды всегда стоит одно главное ядро-сборщик, подготавливающее итоговые данные обо всех вычислениях и сохраняющее их на жесткий диск.
- *MonteCarlo*: агент, имитирующий расчет независимых реализаций методов Монте-Карло на нескольких вычислительных узлах, группа ядер-вычислителей. Каждый агент проводит независимые вычисления согласно схеме вычислений и взаимодействует только с соответствующим *DataAgregator*. Основными характеристиками агента являются временные и статистические свойства, оценки которых получены на основе реальных вычислений.

В результате работы модели собираются следующие отчеты:

- Набор времен, потраченных на каждую итерацию вычислений каждым агентом. Эти времена позволяют получить статистические характеристики протекающих в модели вычислений, для оценки правдоподобия модели.
- Информация о количестве итераций вычислений, совершенных каждым агентом *MonteCarlo*. При помощи данной статистики можно, например, отследить, как влияет количество вычислителей на скорость расчетов.
- Информация об интенсивности получения данных агентами *DataAgregator* от вычислителей, либо нижестоящих *DataAgregator*, в данном случае регистрируется количество полученных за равные промежутки времени пакетов.

Исходные данные для имитационного моделирования получены с использованием библиотеки PARMONC, предназначенной для использования на современных суперкомпьютерах тера- и петафлопсного уровня [9]. Область применения библиотеки: «большие» задачи статистического моделирования в естественных и гуманитарных науках (физика, химия, биология, медицина, экономика и финансы, социология и др.) Библиотека PARMONC установлена на кластерах Сибирского суперкомпьютерного центра (ЦКП ССКЦ СО РАН) и может использоваться на вычислительных системах с аналогичной архитектурой. При этом использование библиотеки

не привязано к каким-то определенным компиляторам языков C и FORTRAN или MPI. Инструкции по использованию библиотеки с примерами можно найти по ссылкам [10].

Как известно, теоретическое ускорение при распараллеливании для методов статистического моделирования практически "идеальное", что подтверждается численными расчетами при числе вычислительных ядер порядка нескольких тысяч [11]. Тем не менее, при числе ядер порядка сотен тысяч или нескольких миллионов вопросы организации счета требуют серьезного исследования, поскольку при этом возникают проблемы с большой загрузкой ядер-сборщиков, которые периодически собирают статистику с ядер-вычислителей. А именно, проведенное имитационное моделирование показало, что при большом числе используемых вычислительных ядер (больше 10000) реальное ускорение от распараллеливания существенно отличается от теоретического, что связано с большой загрузкой выделенных ядер-сборщиков, которые обрабатывают поступающие пакеты данных с ядер-вычислителей. При этом до 1000 ядер ускорение в модели совпадает с ускорением в реальных расчетах. С целью повышения эффективности распараллеливания исследовались различные варианты организации обмена данными между ядрами.

А именно, целесообразно осуществлять периодическую пересылку результатов промежуточного осреднения реализаций, независимо полученных на загруженных ядрах (ядрах-вычислителях), на выделенные ядра (ядра-сборщики), объединенные в многоуровневую структуру. Ядра-сборщики будут периодически получать переданные им данные и осреднять их, передавая затем результаты на ядро (с номером 0), соответствующее вершине многоуровневой структуры (см. рис. 2 и 3). Будем называть такое ядро *главным ядром-сборщиком*; в числе его задач – сохранение осредненных данных на диск. Рассчитанные на главном ядре-сборщике осредненные значения будут соответствовать выборке, полученной совокупно на всех ядрах-вычислителях. Распределенное статистическое моделирование на разных вычислительных ядрах-вычислителях производится в асинхронном режиме. Отправка и получение результатов статистического моделирования также осуществляется в асинхронном режиме [12].

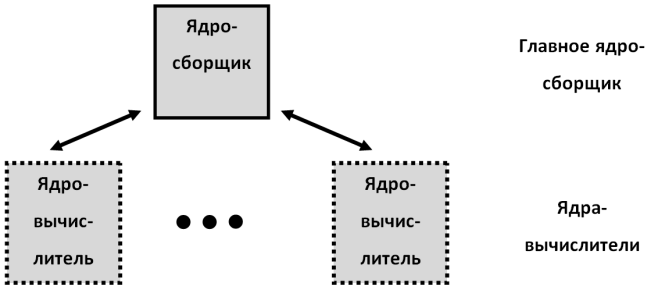


Рис. 2. Вариант организации связей между ядрами: одно главное ядро-сборщик.

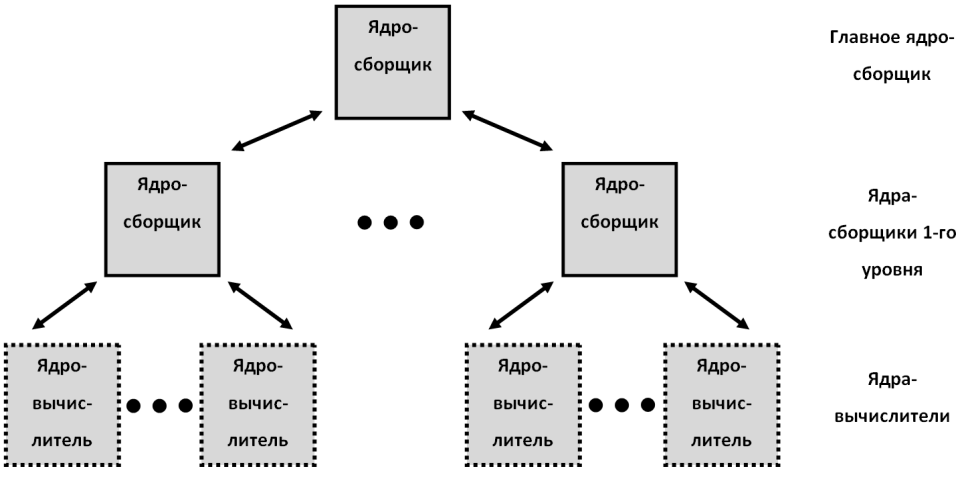


Рис. 3. Вариант организации связей между ядрами: один уровень промежуточных ядер-сборщиков.

На кластере НКС-30Т Сибирского суперкомпьютерного центра с использованием библиотеки PARMONC, был произведен ряд расчетов для общего числа ядер, примерно равного 1000. Реальные затраты машинного времени на независимое моделирование реализаций на ядрах-вычислителях и обмен данными (выборочными средними) с главным ядром-сборщиком были использованы для калибровки имитационной модели в AGNES. По результатам расчетов был сделан вывод, что требуемый уровень относительной статистической погрешности в 0.1% достигается при объеме выборки равном L = 240 000. Среднее время моделирования одной реализации - примерно 12 сек. Для ядер-вычислителей обмен данными с главным ядром-сборщиком происходил после каждой смоделированной на них реализации.

Отображение модели на вычислительный кластер выглядит следующим образом: на каждом сервере запускается JVM и отдельный контейнер JADE. На каждом контейнере запускаются агенты имитирующие расчет методов Монте-Карло. Каждый агент *MonteCarlo* содержит внутри себя группу циклических поведений, каждое из которых имитирует расчет независимых реализаций на одном вычислительном узле. Подобное представление позволяет моделировать расчеты по методу Монте-Карло на 10^7 вычислительных узлах с использованием 10^4 агентов (см. рис. 4).

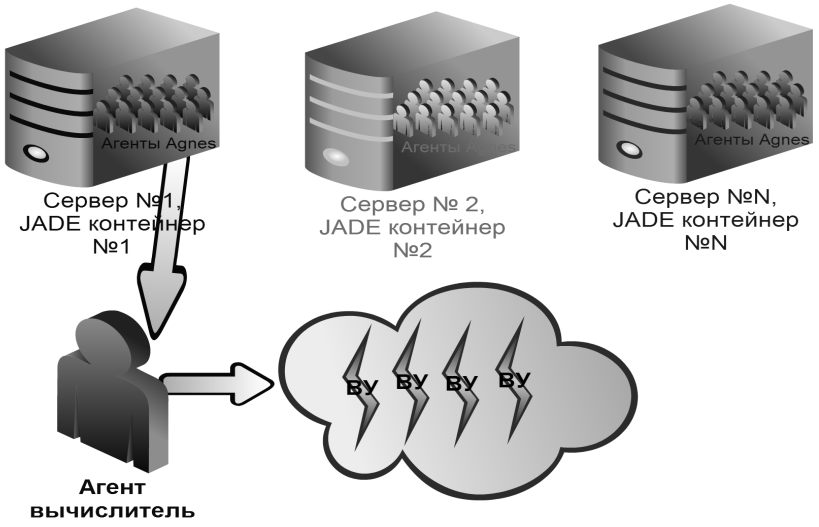


Рис. 4. Схема отображения модели на вычислительный кластер.

5. Результаты расчетов на имитационной модели для методов Монте-Карло

При имитационном моделировании расчетов по методу Монте-Карло за основу взята MPP-архитектура кластера НКС-30Т. При этом исследовалась величина относительного ускорения от распараллеливания при расчётах на M ядрах, определенная следующим образом:

$$S_L(M) = T_L(M_{min})/T_L(M),$$

где $T_L(M)$ – машинное время на главном ядре-сборщике, затраченное на моделирование и сохранение выборочных средних для задачи, в которой моделируется L реализаций случайной оценки; M_{min} – наименьшее число ядер, использованных при расчётах.

В первой серии экспериментов рассматривались два варианта организации обмена данными между ядрами-вычислителями и главным ядром-сборщиком:

- с использованием одного уровня промежуточных ядер-сборщиков (см. рис. 3).
- без использования промежуточных ядер-сборщиков (см. рис. 2);

В первом варианте ядра-вычислители были поделены на M_1 равных частей (M_1 = 10, 20, 100), для каждой из которых данные с ядер-вычислителей всегда отправлялись на «свое» ядро-сборщик. В свою очередь, M_1 ядер-сборщиков отправляли данные на главное ядро-сборщик. Во втором варианте для определенности будем считать, что параметр $M_1 = 0$.

На рис.5 приведена зависимость относительного ускорения $S_L(M)$ от общего числа моделируемых ядер M. Моделирование проводилось до $M = 5*10^5$ ядер, но для показательности на рисунке приведены данные до $M = 10^5$. На рисунке ясно видна закономерность: увеличение числа ядер-сборщиков приводит к увеличению относительного ускорения.

В этой связи интересно исследовать вопрос об оптимальном (в смысле максимального значения относительного ускорения) числе ядер-сборщиков при фиксированном общем числе моделируемых ядер.

Во второй серии экспериментов при фиксированном числе моделируемых ядер $M = 10^6$ варьировалось число ядер-сборщиков M_1, а также соответствующее число ядер-вычислителей в каждой группе, связанной со «своим» ядром-сборщиком. На рис. 6 приведен график зависимости относительного ускорения S_L от числа ядер-сборщиков M_1. Из рисунка видно, что максимальная величина относительного ускорения достигается при M_1, приближенно равном 1000. Это говорит о том, что при меньшем значении M_1 ядра-сборщики перегружены обработкой данных, поступающих от ядер-вычислителей, а при большем числе – перегружено главное ядро-сборщик, занятое обработкой поступающих данных от ядер-сборщиков.

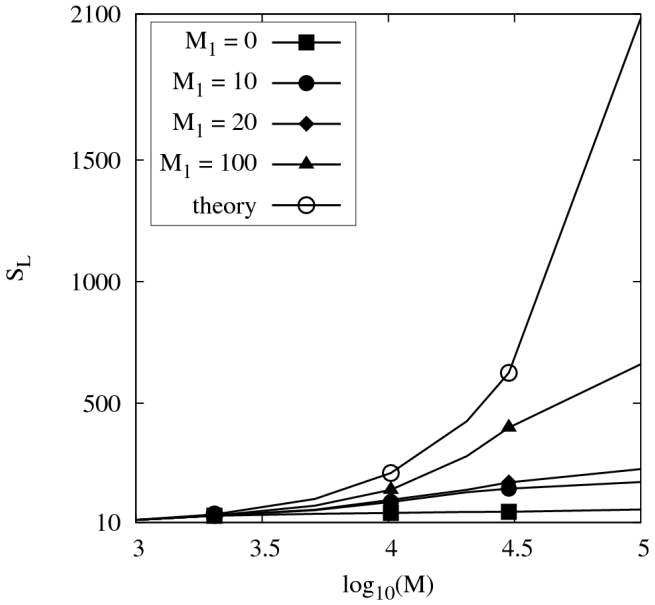


Рис. 5. Зависимость относительного ускорения S_L от общего числа моделируемых ядер M при разном числе ядер-сборщиков M_1 (горизонтальная ось – в логарифмическом масштабе)

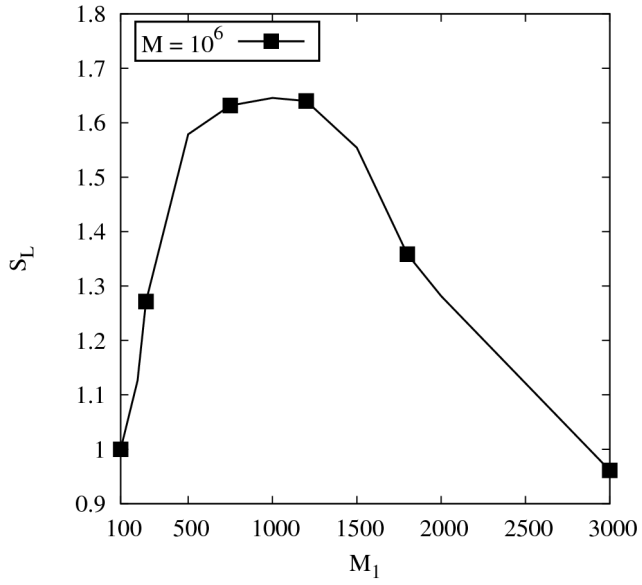


Рис. 6. Зависимость относительного ускорения S_L от числа ядер-сборщиков M_1 при общем числе моделируемых ядер M=10^6 .

6. Численное моделирование 3D сейсмических полей

Аналогичные исследования были проведены и для другого класса алгоритмов, основанного на применении разностного метода. В работе рассмотрен алгоритм численного моделирования 3D сейсмических полей в изотропной неоднородной упругой среде [13]. Такого вида задачи характеризуются большим объемом вычислений, поскольку область моделирования представляется достаточно подробно для проведения 3D моделирования.

Предполагается, что вычислительные модули кластера состоят из нескольких CPU и GPU. Поэтому разработана программа на основе масштабируемого параллельного алгоритма при использовании комбинации технологий программирования CUDA и MPI. Для проведения расчетов различных 3D моделей была рассмотрена следующая организация параллельного алгоритма и программы: 3D область моделирования разделяется на слои, каждый слой рассчитывается независимо на выделенном GPU, а обмены данными между соседними GPU проводятся посредством MPI. При этом вычисления для слоя производятся посредством CUDA в 2D. Способ декомпозиции расчетной области для организации параллельных вычислений представлен на рис 7.

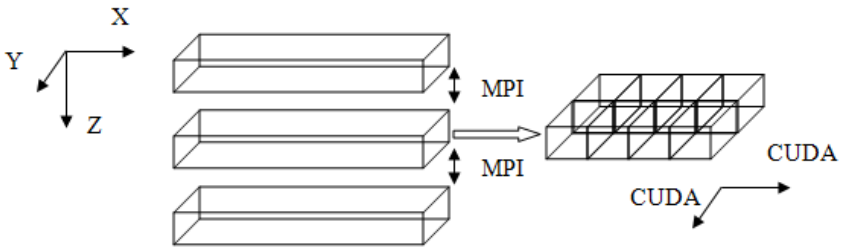


Рис. 7. Схема декомпозиции расчетной области.

7. Результаты имитационного моделирования сеточного метода

Для исследования реализации алгоритма численного моделирования на предполагаемую модель экзафлопсного компьютера выбран следующий критерий масштабируемости – время счета алгоритма меняется незначительно при следующих допущениях: размер 3D модели увеличивается пропорционально количеству вычислительных узлов; каждый вычислительный узел совершает одно и то же количество итераций для своей подобласти. Исследование проводилось с использованием имитационного моделирования работы экзафлопсного суперкомпьютера в вышеупомянутой системе AGNES. Предполагается гибридная архитектура суперЭВМ. Для проверки адекватности результатов имитационного моделирования проводилось их сравнение с результатами работы данного алгоритма на гибридном кластере НКС-30Т+GPU ССКЦ.

Для имитации сеточных методов реализован класс функциональных агентов Grid — узел-вычислитель, имитирующий расчет сеточных методов на одном вычислителе. Моделируются вычисления, когда область исследование режется вдоль одной оси, и полученные области загружаются на вычислители. Таким образом, получается, что у каждого вычислителя есть пересечение по данным максимум с 2-ми вычислителями («крайние» вычислители обмениваются только с одним соседом). Каждый вычислитель, на первом шаге рассчитывает свои граничные области, затем асинхронно передает насчитанные результаты соседям. Расчет внутренних областей идет на втором шаге, получив данные от соседей и просчитав изменение свой области, агент переходит к шагу один.

Общие результаты изменения времени счета в зависимости от количества доступных ядер GPU (при пропорциональном увеличении размера 3D модели) в логарифмическом масштабе приведены на рис 8. Показано хорошее соответствие экспериментальных и модельных результатов на начальном участке кривой (до 30720 ядер). При значительном увеличении количества вычислительных узлов с пропорциональным увеличением размера 3D модели время счета увеличивается, но несущественно (при росте числа узлов от 7680 до 1024000 время увеличилось на 17,5%).

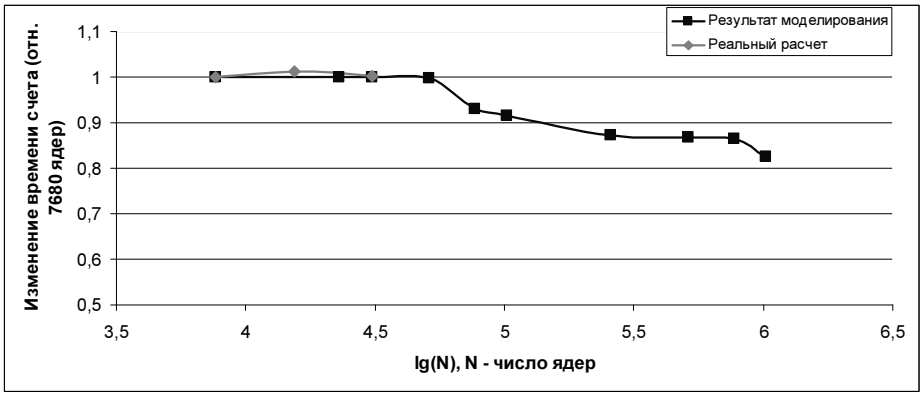


Рис. 8. Изменение времени расчета алгоритма численного моделирования в зависимости от числа вычислительных ядер (горизонтальная ось – в логарифмическом масштабе).

В заключение данного раздела приведем оценки необходимого количества ядер для реальных расчетов и соответственно для имитационной модели. В реальном расчете понадобилось 30720 ядер, а для имитационной модели только 12 вычислительных ядер! А при моделировании работы данного алгоритма с использованием до 1,5 млн. ядер в системе AGNES нам понадобилось только 144 ядра!

8. Заключение

Проведенное имитационное моделирование показало возможность масштабирования алгоритмов на большое число (сотни тысяч и даже миллионы) вычислительных ядер предполагаемого экзафлопсного суперкомпьютера, а также возможность исследования поведения алгоритмов при таком большом масштабировании.

Следует отметить, что эксперименты показали нецелесообразность, а в некоторых случаях и невозможность использования соответствия «агент-ядро», что связано с естественными ограничениями, вызванными затратами на управление взаимодействием агентов в системе JADE. Однако вполне возможно моделирование одним агентом поведения множества вычислительных ядер. В этом случае моделирование, по крайней мере, для рассматриваемых алгоритмов, осуществляется достаточно эффективно без потери точности. Максимальное количество моделируемых ядер достигало 10^6 , что даёт обоснованную надежду на возможность практического применения системы AGNES для моделирования исполнения на супер ЭВМ параллельных программ и других классов.

Таким образом, в настоящее время для оценки масштабируемости вычислительного алгоритма на гибридном кластере можно рекомендовать следующее: составить схему выполнения программы; прогнать параллельную программу на небольшом количестве ядер от сотен до нескольких тысяч ядер; ввести в имитационную модель задержки, отображающие время счета на вычислительных ядрах, полученные из реальных расчетов; протестировать правильность расчета на имитационной модели путем сравнения на начальном участке реального и модельного расчетов; исследовать поведение алгоритма при большом количестве ядер (от сотен тысяч до миллионов вычислительных ядер); провести, при необходимости, коррекцию вычислительной схемы, реализующей данный алгоритм.

Система AGNES установлена в ЦКП ССКЦ ИВМиМГ СО РАН и доступна по ссылке http://www2.sscc.ru/PPP/Mat-Libr/agnes.htm.

Литература

1. Peter Kogge(Ed.). ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, DARPA report, September 28, 2008. Электронный адрес http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.
2. В.П. Иванников, С.С. Гайсарян, А.И. Аветисян, В.А. Падарян. Оценка динамических характеристик параллельной программы на модели // Программирование, №4, 2006., С. 21-37.
3. В.П. Иванников, А.И. Аветисян, С.С. Гайсарян, В.А. Падарян. Прогнозирование производительности MPI-программ на основе моделей // Журнал "Автоматика и телемеханика", 2007, №5, С. 8-17.
4. F. L. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE // Wiley, 2007.
5. D. Podkorytov, A. Rodionov, H. Choo, Agent-based simulation system AGNES (AGent NEtwork Simulator) for networks modeling // Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC'12, 2012.
6. Д.И. Подкорытов. Агентно-ориентированная среда моделирования сетевых систем AGNES // Ползуновский вестник, 2012. № 2/1, с. 93-106.
7. Эксафлопсные суперЭВМ, контуры архитектуры. Труды конференции PACO-2012 [Электронный ресурс]. - Режим доступа: http://paco2012.ipu.ru/procdngs/B101.pdf.
8. М.А. Марченко, Г.А. Михайлов. Распределенные вычисления по методу Монте-Карло // Автоматика и телемеханика. 2007. Вып. 5. С. 157-170.
9. M.A. Marchenko, PARMONC - A Software Library for Massively Parallel Stochastic Simulation // LNCS. 2011. V. 6873. P. 302-315.

10. Страница библиотеки PARMONC на сайте ССКЦ КП СО РАН [Электронный ресурс]. - Режим доступа: http://www2.sscc.ru/SORAN-INTEL/paper/2011/parmonc.htm.

11. Boris Glinsky, Alexei Rodionov, Mikhail Marchenko, Dmitry Podkorytov, Dmitry Weins. Scaling the Distributed Stochastic Simulation to Exaflop Supercomputers // Proceedings of 2012 IEEE 14th International Conference on High Performance Computing and Communications , p. 1131-1136.

12. Б.М. Глинский, А.С. Родионов, М.А. Марченко, Д.И. Подкорытов, Д.В. Винс. Агентно-ориентированный подход к имитационному моделированию суперЭВМ экзафлопсной производительности в приложении к распределенному статистическому моделированию // Вестник ЮУрГУ, 2012. № 18 (277), Вып. 12., с. 93-106.

13. Б.М. Глинский, Д.А. Караваев, В.В. Ковалевский, В.Н. Мартынов Численное моделирование и экспериментальные исследования грязевого вулкана "Гора Карабетова" вибросейсмическими методами. //Вычислительные методы и программирование. М.: Изд-во Моск. Гос. ун-та, 2010, Том 11, №1, С. 99-108.

Вычислительные ресурсы УрО РАН. Состояние и перспективы*

М.Л. Гольдштейн[1], А.В. Созыкин[1,2], Г.Ф. Масич[3], А.Г. Масич[3]

Институт математики и механики УрО РАН[1], Уральский федеральный университет[2], Институт механики сплошных сред УрО РАН[3]

Рассматриваются вопросы создания киберинфраструктуры УрО РАН - распределенных вычислительных ресурсов, информационных систем и устройств хранения данных, объединенных высокоскоростными каналами связи. Киберинфраструктура служит основой для проведения научных исследований на мировом уровне. Описываются вычислительные ресурсы суперкомпьютерного центра, подходы к построению высокоскоростной магистральной сети УрО РАН на основе DWDW технологии и особенности построения вычислительной облачной платформы УрО РАН.

1. Введение

Решение фундаментальных и прикладных ключевых задач, обусловливающих научно-технический прогресс в экономике и бизнесе и ведущих к получению новых знаний и информации, требует новых всё более производительных вычислительных систем, скоростных сетей передачи данных и распределенных инфраструктур коллективного пользования. Обеспечение научно-образовательного сообщества Уральского отделения РАН вычислительным инструментарием мирового уровня, отвечающим актуальным потребностям ученых и соответствующим приоритетным направлениям фундаментальных и прикладных исследований – одна из целей стратегии развития Уральского отделения РАН до 2025г [1].

В качестве ключевых этапов развития вычислительной инфраструктуры УрО РАН на сегодняшний день можно выделить:

- развитие суперкомпьютерного вычислительного центра (СКЦ) ИММ УрО РАН до мирового уровня (уровень Т1, вхождение в списки TOP500, Graph500, Green500) [2],
- построение высокоскоростной магистрали, объединяющей вычислительные ресурсы и системы хранения данных Научных центров УрО РАН в городах Архангельск, Сыктывкар, Ижевск, Пермь и Екатеринбург [3],
- интеграция грид-среды УрО РАН с российской грид-средой (проект Министерства связи и массовых коммуникаций России),
- создание распределенной среды высокопроизводительных вычислений [2],
- развитие облачной вычислительной платформы и создание пакета сервисов для пользователей СКЦ [4].

Работы по данным направлениям деятельности в УрО РАН ведутся в Институте математики и механики (ИММ) УрО РАН и Институте механики сплошных сред (ИМСС) УрО РАН.

2. Вычислительные ресурсы

Главный вычислительный ресурс СКЦ - суперкомпьютер (СК) “УРАН” (5 место в рейтинге TOP50, 17-ая редакция от 18.09.2012г.) имеет гибридную архитектуру:

- графические процессоры GPU: Nvidia Tesla M2090 (80 шт.) и M2050 (160 шт.)
- центральные процессоры: Intel Xeon E5450 3 ГГц (416 шт.), Intel Xeon X5675 3,06 ГГц (60 шт.)
- сети: Infiniband DDR (коммуникационная), GigabitEthernet (ввода-вывода и управления)

* Работа поддержана грантами УрО РАН № № 12-П-1-2012, 12-П-1-1029, 12-С-1-1012 и РФФИ № 11-07-96001-р_урал_а

- системное ПО: Linux, MPI (OpenMPI и MPICH), компиляторы gcc и Intel, система запуска задач SLURM

Прикладное ПО: Matlab, Ansys.

Инженерная инфраструктура:

- кондиционеры Liebert CR-V CR035RA – 4 шт.,
- кондиционер Liebert 50 -1шт.
- ИБП APC Smart-UPS VT 40kVA - 4 шт.
- ИБП APC Symmetra 16kVA – 7 шт.

Структурная схема СК приведена на рис. 1.

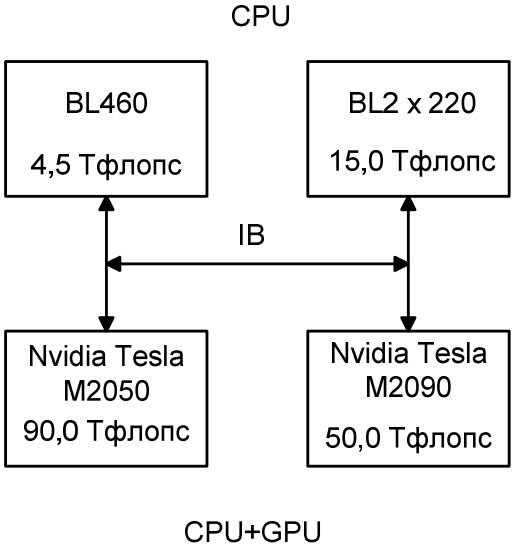


Рис. 1. Структурная схема суперкомпьютера “УРАН” (2012г.)

Развитие СК “УРАН” предполагает дальнейшее наращивание его производительности до 240 Тфлопс (пиковая) и объема дисковой памяти до 190 Тбайт (полная емкость). В качестве элементной базы расширения СК выбраны узлы на основе CPU Intel Xeon X5675 3,06 ГГц + GPU Nvidia Tesla M2090 и дисковые массивы на основе 2-х серверов хранения Supermicro в конфигурации: процессор CPU Intel Socket 1366 Xeon E5607 2.26Ghz tray и 48 жестких дисков HDD Seagate SATA3 3Tb. Структурная схема очередной модификации СК “УРАН” представлена на рис. 2.

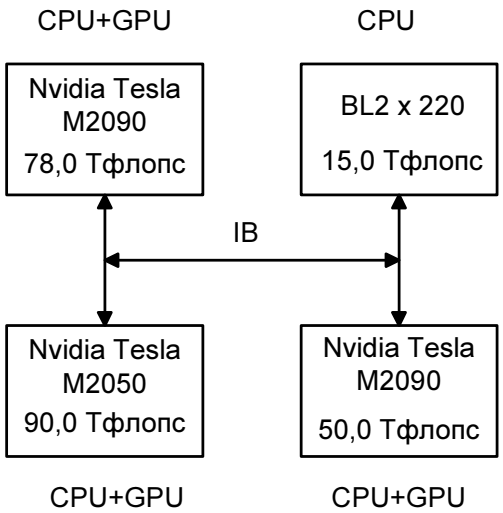


Рис. 2. Структурная схема очередной модификации СК “УРАН” (реализация январь-февраль 2013г.)

Динамика роста ключевых параметров СК “УРАН” приведена на рис. 3.

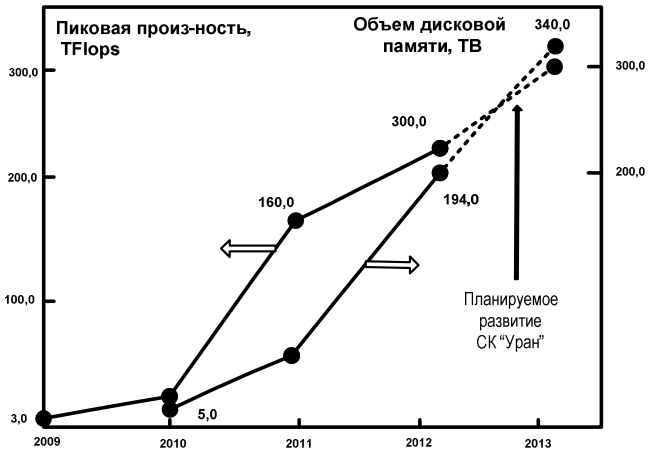


Рис. 3. Динамика роста ключевых параметров СК "УРАН" (2013г.)

Развитие грид-инфраструктуры УрО РАН предполагает создание на первом этапе ВЦ уровня Т2 в ИМСС. Для реализации этого проекта вычислительное поле производительностью 4,5 Тфлопс на модулях BL-460 (см. рис. 1) передаются в ИМСС вместе с дисковой системой хранения на основе сервера хранения Supermicro в конфигурации: процессор CPU Intel Socket 1366 Xeon E5607 2.26Ghz tray и 12 жестких дисков HDD Seagate SATA3 3Tb. Выбор ИМСС обусловлен наличием высокоскоростного (2 х 10Гбит/с + 8 х 1Гбит/с) канала связи между гг. Екатеринбург-Пермь с одной стороны и наличием экспериментальных установок (например, PIV), генерирующих большие потоки данных.

Структурная схема грид УрО РАН (первый этап) представлена на **рис.4.**

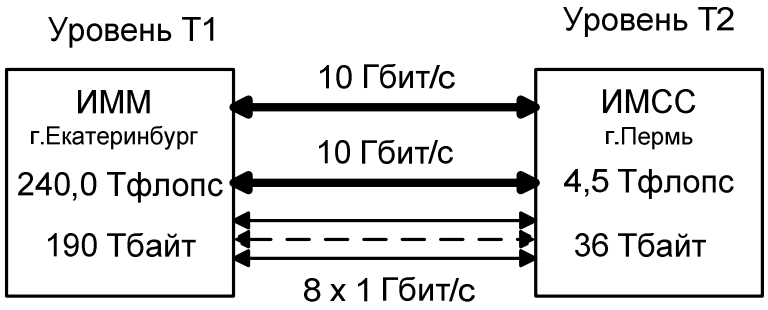


Рис.4. Структурная схема грид УрО РАН (первый этап - 2013г.)

Дальнейшая перспектива развития СКЦ заложена в ряде долгосрочных планов УрО РАН и вкратце выглядит таким образом (**табл. 1**).

Таблица 1. Планы развития вычислительных ресурсов УрО РАН на период 2013-2020гг.

Год	2013	2014	2015	2016	2017	2018	2019	2020
R(peak), в Pflops	0,35- 0,5	0,6-0,8	1,0-1,3	1,5-1,8	2,0-2,3	2,5-3,0	3,5	4,0
Объем дисковой памяти, в PB	0,35- 0,5	0,6-0,8	1,0-1,3	1,5-1,8	2,0-2,3	2,5-3,0	3,5	4,0

3. Распределенная система хранения

Современные экспериментальные установки, которые вместе с СК и магистральными каналами связи формируют киберинфопространсто, в результате проведения научных исследо-

ваний создают большие объемы данных. Эти данные нужно хранить и быстро передавать на СК и вычислительные кластеры для обработки. Для этих целей в УрО РАН создается распределенная система хранения (РСХ). В качестве основы для создания такой системы выбрано программное обеспечение dCache [5], которое активно используется для создания РСХ в крупных международных проектах, таких как Worldwide LHC Computing Grid, European Grid Infrastructure, NorduGrid и др. Отличительной особенностью dCache является поддержка не только специфичных для грид протоколов доступов к данным (SRM, dCap, xRoot), но и открытых протоколов (NFS, HTTP). В результате РСХ на основе dChache можно использовать как в составе грид, так и подключать к обычным вычислительным кластерам и персональным компьютерам.

РСХ УрО РАН строится из отдельных серверов хранения, которые представляют собой серверы стандартной архитектуры под управлением Linux с большим количеством дисков (12-24 шт.). Объединение серверов хранения в единую систему выполняется с помощью dCache. На первом этапе создания РСХ серверы хранения устанавливаются в ИММ УрО РАН и ИМСС УрО РАН (рис. 5.), в дальнейшем планируется установка серверов хранения в других регионах присутствия УрО РАН.

Рис. 5. Схема распределенной системы хранения данных УрО РАН (первый этап)

На первом этапе создания к РСХ подключаются суперкомпьютер «УРАН» и вычислительный кластер в ИМСС УрО РАН по протоколу NFS. Также апробируется подключение к РСХ экспериментальных установок в ИМСС УрО РАН. В дальнейшем, при подключении СКЦ ИММ УрО РАН к грид-проектам, планируется использование грид-протоколов РСХ.

4. Магистральные каналы связи

Ключевой вопрос построения скоростных научно-образовательных сетей – создание собственных или аренда существующих каналов связи. Выполненные в ИМСС УрО РАН исследования позволили найти экономически эффективные решения построения собственной скоростной оптической магистрали «Архангельск–Сыктывкар–Ижевск–Пермь–Екатеринбург», названной «Инициатива GIGA UrB RAS» [3]. Отличительная особенность этих решений - преодоление сложившейся в России негативной практики аренды дорогостоящих каналов связи путем использования темных волокон (dark fiber) в магистралях национальных операторов связи и использовании собственного оборудования спектрального уплотнения каналов. Цель – достижение стратегической обеспеченности научно-образовательного сообщества территории в коммуникационном сервисе.

Рис. 6. Инициатива GIGA UrB RAS

4.1. Этапы становления проекта

2008 год – в ИМСС УрО РАН разработан эскизный проект «Инициатива GIGA UrB RAS» (рис. 5) [3].

2009 год – в работе [6] сформулирована LambdaGrid парадигма распределенных вычислений, а в УрО РАН сформирован проект ГИГА, включающий помимо создания скоростной оптической сети, развитие суперкомпьютерных ресурсов, хранилищ данных и систем визуализации.

2010 год – на ИМСС УрО РАН возложено исполнение проекта ГИГА в части создания скоростной научно-образовательной оптической магистрали «Архангельск–Екатеринбург» [7]. В этом же году введен в эксплуатацию участок магистрали «Пермь–Екатеринбург» на скорости 1 Гбит/с по технологии Ethernet на двух «темных» оптических волокнах.

2012 год – выполнена структурная оптимизация DWDM тракта в рамках выделенного бюджета и проведена пуско-наладка DWDM системы 20 Гбит/с (2 λ канала по 10 Гбит/с) на участке «Пермь-Екатеринбург», с возможностью масштабирования до 16 лямбда каналов по 10-40Гбит/с) [8].

4.2. DWDM тракт Пермь-Екатеринбург

Используется DWDM оборудование компании ECI-Telecom [9]: платформы XDM-2000 на оконечных узлах и XDM-40 на промежуточных узлах. DWDM мультиплексоры обеспечивают возможность наращивания до 16 λ-каналов любой емкости каждый (10/40 Гбит/с). Комплектация установленного оборудования обеспечивает передачу двух λ-каналов по 10 Гбит/с в каждом со следующими характеристиками:

- Первый λ-канал 10 Гбит/с соединяет по схеме точка-точка коммутаторы CESR AS9215 с 24-мя портами GE и 4-мя портами 10GE с соответствующими «цветными» транспондерами для сопряжения с DWDM оборудованием.
- Структура второго λ -канала 10 Гбит/с: гарантированная и независимая передача 8x1GE каналов с терминацией в клиентские порты с применением соответствующих SFP модулей.

Спецификация промежуточных узлов обеспечивает только оптическое усиление сигнала, однако закладываемое DWDM оборудование предполагает возможность установки ROADM мультиплексоров для выделения и маршрутизации λ -каналов не только на оконечных, но и на промежуточных узлах. Система мониторинга и управления магистральной ВОЛС (Sun Fire V245 и программное обеспечение LightSoft) располагается в основном центре управления сетью в ИМСС УрО РАН (Пермь).

Настраиваемый коммуникационный сервис для нужд конкретных проектов и инициатив предоставляет следующие классы услуг передачи данных:

- λ услуга – предоставление в пользование фиксированного диапазона длин волн ("лямбда") на участке DWDM магистрали;
- FrameNet услуга – представление двухточечного или многоточечного Ethernet канала связи между Ethernet портами DWDM магистрали;
- PacketNet услуга – присоединение к Public Интернет;

λ услуга предусматривает, что оконечные точки канала находятся в двух различных узлах DWDM магистрали. Оконечными точками канала являются интерфейсные платы узлов DWDM магистрали. Эта услуга может предполагать установку лямбда каналообразующего оборудования в DWDM мультиплексоре. λ услуга обеспечивает точка-точка соединение на физическом уровне эталонной модели взаимодействия открытых систем (L1 OSI RM).

FrameNet услуга предполагает использование доступных посредством гигабит Ethernet GE (LX или на ZX) интерфейсов, таких как 1-10 GE портов оконечного оборудования магистрали. Эта услуга обеспечивает точка-точка и многоточечную доставку на Ethernet (канальном) уровне (L2 OSI RM). Доступны два базовых типа услуг FrameNet: (1) специализированная Ethernet услуга, гарантирующая заказанную пропускную способность; (2) неспециализированная Ethernet услуга, не гарантирующая постоянную пропускную способность.

PacketNet услуга предоставляет базовую услугу IP маршрутизации для доступа к public Интернет, обеспечивая соединения на сетевом уровне (L3 OSI RM). Прием анонсов IP-сетей, как и анонсирование IP-сетей, выполняется согласно политикам маршрутизации, отраженным в RIPE NCC.

Предоставляемый магистралью λ сервис позволяет создавать принципиально новый инструментарий для решения задач, основанный на зарождающихся LambdaGrid парадигмах распределенных вычислений и использующих параллелизм гарантированных и независимых каналов связи.

5. Вычислительная облачная платформа УрО РАН

Вычислительная облачная платформа (ВОП) УрО РАН создается для обеспечения гибкости при проведении вычислений. Пользователям иногда требуются оборудование или программное обеспечение в конфигурации, отличной от той, что предоставляется суперкомпьютерами. Например, пользователям может быть нужна операционная система Windows (а суперкомпьютеры работают под Linux), или набор выделенных серверов (на суперкомпьютере ресурсы предоставляются системой запуска задач во временное пользование).

Важным направлением использования ВОП является установка пакетов прикладных программ, интегрированных с суперкомпьютерами. Пакеты прикладных программ устанавливаются на виртуальные машины ВОП и настраиваются таким образом, чтобы задачи можно было запускать на счет на суперкомпьютерах прямо из графического интерфейса пакетов. Схема ВОП показана на рис. 7.

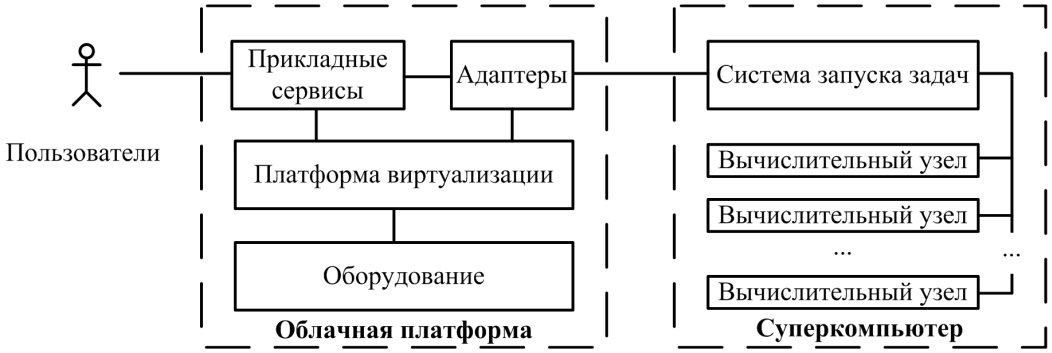


Рис. 7. Структура вычислительной облачной платформы УрО РАН

В настоящее время ВОП используется для следующих проектов:

- Виртуальная Web-лаборатория «Параллельное программирование в Matlab». На ВОП установлен Matlab, интегрированный с кластером. Создается Web-интерфейс, который будет служить единой точкой доступа к Matlab в облачной конфигурации, учебному курсу по параллельному программированию в Matlab и практическим руководством по применению Matlab на СК «УРАН».
- Обработка больших объемов изображений на параллельных вычислительных системах [10]. Для проекта в облачной платформе выделено четыре сервера, на которые установлен Hadoop. С помощью Hadoop выполняется автоматизация распараллеливания обработки изображений, а распределенная файловая система HDFS (входит в состав Hadoop) позволяет реализовать подход перемещения вычислений к данным.
- Разработка морфологического анализатора русского языка [11]. Для проекта в ВОП создан набор виртуальных машин, работающих в режиме распределенных вычислений без скоростного интерконнекта. Морфологический анализатор реализован в виде Web-сервиса.

Первая очередь реализации ВОП построена на базе четырех серверов Fujitsu-Siemens RX330 в каждом по 2 процессора AMD Opteron 2220, 8 ГБ памяти, жесткий диск 250 ГБ, сеть Gigabit Ethernet. В качестве системы хранения используется EMC Celerra NS-480. Платформа виртуализации создана на основе бесплатно распространяемого программного обеспечения oVirt (Open Source Vitrualization Manager [12]).

Важным аспектом реализации ВОП является то, что для ее создания используются серверы, ранее входившие в состав вычислительных кластеров, но производительность которых, по современным меркам, делает их дальнейшее использование для проведения высокопроизводительных вычислений нерациональным. Применение этих серверов для создания ВОП позволило продлить срок эксплуатации оборудования.

6. Заключение

Заложены основы построения киберинфраструктуры Уральского отделения РАН, объединяющей по скоростной оптической магистрали вычислительные ресурсы и системы хранения данных научных центров отделения на территории от Архангельска до Екатеринбурга. На основе суперкомпьютерного центра ИММ УрО РАН сформулирована трехуровневая иерархия распределенных по научным центрам вычислительных ресурсов. Описана облачная вычислительная платформа УрО РАН, в рамках которой пользователям предоставляются пакеты прикладных программ, интегрированные с суперкомпьютерами.

Литература

1. Стратегия развития Уральского отделения Российской академии наук до 2025г. Екатеринбург: УрО РАН, 2010. 236 с.

2. Гольдштейн М.Л., Созыкин А.В. Концептуальная модель и прототип ГРИД УрО РАН // Системы управления и информационные технологии, №3.1(49), 2012. – С. 132-136.

3. Масич А.Г., Масич Г.Ф. Инициатива GIGA UrB RAS // Вычислительные технологии. 2008. Т. 13. Вестник КазНУ им. Аль-Фараби. Серия математика, механика, информатика 2008. №3(58). Совместный выпуск. - Ч.II. - С.413-418.

4. Созыкин А.В., Гольдштейн М.Л. Модель взаимодействия прикладных облачных сервисов и суперкомпьютеров // Вестн. Пермского университета. Сер. Математика. Механика. Информатика. 2012. Вып. 3 (11). С. 65 - 69.

5. Millar P. dCache, agile adoption of storage technology // International Conference on Computing in High Energy and Nuclear Physics. New York. 2012.

6. Масич А.Г., Масич Г.Ф. От «Инициативы GIGA UrB RAS к Киберинфраструктуре УрО РАН. // Вестник Пермского научного центра. – Пермь: Изд-во ПНЦ УрО РАН, 2009. С. 41-56.

7. Матвеенко В.П., Масич А.Г., Масич Г.Ф. Региональная научно-образовательная оптическая сеть УрО РАН – фундамент киберинфраструктуры // Материалы X Международной конференции «Высокопроизводительные параллельные вычисления на кластерных системах (HPC2010)». – Пермь: Изд-во ПГТУ, 2010. Т.1. С. 11-21.

8. Масич А.Г., Масич Г.Ф., Матвеенко В.П., Тирон Г.Г. Инициатива GIGA UrB RAS: методология построения и архитектура научно-образовательной оптической магистрали Уральского отделения РАН // Межд. конф. Математические и информационные технологии, MIT-2011. Белград, 2012. С. 257-265.

9. Масич А.Г., Масич Г.Ф. Аспекты реализации научно-образовательной оптической магистрали УрО РАН // Труды XIX Всероссийской научно-методической конференции «Телематика’2012». - С-Пб.: изд-во ИТМО, 2012. – С.247-250.

10. Созыкин А.В., Гольдштейн М.Л. Система обработки изображений с автоматическим распараллеливанием на основе MapReduce // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование», № 27, 2012. С. 109-118.

11. Усталов Д.А., Гольдштейн М.Л. Распределенная инструментальная среда морфологического анализа для обработки русского языка // Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование», № 27, 2012. С. 119-127.

12. oVirt Project. URL: http://www.ovirt.org/ (дата обращения 28.11.2012).

Высокопроизводительное моделирование распространения электромагнитного поля с использованием технологии CUDA*

Д.А. Жердев, В.А. Фурсов

Самарский государственный аэрокосмический университет имени С.П. Королева, Институт систем обработки изображений РАН

Решается задача моделирования диаграмм рассеяния техногенных объектов на подстилающей поверхности. Для формирования электромагнитного поля в непосредственной близости от объекта используется метод разностного решения уравнений Максвелла в 3-х измерениях затем с использованием разностных соотношений решается задача определения амплитуды поля в дальней точке. Программный комплекс реализован в CUDA-среде. Проведены вычислительные эксперименты по оценке ускорения и эффективности реализации параллельной программы на процессорах с графическими ускорителями.

1. Постановка задачи

Радиолокационные портреты объектов разной природы на подстилающих поверхностях находят широкое применение, например, в системах навигации по картам местности [1], [2], [3], [4], в задачах определения типа техногенного объекта [5], [6] и др. Для определения типа техногенного объекта часто используют так называемую эффективную площадь рассеяния (ЭПР) и/или диаграмму обратного рассеяния (ДОР), определяемую как зависимость ЭПР от угла отражения. Определение указанных характеристик обычно осуществляют на специализированных полигонах или в безэховых камерах, путем регистрации тестовых облучений. Проведение экспериментальных исследований рассеивающих свойств радиолокационных целей и составление баз данных, содержащих диаграммы рассеяния от объектов, является весьма трудной и дорогостоящей работой. Поскольку поле рассеяния обычно весьма чувствительно к форме и ракурсу наблюдения объекта, для надежного распознавания объектов требуется создавать базы данных огромного объема.

Поэтому актуальной является разработка методик и инструментальных средств математического моделирования радиолокационных характеристик техногенных объектов. При таком подходе для определения типа техногенного объекта в базе данных достаточно хранить лишь модель объекта, с использованием которой в каждом конкретном случае определяется эталонная диаграмма рассеяния для текущих характеристик атмосферы, подстилающей поверхности, ракурса наблюдения и др. При этом важнейшим является требование оперативности получения результата, т.к. распознавание объектов по диаграммам рассеяния часто должно осуществляться в реальном времени.

Существуют несложные в вычислительном отношении методы решения задачи рассеяния электромагнитного поля, которые могли бы обеспечить требуемую оперативность. Одним из таких методов является метод геометрической оптики, обеспечивающий получение простых расчетных формул [7]. Данный метод не учитывает волновую природу задачи, которая является причиной интерференции и изменения количества рассеиваемой энергии в зависимости от ракурса цели. Приближенные теории опираются на точную электромагнитную теорию в той или иной ее форме вводят упрощающие аппроксимации, например в интеграле Чу-Стреттона [8], для оценки распределения поверхностного тока. Этот метод базируется на представлениях физической оптики, согласно которой локальная плотность тока в каждой точке облученной части геометрического тела принимается равной плотности тока в этой точке, если бы он протекал по

* Работа выполнена при поддержке РФФИ (проекты № 11-07-12051-офи-м, № 12-07-00581-а).

бесконечной касательной плоскости. Однако методы физической оптики неприменимы там, где требуется точное определение поляризации [9].

Помимо токов, рассматриваемых теорией физической оптики, Уфимцев постулировал так же существование реберных токов [5], что существенно повышает точность вычислений с учетом областей, где направление рассеяния удалено от направления зеркального отражения. Но коэффициенты дифракции Уфимцева плохо себя ведут в переходных областях, вблизи границ затененных и освещенных частей поверхности.

В настоящей работе для моделирования распространения электромагнитного поля используется конечно-разностный явный метод решения уравнений Максвелла во временной области, который позволяет строить достаточно точные радиолокационные портреты объектов. Этот метод хорошо декомпозируется по данным, что является предпосылкой для его оперативной реализации на высокопроизводительных многоядерных гибридных вычислительных системах. В частности, в настоящей работе исследуются показатели эффективности реализации разработанной технологии моделирования в CUDA-среде.

2. Разностная схема решения уравнений Максвелла

В данной работе исследуется метод конечных разностей решения уравнений Максвелла. Конечный набор точек, в которых будут рассчитаны приближенные значения, соответствует дискретной сетке

$$(x, y, z, t) = (i\Delta x, j\Delta y, k\Delta z, n\Delta t),$$

где $\Delta x, \Delta y, \Delta z, \Delta t$ - шаги по пространству вдоль оси x, y, z и по времени t соответственно. Пусть $x \in [0, L_x]$, $y \in [0, L_y]$, $z \in [0, L_z]$, где L_x, L_y, L_z - расстояние в метрах. Тогда $N_x = L_x / \Delta x$ число узлов сетки в направлении x, а $N_y = L_y / \Delta y$, $N_z = L_z / \Delta z$ число узлов в направлении y и z соответственно. Обозначим набор узлов дискретной сетки, как $\Omega = \{i \in [0; N_x], j \in [0; N_y], k \in [0; N_z]\}$.

Для описания изменения электромагнитного поля в окрестности Θ исследуемого объекта используем уравнения Максвелла. Считаем, что среда изотропна и недиспергирующая

$$rot\vec{E} = -\mu_0\mu\frac{\partial\vec{H}}{\partial t}, \quad (1)$$

$$rot\vec{H} = \sigma\vec{E} + \varepsilon_0\varepsilon\frac{\partial\vec{E}}{\partial t}, \quad (2)$$

$$div\vec{E} = \frac{\rho}{\varepsilon\varepsilon_0}, \quad (3)$$

$$\mu\mu_0 div\vec{H} = 0. \quad (4)$$

В случае моделирования свободного от сторонних электрических зарядов пространства $\rho = 0$, а так же с учетом граничных условий

$$E_\tau^{(1)} - E_\tau^{(2)} = 0,$$

$$H_n^{(1)} - H_n^{(2)} = 0,$$

достаточно двух уравнений (1) и (2).

Элемент пространства Ω_o будем называть объектом, если его материальные параметры ε, μ или σ отличаются от параметров среды, в которой распространяется электромагнитная волна. Окрестностью исследуемого объекта является элемент пространства, определяемый как $\Theta = \Omega - \Omega_0$.

Для того чтобы моделировать распространение электромагнитного поля в пространстве на вычислительной машине, нужно записать разностные аналоги для формул (1) и (2).

$$H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})=\frac{\Delta t}{\mu\mu_0}\left(\frac{E_y^n(i,j+\frac{1}{2},k+1)-E_y^n(i,j+\frac{1}{2},k)}{\Delta z}-\frac{E_z^n(i,j+1,k+\frac{1}{2})-E_z^n(i,j,k+\frac{1}{2})}{\Delta y}\right)+$$
$$+H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2}), \quad (5)$$

$$H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})=\frac{\Delta t}{\mu\mu_0}\left(\frac{E_z^n(i+1,j,k+\frac{1}{2})-E_z^n(i,j,k+\frac{1}{2})}{\Delta x}-\frac{E_x^n(i+\frac{1}{2},j,k+1)-E_x^n(i+\frac{1}{2},j,k)}{\Delta z}\right)+$$
$$+H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}), \quad (6)$$

$$H_z^{n+\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k)=\frac{\Delta t}{\mu\mu_0}\left(\frac{E_x^n(i+\frac{1}{2},j+1,k)-E_x^n(i+\frac{1}{2},j,k)}{\Delta y}-\frac{E_y^n(i+1,j+\frac{1}{2},k)-E_y^n(i,j+\frac{1}{2},k)}{\Delta x}\right)+$$
$$+H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k), \quad (7)$$

$$E_x^n(i+\frac{1}{2},j,k)=\frac{\Delta t}{\varepsilon\varepsilon_0}\left(\frac{H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k)-H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j-\frac{1}{2},k)}{\Delta y}-\frac{H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})}{\Delta z}+\right.$$
$$\left.+\frac{H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k-\frac{1}{2})}{\Delta z}-\sigma E_x^{n-1}(i+\frac{1}{2},j,k)\right)+E_x^{n-1}(i+\frac{1}{2},j,k), \quad (8)$$

$$E_y^n(i,j+\frac{1}{2},k)=\frac{\Delta t}{\varepsilon\varepsilon_0}\left(\frac{H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})-H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k-\frac{1}{2})}{\Delta z}-\frac{H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k)}{\Delta x}+\right.$$
$$\left.+\frac{H_z^{n-\frac{1}{2}}(i-\frac{1}{2},j+\frac{1}{2},k)}{\Delta x}-\sigma E_y^{n-1}(i,j+\frac{1}{2},k)\right)+E_y^{n-1}(i,j+\frac{1}{2},k), \quad (9)$$

$$E_z^n(i,j,k+\frac{1}{2})=\frac{\Delta t}{\varepsilon\varepsilon_0}\left(\frac{H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})-H_y^{n-\frac{1}{2}}(i-\frac{1}{2},j,k+\frac{1}{2})}{\Delta x}-\frac{H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})}{\Delta y}+\right.$$
$$\left.+\frac{H_x^{n-\frac{1}{2}}(i,j-\frac{1}{2},k+\frac{1}{2})}{\Delta y}-\sigma E_z^{n-1}(i,j,k+\frac{1}{2})\right)+E_z^{n-1}(i,j,k+\frac{1}{2}). \quad (10)$$

Разностные формулы (5) – (10) имеют второй порядок аппроксимации по времени и пространству [10]. В трехмерном пространстве расположение компонент электромагнитного поля в узлах сетки представлено на рисунке 1.

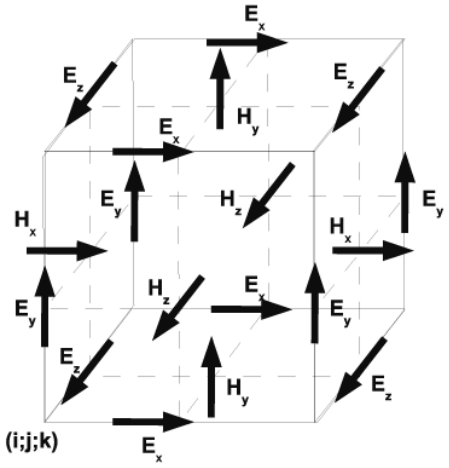


Рис. 1. Расположение узлов на разностной схеме

Для устойчивости данной разностной схемы должно выполняться следующее соотношение для шагов по пространству и времени

$$\Delta t \le \frac{CF}{c_{\max}\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}}},$$

где CF – фактор Куранта (меньше единицы).

Описанная разностная схема решения уравнений Максвелла позволяет решить задачу моделирования электромагнитного излучения в окрестности Θ исследуемого объекта.

Для нахождения поля на значительном удалении от рассеивающего объекта, когда к принимающей антенне приходят плоские волны, применяется метод преобразования ближнего поля в дальнее поле, описанный в работах [11], [12] для случая двух измерений. В данном случае для построения вычислительной схемы моделирования в трех измерениях использовалась методика, описанная в работе [13].

3. Параллельная реализация разностных уравнений Максвелла

Особенностью архитектуры CUDA [14] является блочно-сеточная организация. Все потоки, выполняющие ядро, объединяются в блоки, а блоки, в свою очередь, объединяются в сетку. Традиционная схема работы в CUDA-среде включает этапы подготовки памяти для видеокарты, загрузки в нее исходных данных, запуске ядра выполнения и завершении работы с памятью.

Разработанный здесь алгоритм может работать со всеми видеокартами, начиная с Compute capability 1.1 и выше, что является несомненным достоинством. Один из алгоритмов вычисления построен на глобальной памяти. Все массивы в программе имеют линейную размерность. Доступ к элементам массива (с учетом того, что решается трехмерная задача) осуществляется по формуле

$index = i + j * N_x + k * N_x * N_y$, где $i \in [0, N_x - 1]$, $j \in [0, N_y - 1]$ и $k \in [0, N_z - 1]$.

Перед вычислениями определяем место для массивов, характеризующих электромагнитное поле Hx, Hy, Hz, Ex, Ey, Ez. Согласно разностным уравнениям (5) – (10), массивы поля $\vec{H}$ имеют размерность $N_x * N_y * N_z$, а массивы поля $\vec{E}$ - $(N_x+1)*(N_y+1)*(N_z+1)$. Так же нужно выделить память для параметров среды ε, μ и σ такой же размерности, как и массивы поля $\vec{H}$. В алгоритме, построенном с использованием глобальной памяти, сетка CUDA вычислений имеет двумерную структуру, а блок линейную структуру. Алгоритм моделирования с использованием запуска ядер на CUDA, на примере вычисления H_x компоненты поля, выглядит следующим образом. В программе, использующую глобальную память, для моделирования дис-

кретной сетки размером 100×100×100 используется сетка размером 100×100 и блок размером 100 потоков. На вычисление запускается два ядра для расчета $\vec{H}$ и $\vec{E}$ в пространстве Ω.

```
dim3 dimGridH(nx, ny);
dim3 dimBlockH(nz);

kernelHx<<<dimGridH, dimBlockH>>>(Ey, Ez, Hx, mj, dx, dy, dz, dt);
```

Рис. 2. Алгоритм одного временного шага вычисления компоненты электромагнитного поля по схеме (5) – (10)

При организации вычислений с использованием быстрой разделяемой памяти CUDA, распределяемый на блок по 16 Kb, можно действовать тремя разными способами:

1. В соответствии с разностной схемой «положить» в вычислительный блок все необходимые значения исходных данных для моделирования поля.

2. Значения «забегающие вперед» брать из глобальной памяти. Шаблон данного метода представлен на рисунке 3.

3. Граничные значения обрабатывать особым образом. Декомпозиция по данным на уровне блока одной из проекции представлена на рисунке 3.

На рисунках 3, 4 закрашенными обозначены элементы - берущиеся из глобальной памяти, а без цвета - берущиеся из разделяемой.

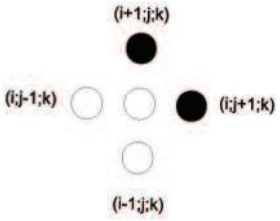


Рис. 3. Шаблон вычислений по 2 методу

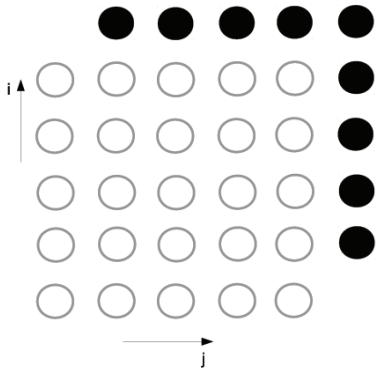


Рис. 4. Декомпозиция данных по 3 методу

В вычислениях с разделяемой памятью, организация блока имеет трёхмерную структуру размера 6×6×6 ячеек. В данном случае возможно задание трёхмерной структуры сетки, но на Compute capability 1.1 разрешена, только двумерная структура, этот недостаток удалось обойти посредством линейного «вытягивания» сетки в одном направлении в нужное число раз.

Во втором и третьем способах память для всех массивов разделяемой памяти выделяется одна и та же BLOCK_SIZE*BLOCK_SIZE*BLOCK_SIZE. В программе, использующую глобальную память, для моделирования дискретной сетки размером 120×120×120 используется сетка размером 6×6×6 и блок размером 400×20 потоков. На вычисление запускается два ядра для расчета $\vec{H}$ и $\vec{E}$ в пространстве Ω.

В третьем случае добавляется только условие на границе. Алгоритм вычисления следующего временного шага компоненты H_x векторного поля с использованием разделяемой памяти и третьего способа организации вычислений представлен ниже.

Для оценки быстродействия программного комплекса были проведены сравнительные эксперименты с использованием CPU и GPU. Численные эксперименты на GPU проводились на видеокарте Nvidia GeForce 8800 GT 1024 Mb RAM, Compute capability 1.1, а так же на CPU AMD Athlon 64 X2 5600+.

На рисунке 6 представлены результаты замера времени выполнения программы на CPU и GPU с использованием глобальной, а так же трех разных способов вычислений с разделяемой памятью. Эксперименты проводились для различных размеров дискретной сетки.

```
#define BLOCK_SIZE = 6;

__shared__ float s_Hx[BLOCK_SIZE*BLOCK_SIZE*BLOCK_SIZE];
__shared__ float s_Ey[BLOCK_SIZE*BLOCK_SIZE*BLOCK_SIZE];
__shared__ float s_Ez[BLOCK_SIZE*BLOCK_SIZE*BLOCK_SIZE];
__shared__ float s_mj[BLOCK_SIZE*BLOCK_SIZE*BLOCK_SIZE];

int m = blockIdx.x / sizeX;
int i = m*BLOCK_SIZE + threadIdx.x;
int j = blockIdx.y*BLOCK_SIZE + threadIdx.y;
int k = (blockIdx.x - m * sizeX)*BLOCK_SIZE + threadIdx.z;
int index = i + j * (sizeX * BLOCK_SIZE + 1) + k * (sizeX *
BLOCK_SIZE + 1) * (gridDim.y * BLOCK_SIZE + 1);
int indexS = threadIdx.x + threadIdx.y*BLOCK_SIZE +
threadIdx.z*BLOCK_SIZE*BLOCK_SIZE;
int indexS_Ey = threadIdx.x + (threadIdx.y + 1)*BLOCK_SIZE +
threadIdx.z*BLOCK_SIZE*BLOCK_SIZE;
int indexS_Ez = threadIdx.x + threadIdx.y*BLOCK_SIZE +
(threadIdx.z + 1)*BLOCK_SIZE*BLOCK_SIZE;

      s_Hx[indexS] = Hx[index];
      s_Ey[indexS] = Ey[index];
      s_Ez[indexS] = Ez[index];
      s_mj[indexS] = mj[index];
      __syncthreads();
if(threadIdx.x == BLOCK_SIZE-1 || threadIdx.y == BLOCK_SIZE-1 ||
threadIdx.z == BLOCK_SIZE-1){
      s_Hx[indexS] += dt / s_mj[indexS] * ((Ey[index + (sizeX *
BLOCK_SIZE + 1) * (gridDim.y * BLOCK_SIZE + 1)] - s_Ey[indexS])/dz
- (Ez[index + (sizeX * BLOCK_SIZE + 1)] - s_Ez[indexS])/dy);
}
else{
      s_Hx[indexS] += dt / s_mj[indexS] * ((s_Ey[indexS_Ez] -
s_Ey[indexS])/dz - (s_Ez[indexS_Ey] - s_Ez[indexS])/dy);
}
            __syncthreads();
      Hx[index] = s_Hx[indexS];
```

Рис. 5. Алгоритм вычисления следующего шага по времени компоненты напряженности магнитного поля

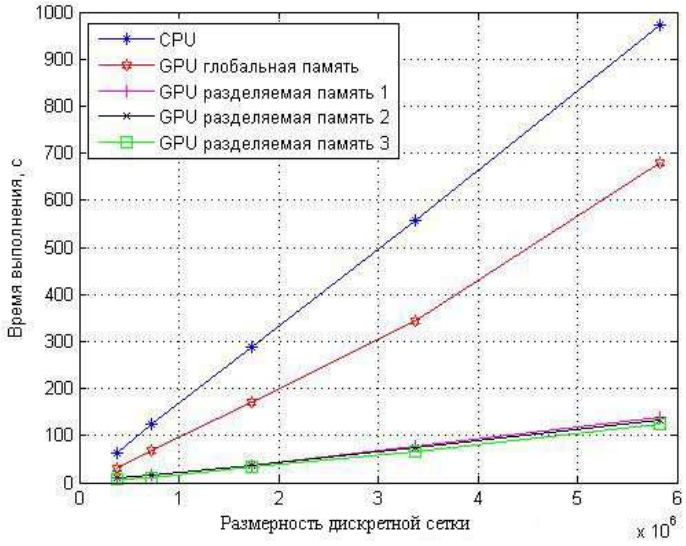


Рис. 6. Сравнение скорости выполнения

4. Примеры радиолокационных портретов

При проведении численного моделирования была создана 3-х мерная модель на подстилающей поверхности, показанные на рисунке 7, а. Данный объект состоит из замкнутых объёмных тел, скомпонованных так, что являются моделью самолёта соответственно. Модели сохраняются в сеточном виде, где одному полигону соответствует замкнутая элементарная фигура (шар, эллипсоид, цилиндр и т.д.). При загрузке в программу моделирования для полигона на основе множества уравнений плоскостей определяется: лежит точка внутри тела или нет.

Для внешнего пространства, в котором распространяется электромагнитное излучение, параметры среды принимались следующими:

$$\varepsilon = 1, \quad \mu = 1, \quad \sigma = 0.$$

Моделируемые объекты задавались идеальными проводниками.

Эксперименты проводились с тремя типами подстилающих поверхностей:

- первый тип: $\varepsilon = 8.7, \mu = 1, \sigma = 0.0001$;
- второй тип: $\varepsilon = 5.7, \mu = 1, \sigma = 0.004$;
- третий тип: $\varepsilon = 3.7, \mu = 1, \sigma = 0.008$.

При проведении эксперимента амплитуда электрического поля была задана 1 В/м, амплитуда магнитного поля соответственно 0,0027 А/м. На объект падала плоская электромагнитная волна, длина волны см. Линейные размеры объектов превышали длину волны в 100-500 раз.

На рисунках 7, б, в, г приведены полученные в ходе экспериментов соответствующие указанным объектам диаграммы рассеяния электромагнитного поля, усреднённые по набору углов.

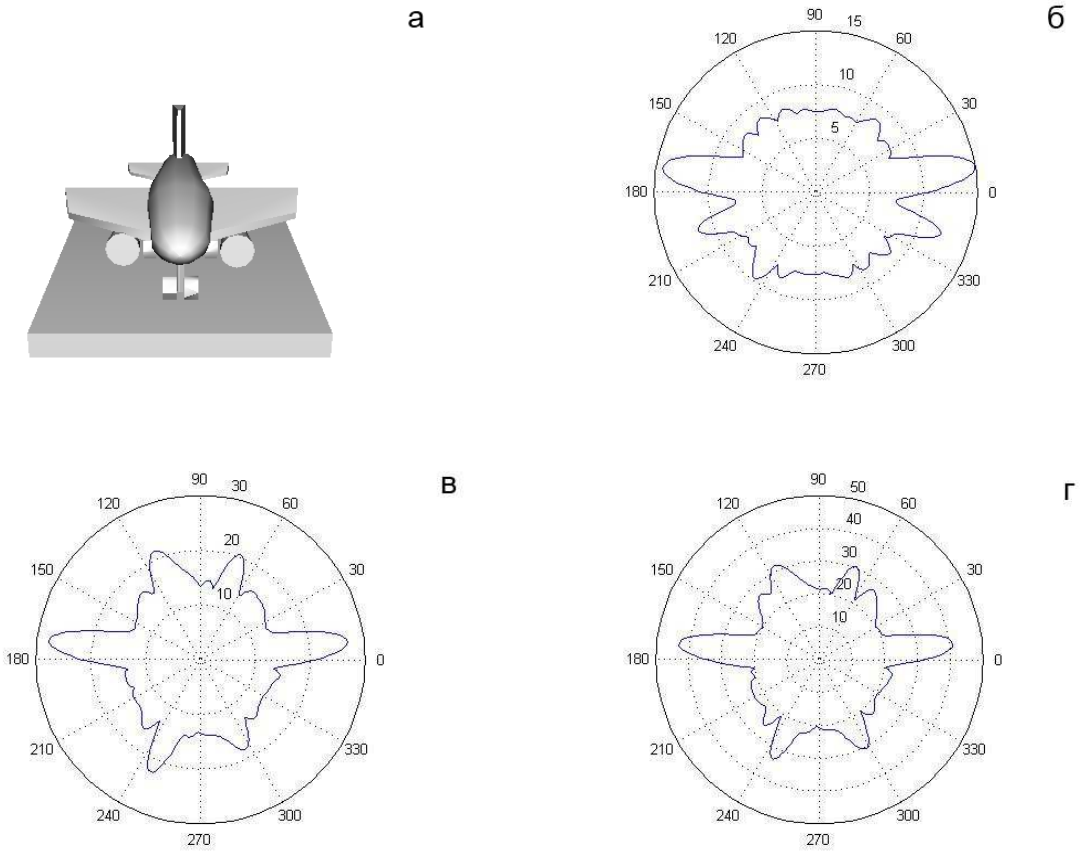


Рис. 7. Результаты моделирования: а) модель самолета;
диаграммы рассеяния на подстилающих поверхностях: б) 1-го типа, в) 2-го типа, г) 3-го типа

6. Заключение

Полученные результаты открывают новые перспективы в области радиолокационного наблюдения. Возможность с высоким быстродействием строить распределение отраженного электромагнитного излучения в трех измерениях позволит существенно повысить точность и надежность решения задач классификации объектов по радиолокационным портретам.

Несмотря на существование открытых пакетов трёхмерного моделирования уравнений Максвелла на графических картах, например, B-CALM [15], данная разработка актуальна. Разработанное программное обеспечение специально ориентировано на решение задачи построения диаграмм ЭПР и ОДР и за счет оптимизации кода с учетом особенностей этой задачи, позволит повысить оперативность систем навигации, тематической обработки данных радиолокационного зондирования Земли и др.

Литература

1. Костоусов В.Б., Костоусов А.В. Моделирование процесса наведения движущихся объектов по радиолокационным изображениям // Гироскопия и навигация. 2004. Вып. 2. С. 37-47
2. Иванов Н.М., Лысенко Л.Н. Баллистика и навигация космических аппаратов // М.: Дрофа. 2004. 544 с.
3. Анучин О.Н., Емельянцев Г.И. Интегрированные системы ориентации и навигации для морских подвижных объектов // М.: Электроприбор. 1999. 357 с.
4. Алешин Б.С., Времеенко К.К. Ориентация и навигация подвижных объектов // М.: Физматлит. 2004. 424 с.
5. Уфимцев П.Я. Теория дифракционных краевых волн в электродинамике. Введение в физическую теорию дифракции // М.: Бином. 2012. 372 с.
6. Лагарьков А.Н., Погосян М.А. Фундаментальные и прикладные проблемы стелс-технологий // Вестник российской Академии Наук том 73, № 9. 2003. 848 с. URL: http://vivovoco.rsl.ru/VV/JOURNAL/VRAN/03_10/STELLS.HTM (дата обращения: 29.11.2012).
7. Spencer R. C. Optical Theory of the Corner Reflector // MIT Radiation Lab. 1944. 433 p.
8. Stratton J. A. Electromagnetic theory // McGRAW-HILL book company Inc. 1941. 631 p.
9. Siegel K. M., Bowman J.J. Methods of Radar Cross Section Analysis // Academic Press Inc. 1968. 426 p.
10. Kane Yee. Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media // Antennas and Propagation. 1995. V. 14. P. 302–307.
11. Gonz´alez Garc´ıa., Garc´ıa Olmedo, Go´mez Mart´ın. A time-domain near-to far-field transformation for fdtd in two dimensions // MICROWAVE AND OPTICAL TECHNOLOGY LETTERS. 2000. V. 27. P. 427-432.
12. John B. Scheneider. Understanding the Finite-Difference Time-Domain Method // Scholl of electrical engineering and computer science Washington State University. URL: http://www.eecs.wsu.edu/~schneidj/ufdtd/ (request data: 29.11.2012).
13. Taflove A., Hagness S. Computational Electrodynamics: The Finite-Difference Time-Domain Method // Boston: "Arthech House" Publisher. 2005. 1006 P.
14. NVIDIA CUDA. Nvidia CUDA C Programming Guide // Version 4.2. 16.4.2012. URL: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf (request data: 29.11.2012).
15. Belgium-California Light Machine. An Open-Source GPU-based 3D-FDTD with Multi-Pole Dispersion for Plasmonics // Pierre Wahl, Dany Ly-Gagnon, Christof Debaes, David Miller, Hugo Thienpont. Vrjje Universiteit Brussel. Stanford University. URL: ftp://ftp.heanet.ie/mirrors/sourceforge/b/b-/b-calm/NUSODTALK.pdf (request data: 29.11.2012).

Распараллеливание итерационных методов решения вариационных неравенств*

Д.Н. Запорожец

Омский государственный технический университет

Рассмотрены эффективные методы решения вариационных неравенств: градиентный метод с постоянным шагом, одношаговый и двухшаговый экстраградиентные методы. Предложен итерационный метод с памятью как подход к распараллеливанию итерационных методов решения вариационных неравенств. Проведено сравнение эффективности полученных методов на тестовых задачах.

1. Введение

Вариационные неравенства являются удобным инструментом для решения многих задач. В терминах вариационных неравенств можно сформулировать задачи из различных областей, таких как: математическая физика, механика, экономика и другие.

Универсальным способом решения вариационных неравенств являются итерационные методы, в частности – градиентные. Однако для сходимости градиентных методов требуется выполнение условий сильной монотонности оператора вариационного неравенства или компактности исходного множества. Ослабить эти условия, а значит, расширить класс решаемых задач, позволяют одношаговый экстраградиентный метод, независимо предложенный Г. М. Корпелевич [1] и А.С. Антипиным [2] и двухшаговыйэкстраградиентный метод, предложенный Н. В. Меленьчуком [3, 4]. Экстраградиентные методы применимы при условии монотонности оператора вариационного неравенства и замкнутости исходного множества, а также эти методы обладают важным свойством сходимости из любой начальной точки.

Общим недостатком итерационных методов является строгая последовательность процесса и использование на каждой итерации только той информации, что получена на этой же итерации, невозможность начать вычисление очередного шага, пока не закончится вычисление предыдущего. Предложенные автором статьи итерационные методы с памятью [5] запоминают информацию о направлении на предыдущей итерации и используют её при вычислении очередной итерации. В связи с этим, в итерационных методах с памятью возможно использование параллельных вычислений на каждой итерации, что позволит сократить время решения задачи.

В данной статье этот подход применяется к градиентному методу, к одношаговому экстраградиентному методу и к двухшаговому экстраградиентному методу решения вариационных неравенств, проводится сравнение эффективности этих методов.

2. Постановка задачи и описание методов решения

Решить *вариационное неравенство* – значит найти такой вектор z^*, удовлетворяющий условиям:

$$(G(z^*), z - z^*) \geq 0, \forall z \in \Omega,$$

где $G(z): R^n \to R^n$, Ω – выпуклое, замкнутое множество.

Рассмотрим методы решения вариационных неравенств: градиентный метод с постоянным шагом, одношаговый экстраградиентный метод и двухшаговыйэкстраградиентный метод.

Градиентный метод с постоянным шагом имеет следующую вычислительную схему:

$$x_k = P_\Omega(x_{k-1} - \alpha G(x_{k-1})).$$

Здесь α– длина шага метода, G – оператор вариационного неравенства.

* Исследование выполнено при поддержке Министерства образования и науки Российской Федерации, соглашение № 14.B37.21.1123 и при поддержке РФФИ, проект № 12-07-00326

Одношаговый экстраградиентный метод задается следующими рекуррентными соотношениями:

$$\dot{x}_k = P_\Omega(x_{k-1} - \alpha G(x_{k-1})),$$
$$x_k = P_\Omega(x_{k-1} - \alpha G(\dot{x}_k)).$$

Основное отличие одношагового экстраградиентного метода от градиентного с постоянным шагом заключается в вычислении прогнозной точки $\dot{x}_k$. Для вычисления очередной точки используется направление из прогнозной точки.

Двухшаговыйэкстраградиентный метод описывается рекуррентными выражениями:

$$\dot{x}_k = P_\Omega(x_{k-1} - \alpha G(x_{k-1})),$$
$$\ddot{x}_k = P_\Omega(x_k - \alpha G(\dot{x}_k)),$$
$$x_k = P_\Omega(x_{k-1} - \alpha G(\ddot{x}_k)).$$

В отличие от одношагового экстраградиентного метода двухшаговый метод для вычисления очередной точки использует две прогнозных точки $\dot{x}_k$ и $\ddot{x}_k$.

Легко заметить, что все вышеперечисленные схемы строго последовательны. Вычисление очередной точки происходит в два этапа. Сначала вычисляется направление движения из текущей точки, затем делается шаг метода из текущей точки по вычисленному направлению. При этом на старте каждой итерации используется только точка, полученная на предыдущем шаге.

Итерационные методы с памятью запоминают направление движения метода на каждом шаге и используют его для вычисления очередной точки на следующем шаге. Рассмотрим их схему. Пусть итерационный процесс имеет вид

$$x_{k+1} = F(x_k),$$

где F – некоторая процедура, последовательность действий, необходимая для получения следующей точки.

Определение. Вектором движения итерационного процесса на k итерации называется разность точек, полученных на k и k–1 итерациях.

$$r_k = x_k - x_{k-1}.$$

Используя введенное определение можно на k итерации построить вспомогательную точку $\hat{x}_k$ путем прибавления вектора r_k, то есть

$$\hat{x}_k = x_k + r_k.$$

Тогда вычислительная схема метода с памятью выглядит следующим образом:

$$x_{k+1} = \begin{cases} F(\hat{x}_k), \|F(x_k) - \hat{x}_k\| \le \|F(x_k) - x_k\| \dfrac{(F(x_k) - \hat{x}_k, F(x_k) - x_k)}{\|F(x_k) - \hat{x}_k\| \times \|F(x_k) - x_k\|} \\ F(x_k), \text{иначе} \end{cases}$$

Выполнение условия

$$\|F(x_k) - \hat{x}_k\| \le \|F(x_k) - x_k\| \frac{(F(x_k) - \hat{x}_k, F(x_k) - x_k)}{\|F(x_k) - \hat{x}_k\| \times \|F(x_k) - x_k\|}$$

будем называть критерием одного направления.

Суть предлагаемого подхода заключается в том, что используя вычисленное ранее направление r_k, можно за линейное время вычислить вспомогательную точку $\hat{x}_k$. Затем вычислительный процесс можно распараллелить и на одном процессоре вычислить $F(x_k)$, в то время, как на другом –$F(\hat{x}_k)$. И если $F(x_k)$ и $\hat{x}_k$ оказываются достаточно близки, а именно, выполняется критерий одного направления, то в качестве следующей точки следует брать $F(\hat{x}_k)$. Геометрическая схема метода представлена на рисунке 1.

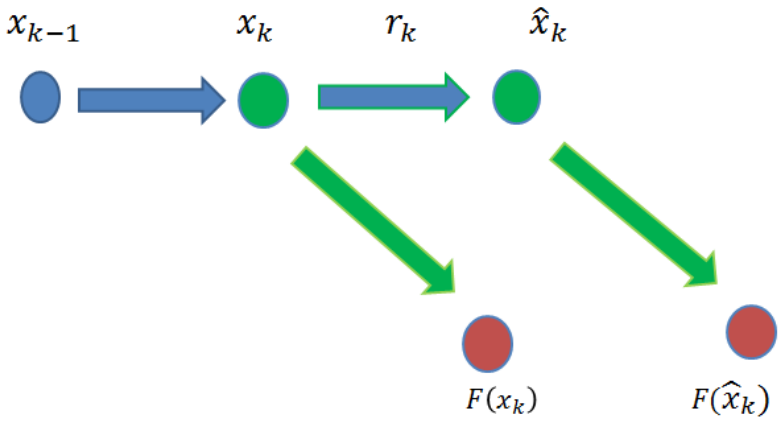


Рис. 1. Геометрическая схема итерационного метода с памятью

Данный подход позволяет существенно сократить число итераций в задачах, где вектор движения итерационного процесса меняется не часто. Также возможно его применение совместно с распараллеливанием трудоемких операций в вычислительной процедуре *F*. Применение предлагаемого подхода к градиентному методу с постоянным шагом, одношаговому экстраградиентному методу и двухшаговому экстраградиентному методу позволяет получить новый градиентный метод с постоянным шагом с памятью, одношаговый экстраградиентный метод с памятью и двухшаговый экстраградиентный метод с памятью.

3. Программная реализация итерационных методов с памятью

Методы реализованы на языке программирования C++. Так как степень параллелизма итерационных методов с памятью равна двум, то для их реализации достаточно системы с общей памятью, поэтому использовалась библиотека OpenMP 2.0.

Ниже на рисунке 2 приведен код двухшагового экстраградиентного метода с памятью. В коде переменная *dAlfa* отвечает за длину шага метода. Функция *AddVectors* складывает два вектора, которые передаются первым и вторым аргументами, результат сложения записывается в вектор, передаваемым 3 аргументом. Функция *Calc2StepIter* делает одну итерацию двухшаговым экстраградиентным методом. В качестве первого параметра функция принимает начальный вектор, вторым параметром передается вектор, который будет содержать конечную точку и третьим параметром передается длина шага метода. Функция *SubVectors* находит вектор, являющийся разностью двух других векторов. Первым параметром передается уменьшаемое, вторым параметром - вычитаемое, а разность будет содержаться в третьем параметре. Функция *NormSubVectors* вычисляет норму разности двух векторов без непосредственного вычисления вектора разности. Функция *MultyScalarVectors* вычисляет скалярное произведение векторов. Для реализации итерационного метода с памятью применительно к другому методу необходимо изменить правила вычисления константы *dAlfa* и заменить функцию *Calc2StepIter*.

```
doubledAlfa = 1.0/(1.05*sqrt(3.0)*L);
doubleCountIter22Step= 0;
do{
   AddVectors(&uk,&remdir,&vk);
   #pragma omp parallel if (proc_num > 1) num_threads(proc_num)
   {
       #pragma omp sections
       {
       #pragma omp section
             Calc2StepIter(&uk,&uk1, dAlfa);
       #pragma omp section
             Calc2StepIter(&vk,&vk1,dAlfa);
       }
   }
   Vector v(uk.GetSize());
   SubVectors(&uk1, &uk, &v);
   if ((remdir.Norm() != 0.0) && (NormSubVectors(&uk1,&vk) < MultyScalarVec-
tors(&v, &remdir) /remdir.Norm()))
   {
       uk1.SetEqual(&vk1);
       SubVectors(&vk1,&vk,&remdir);
   }
   Else
       SubVectors(&uk1,&uk,&remdir);

   dNorm = NormSubVectors(&uk1,&uk);
   uk.SetEqual(&uk1);
   CountIter22Step++;
}
       while (dNorm > eps);
```

Рис. 2. Двухшаговый экстраградиентный метод с памятью

4. Результаты распараллеливания итерационных методов

Рассмотрим, как применение итерационных методов с памятью влияет на решение задач. В качестве тестовых примеров возьмем некоторые примеры из монографии И. В. Коннова [6], а так же классические для методов оптимизации функции Розенброка и Химмельблау.

Линейная задача дополнительности (ЛЗД) заключается в том, что необходимо найти вектор u^* из R^n_+ и Mu^*+q из R^n_+, являющийся решением

$$< u^*, Mu^* + q > = 0$$

Матрицу M зададим следующим образом

$$M = \begin{cases} 2, i < j \\ 1, i = j, \\ 0, i > j \end{cases}$$

а вектор q возьмем равным $q = (-1,\ldots,-1)$. В качестве допустимой области Ω выступает R^n_+. Решением такой задачи является $u^*=(0,\ldots,0,1)$. В качестве начальной точки выберем $x_0 = (0,\ldots,0)$.

Результаты вычислений представлены в таблице 1. Векторное поле оператора H и траектории методов показаны на рисунке 3. Штриховой линией показана траектория одношагового экстраградиентного метода, штрих-пунктирной линией – двухшагового экстраградиентного метода.

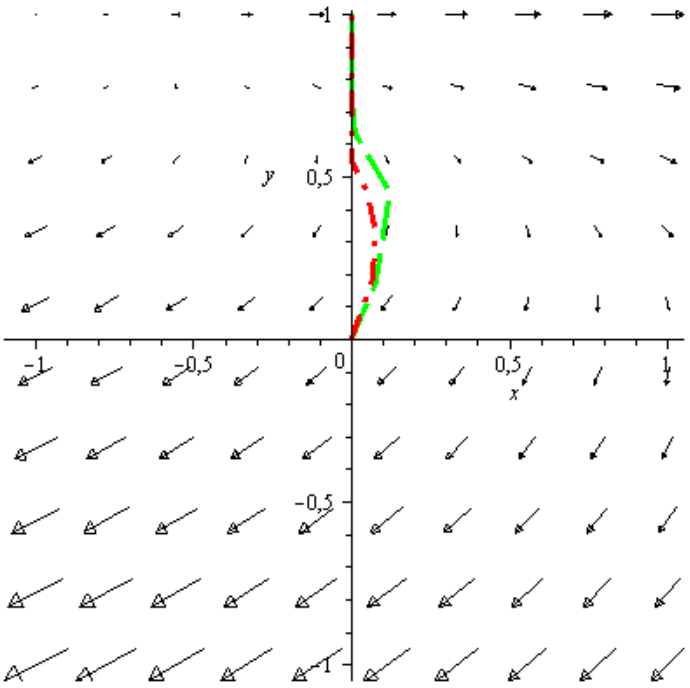


Рис. 3. Траектории экстраградиентных методов при решении ЛЗД

Таблица 1. Результаты решения ЛЗД

Размерность задачи	Исходный метод	Количество итераций исходным методом / с памятью	Эффективность
10	Градиентный	212/145	1.46
10	Одношаговый	302/203	1.49
10	Двухшаговый	371/195	1.9
100	Градиентный	1768/1192	1.48
100	Одношаговый	2406/1625	1.48
100	Двухшаговый	2879/1515	1.9
500	Градиентный	7194/4842	1.49
500	Одношаговый	9666/6498	1.49
500	Двухшаговый	11474/5995	1.91

Результаты показывают, что для приведенной линейной задачи дополнительности градиентный метод с памятью эффективнее градиентного на *46%*, одношаговый экстраградиентный метод с памятью эффективнее одношагового экстраградиентного метода на *49%*, а двухшаговый экстраградиентный метод с памятью эффективнее двухшагового экстраградиентного метода на *90%.* Такая закономерность обуславливается тем, что векторное поле оператора *H* меняется незначительно, и как следствие, в траекториях методов много шагов, идущих в одном направлении.

Решим нелинейное вариационное неравенство (НВН) с оператором

$$G(u) = \begin{pmatrix} (1+u_n)^3 u_1 \\ (1+u_{n-1})^3 u_2 \\ \cdots \\ (1+u_1)^3 u_n \end{pmatrix}$$

и допустимой областью Ω, которая задается условием

$$h(u) = max(h_1(u), h_2(u)),$$

где

$$h_1(u) = (u_1 - 3)^2 + \sum_{i=2}^{n} u_i^2 - 9$$

$$h_2(u) = (u_1 + 1)^2 + \sum_{i=2}^{n} u_i^2 - 25$$

Решением вариационного неравенства является u^*=(0, 0, …, 0). В качестве стартовой точки выберем x_0 = (3, 3, …, 3). Результаты вычислений представлены в таблице 2. Векторное поле оператора вариационного неравенства и траектории методов при решении нелинейного вариационного неравенства размерности 2 показаны на рисунке 4. Штриховой линией показана траектория одношагового экстраградиентного метода, штрих-пунктирной линией – двухшагового экстраградиентного метода.

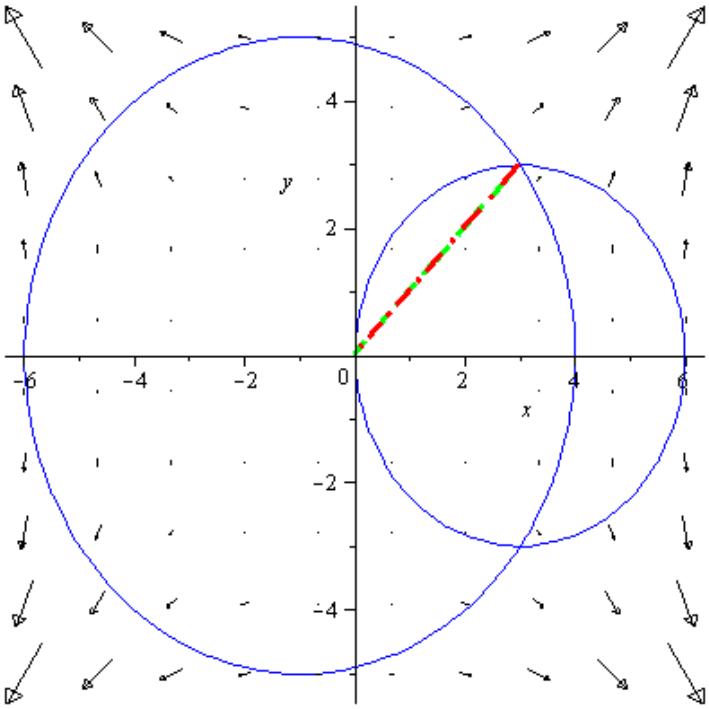


Рис. 4. Траектории экстраградиентных методов при решении НВН

Таблица 2. Результаты решения НВН

Размерность задачи	Исходный метод	Количество итераций исходным методом / с памятью	Эффективность
10	Градиентный	2379/1585	1.5
10	Одношаговый	3264/2175	1.5
10	Двухшаговый	3926/2099	1.59
100	Градиентный	2309/1538	1.5
100	Одношаговый	3163/2108	1.5
100	Двухшаговый	3801/2037	1.86
1000	Градиентный	2316/1544	1.5
1000	Одношаговый	3174/2116	1.5
1000	Двухшаговый	3814/2044	1.86

Результаты решения нелинейного вариационного неравенства подтверждают эффективность применения итерационных методов с памятью. Градиентный метод с памятью эффективнее градиентного на 50%, одношаговый экстраградиентный метод с памятью эффективнее одношагового экстраградиентного метода на 50%, а двухшаговый экстраградиентный метод с памятью эффективнее двухшагового экстраградиентного метода на 86%. Примечательно, что двухшаговый экстраградиентный метод с памятью решает задачу быстрее, чем одношаговый экстраградиентный метод с памятью, притом, что на той же задаче одношаговый экстраградиентный метод эффективнее двухшагового экстраградиентного метода.

Рассмотрим задачу поиска минимума функции Розенброка. Функция Розенброка — функция, используемая для оценки производительности алгоритмов оптимизации, предложенная Ховардом Розенброком в 1960 году. Функция Розенброка для двух переменных определяется как:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

Известно, что функция достигает своего глобального минимума 0 в точке (1,1). Оптимизационную задачу поиска минимума функции Розенброка можно записать в виде вариационного неравенства, для этого в качестве оператора вариационного неравенства H нужно взять вектор

$$\begin{pmatrix} -400x(y-x^2) + 2x - 2 \\ 200(y-x^2) \end{pmatrix}$$

Функция Розенброка не является выпуклой. Граница области выпуклости походит вдоль кривой $y = x^2 + \frac{1}{200}$. Ниже этой кривой функция строго выпукла, а значит, имеет только одну точку минимума в этой области. Выше границы выпуклости функция Розенброка вогнута, а значит, не может иметь точек минимума. В качестве начальной точки возьмем (-1, *2*). Результаты решения задачи представлены в таблице 3.

Таблица 3. Результаты решения функции Розенброка

Размерность задачи	Исходный метод	Количество итераций исходным методом / с памятью	Эффективность
2	Градиентный	298307/198867	1.5
2	Одношаговый	373296/248861	1.5
2	Двухшаговый	423501/271004	1.56

В задаче поиска минимума функции Розенброка итерационные методы с памятью так же демонстрируют свою эффективность. Градиентный метод с памятью эффективнее градиентного на 50%, одношаговый экстраградиентный метод с памятью эффективнее одношагового экстраградиентного метода на 50%, а двухшаговый экстраградиентный метод с памятью эффективнее двухшагового экстраградиентного метода на 56%.

Другой классической тестовой функцией для методов оптимизации служит функция Химмельблау. Функция задается следующим образом:

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)$$

Известно, что 0 является глобальным минимумом функции и достигается в четырех точках (-3.78; -3.28), (-2.8; 3.13), (3.58; -1.85), (3;2).

Задачу поиска минимума функции Химмельблау можно записать в виде вариационного неравенства, для этого в качестве оператора вариационного неравенства необходимо взять

$$H = \begin{pmatrix} 4x(x^2 + y - 11) + 2(x + y^2 - 7) \\ 2(x^2 + y - 11) + 4y(x + y^2 - 7) \end{pmatrix},$$

а в качестве допустимой области будем рассматривать первую четверть координатной плоскости. Тогда (3, 2) .будет единственной точкой минимума. Функция Химмельблау не является выпуклой. На рисунке 5 показаны области выпуклости и вогнутости функции. В таблице 4 показаны результаты решения задачи.

Таблица 4. Результаты решения функции Химмельблау

Размерность задачи	Исходный метод	Количество итераций исходным методом / с памятью	Эффективность
2	Градиентный	169/112	1.5
2	Одношаговый	247/163	1.52
2	Двухшаговый	308/162	1.9

В задаче поиска минимума функции Химмельблау, так же как и для функции Розенброка, итерационные методы с памятью оказываются эффективными. Градиентный метод с памятью эффективнее градиентного на 50%, одношаговый экстраградиентный метод с памятью эффективнее одношагового экстраградиентного метода на 52%, а двухшаговый экстраградиентный метод с памятью эффективнее двухшагового экстраградиентного метода на 90%.

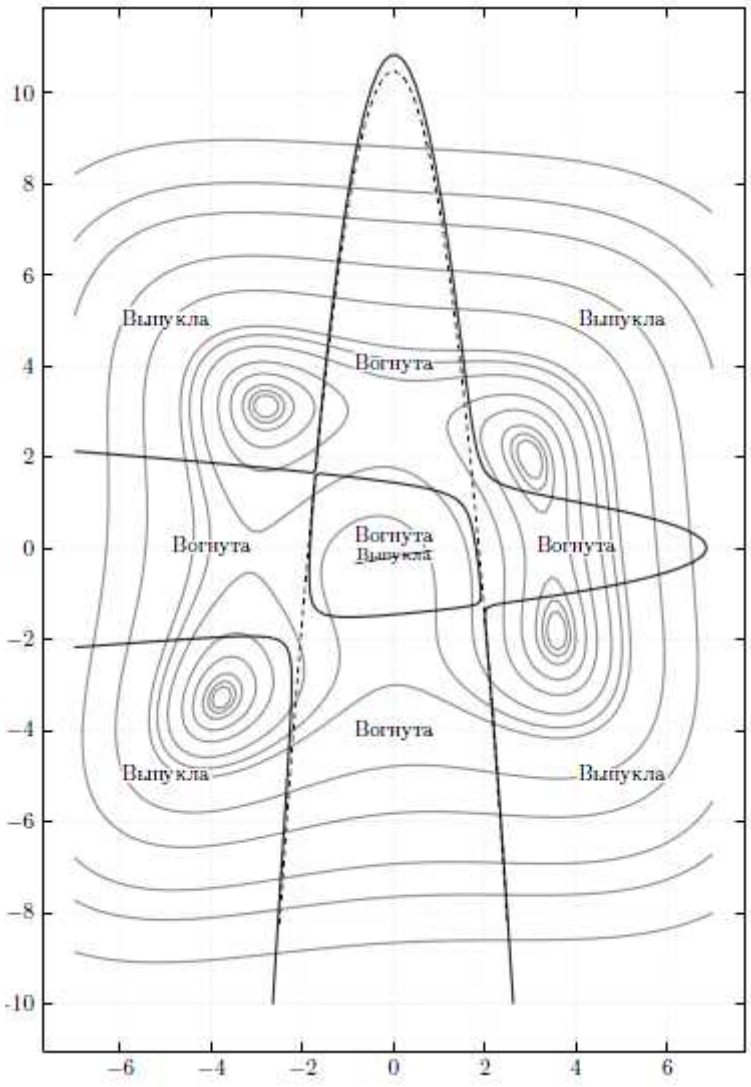


Рис. 5. Линии уровня и области выпуклости функции Химмельблау

5. Заключение

Повышение эффективности вычислительных методов решения вариационных неравенств актуально, поскольку с помощью вариационных неравенств могут быть сформулированы задача теории игр, задача о седловой точке, задача линейного программирования, линейная задача дополнительности и многие другие.

Рассмотрены следующие методы решения вариационных неравенств: градиентный метод, одношаговый и двухшаговый экстраградиентные методы и итерационный метод с памятью. Экстраградиентные методы по сравнению с градиентным обладают тем достоинством, что сходятся из любой точки и позволяют решать более широкий класс задач, за счёт использования прогнозных точек.

Применяя к любому вычислительному процессу итерационный метод с памятью можно существенно увеличить скорость получения решения за счёт использования информации о направлении на предыдущем шаге и применения параллельных вычислений.

В рассмотренных примерах применение итерационного метода с памятью к градиентному методу с постоянным шагом уменьшило количество итерации на46%, для одношагового экстраградиентного метода на49%, а для двухшагового экстраградиентного метода – на56%.

В дальнейших исследованиях планируется рассмотреть другие итерационные методы, а также расширить круг тестовых задач.

Литература

1. Корпелевич Г. М. Экстраградиентный метод для отыскания седловых точек и других задач // Экономика и математические методы. 1976. Т. 12, №4. С. 747–756.
2. Антипин А.С. Экстрапроксимальный метод решения равновесных и игровых задач // Журнал вычислительной математики и математической физики. 2005. Т.45, №11,12. С. 1969-1990, 2102-2111.
3. Зыкина А. В., Меленьчук Н. В. Двухшаговый экстраградиентный метод для вариационных неравенств. // Известия вузов. Математика. 2010. №9. С. 82–85.
4. Зыкина А.В., Меленьчук Н. В, Запорожец Д.Н. Сравнительный анализ экстраградиентных методов решения вариационных неравенств для некоторых задач. // Автоматика и телемеханика. – 2012. – № 4. – С. 32–46.
5. Запорожец Д.Н. Экстраградиентные методы с памятью. // XV Байкальская международная школа-семинар "Методы оптимизации и их приложения". Т. 2: Математическое программирование. Иркутск: РИО ИДСТУ СО РАН, 2011. С. 92 – 95.
6. Konnov I. V. Combined relaxation methods for variational inequalities. Berlin etc.: Springer, 2001

Web-портал для проведения виртуальных экспериментов в распределенных вычислительных средах*

Е.А. Захаров

Южно-Уральский государственный университет

Системы суперкомпьютерного моделирования, как правило, требуют соблюдения определенного, зачастую, далеко не тривиального технологического цикла, что ограничивает круг потенциальных пользователей системы. Для снижения влияния этого фактора и решения проблем интеграции систем компьютерного проектирования в вычислительной среде была предложена технология DiVTB, позволяющая инкапсулировать процесс постановки и решения определенного класса задач. В результате взаимодействие пользователя с вычислительной системой, необходимое для постановки задачи и получения результатов ее моделирования, сводится к уточнению ряда параметров из фиксированных диапазонов. В статье рассматриваются архитектурные и алгоритмические особенности реализации компонента DiVTB Portal.

1. Введение

На **Рис. 1** изображена архитектура системы DiVTB (Distributed Virtual Test Bed - Распределенный Виртуальный Испытательный Стенд, РаВИС) [1, 2]. Одним из немаловажных компонентов этой системы является интерфейс пользователя, предоставляемый ему для взаимодействия с вычислительной средой и распределенными ресурсами. В качестве приложения, играющего роль такого интерфейса, ранее была начата разработка компонента DiVTB Portal [3].

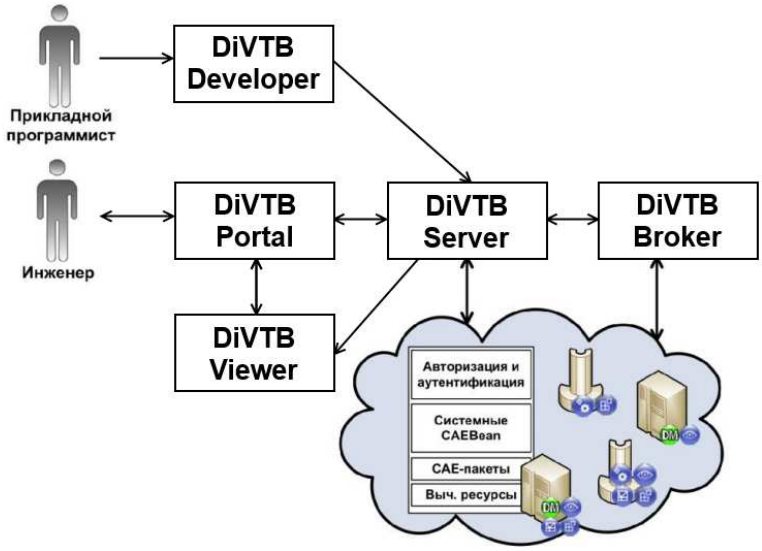


Рис. 1. Архитектура системы DiVTB

Компонент DiVTB Portal тесно взаимодействует с DiVTB Server [4], который отвечает за исполнение задач, а так же хранение данных, включая исходные файлы доступных для исполнения испытательных стендов.

Исполнение задач сервером DiVTB происходит на основе ресурсов, предоставляемых системой DiVTB Broker [6, 7]. Данная система производит сбор, анализ и распределение нагрузки в вычислительной среде между доступными вычислительными ресурсами.

Исходные файлы доступных для исполнения испытательных стендов формируются еще одним компонентом системы - DiVTB Developer [5]. Данный компонент предоставляет разра-

* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 11-07-00478-а и Министерства образования и науки РФ (государственное задание 8.3786.2011).

ботчику интерфейс для формирования файлов проектов, включая, как разработку интерфейса испытательного стенда задач для DiVTB Portal, так и разработку описания процесса исполнения задачи для DiVTB Server.

Как правило, результаты работы инженерных пакетов представляют собой не малые объемы данных, что затрудняет их передачу конечному пользователю системы, а так же требуют дальнейшей обработки для проведения анализа. Все это предъявляет дополнительные требования к производительности вычислительных устройств пользователей. Данную задачу решает последний компонент системы DiVTB - сервис интерактивной визуализации DiVTB Viewer [8].

2. Архитектура DiVTB Portal

Инженер, взаимодействуя с web-интерфейсом DiVTB Portal (**Рис. 2**), получает возможность просматривать список доступных ему испытательных стендов, задавать параметры для конкретной задачи и запускать ее на исполнение. Web-интерфейс отвечает за аутентификацию пользователей системы и определение прав доступа к ее компонентам.

Менеджер задач включает в себя функционал, как запуска заданий на исполнение, так и управления уже существующими задачами пользователя, независимо от их текущего состояния.

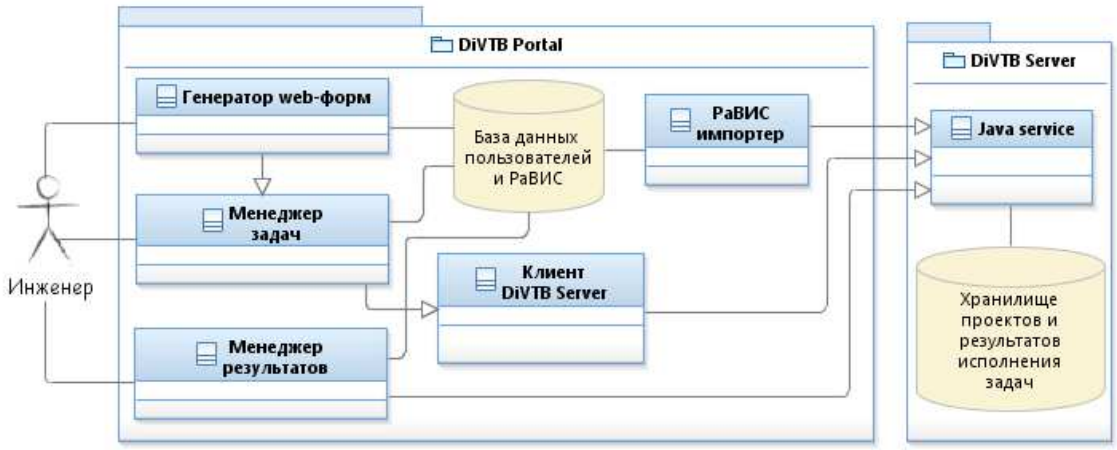


Рис. 2. Архитектура DiVTB Portal

Генератор web-форм отвечает за хранение проблемных оболочек DiVTB и автоматическую генерацию форм, необходимых для постановки задач. Сформированные задачи в дальнейшем могут быть запущены на исполнение с помощью менеджера задач.

Клиент DiVTB Server содержит в себе основы реализации взаимодействия с DiVTB Server и предоставляет всем остальным компонентам web-портала, которым, для реализации своего функционала, необходимо взаимодействие с сервером, удобный интерфейс.

Менеджер результатов исполнения задач, позволяет пользователю DiVTB Portal получить как результаты решения задачи в том виде, в котором их предоставляет конкретный пакет, так и информацию о ходе самого процесса проведения расчетов. *Импортер РаВИС* (распределенных виртуальных испытательных стендов) позволяет обновлять и поддерживать в актуальном состоянии список доступных распределенных виртуальных испытательных стендов, хранящихся на DiVTB Server.

3. Взаимодействие web-портала с компонентом DiVTB Server

Согласно архитектуре системы DiVTB (**Рис. 1**), разрабатываемый web-портал осуществляет взаимодействие с двумя компонентами: DiVTB Server и DiVTB Viewer. В текущей реализации системы DiVTB, DiVTB Server предоставляет web-порталу доступ к следующему функционалу:

- Получение оболочек поддерживаемых распределенных виртуальных испытательных стендов;

- Формирование задач на основе конкретного испытательного стенда, путем задания различных параметров;
- Запуск и отслеживание состояния задач пользователей DiVTB Portal;
- Получение результатов исполнения задачи в виде zip-архива, содержащего файлы, соответствующие используемым программным пакетам.

3.1 Импорт доступных испытательных стендов в DiVTB Portal

С течением времени список испытательных стендов, доступных в системе DiVTB, меняется. Это обусловлено появлением новых, изменением существующих или удалением уже не используемых стендов. Удаление испытательного стенда или изменение его параметров могут привести к потере данных ранее запущенных пользователями задач. Отсюда вытекает необходимость поддержания локальной базы данных для DiVTB Portal, хранящей различные версии когда-либо использовавшихся испытательных стендов.

Все доступные для исполнения испытательные стенды размещаются в хранилище проектов DiVTB Server. Каждый испытательный стенд, имеет свой уникальный идентификатор uid, опираясь на который DiVTB Portal может определить наличие или отсутствие его в своей локальной базе данных. Для поддержания базы данных доступных распределенных виртуальных испытательных стендов системы DiVTB в web-портал был добавлен компонент осуществляющий их импорт из хранилища DiVTB Server. Для достижения данной цели были использованы методы компонента DiVTB Server, приведенные на **Рис. 3**.

```
String[] getProjectIDs();
ProblemCaebean getProblemCaebean(String caebeanUid);
```

Рис. 3. Методы компонента DiVTB Server, используемые при реализации подсистемы импорта испытательных стендов в DiVTB Portal

При осуществлении импорта испытательных стендов происходит получение идентификаторов всех доступных проектов с помощью метода getProjectIDs. Далее для каждого идентификатора проверяется его наличие или отсутствие в базе данных DiVTB Portal (**Рис. 4**).

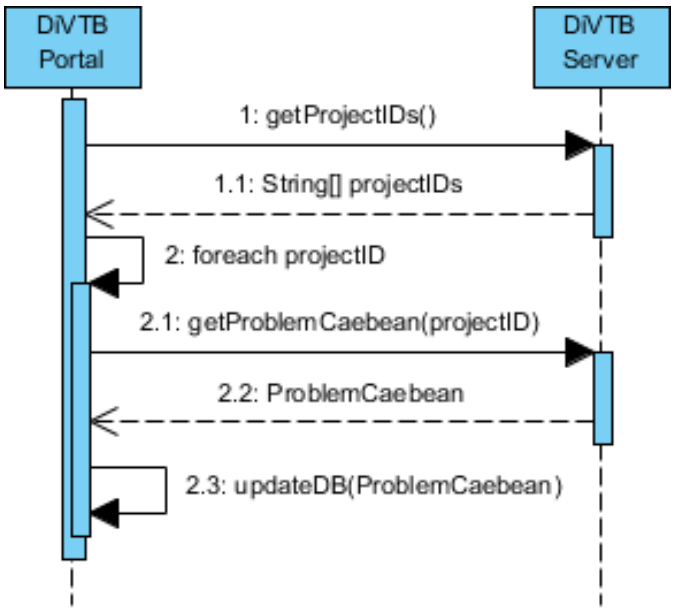


Рис. 4. Процесс обновления базы данных доступных испытательных стендов DiVTB Portal

В случае отсутствия проекта в базе данных у DiVTB Server запрашивается проблемная оболочка для данного проекта вызовом метода getProblemCaebean. Полученный испытательный стенд размещается в базе данных DiVTB Portal, после чего его можно сделать доступным для выбранных пользователей системы.

В случае, когда проект присутствует в базе данных DiVTB Portal, но отсутствует в базе данных DiVTB Server, данный проект переводится в состояние "не активен". Данное состояние означает невозможность для пользователей портала запускать задачи, основанные на данном испытательном стенде, при этом сохраняется возможность просмотра параметров запуска и результатов решенных ранее задач.

3.2 Формирование и запуск задачи

На основе выбранного пользователем испытательного стенда может быть сформирована и запущена на исполнение задача. Для реализации данной возможности были использованы методы, предоставляемые компонентом DiVTB Server, приведенные на **Рис. 5**.

```
String createInstance();
void uploadUpdates(String instanceUid,
                   Base64Binary zipArchive);
void submitJob(String instanceUid, ProblemCaebean job);
```

Рис. 5. Методы компонента DiVTB Server, использованные при реализации функции формирования и запуска задач порталом

Для формирования задачи необходимо создать instance в DiVTB Server, в рамках которого осуществляется вся дальнейшая работа. Сперва вызывается метод createInstance, который создает новый instance и возвращает присвоенный ему уникальный идентификатор uid. Полученный идентификатор, для удобства, используется порталом в качестве идентификатора задачи. Затем пользователь DiVTB Portal задает параметры испытательного стенда с помощью автоматически сгенерированной формы. Данные параметры дополняют испытательный стенд, тем самым формируя задачу, которая будет запущена в рамках созданного ранее instance вызовом метода submitJob.

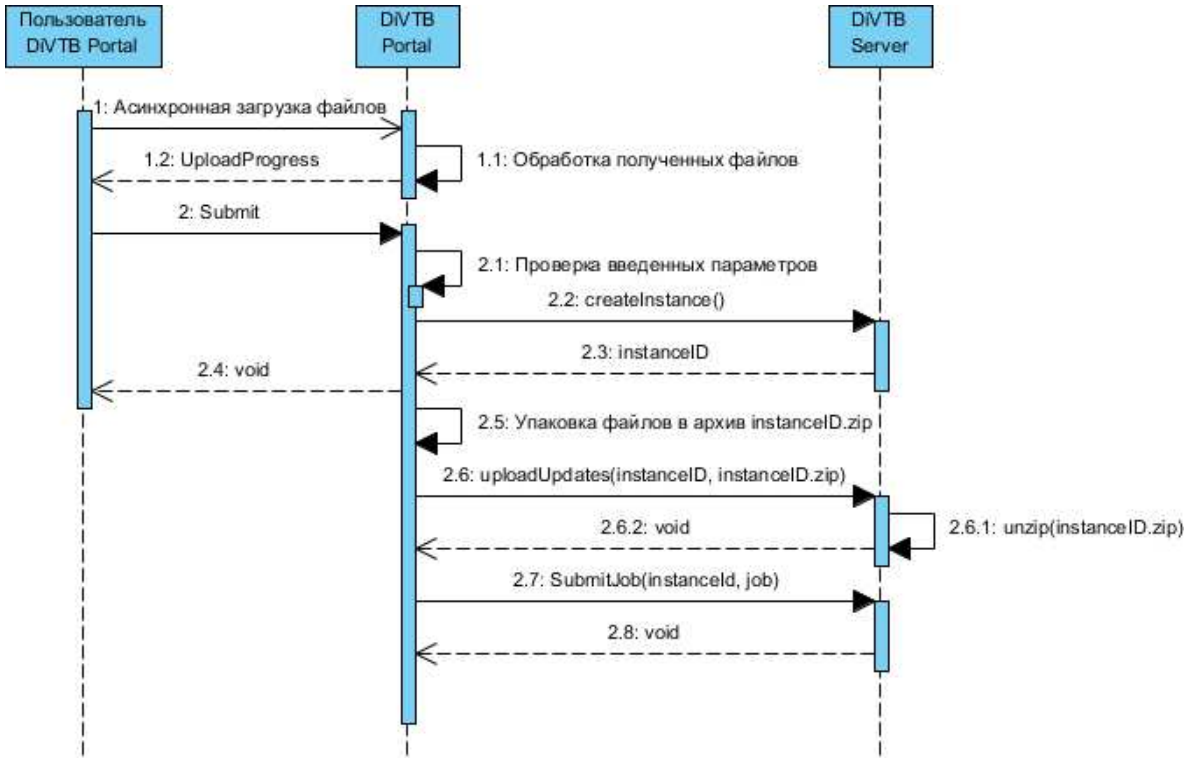


Рис. 6. Процесс передачи файлов в качестве параметров испытательного стенда

В процессе взаимодействия с пользователями системы DiVTB выяснилось, что для постановки некоторых задач требуется возможность передачи файлов ресурсам системы DiVTB в качестве входных параметров. Данный функционал был реализован в виде дополнительного действия по передаче zip-архива, содержащего входные файлы. На **Рис. 6** изображен процесс

передачи файлов в качестве параметров испытательного стенда. На этапе задания параметров пользователь выбирает файлы необходимые для загрузки в качестве входных параметров испытательного стенда. Данные файлы асинхронно загружаются в DiVTB Portal, где временно хранятся до момента запуска задачи на исполнение (**Рис. 7**).

Рис. 7. Интерфейс пользователя DiVTB Portal при асинхронной загрузке файлов

В момент запуска все исходные параметры, включая файлы, проходят процесс проверки на корректность. После этого файлы, при наличии, упаковываются в zip-архив. Далее для отправки архива с файлами DiVTB Portal вызывает метод uploadUpdates, который загружает и распаковывает его в "рабочей директории" задачи на DiVTB Server. В свою очередь, при запуске задачи параметры, имеющие тип "File", будут содержат имена загруженных пользователем файлов.

3.3 Мониторинг и управление задачами пользователя

Все задачи в системе DiVTB могут находиться в одном из пяти состояний:

- NOT_STARTED - начальное состояние, соответствующее еще не запущенной задаче;
- RUNNING - промежуточное состояние, присваивающееся задаче непосредственно в момент вызова метода submitJob;
- HELD - промежуточное состояние, присваивающееся задачам, исполнение которых временно приостановлено по каким-либо причинам;
- SUCCESSFULL - конечное состояние, соответствующее успешному завершению исполнения задачи;
- FAILED - конечное состояние, соответствующее экстренному завершению исполнения задачи в следствие возникновения каких-либо ошибок.

На **Рис. 8** приведены методы DiVTB Server, которые были использованы в DiVTB Portal для обновления информации о состоянии и работы с результатами уже завершенных задач пользователей.

```
String getStatus(String instanceUid);
long getExecTime(String instanceUid);
Base64Binary getResultsArch(String instanceUid);
Base64Binary getWorkArch(String instanceUid);
void removeResultDir(String instanceUid);
void removeWorkDir(String instanceUid);
```

Рис. 8. Методы компонента DiVTB Server, использованные при реализации работы с задачами пользователя DiVTB Portal

При обращении пользователя портала к задачам, имеющим не конечное состояние, портал использует метод getStatus для получения текущего состояния каждой задачи. В случае изменения состояния с начального или промежуточного на конечное, помимо самого состояния, дополнительно запрашивается время исполнения задачи с помощью вызова метода getExecTime.

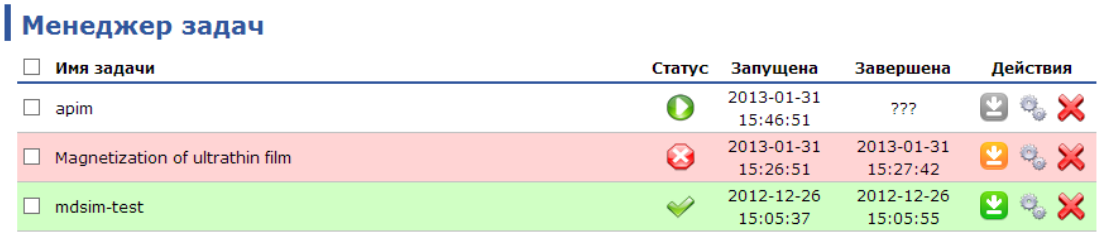


Рис. 9. Интерфейс пользователя DiVTB Portal при получении результатов исполнения задач с различным конечным состоянием

При достижении задачей одного из конечных состояний, пользователю предоставляется возможность получить либо финальный результат, либо промежуточные итоги исполнения, в зависимости от факта успешности завершения (**Рис. 9**).

При удалении пользователем DiVTB Portal какой-либо задачи так же удаляются и результаты ее исполнения, хранящиеся в DiVTB Server. Конечный результат имеется только у задач, находящихся в состоянии SUCCESSFULL. Результаты этих задач будут удаляться вызовом метода removeResultDir, а во всех остальных случаях - вызовом метода removeWorkDir.

4. Интерфейс постановки задачи

При работе с интерфейсом DiVTB Portal пользователь может столкнуться с тремя различными способами задания параметров задач:

- Простое указание значения параметра (любые числа и строки);
- Выбор из ограниченного набора значений параметра;
- Передача файла в качестве параметра задачи.

Генератор web-форм DiVTB Portal позволяет пользователям задавать параметры испытательных стендов, путем указания их значений в текстовом поле ввода (**Рис. 10**). Параметры, в данном случае, могут иметь различный тип, в том числе: String, Int, Float, File и т.д. Указание о том, какой формат имеет параметр размещаются разработчиком интерфейса в комментариях.

Рис. 10. Пример элемента интерфейса DiVTB Portal для задания значения параметра

При создании испытательного стенда в DiVTB Developer для любого параметра, за исключением параметров файлового типа, может быть ограничено множество их значений, с помощью перечисления всех возможных значений. При наличии такого перечисления поле ввода значения параметра будет представлено пользователю системы в виде элемента выбора значения из предложенного списка (**Рис. 11**).

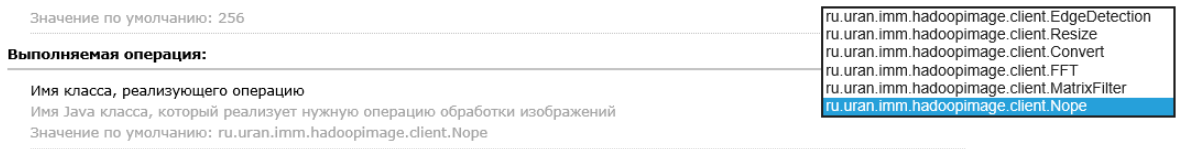


Рис. 11. Пример элемента интерфейса DiVTB Portal для задания значения параметра, при наличии ограниченного набора его возможных значений

Процесс задания и передачи параметров файлового типа был рассмотрен ранее в разделе 3.2 Формирование и запуск задачи. На **Рис. 7** изображен пример интерфейса портала до выбора файла пользователем и после его загрузки в хранилище файлов портала. Параметры типа File сочетают в себе 2 функции:

- обеспечивают загрузку выбранных файлов на вычислительный узел при запуске задачи;

- передают исполнителю задачи одноименный параметр типа String, в котором содержится имя загруженного пользователем файла.

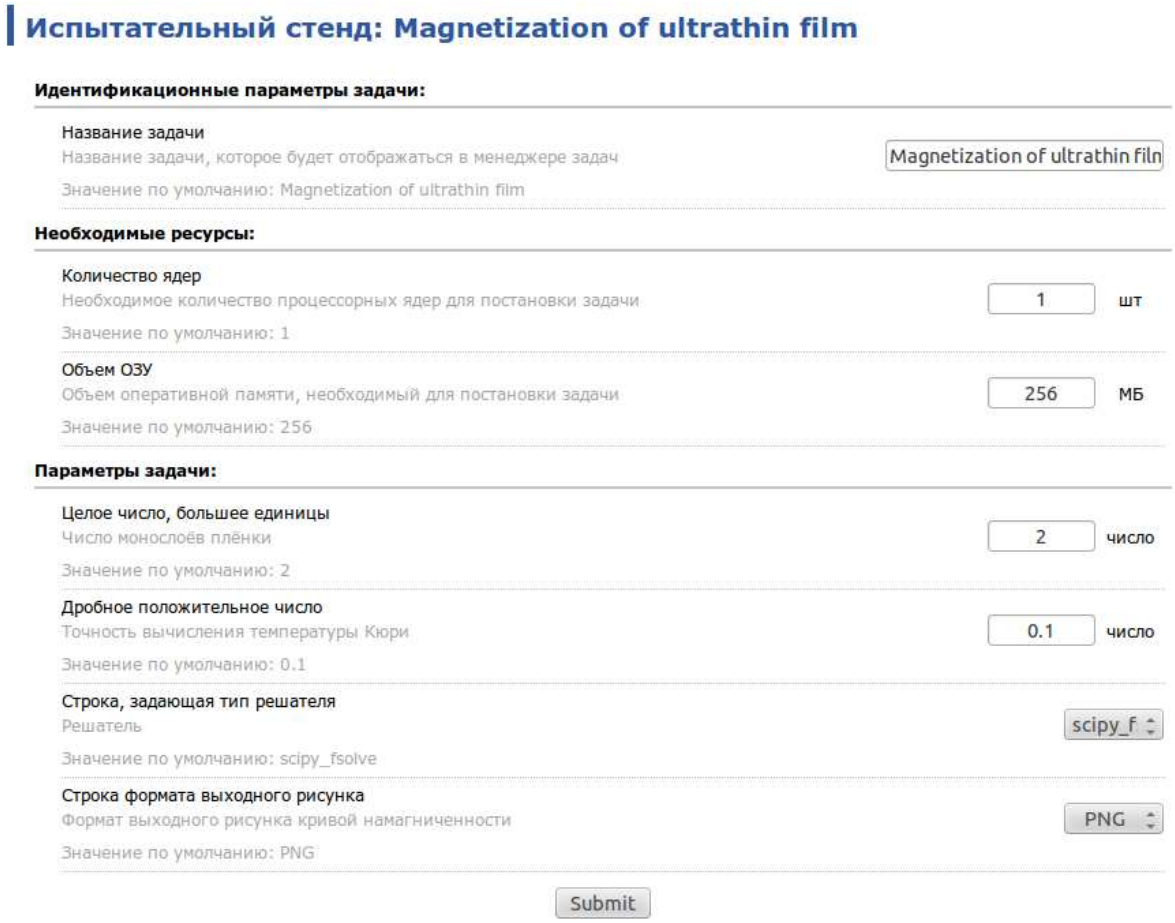


Рис. 12. Пример интерфейса DiVTB Portal для задачи "Magnetization of ultrathin film"

На **Рис. 12** приведен пример интерфейса испытательного стенда DiVTB Portal для задачи «Magnetization of ultrathin film».

5. Заключение

В данной работе были рассмотрены архитектурные особенности реализации компонента DiVTB Portal. Были освещены особенности взаимодействия разработанного портала с компонентом DiVTB Server.

Дальнейшее развитие DiVTB Portal будет направлено на совершенствование архитектуры, развитие функций администрирования портала и интеграцию с сервисом интерактивной визуализации DiVTB Viewer.

Литература

1. Радченко Г.И. Распределенные виртуальные испытательные стенды: использование систем инженерного проектирования и анализа в распределенных вычислительных средах // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". 2011. № 37(254). Вып. 10. С. 108-121.
2. Radchenko G., Hudyakova E. A Service-Oriented Approach of Integration of Computer-Aided Engineering Systems in Distributed Computing Environments // UNICORE Summit 2012 Proceedings. Forschungszentrum Julich, 2012. P. 57-66.

3. Захаров Е.А. Веб портал для проведения виртуальных экспериментов в распределенных вычислительных средах // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (17-22 сентября 2012 г., г. Новороссийск). М.: Изд-во МГУ, 2012. С. 257-260.

4. Савченко Д.И., Радченко Г.И. DiVTB Server: среда выполнения виртуальных экспериментов. См. настоящий сборник

5. Худякова Е.С., Радченко Г.И. Веб-система разработки и внедрения распределенных виртуальных испытательных стендов // Материалы XII Всероссийской конференции "Высокопроизводительные параллельные вычисления на кластерных системах" (Нижний Новгород, 26-28 ноября 2012 г.). Нижний Новгород: Изд-во Нижегородского госуниверситета, 2012. С. 430-434.

6. Шамакина А.В. Брокер ресурсов для поддержки проблемно-ориентированных сред // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. № 46(305). Вып. 1. С. 88-98.

7. Shamakina A. Brokering Service for Supporting Problem-Oriented Grid Environments // UNICORE Summit 2012 Proceedings, Forschungszentrum Julich, 2012. P. 67-75.

8. Диков Д.А., Радченко Г.И. Интеграция сервиса интерактивной визуализации для распределенных виртуальных испытательных стендов с системой CAEBeans // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XII Всероссийской конференции (Нижний Новгород, 26-28 ноября 2012 г.). Нижний Новгород: Изд-во Нижегородского госуниверситета, 2012. С. 134-138.

Задача прогнозирования нагрузки для повышения энергетической эффективности вычислительного кластера*

Е.Е. Ивашко, А.С. Румянцев, А.Л. Чухарев

Федеральное государственное бюджетное учреждение науки
Институт прикладных математических исследований
Карельского научного центра Российской академии наук

Исследуется задача прогнозирования нагрузки для построения системы повышения энергоэффективности вычислительного кластера. Предложены величины, отражающие качество восприятия системы пользователем, способ построения прогноза и метод оценки качества прогноза.

Введение

Многопроцессорные системы (МС) представляют собой активно развивающееся в настоящее время направление наращивания мощности вычислителей [1]. Среди МС следует выделить вычислительные кластеры (ВК) и системы распределенных вычислений (СРВ). Архитектура ВК такова, что множество процессорных ядер, оперативная память, дисковая подсистема, быстрая вычислительная сеть воспринимаются пользователем как единый аппаратный ресурс. ВК позволяют вести одновременный расчет заявки на множестве процессоров. СРВ (например, грид-системы) обладают менее тесно связанными вычислителями. Типичная заявка для СРВ состоит из группы относительно независимых заданий, каждое из которых выполняется на отдельном процессоре.

Для целей оптимального разделения ресурсов МС между конкурирующими заявками, в МС используются менеджеры очередей (МО). Это программные системы, которые на основе данных о текущей нагрузке в МС и некоторой дополнительной информации управляют выбором задач из очереди, вычислением, приостановкой, завершением задач с учетом относительных приоритетов. Для МО характерно два типа поведения: нацеленные на получение наиболее плотного расписания и нацеленные на снижение времени отклика системы. Использование МО первого типа приводит к росту загрузки системы, однако, с точки зрения пользователя такая система может не являться комфортной для работы. МО второго типа нацелены на повышение комфорта пользователя, в то же время это может приводить к появлению простоев оборудования и снижению загрузки (за счет эффекта обратной связи загрузка в такой системе может увеличиваться с повышением комфорта пользователя). Таким образом, целью владельца, как правило, является эксплуатация оборудования без простоя, поскольку эксплуатационные расходы кластера являются весьма высокими. Целью пользователя является минимизация времени отклика.

ВК обладают высоким энергопотреблением, в то же время, существенная часть ВК недогружена [2], что дает возможность временно переводить часть устройств в режим пониженного энергопотребления (РПЭ). Нахождение компромисса между загрузкой кластера, временем отклика системы и энергопотреблением системы — комплексная задача, подразумевающая оптимальную настройку параметров системы, выбор МО, взаимодействие с пользователями. В этой связи отметим систему Moab HPC Suite Enterprise Edition [3], которая, по заявлению производителя, частично решает проблемы снижения энергопотребления системы за счет более плотной упаковки расписания, прогнозирования рабочих циклов

*Исследования выполнены при поддержке РФФИ (проект 12-07-31147) и Фонда содействия малым формам предпринимательства в научно-технической сфере (г/к №10491р/16862 от 08.06.2012).

нагрузки и выбора для расчета задачи наиболее энергоэффективных (с максимальным соотношением числа флопсов на ватт) узлов. Однако, такие системы повышения энергоэффективности (СПЭ), как правило, работают со сложным набором правил (см. также [4]), т. е. требуют индивидуального подхода в каждом отдельном случае.

1. Вероятностное моделирование процесса нагрузки ВК

Большой поток вычислительных задач и заранее неизвестное время выполнения задачи в МС приводят к необходимости использования вероятностных моделей для описания функционирования МС. Параллельные вычисления в качестве основы для вероятностной модели впервые упомянуты в работе [10]. Отметим работы [5–7], исследующие возможности моделирования ВК, а также работу [8], в которой проведено сравнение имеющихся моделей ВК на основе данных реальных лог-файлов ВК. Наконец, отметим работу [9], которая содержит обзор имеющихся в литературе моделей массово-параллельных МС, а также анализ некоторой новой модели на основе марковских цепей.

При этом, как правило, рассматривается следующая общая постановка задачи. В МС в случайные моменты времени $t_i, i \geqslant 1$ поступает поток заявок. Заявке i требуется (случайное или детерминированное) число процессоров N_i для выполнения заданий (в дальнейшем используем термин процессор для разделяемых вычислителей, которыми могут быть процессорные ядра, графические ускорители и пр.). Если $N_i \equiv 1$, говорят о классической МС типа GI/G/m. Если заявке с номером i одновременно требуется случайное число $N_i \geqslant 1$ процессоров (т. е. задания должны начать выполнение одновременно), то системы можно разделить на МС с независимым освобождением процессоров [11, 12] и МС с одновременным освобождением процессоров [13, 14]. В системах второго типа (наиболее подходящих для описания ВК) времена обслуживания на всех N_i процессорах идентичны, т. е. используется одна и та же реализация времени обслуживания S_i, что существенно усложняет анализ [14]. Для таких систем известны лишь численные результаты [13], а при отсутствии буфера — также некоторые аналитические результаты [10, 15, 16].

В работах [17, 18] предложена и исследована новая модель ВК, представляющая собой модификацию модели Кифера–Вольфовица для вектора нагрузки W_i в момент прихода заявки i, которой требуется одновременно N_i процессоров на время S_i. Рекурсия для вектора нагрузки имеет вид

$$W_{i+1} = R(W_{i,N_i} + S_i - T_i, \ldots, W_{i,N_i} + S_i - T_i, W_{i,N_i+1} - T_i, \ldots, W_{i,m} - T_i)^+, \qquad (1)$$

где $R(\cdot)$ есть отображение упорядочивания координат вектора по возрастанию и $(\cdot)^+ = \max(\cdot, 0)$. Проверка адекватности модели проводилась с использованием численного моделирования на основе лог-файлов ВК. Отметим, что основную сложность при использовании модели (1) составляет построение модели управляющей последовательности $\{T_i, S_i, N_i\}$, которая обладает сложной структурой взаимосвязей между с. в., не являющимися н. о. р.

Напомним особенности функционирования m-процессорного ВК в режиме разделения доступа между пользователями из множества пользователей кластера $U = \{u_1, \ldots, u_n\}$ с использованием сессий удаленного доступа. Пусть $\tau_i(u), i \geqslant 1$ есть времена начала сессий доступа к ВК и $\sigma_i(u)$ есть продолжительность сессии с номером i для пользователя u. Тогда активные в момент времени t сессии пользователя u (т.е. такие, что $\tau_i(u) \leqslant t \leqslant \tau_i(u) + \sigma_i(u)$) порождают поток заявок, каждая из которых характеризуется временем прихода $t_j(u)$, требуемым числом процессоров $N_j(u)$ и временем обслуживания $S_j(u)$ (которое может быть заранее не известно, либо известно приближенно). Суперпозиция потоков заявок от всех активных пользователей в каждый момент времени представляет собой управляющий поток заявок в системе. Будем называть загрузкой ВК величину

$$\rho = \lambda \frac{E(SN)}{m},$$

где λ есть интенсивность входного потока заявок (в случае однородного потока $\lambda = \frac{1}{ET}$), а без индексов обозначены типичные элементы последовательностей случайных величин.

2. Прогнозирование нагрузки ВК

Поскольку модель процесса нагрузки хорошо описывает поведение ВК на небольших, не сильно нагруженных системах [19], то указанную модель можно использовать в качестве источника информации о поведении процесса нагрузки при заданной реализации управляющей последовательности заявок. В случае, когда используется особая дисциплина обслуживания, несколько очередей или имеются другие особенности системы, возможно использование МО в режиме симуляции [20].

2.1. Оценка качества обслуживания на ВК

В качестве метрик, отражающих качество обслуживания на вычислительном кластере, как правило, используют следующие величины [2]: время отклика системы $D_i + S_i$; замедление заявки $\frac{D_i+S_i}{S_i}$; загрузка системы ρ; допустимая загрузка системы $\rho_{\max}$, не приводящая к неограниченному росту очереди. Отметим, что интенсивность входного потока заявок λ также косвенно отражает качество обслуживания на кластере. Это связано с наличием эффекта обратной связи между потоком заявок и состоянием системы [2].

Пусть в результате функционирования СПЭ на ВК часть устройств переводится в РПЭ, время выхода из которых различается. В случае необходимости задействования таких ресурсов для обслуживания заявки i обозначим $V_i \geqslant 0$ время до полной готовности всех ресурсов к обслуживанию данной заявки. Отметим, что в первом приближении можно считать распределение V_i дискретным. Тогда время отклика системы будет составлять величину $D_i + V_i + S_i$, среднее замедление заявки составит $\frac{D_i+V_i+S_i}{S_i}$. При этом с ростом значений V_i повышается экономия энергии (т.к. устройства будут дольше находиться в РПЭ), но ухудшается качество обслуживания в системе с точки зрения пользователя.

Необходимо отметить, что в некоторых случаях может наблюдаться одновременный рост средней загрузки в системе и снижение среднего замедления, либо снижение средней загрузки и рост среднего замедления [22]. Поэтому целесообразно определить величину, отражающую качество восприятия (Quality of Experience) системы пользователем на заявке с номером i следующим образом:

$$R_i = \alpha(D_i + V_i + S_i) + \beta\frac{D_i + S_i + V_i}{S_i},$$

где α, β— некоторые параметры, $0 \leqslant \alpha, \beta \leqslant 1,\ \alpha + \beta = 1$. В пределе при $\alpha = 1$ метрика R_i соответствует времени отклика системы для заявки i, а при $\beta = 1$ соответствует замедлению заявки. Тогда качество обслуживания есть некоторая функция $F(R_1, \ldots, R_n)$. В простейшем случае можно полагать

$$F(R_1, \ldots, R_n) = \frac{1}{n}\sum_{i=1}^{n} R_i.$$

Для оценки изменения качества обслуживания в системе с введением системы повышения энергетической эффективности можно воспользоваться отношением

$$\frac{F(R_1, \ldots, R_n)}{F(\hat{R}_1, \ldots, \hat{R}_n)},$$

где $\hat{R}_i = \alpha(D_i + S_i) + \beta\frac{D_i+S_i}{S_i}$ есть качество восприятия на заявке i без использования системы повышения энергоэффективности. Тогда можно полагать, что указанная система

не причиняет ущерб качеству обслуживания, если

$$P\left(\frac{F(R_1,\ldots,R_n)}{F(\hat{R}_1,\ldots,\hat{R}_n)}\in[1,1+\varepsilon)\right)=1-\delta,$$

где $0\leqslant\varepsilon,\delta\ll 1$ — некоторые параметры.

2.2. Метод построения прогноза

Предположим, что в момент времени T в системе имеется свободный ресурс из K устройств. Если в системе нет очереди, то число свободных устройств до ближайшего прихода заявки не уменьшится, а в момент прихода заявки это значение может уменьшиться. Поэтому целесообразно принимать решение о переводе устройств в РПЭ до ближайшего прогнозируемого прихода. Если же в системе очередь заявок не пуста, то ближайшее уменьшение числа свободных устройств возможно либо в момент ближайшего прихода заявки (если $N_j<K$ и используется дисциплина обслуживания, отличная от FIFO), либо в момент ближайшего ухода заявки и начала обслуживания следующих по порядку очереди заявок. Таким образом, перевод устройств в РПЭ может производиться до минимального по времени момента из двух: ближайшего прогнозируемого прихода и ближайшего прогнозируемого ухода. Обозначим $\hat{t}$ момент времени, в который в соответствии с прогнозом ожидается уменьшение свободного ресурса устройств. Пусть $\hat{N}$ есть ожидаемый размер уменьшения ресурса в момент $\hat{t}$. Целесообразно производить построение прогноза в момент T окончания обслуживания очередной заявки (при этом могут начать обслуживание одновременно несколько заявок из очереди). Если в момент $T+0$ остался свободный ресурс, то можно планировать его перевод в РПЭ на время $\hat{t}-T$. Корректировка прогноза может производиться в следующие ключевые моменты: приход или уход очередной заявки, начало или окончание сессии пользователя.

Предположим, что каждое устройство имеет несколько режимов, C_0 (режим обычной работы), $C_1,\ldots,C_k$ (РПЭ, больший индекс соответствует большей экономии). Каждый РПЭ характеризуется величиной t_k времени, такого что на время, меньшее t_k, переход в режим C_k нецелесообразен (в частности, t_i связано с временем ухода и возвращения из состояния C_i). Тогда, на основании прогноза $\hat{t},\hat{N}$ необходимо построить вектор $\kappa=(\kappa_0,\ldots,\kappa_k)\in\mathbb{Z}_+^{k+1}$, такой что $\kappa_0+\cdots+\kappa_k=K$, в котором κ_i есть количество устройств, которое необходимо перевести в состояние C_i, а также вектор $\tilde{t}=(\tilde{t}_0,\ldots,\tilde{t}_k),0\leqslant\tilde{t}_i\leqslant\hat{t}$, состоящий из времен выхода из соответствующих состояний. Таким образом,

$$(\kappa,\tilde{t})=\varkappa(\hat{t},\hat{N},t_1,\ldots,t_k).$$

В простейшем случае можно определить функцию $\varkappa$, исходя из следующих соотношений. Если $\hat{t}<t_1$, то $\kappa=(K,0,\ldots,0),\tilde{t}=(\hat{t},0,\ldots,0)$. Если $t_i\leqslant\hat{t}\leqslant t_{i+1}$, то $\kappa_i=K-\hat{N},\kappa_{i-1}=\hat{N},\tilde{t}_i=\hat{t},\tilde{t}_{i-1}=\hat{t}-t_{i-1}$, а все остальные величины искомых векторов нулевые. В более общем случае, функция $\varkappa$ позволяет управлять качеством обслуживания в системе, поэтому более точное определение функции $\varkappa$ является предметом дальнейших исследований, в том числе с использованием численных экспериментов.

Опишем более подробно способ построения величин $\hat{t}$ и $\hat{N}$. Рассмотрим систему, в которой очередь пуста. Тогда в момент построения прогноза T пользователь $u\in U$ является активным, если найдется индекс i такой, что $\tau_i(u)\leqslant T\leqslant\tau_i(u)+\sigma_i(u)$, в противном случае пользователь не активен. Для активного пользователя u известно ближайшее к моменту T время постановки задачи в очередь, т.е. $\max\{t_j(u):t_j(u)<T\}$. Для не активного пользователя u известно время окончания ближайшей сессии, $\tau_i(u)+\sigma_i(u)$. Для каждого активного пользователя произведем реализацию $\hat{t}(u)-t_i(u)$ случайной величины (с.в.) с ф.р. $F_A(x)$, т.е. возможное время до прихода ближайшей заявки от данного пользователя, а также реализацию с.в. $\hat{\sigma}(u)$ с ф.р. $F_{ON}(x)$ длительности активной сессии. Если $\hat{t}(u)>t_i(u)+\hat{\sigma}(u)$, то

пользователь завершит сессию до постановки задачи, поэтому его необходимо рассмотреть повторно вместе с не активными. В противном случае величина $\hat{t}(u)$ представляет собой ближайшее время прихода задачи от данного пользователя. Одновременно необходимо произвести реализацию величины $\hat{N}(u)$ числа требуемых процессоров для заявки, пришедшей в момент времени $\hat{t}(u)$ от пользователя u (отметим также, что для особых дисциплин обслуживания также необходимо произвести выборку из распределения $F_B(x)$ прогнозируемого времени выполнения $\hat{S}(u)$ ближайшей заявки от пользователя u для расстановки относительных приоритетов в очереди). Далее для каждого не активного пользователя реализуем величину $\hat{\tau}(u) - \tau_i(u) - \sigma_i(u)$ с ф.р. $F_{OFF}(x)$ длительности периода неактивности пользователя, тем самым получаем момент начала новой сессии, а также аналогично случаю с активным пользователем находим время прихода ближайшей заявки данного пользователя $\hat{t}(u)$ реализацией с.в. $\hat{t}(u) - \hat{\tau}(u)$ с ф.р. $F_A(x)$ (отметим, что первый интервал прихода после начала новой сессии может иметь другое распределение, в стационарном случае ф.р. имеет вид $\frac{1}{E(\hat{t}(u)-\hat{\tau}(u))}\int_0^x F_A(y)\,dy$). Аналогично, реализуем $\hat{N}(u)$. При этом предполагается отсутствие сессий без постановки задачи в очередь. Тогда прогнозируемые значения $\hat{t}, \hat{N}$ могут быть вычислены как значения функционалов

$$\hat{t} = \varphi(\hat{t}_1(U), \ldots, \hat{t}_n(U)),$$
$$\hat{N} = \psi(\hat{N}_1(u), \ldots, \hat{N}_n(u))$$

где $\hat{t}_i(U)$ есть вектор ближайших прогнозируемых приходов заявок пользователей U после момента времени T, полученный в i-той реализации описанного выше процесса ($N_i(U)$ в очевидных обозначениях). Отметим, что точность прогноза зависит как от числа реализаций n, так и от вида функционалов $\varphi(\cdot), \psi(\cdot)$. В простейшем случае можно полагать $\phi(\cdot) = \psi(\cdot) = \min(\cdot)$. В то же время можно предложить также более гибкий и одновременно простой вид функционала, который позволяет управлять энергоэффективностью, а именно

$$\varphi(\hat{t}_1(U), \ldots, \hat{t}_n(U)) = \min_{1 \leqslant i \leqslant n} \hat{t}_i^{(j)}(U),$$

где $x^{(j)}$ есть j-я порядковая статистика вектора x. Меняя значение j, можно (при возрастании j) увеличивать энергоэффективность, одновременно повышая риск нарушения прогноза и снижая качество обслуживания. Функционал $\psi(\cdot)$ можно определить аналогично.

Отличие случая с непустой очередью заключается в том, что кроме вычисления прогнозируемых векторов $\hat{t}_i(U), 1 \leqslant i \leqslant n$ приходов ближайших заявок и размеров $\hat{N}_i(U)$, необходимо также прогнозировать ближайшее время завершения вычисления заявки на ВК (среди заявок, уже занимающих ресурсы ВК), а также рассматривать заявки, которые уже находятся в очереди, исследуя возможность их постановки на выполнение в момент ближайшего прогнозируемого ухода заявки. При этом необходимо учитывать возможность одновременного начала выполнения нескольких заявок на освободившемся ресурсе. Тогда ближайшее прогнозируемое время $\hat{t}$ уменьшения свободного ресурса необходимо вычислять в виде функционала вида

$$\hat{t} = \varphi'(\hat{t}_1(U), \ldots, \hat{t}_n(U), \hat{S}, N_{i_1}, \ldots, N_{i_s}),$$

где $\hat{t}_i(U)$ есть вектор ближайших прогнозируемых приходов заявок от пользователей U (определяемый аналогично случаю пустой очереди), $\hat{S}$ есть ближайшее прогнозируемое время ухода заявки с обслуживания, $\hat{\nu}$ есть величина свободного ресурса в момент ухода заявки (с учетом освободившихся процессоров), а $N_{i_1}, \ldots, N_{i_s}$ есть число процессоров, требуемых заявкам $i_1, \ldots, i_s$, находящимся в очереди в момент времени T. Прогнозируемое уменьшение ресурса $\hat{N}$ определяется аналогично в виде функционала

$$\hat{N} = \psi'(\hat{t}_1(U), \ldots, \hat{t}_n(U), \hat{S}, N_{i_1}, \ldots, N_{i_s}),$$

где в простейшем случае при реализации события {ближайший уход состоится раньше, чем ближайший приход} величина $\hat{N}$ определяется из соотношения

$$\hat{N} = \max_{b_k \in \{0,1\}, 1 \leqslant k \leqslant s} \left\{ \nu = \sum_{j=1}^{s} b_j N_{i_j} : \nu < \hat{\nu} \right\}.$$

2.3. Оценка качества прогноза нагрузки

Пусть в момент времени T появляется возможность перевести часть устройств в РПЭ. Тогда СПЭ должна с учетом прогноза $(\kappa, \tilde{t})$ рассчитать времена $\delta = (\delta_1, \ldots, \delta_m)$ простоя процессоров с номерами $i = 1, \ldots, m$. В случае более раннего, чем прогнозируемое, востребования ресурсов некоторых процессоров, СПЭ должна фиксировать максимальное время V_j до полной готовности всех ресурсов к принятию вновь пришедшей заявки с номером j, что отразится на качестве обслуживания в системе. Предположим, что заявка j поступает в систему в момент времени $t_j > T$. Пусть $b(j) = (b_1(j), \ldots, b_m(j))$, где $b_i(j) = 1$, если заявке j требуется устройство i и $b_i(j) = 0$ иначе, $i = 1, \ldots, m$. Тогда метрика ошибки прогноза нагрузки в момент прихода заявки j может быть некоторой функцией вида

$$Q(T, \delta, b(j), t_j, V_j).$$

Отметим, что целесообразно в качестве метрики для ошибки прогноза для узла i принять величину $\frac{(\delta_i - t_j + T)^+}{\delta_i}$, где $(\cdot)^+ = \max(0, \cdot)$. Тогда можно предложить в качестве метрики для ошибки прогноза использовать величину

$$Q(T, \delta, b(j), t_j, V_j) = CV_j \sum_{i=1}^{m} b_i(j) \frac{(\delta_i - t_j + T)^+}{\delta_i},$$

где в качестве константы C можно положить, например, $\frac{1}{|b(j)|}$. Тогда если прогноз нарушен незначительно ($t_j - T \approx \delta_i$ и $\delta_i \gg 0$), то $Q(T, \delta, b(j), t_j, V_j) \approx 0$, т.е. ошибка прогноза незначительна. Если же $t_j - T \approx 0$ и $\delta_i \gg 0$, то ошибка прогноза будет соответствовать накладным расходам от использования системы снижения энергопотребления, т.е. $Q(T, \delta, b(j), t_j, V_j) \approx V_j$. В то же время, если $V_j \approx 0$, то ошибку прогноза можно считать незначительной.

Для оценивания качества прогноза нагрузки, необходимо минимизировать ошибку прогноза на достаточно большом промежутке времени, что требует введения некоторой оценки, например, интегрального вида

$$Q = \int_{T_1}^{T_2} Q(T, \delta, b(j), t_j, V_j)\, dT.$$

2.4. Оценка энергетической эффективности

Наиболее популярной среди метрик оценки энергоэффективности ВК является PUE (Power Usage Efficiency) [21], которая отражает отношение энергии, потребляемой всем вычислительным комплексом (включая инфраструктуру) к энергии, потребляемой IT-компонентами комплекса (вычислителями, системами хранения, сетевой компонентой):

$$\text{PUE} = \frac{\text{Total facility power}}{\text{IT equipment power}}.$$

Необходимо отметить, что при внедрении СПЭ будет происходить рост PUE в связи с уменьшением суммарного и мгновенного значений энергопотребления. Таким образом, данный коэффициент будет некорректно отражать экономический эффект от внедрения указанной

системы. Поэтому для применения на ВК с СПЭ можно предложить метрику, отражающую коэффициент полезного действия системы повышения энергоэффективности комплекса в виде отношения суммарного энергопотребления за некоторый период времени с установленной системой к суммарному потреблению за этот же период без нее, т.е.

$$\frac{E(V,\rho)}{E_0}.$$

3. Постановка задачи и перспективы исследования

Суммируя вышеизложенное, можно поставить следующую задачу оптимизации: минимизация суммарного энергопотребления при сохранении качества обслуживания.

$$E(V,\rho) \to \min,$$

$$P\left(\frac{F(R_1,\ldots,R_n)}{F(\hat{R}_1,\ldots,\hat{R}_n)} \in [1,1+\varepsilon)\right) = 1-\delta.$$

В дальнейших исследованиях планируется провести ряд численных экспериментов для оценки качества прогноза нагрузки на основе данных лог-файлов ВК, в частности, ВК ЦКП КарНЦ РАН «Центр высокопроизводительной обработки данных» [23] с целью подбора оптимального вида функционалов, влияющих на качество прогноза, а также оптимальных значений параметров СПЭ.

Литература

1. Parkhurst J., Darringer J., Grundmann B. From single core to multi-core: preparing for a new exponential // Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design. ICCAD'06. New York, NY, USA: ACM, 2006. Pp. 67–72.

2. Feitelson D. G. Workload modeling for computer systems performance evaluation (web draft): URL: http://www.cs.huji.ac.il/ feit/wlmod/wlmod.pdf (дата обращения 06.12.2012)

3. Adaptive Computing: URL: http://www.adaptivecomputing.com (дата обращения 06.12.2012)

4. HP.com - HP Power Management Software: URL: http://h18004.www1.hp.com/products/servers/management/dynamic-power-capping/index.html (дата обращения: 17.08.2012).

5. Jann J., Pattnaik P., Franke H. et al. Modeling of Workload in MPPs // Job Scheduling Strategies for Parallel Processing, IPPS'97 Workshop, Geneva, Switzerland, April 5, 1997, Proceedings. Vol. 1291 of Lecture Notes in Computer Science. Springer, 1997. Pp. 95–116.

6. Feitelson D. G. Packing Schemes for Gang Scheduling // In Job Scheduling Strategies for Parallel Processing. Springer-Verlag, 1996. Pp. 89–110.

7. Gehring J., Preiss T. Scheduling a Metacomputer with Uncooperative Sub-schedulers // Proceedings of the Job Scheduling Strategies for Parallel Processing. IPPS/SPDP '99/JSSPP '99. London, UK, UK: Springer-Verlag, 1999. Pp. 179–201.

8. Talby D., Feitelson D. G., Raveh A. A Co-Plot analysis of logs and models of parallel workloads // ACM Transactions on Modeling and Computer Simulation (TOMACS). 2007. Vol. 17, no. 3.

9. Krampe A., Lepping J., Sieben W. A hybrid Markov chain model for workload on parallel computers // Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. HPDC '10. New York, NY, USA: ACM, 2010. Pp. 589–596.

10. Dijk N., Smeitink E. A non-exponential queueing system with batch servicing. Researchmemorandum, No. 13. Free University, Amsterdam. 1988.

11. Brill P., Green L. Queues in which customers receive simultaneous service from a random number of servers: A system point approach // Management Science. 1984. Vol. 30, no. 1. Pp. 51–68.

12. Green L. A Queueing System in Which Customers Require a Random Number of Servers // Operations Research. 1980. Vol. 28, no. 6. Pp. 1335–1346.

13. Kim S. M/M/s Queueing System Where Customers Demand Multiple Server Use: Dr. Sci. dissertation / Southern Methodist University. 1979.

14. Green L. Comparing operating characteristics of queues in which customers require a random number of servers // Management Science. 1980. Vol. 27, no. 1. Pp. 65–74.

15. Whitt W. Blocking when service is required from several facilities simultaneously // AT&T Technical Journal. 1985. Vol. 64, no. 8. Pp. 1807–1856.

16. Kaufman J. Blocking in a shared resource environment // IEEE Transactions on Communications. 1981. Vol. 29, no. 10. Pp. 1474–1481.

17. Морозов Е., Румянцев А. Некоторые модели многопроцессорных систем обслуживания с тяжелыми хвостами // Параллельные вычислительные технологии 2011: сборник трудов Международной научной конференции. Челябинск: ЮУрГУ, 2011. С. 555–566.

18. Румянцев А. О стохастическом моделировании вычислительного кластера // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем: Тезисы докладов Всероссийской конференции с международным участием (18–22 апреля 2011). Москва: РУДН, 2011. С. 46–47.

19. Морозов Е. В., Румянцев А. С. Вероятностные модели многопроцессорных систем: стационарность и моментные свойства // Информатика и ее применения. 2012. Т. 6. №3. С. 99–106.

20. Alejandro Luciero. Simulation of batch scheduling using real production-ready software tools: URL: www.bsc.es/media/4856.pdf (дата обращения 06.12.2012)

21. Green Grid metrics: describing data center power efficiency // Whitepaper. Green Grid, 2007.

22. Shmueli E., Feitelson D.G. On Simulation and Design of Parallel-Systems Schedulers: Are We Doing the Right Thing? // IEEE Transactions on Parallel and Distributed Systems. 2009. Pp. 983–996.

23. ЦКП КарНЦ РАН "Центр высокопроизводительной обработки данных": URL: http://cluster.krc.karelia.ru (дата обращения 06.12.2012)

Математическое моделирование отрывных течений в кольцевых соплах

М.А. Карташева

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно−Уральский государственный университет»
(национальный исследовательский университет)

Рассмотрены отрывные течения, возникающие при движении продуктов сгорания по тракту кольцевого сопла внешнего расширения. Предложена модель отрывной донной области, учитывающая основные физические факторы, действующие при течении продуктов сгорания в кольцевом сопле. Проведено математическое моделирование отрывных течений в кольцевых соплах различных конфигураций на различных режимах работы

1. Введение

Одной из важных задач математического моделирования течений в кольцевых соплах является моделирование процессов в отрывных зонах кольцевых сопел с укороченным центральным телом (рис. 1).

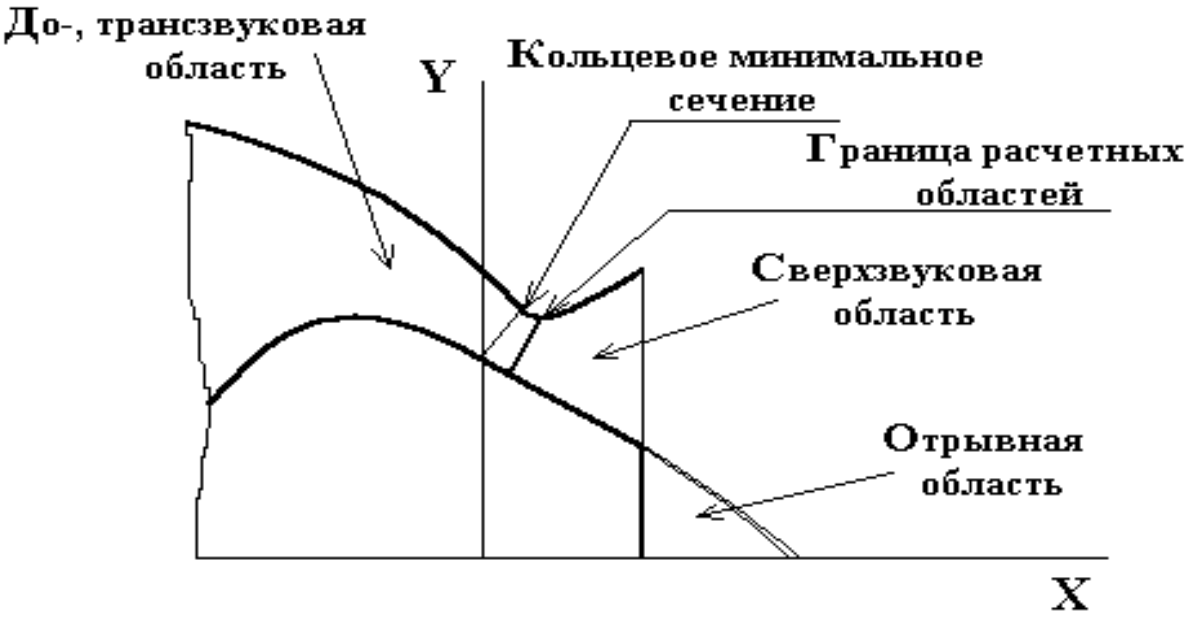


Рис. 1. Схема течения в кольцевом сопле с укороченным центральным телом

Дело в том, что при значительном укорочении центрального тела кольцевого сопла вклад профилированной части центрального тела в суммарную тягу сопла снижается вследствие уменьшения его длины, а вклад тяги, создаваемой торцем центрального тела, напротив, увеличивается вследствие роста площади торца и величины давления, действующего на него.

Расчет течения в отрывной зоне кольцевого сопла с укороченным центральным телом зависит от типа течения в донной области: является донная область открытой или замкнутой. В случае открытой донной области специальных расчетов параметров донной области не требуется, так как давление в ней с достаточной степенью точности равно давлению внешней среды. Поэтому основное внимание следует уделить исследованию динамики газа в замкнутых отрывных областях за торцем укороченного центрального тела.

Расчет параметров течения в рассматриваемой области может быть осуществлен двумя принципиально различными способами: с помощью прямого численного моделирования либо с помощью специально разработанных моделей отрывной донной области [1].

Математическое моделирование течения в отрывной области с помощью численных методов, в частности методов сквозного счета, позволяет определить параметры течения, не выделяя особенностей (разрывов) потока. Однако, при использовании разностных схем, адекватно описывающих течение в основном потоке газа, истекающем из кольцевого минимального сечения, где числа Рейнольдса составляют $Re \sim 10^5 - 10^7$, для расчета параметров отрывной области, где велико влияние сил вязкости и $Re \sim 10^2 - 10^4$, в этом случае необходимо внести изменения в вычислительный алгоритм, реализующие учет имеющихся в этой области течения вязких эффектов. То есть, потребуется разработка двух вычислительных алгоритмов (для расчета параметров «вязкой» и «невязкой» областей течения), что существенно усложняет задачу моделирования процессов в рассматриваемой области течения. Такое усложнение часто приводит к снижению точности расчетов в связи с необходимостью расчета параметров области вязко–невязкого взаимодействия, положение которой заранее неизвестно и должно быть определено непосредственно в процессе расчета.

Возможен также расчет параметров отрывной донной области с использованием алгоритмов для расчета невязких течений. При этом неизбежно снижается точность расчета параметров исследуемой области, но снимаются проблемы связанные со сходимостью и устойчивостью вычислительного алгоритма. Такие проблемы обычно возникают при «сшивании» различных вычислительных алгоритмов. Поэтому применение методов расчета невязких течений в данном случае является оправданным. В рамках данного исследования для расчета параметров отрывной донной области за торцем укороченного центрального тела кольцевого сопла внешнего расширения использован метод установления с применением схемы С.К. Годунова (модификация В.П. Колгана).

Другой способ заключается в разработке специальных методов расчета параметров отрывной области с использованием упрощающих положений, аналитических соотношений и эмпирических формул. Рассматриваемый подход имеет преимущества при исследовании сложных газодинамических и тепловых процессов в отрывных зонах в случаях, когда имеется необходимый объем экспериментальных исследований, позволяющий верифицировать разработанную модель отрывной донной области.

К методам расчета параметров отрывной донной области, реализующим данный подход, следует отнести методы разделяющей линии тока, предложенные Корстом и Чепменом, теорию смешения Крокко–Лиза, метод Карашимы и «интегральные» методы [2]. В этих моделях поток при больших числах Рейнольдса априорно разделен на области внешнего невязкого и внутреннего вязкого течений типа пограничного слоя и следа. Потенциальный внешний поток обтекает так называемое «тело вытеснения», распределение давления на котором, в первом приближении, совпадает с распределением давления на границе вязкого потока, при этом давление в отрывной области, а также поперек пограничного слоя и следа считается неизменным.

Метод разделяющей линии тока (РЛТ) основан на использовании простейшей модели тела вытеснения, состоящего только из двух областей: донной отрывной области, давление в которой предполагается постоянным и равным донному, и дальнего следа, давление в котором равно давлению невязкого потока. В этом случае основным параметром, определяющим границы областей такого течения, и является величина донного давления. Для ее определения необходимо рассмотреть течение в автомодельном вязком слое, развивающимся вдоль границы невязкого потока в изобарической области, при этом формулируется условие присоединения разделяющейся линии тока вблизи оси симметрии потока. Сама присоединяющаяся линия тока выбирается с учетом условия сохранения массы в отрывной области. Существует достаточно большое количество расчетных методов, относящихся к методам РЛТ, отличающихся друг от друга различиями в учете тех или иных факторов, например, толщины пограничного слоя в точке отрыва потока, а также различными зависимостями (в том числе эмпирическими) для профиля скорости в вязком слое смешения, зависимостями между параметрами вязкого слоя в сечениях его отрыва и присоединения.

Благодаря применению эмпирических и полуэмпирических зависимостей методы РЛТ позволяют получить достаточно хорошее совпадение расчетных и экспериментальных значений

величин донного давления в области отрыва потока. Как указано в работе [3], применение методов РЛТ достаточно эффективно в широком диапазоне сверхзвуковых скоростей и сравнительно небольших толщинах пограничного слоя, что вполне соответствует условиям работы рассматриваемых кольцевых сопел внешнего расширения летательных аппаратов.

Интегральные методы расчета отрывных течений основаны на использовании более сложной модели тела вытеснения, чем в методе РЛТ. В этом случае тело вытеснения включает в себя реальное тело, увеличенное на толщину вытеснения безотрывного пограничного слоя, область отрыва потока, возвратных течений и присоединения потока в ближнем следе, и определяется взаимодействием вязкого и невязкого потоков на значительной длине до сечения запирания следа. Данные методы основываются на априорной схеме течения, в соответствии с которой выделяются характерные области и слои невязкого и вязкого течений с заданными профилями скорости, температуры и другими параметрами течения, зависящими от некоторого числа свободных формпараметров [3].

2. Модель отрывной донной области течения в кольцевом сопле

В настоящей статье основное внимание уделено расчету параметров отрывной зоны за торцем укороченного центрального тела кольцевого сопла внешнего расширения. Для расчета параметров отрывной области целесообразно использовать более простые и наглядные методы разделяющей линии тока.

В настоящей статье для расчета параметров отрывной области предложен метод, относящийся к методам разделяющей линии тока и учитывающий свойства газового потока и различные физические процессы, протекающие при обтекании газом укороченного центрального тела. Метод построен на основе классического метода Корста [2] с использованием положений, разработанных в работах [4], с учетом осесимметричности течения. Схема течения, соответствующая рассматриваемому методу Корста, представлена на рис. 2.

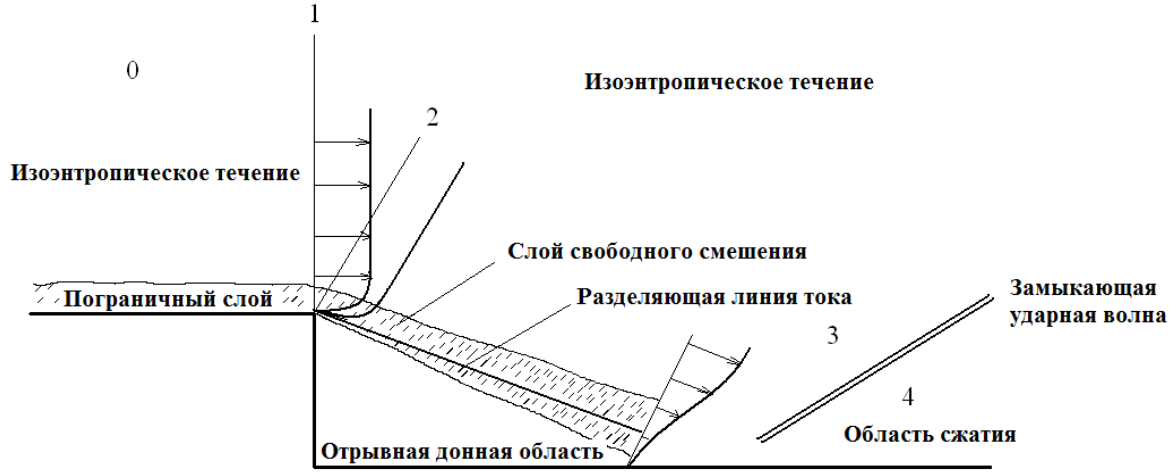


Рис. 2. Схема течения для расчета величины донного давления

Основные положения метода заключаются в следующем: поток, набегающий на донную часть вдоль двумерной поверхности, является звуковым или сверхзвуковым и остается сверхзвуковым после отрыва от угла. Образуются четыре области течения:

1. область между сечениями 0 и 1, поток набегает на заднюю кромку;
2. область между сечениями 1 и 2, поток расширяется при обтекании задней кромки;
3. область между сечениями 2 и 3, вблизи границ сжимаемой струи происходит смешение при постоянном давлении;
4. область 3 и 4, повторное сжатие в плоском скачке в конце отрывного течения.

В каждой из указанных газодинамических областей используются соответствующие соотношения, описывающие процессы в этих областях, которые представлены в работе [2]. Основные допущения теории Корста следующие:

– в области вязкого течения, на которое воздействует прилегающий почти однородный невозмущенный поток, статическое давление равно давлению в невозмущенном потоке: $p = p_i$;
– расширение внешнего потока между сечениями 1 и 2 происходит в соответствии с решением Прандтля–Майера;
– смешение на границе струи между сечениями 2 и 3 происходит при постоянном давлении: $p_2 = p_3$; безразмерный профиль скорости в слое смешения описывается соотношением: $\varphi = \frac{1}{2}\left(1 + erf\eta\right)$, где $\eta = \sigma\left(y/x\right)$, $C_{2a} = \left\{1 + \frac{2}{(k-1)M_н^{\,2}}\right\}$ и представлен на рис. 3;
– возрастание давления в области повторного сжатия в конце отрывной области течения определяется примыкающим внешним течением, а сжатие во внешнем потоке – плоским косым скачком между областями 3 и 4;
– в отрывной области масса газа должна сохраняться, условие смыкания линий тока получается из условия сохранения массы в отрывной зоне и применяется к линии тока, которая приходит в критическую точку области замыкания.

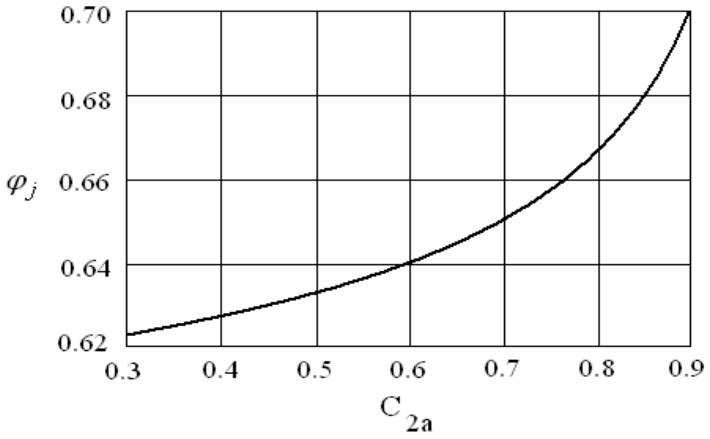


Рис. 3. Безразмерная скорость вдоль разделяющей линии тока

Рассмотренный метод расчета параметров донной области может быть положен в основу метода определения величины тяги, создаваемой торцем укороченного центрального тела. При этом метод для его эффективного применения должен быть адаптирован к условиям течения газа, существующим в кольцевых соплах. Описанная двумерная схема течения должна быть модифицирована для осесимметричного случая. Такая модификация предполагает учет осесимметричности газового потока с помощью уравнения расхода, определения осевой и радиальной координат точки присоединения разделяющей линии тока. Для осесимметричных течений присоединение и разворот невязкого потока при обтекании тел происходит при достижении разделяющей линией тока половины радиуса торца [5]. Для расчета параметров отрывной области за торцем центрального тела кольцевого сопла также возможно применение данного результата.

Дальнейшая модификация рассматриваемой модели связана с уже упомянутым учетом физических процессов и свойств газа при его движении в кольцевом сопле. Прежде всего, это учет влияния толщины начального пограничного слоя на величину донного давления [6]. Учет начальной толщины пограничного слоя в точке отрыва потока на выходной кромке укороченного центрального тела кольцевого сопла заключается в учете массы газа, содержащейся в пограничном слое, с помощью закона сохранения массы газа в области отрыва, а также корректировки безразмерного профиля скорости газа φ в слое смешения. Координата сечения присоединения потока при этом устанавливается из равенства площадей прямого и обратного тока в отрывной донной области.

Предлагаемый метод также учитывает особенности течения за торцем укороченного центрального тела в соответствии с подходом к расчету параметров отрывной области, изложенным в работе [7]. Используется математическая модель вязко–невязкого взаимодействия, основанная на рассмотрении вязкого течения в приближении пограничного слоя и невязкого течения при одной и той же величине донного давления p_d. Такая модель в целом соответствует идеологии взятого за основу метода Корста и может быть использована для его модификации. Ее основные положения заключаются в следующем:

– профили скорости пограничного слоя перед и за волной разрежения (сжатия) описываются степенными законами, величина полного давления постоянна вдоль линии в пограничном слое в пределах волны;
– за уступом, до точки соединения потоков, донные давление p_d и энтальпия H_d постоянны;
– для каждой пары величин p_d и H_d параметры на внешних границах зон смешения и максимальное давление в области соединения двух потоков равны параметрам соответствующих невязких потоков.

Безразмерный профиль скорости в слое смешения описывается соотношением:

$$\varphi = \frac{1}{2}\left[1 + erf\left(\eta - \eta_x\right)\right] + \frac{1}{\sqrt{\pi}} \int\limits_{\eta-\eta_x}^{\eta} \left(\frac{\eta - \beta}{\eta_x}\right)^{1/n} e^{-\beta^2} d\beta, \qquad (1)$$

где $\eta = \eta_x \dfrac{y}{\delta_1}$; $\eta_x = \dfrac{\sigma\delta}{1.5\left(x^2 + 4.4\sigma^2 x\delta\varepsilon_0^{-}\right)^{0.5}}$, β – параметр интегрирования, δ_1 – толщина пограничного слоя в точке отрыва потока.

В качестве условия присоединения потока, в отличие от модели Корста, в которой в качестве такого условия использовалось равенство полного давления в газе в слое смешения статическому давлению газа за косым скачком уплотнения за точкой присоединения потока, в данной модели используется другое условие, которое учитывает работу сил трения в пограничном слое и вязком слое смешения:

$$\frac{p_o}{p_d} = \frac{p_c}{p_d}\frac{1}{\overline{p}_{cr}}, \qquad (2)$$

где p_o – полное давление газа в слое смешения (на разделяющей линии тока), p_d – донное давление на торце центрального тела, p_c – статическое давление за косым скачком уплотнения в области присоединения потока, $\overline{p}_{cr}$ – критический перепад давления на скачке уплотнения, определяемый как мера работы сил трения в пограничном слое и зоне смешения.

Рассмотренный подход был развит в работе [4], в которой предложен приближенный метод расчета донного давления и энтальпии за плоским или осесимметричным уступом, обтекаемым сверхзвуковым потоком. Метод основан на использовании модели вязко–невязкого взаимодействия и позволяет определять параметры донного течения для произвольных толщин начального пограничного слоя при различных числах Маха. Метод является логическим продолжением работы [7] и вносит уточнения в соотношения параметров и условия, использованные ранее.

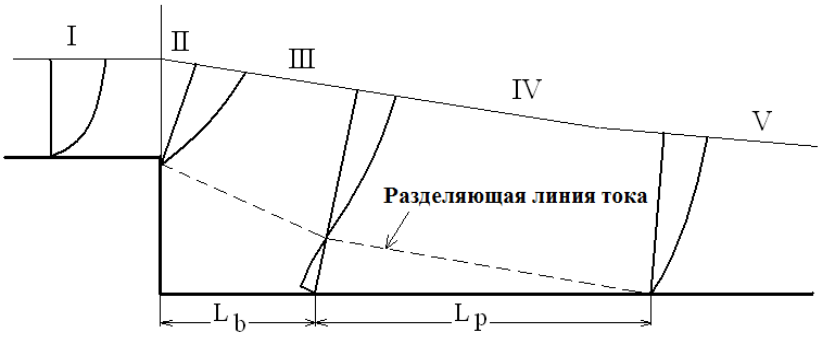


Рис. 4. Схема отрывного течения за осесимметричным донным уступом [6].

Течение за уступом содержит следующие характерные области (рис. 4):

I – набегающего сверхзвукового потока с пограничным слоем;

II – перехода потока и пограничного слоя через веер волн разрежения или скачок уплотнения у кромки уступа;

III – изобарическую длиной L_b, с давлением, приблизительно равным донному давлению p_d;

IV – повышения давления длиной L_p;

V – выравнивания давления.

Отличие от рассматриваемого ранее метода состоит в следующем: ранее предполагалось, что до точки встречи оторвавшегося потока на оси летательного аппарата давление постоянно и равно донному давлению, а в точке смыкания потока давление возрастает в замыкающем скачке уплотнения. В предлагаемом методе повышение давления в области присоединения происходит плавно, причем длина этой области в несколько раз больше, чем длина изобарической области за точкой отрыва потока. Длина L_p этой области определяется с помощью уравнения импульсов пограничного слоя в интегральной форме для осесимметричного течения – в виде уравнения баланса сил давления и трения, записанного для окрестности точки присоединения, где $u \approx v \approx 0$. Следствием внесенных изменений в модель отрывной области течения является несколько иной вид условия присоединения на оси потока и новое выражение для безразмерного профиля скорости на разделяющей линии тока. Соответствующие соотношения приведены в работе [4]. Экспериментальные данные, приведенные в данной работе, свидетельствуют о хорошем согласовании полученных в результате использования предлагаемой модели донной области расчетных данных с результатами экспериментов.

В настоящей работе предложена модель течения в отрывной донной области, основанная на модифицированной модели Корста, с учетом осесимметричности течения, толщины начального пограничного слоя в точке отрыва потока, работы сил трения в слое смешения и протяженности зоны повышения давления в области присоединения потока.

Предложенная модель реализована в вычислительном алгоритме и модуле расчета параметров отрывной области и величины донной тяги кольцевого сопла.

3. Результаты численных параметрических исследований

С помощью разработанного алгоритма проведены методические численные параметрические исследования зависимости газодинамических параметров и параметров теплового состояния в осесимметричной отрывной донной области при различных характеристиках потока в точке отрыва, характерных для течений в кольцевых соплах внешнего расширения и внешнего обтекания летательных аппаратов в диапазоне значений чисел Маха M=1.0…15.0. Полученные результаты представлены на рис. 5,6,7.

На рис. 8, 9, 10 представлены результаты математического моделирования параметров отрывной донной области (величины донного давления) с помощью методов сквозного счета. Сплошной тонкой линией обозначены результаты расчета по схеме С.К. Годунова–В.П. Колгана [8], пунктирной линией обозначены результаты расчета методом «крупных частиц» [9].

Проведено сравнение полученных результатов с результатами расчетов по предложенной модели отрывной донной области. Анализ результатов расчетов показывает хорошее согласование результатов, полученных с помощью численных методов и предложенной полуаналитической моделью.

Имеющиеся расхождения в результатах расчетов не превышают нескольких процентов практически во всем сравниваемом диапазоне. Для определения величины давления на торце укороченного центрального тела (либо днище летательного аппарата) такую погрешность следует считать достаточно низкой, так как современные эмпирические и полуэмпирические (учитывающие результаты экспериментальных исследований) методики предназначены, в основном, для работы в узких диапазонах параметров течения, давая при этом значительную погрешность (иногда в 2–3 раза). При этом следует отметить простоту предложенной модели, построенной по модульному принципу с учетом большинства значимых физических факторов,

действующих в рассматриваемой отрывной области течения. Это позволяет использовать предложенную модель для расчета параметров отрывной донной области за торцем укороченного центрального тела кольцевого сопла внешнего расширения , а также донной области летательных аппаратов в широком диапазоне полетных условий.

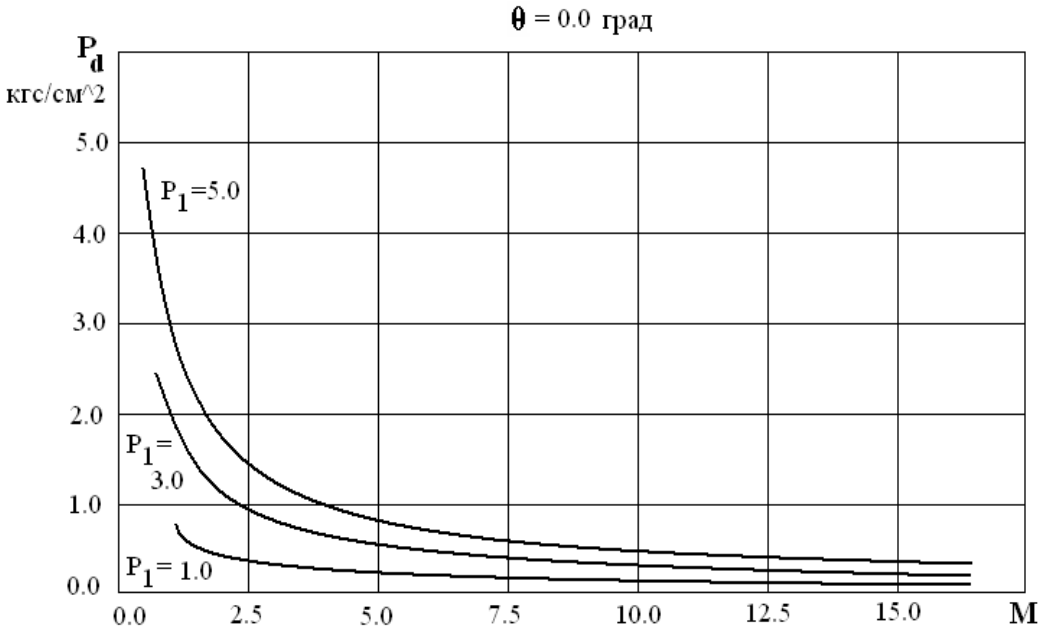


Рис. 5. Зависимость величины донного давления от числа Маха при различных давлениях в точке отрыва

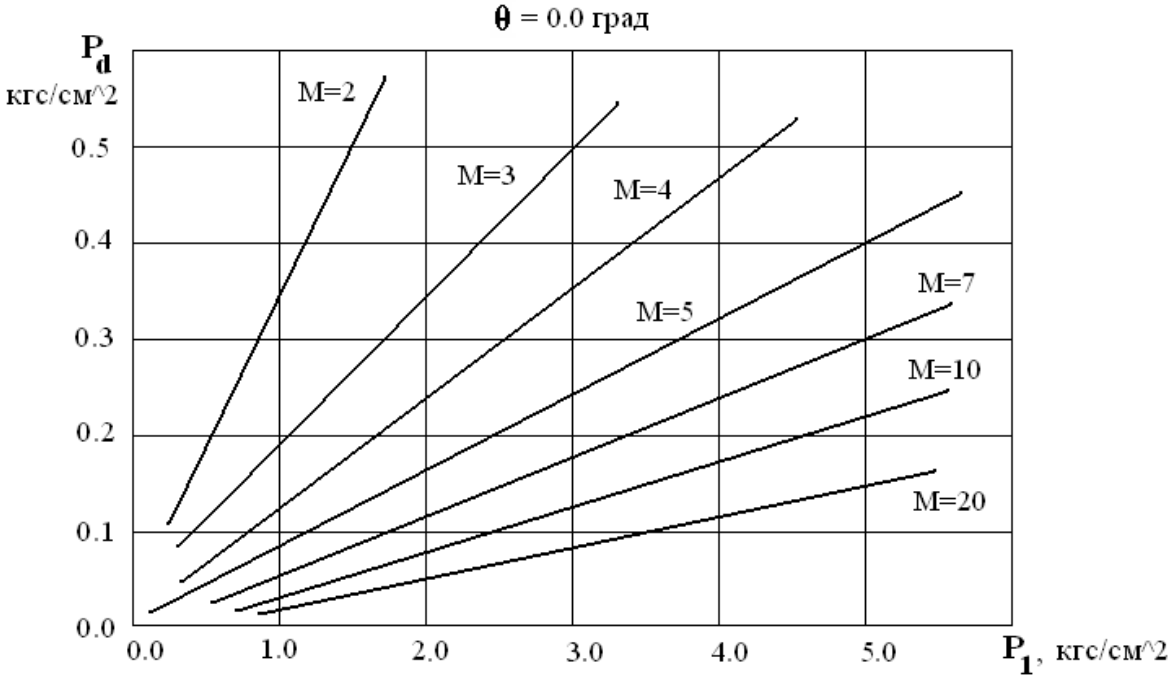


Рис. 6. Зависимость величины донного давления от давления в точке отрыва при различных числах Маха

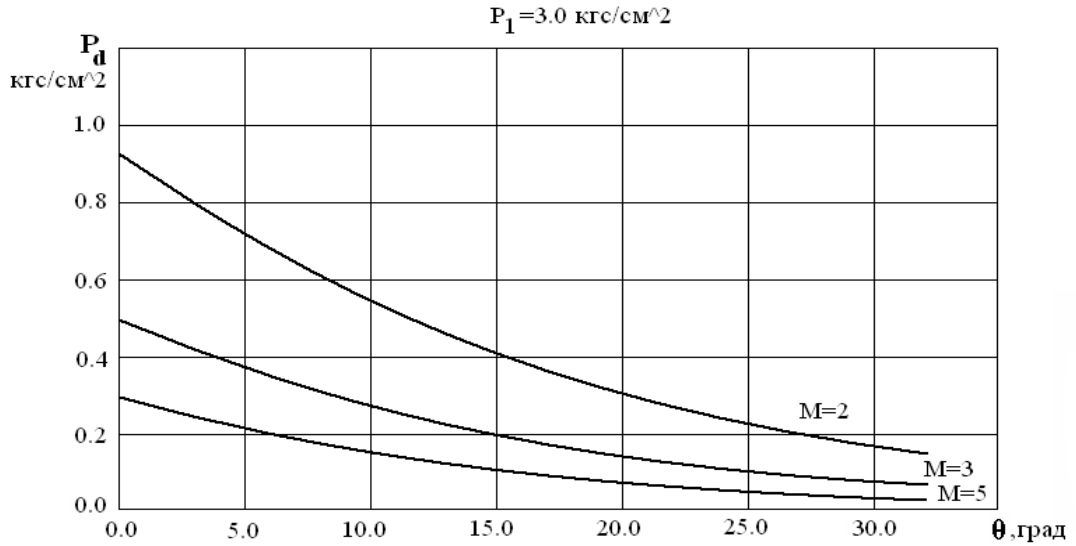


Рис. 7. Зависимость величины донного давления от угла наклона обтекаемой поверхности к продольной оси сопла (летательного аппарата) в точке отрыва потока

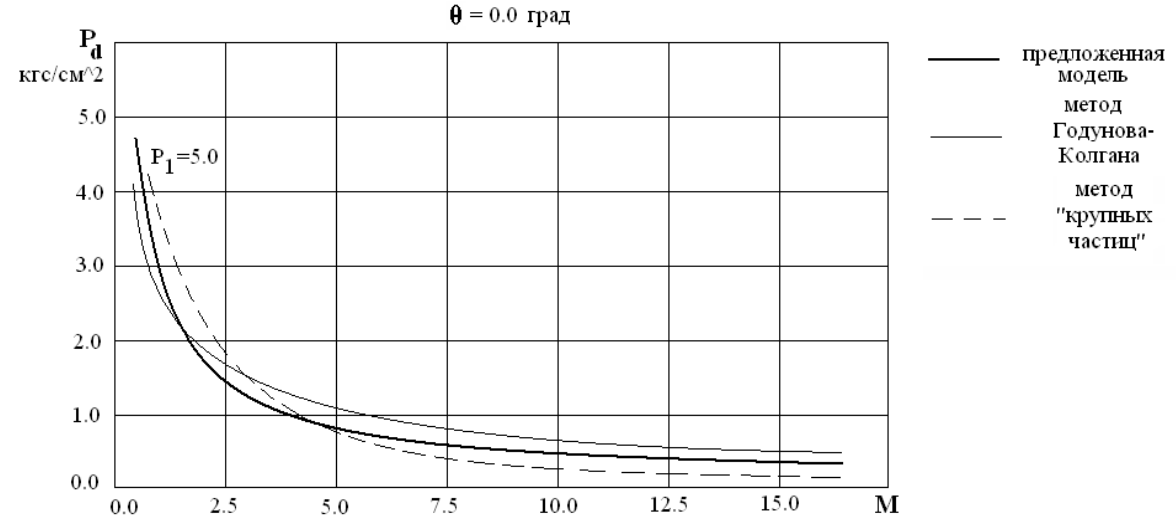


Рис. 8. Результаты расчета зависимости величины донного давления от числа Маха различными методами

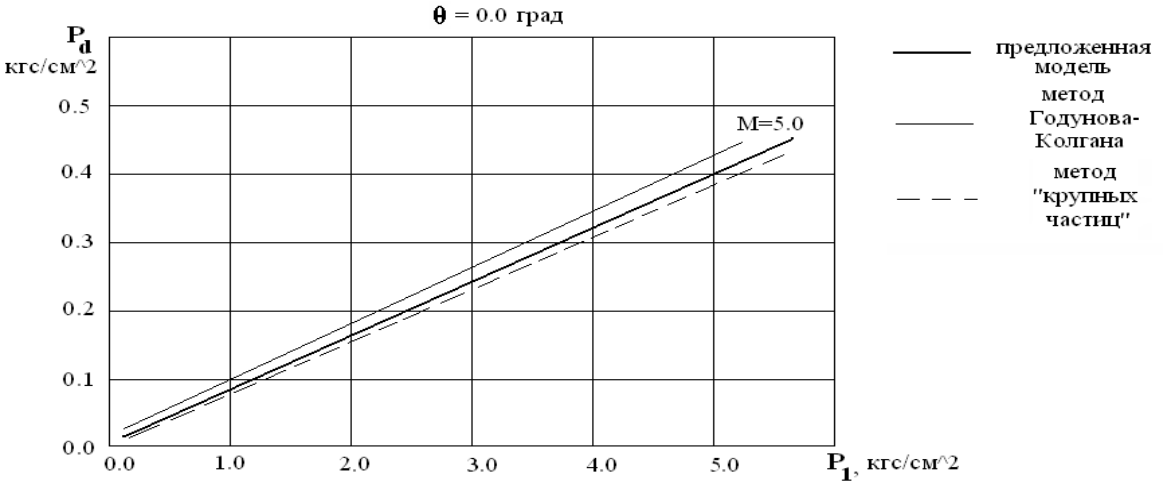


Рис. 9. Результаты расчета зависимости величины донного давления от давления в точке отрыва потока различными методами

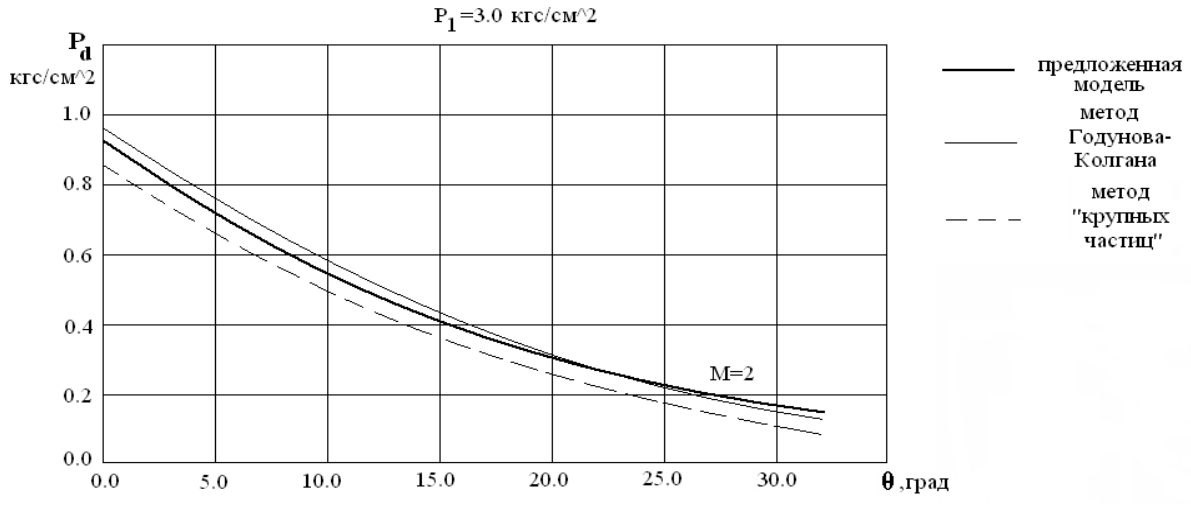


Рис. 10. Результаты расчета зависимости величины донного давления от угла наклона обтекаемой поверхности к продольной оси в точке отрыва потока различными методами

4. Заключение

По результатам проведенных исследований разработаны методические рекомендации по использованию предложенной модели отрывной донной области для исследования характеристик летательных аппаратов различного назначения различных конструкций.

Результаты математического моделирования получены с применением высокопроизводительных вычислений на суперкомпьютере «СКИФ-Аврора ЮУрГУ» ФГБОУ ВПО «ЮУрГУ» (НИУ) с производительностью до 117 TFlops.

Литература

1. Карташев, А.Л. Математическое моделирование течений в кольцевых соплах: монография / А.Л. Карташев, М.А. Карташева. – Челябинск: Издательский центр ЮУрГУ, 2011. – 158 с.
2. Чжен, П. Отрывные течения/ П. Чжен. – М.: Мир, 1973. – Т. 3. – 333 с.
3. Гогиш, Л.В. Турбулентные отрывные течения/ Л.В. Гогиш, Г.Ю. Степанов. – М.: Наука,1979. – 367 с.
4. Аукин, М.К. Расчет донного давления и энтальпии за плоским или осесимметричным уступом, обтекаемым сверхзвуковым потоком, с учетом влияния начального пограничного слоя/ М.К. Аукин, Р.К. Тагиров// Изв. РАН. МЖГ. – 1999. – № 2. – С.110–119.
5. Швец, А.И. Газодинамика ближнего следа/ А.И. Швец, И.Т. Швец. – Киев: Наукова думка, 1976. – 382 с.
6. Тагиров, Р.К. Влияние начального пограничного слоя на донное давление/ Р.К. Тагиров// Изв. АН СССР. – МЖГ. – 1966. – № 2. – С.145–148.
7. Масалов, В.К. Расчет донного давления и энтальпии за уступом, обтекаемым двумя сверхзвуковыми потоками, с учетом влияния пограничных слоев и тепловых потоков/ В.К. Масалов, Р.К. Тагиров// Изв. АН СССР. – МЖГ. – 1991. – № 5. – С. 167–176.
8. Численное решение многомерных задач газовой динамики/С.К. Годунов, А.В. Забродин, М.Я. Иванов и др. – М.: Наука, 1976. – 400 с.
9. Белоцерковский, О.М. Метод крупных частиц в газовой динамике/ О.М. Белоцерковский, Ю.М. Давыдов. – М.: Наука. Главная редакция физико–математической литературы, 1982. – 392 с.

Особенности внутреннего представления системы САПФОР*

Н.А. Катаев

Институт Прикладной Математики имени М.В. Келдыша РАН

Система САПФОР помогает пользователю пройти полный путь распараллеливания программы от исследования последовательной версии программы через ее преобразование и получение потенциально параллельной версии до распараллеливания с использованием моделей DVM, DVMH, OpenMP. Тесное взаимодействие с пользователем на этапах анализа и преобразования последовательной программы предъявляет определенные требования к архитектуре САПФОР. Внутреннее представление системы должно обеспечивать возможность предоставления пользователю исчерпывающей информации о каждом этапе распараллеливания программы. В рамках проводимого исследования было разработано внутреннее представление САПФОР, единое для языков Фортран и Си.

1. Введение

Широкое распространение технологий параллельного программирования и, одновременно, трудность их повсеместного использования при написании прикладных программ ведут к необходимости создания средств параллельного программирования, автоматизирующих процесс разработки параллельных программ. Использование полностью автоматических средств, например, автоматически распараллеливающих компиляторов для мультипроцессоров, поставляемых компаниями Intel, Microsoft, Sun Microsystems, IBM и др. не достаточно для получения максимально эффективного параллельного кода для сложных вычислительных задач.

Возможным решением является создание систем, помогающих пользователю в процессе разработки параллельных программ. Такие системы позволяют контролировать корректность принимаемых пользователем решений, собирают информацию об узких местах в программе, обладающих недостаточным параллелизмом, проблемах, мешающих распараллеливанию. Крайне желательным является прогнозирование характеристик выполнения параллельных программ на вычислительных системах разной конфигурации, подбор различных параметров, например, решетки процессоров, для максимального использования ресурса вычислительных систем.

Системы, автоматизирующие процесс распараллеливания, могут предоставлять пользователю информацию рекомендательного характера относительно принятия решений о распараллеливании или преобразовании последовательной программы с целью устранения проблем, мешающих распараллеливанию.

Одним из средств облегчающих разработку параллельных программ является среда разработки параллельных приложений Intel® Parallel Studio [1]. Ее недостатком является то, что непосредственно распараллеливание выполняет пользователь. На основе проведенного анализа и профилирования последовательной программы среда лишь дает рекомендации и указывает места, на которые надо обратить внимание. Для пользователя этап распараллеливания является наиболее трудоемким, требует детального знания соответствующих технологий параллельного программирования и подвержен ошибкам, наличие которых хотя и может быть установлено средствами Intel® Parallel Studio, устранено все равно должно быть пользователем вручную.

Между тем наличие более точных данных о последовательной программе, полученных в результате анализа статического и динамического и в виде знаний, предоставляемых системе пользователем, позволяют проводить распараллеливание потенциально параллельной программы полностью автоматически. Данный подход применяется в системе САПФОР [2], взаимо-

* Работа поддержана грантом Президента РФ МК-6772.2012.9 и грантами РФФИ № 12-01-33003, 12-07-31205 и 12-07-31260.

действуя с которой пользователь подготавливает прикладную программу к распараллеливанию, не выходя за рамки последовательной версии программы.

2. САПФОР

Система Автоматизированного Распараллеливания Фортран Программ (САПФОР) разрабатывается в Институте Прикладной Математики им. М.В. Келдыша РАН. Входным языком системы является Fortran, а результат распараллеливания представляет собой программу на языке Fortran OpenMP, Fortran DVM/OpenMP или Fortran DVMH [3, 4, 5]. Все эти языки являются расширением стандартного языка Fortran 95 директивами параллелизма. Высокий уровень выходного языка позволяет продемонстрировать пользователю результат распараллеливания в понятных для него терминах, кроме того директивы распараллеливания могут быть проигнорированы компиляторами, не поддерживающими соответствующий язык. Для данных языков существуют развитые средства отладки функциональности и эффективности.

Отличительной особенностью САПФОР является использование автоматически распараллеливающего компилятора [6], преобразующего потенциально параллельную программу в ее параллельную версию для заданной ЭВМ. При этом предварительный анализ программы и приведение программы к потенциально параллельному виду может выполняться в полуавтоматическом режиме. Для уточнения свойств последовательной программы, выявленных анализатором, используется либо диалоговая оболочка, либо специальные аннотации в тексте программы. В САПФОР наиболее сложный этап распараллеливания (получение параллельной версии программы) выполняется полностью автоматически.

Основными компонентами САПФОР являются анализаторы последовательных программ, блоки преобразования последовательных программ в параллельные программы (эксперты), диалоговая оболочка для взаимодействия с пользователем, генератор кода, создающий на основе принятых экспертом решений параллельную версию программы.

Связующим звеном между всеми компонентами системы, обеспечивающим поддержку итерационного распараллеливания прикладных программ является внутреннее представление САПФОР. Схема САПФОР показана на **Рис. 1**.

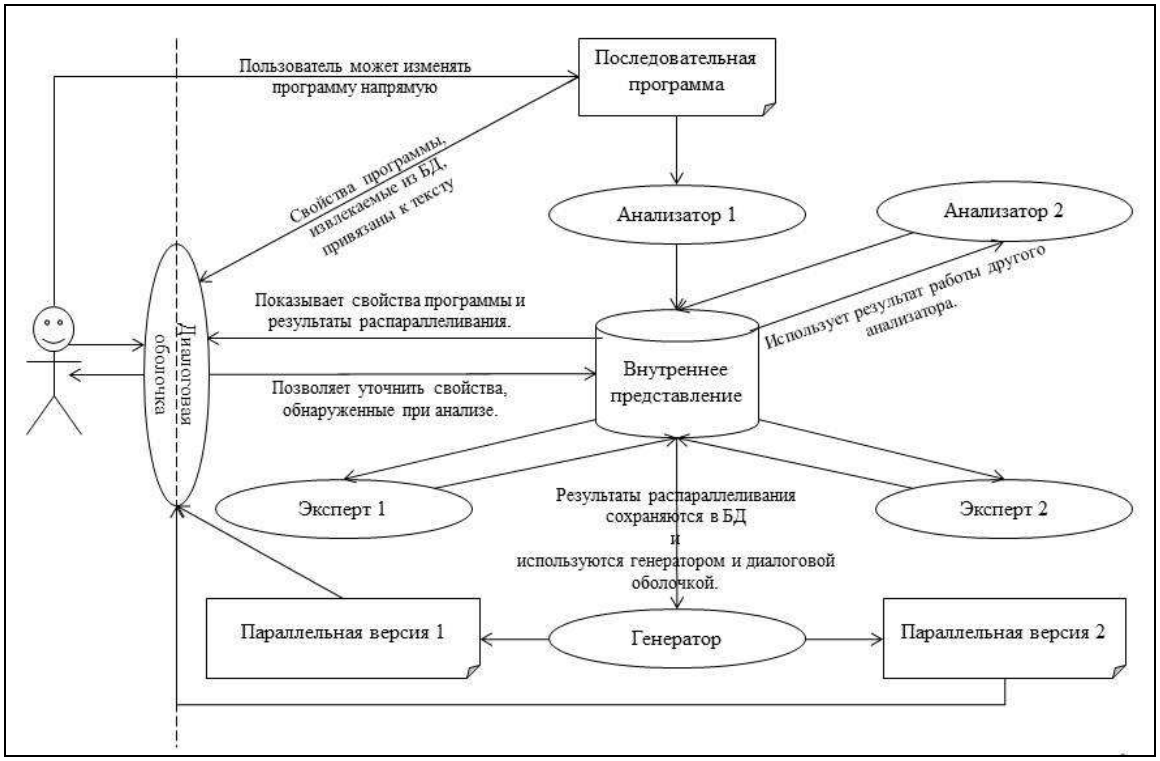


Рис. 1. Схема САПФОР

В статье рассматривается разработанное в рамках данного исследования Универсальное Высокоуровневое Внутреннее Представление (УВВП) процесса распараллеливания программы в системе САПФОР. УВВП САПФОР позволяет использовать в компонентах системы единые алгоритмы функционирования для языков программирования Fortran и C и обеспечивает возможность предоставление пользователю необходимой для распараллеливания информации в терминах исходной программы на протяжении всего пути распараллеливания прикладной программы пользователя.

Недостатки существующих представлений программ

В статье [7] был рассмотрен статический анализатор САПФОР, выполняющий анализ исходной программы на основе информации, содержащейся только в базе данных системы. Подчеркивалось, что данный анализатор потенциально не зависит от языка, на котором написана анализируемая программа. В анализаторе использовались общие алгоритмы анализа приватных скалярных переменных, применимые как для программ, написанных на языке программирования Fortran, так и для программ, написанных на языке программирования C.

Применение единых алгоритмов для различных языков программирования возможно также при выявлении других особенностей программы [7], требующихся для эффективной работы эксперта САПФОР. Решение о распараллеливании прикладной программы также можно принимать независимо от исходного языка. Такой подход используется, например, при оптимизации программ в системе компиляторов GCC [8] и LLVM [9].

Использование особенностей соответствующего языка (синтаксиса, семантики) необходимо на стадии взаимодействия с пользователем и при получении конечной версии параллельной программы. В этом состоит отличие внутреннего представления САПФОР от низкоуровневых представлений GIMPLE GCC и IR LLVM. Низкоуровневые представления должны быть адаптированы для извлечения информации о программе в терминах языка программирования, на котором написана прикладная программа пользователя.

Попытка использования представления GMIPLE GCC совместно с существующей базой данных САПФОР была первым шагом к использованию единого внутреннего представления. Был разработан анализатор [10], использующий в качестве основного средства анализа Универсальную Библиотеку Трансляции (УБТ) [11] и использующий ее для заполнения базы данных САПФОР. В УБТ анализ программ выполняется на основе внутреннего представления GIMPLE системы компиляторов GCC.

В GIMPLE GCC одному оператору исходной программы в общем случае соответствует группа операторов GIMPLE. Более того использование формы однократного присваивания (SSA) и операторов простого вида приводит к появлению дополнительных переменных. При сохранении информации в базе данных все временные переменные должны быть удалены, а операторы приведены к виду операторов исходной программы.

В GIMPLE представлении все использующиеся в программе массивы линеаризованы, но для корректного распараллеливания и распределения данных в языке Fortran DVM (C DVM) необходимо знать размерности массивов и используемые по каждому измерению индексы.

Также необходимо определение индуктивных переменных с границами и шагом и наличия тесной вложенности циклов. Для последнего необходимо восстановление высокоуровневых заголовков циклов программы из соответствующих им групп операторов GIMPLE.

Внутреннее представление должно отражать полный путь распараллеливания программы от исследования последовательной версии программы через ее преобразование и получение потенциально параллельной версии до распараллеливания с использованием моделей DVM, DVMH, OpenMP. Это делает невозможным использование GIMPLE GCC или IR LLVM без организации дополнительного средства хранения информации, отражающей результаты взаимодействия с пользователем, решения принимаемые компонентами САПФОР, выявленные особенности прикладной программы.

Для применения единых алгоритмов информации, содержащейся в существующей базе данных, было не достаточно. Основными недостатками использующейся базы данных были:

1. Отсутствие полной информации об исполняемых операторах программы. Для операторов сохранялась семантика доступа к памяти программы, но семантика испол-

няемых операций отсутствовала. В результате было невозможно, используя только базу данных, проанализировать ветвления в программе, оценить вероятность выполнения различных ветвей, выполнить распространение констант, определить индукционные переменные циклов.
2. Ориентированность базы данных на язык Fortran.
3. Невозможность хранения информации о точках и участках программы (например, циклах), когда данная информация зависит от пути выполнения программы, по которому данная точка или цикл были достигнуты. Путь выполнения программы определяется графом управления и графом вызовов,

В результате было разработано Универсальное Высокоуровневое Внутреннее Представление (УВВП) системы САПФОР, обладающее следующими особенностями:

1. Является связующим звеном между всеми компонентами системы.
2. Обеспечивает поддержание итерационного процесс распараллеливания программы. УВВП САПФОР содержит информацию о каждом этапе распараллеливания программы (результаты анализа: статического и динамического, выполняемые преобразования, указания пользователя, принятые экспертом решения о распараллеливании). С помощью диалоговой оболочки системы данная информация может быть наглядно представлена пользователю в терминах исходной программы. УВВП САПФОР содержит полное описание программы и позволяет восстановить исходный код программы, возможно, с учетом преобразований последовательной программы и директив распараллеливания, указанных экспертом САПФОР.
3. Не зависит от языка программирования. В качестве поддерживаемых языков рассматриваются языки Fortran и C, при этом используемые структуры данных предусматривают возможность адаптации УВВП САПФОР к другим языкам программирования.

Структура внутреннего представления

УВВП САПФОР состоит из четырех уровней, независящих от используемого в распараллеливаемой программе языка программирования:

1. Уровень исходного кода программ.
2. Уровень особенностей программы.
3. Уровень преобразования программы.
4. Уровень распараллеливания программы.

Уровень (1) исходного кода программ описывает прикладную программу в том виде, в каком она была написана пользователем системы САПФОР. Данного уровня достаточно для восстановления исходного кода программы. Уровень является единственным, заполнение которого зависит от используемого в распараллеливаемой программе языка программирования. Для заполнения данного уровня достаточно средств, разбирающих структуру программы (выполняющих лексический, синтаксический и семантический анализ), но не использующих сложные алгоритмы анализа, такие как поиск зависимостей по данным, поиск приватных переменных и др. В качестве front-end строящего первый уровень внутреннего представления для Fortran используется библиотека Sage++[12], для C предполагается использовать возможности Clang [13] компилятора LLVM.

Уровень (2) особенностей программы описывает свойства программы, выявленные анализаторами системы (статическим и динамическим). Для всех циклов программы необходимо выявлять информацию следующего вида: зависимости по данным (flow, anti, output), регулярные зависимости по массивам, редукционные зависимости между витками цикла, приватные переменные (скаляры и массивы), индукционные переменные, время последовательного выполнения циклов (а также отдельных витков), обнаружение неявных циклов.

Для повышения эффективности проводимого распараллеливания, анализатором строится расширенное дерево циклов, в котором для каждой вершины-цикла прослеживается путь от входной точки программы. Каждому потенциально возможному выполнению цикла, определяемому последовательностью вызовов процедур (путем графа вызовов), соответствует отдельная вершина в расширенном дереве циклов. Для обнаруженных особенностей программы в УВВП

САПФОР осуществляется связывание с соответствующими вершинами расширенного дерева циклов, для которых данные особенности имеют место.

При необходимости анализ выполняется отдельно для каждой вершины расширенного дерева циклов. С точки зрения полноты информации, использование расширенного дерева циклов эквивалентно выполнению инлайн подстановки, при этом, данная операция скрыта от пользователя и ему предоставляется информация в терминах его исходной программы, но с указанием пути выполнения программы, по которому может быть достигнут соответствующий цикл.

Уровень преобразования программы описывает преобразования, последовательной версии программы, которые необходимы для выполнения наиболее эффективного распараллеливания. Для разных вариантов распараллеливания возможно использования различных наборов преобразований, приводящих программу к потенциально параллельному виду, пригодному для использования автоматического распараллеливающего компилятора, входящего в состав САПФОР. Необходимость выполнения тех или иных преобразований определяется тем вкладом, который они могут внести в повышение эффективности распараллеливания, оцениваемой экспертом при построении соответствующих вариантов распараллеливания. По каждому преобразованию может быть получена информация, объясняющая пользователю необходимость его применения.

Уровень распараллеливания программы содержит варианты распараллеливания, построенные экспертом, оценки эффективности параллельных версий программ, описание целевой архитектуры, рекомендованных параметров запуска (например, описание решетки процессоров), описание причин, снижающих эффективность распараллеливания программы.

На **Рис. 2** приведено соответствие между компонентами системы САПФОР и уровнями УВВП, за построение которых они отвечают.

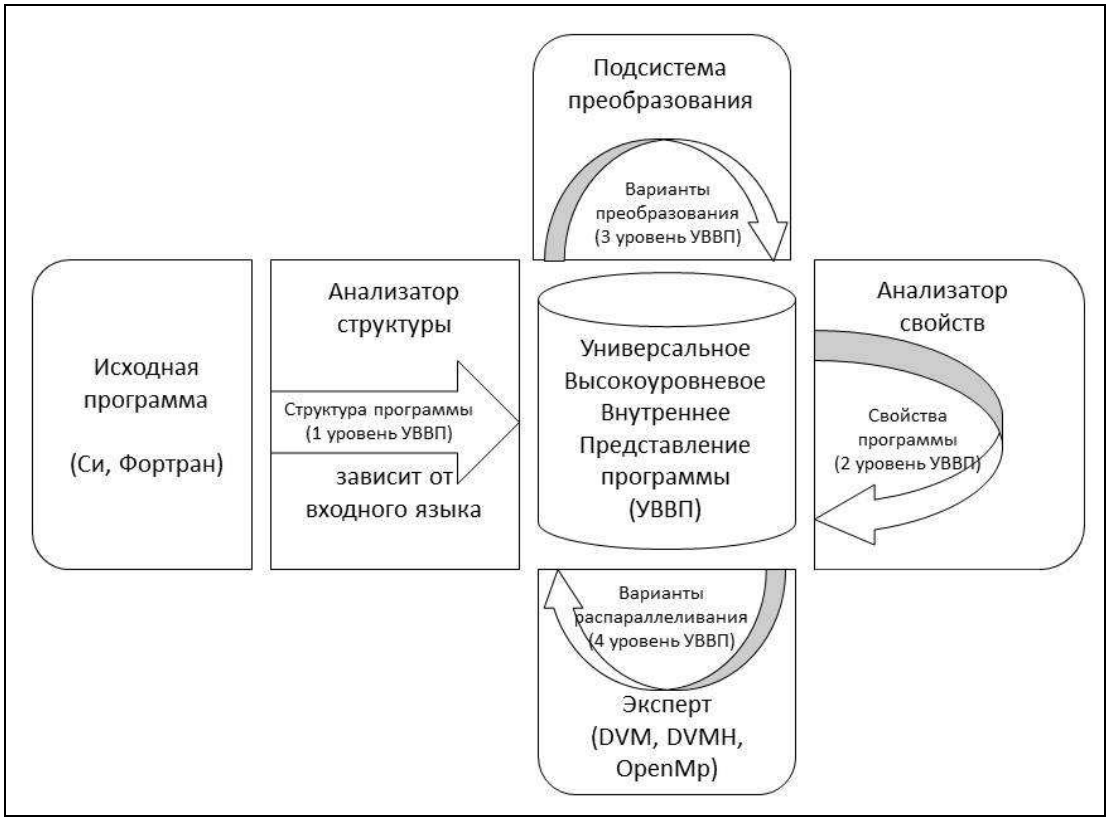


Рис. 2. Уровни УВВП САПФОР

Программная реализация

УВВП САПФОР хранится в виде SQL базы данных, построенной на основе свободно распространяемой библиотеки SQLite [14]. Такой способ реализации УВВП обусловлен тем, что:

1. Система САПФОР образована разными программными компонентами, не имеющими общего адресного пространства и асинхронно взаимодействующими между собой. Компоненты САПФОР могут быть разнесены в пространстве, например, динамический анализ можно запускать на суперкомпьютере для повышения производительности и точности анализа.
2. Во внутреннем представлении содержится большой объем количественных характеристик и других особенностей программы, к которым необходимо обеспечить удобный доступ, для чего может быть использован язык запросов SQL.
3. Использование механизмов SQLite позволяет контролировать непротиворечивость внутреннего представления при его модификации компонентами системы.

Для доступа к базе данных внутри компонент системы САПФОР разработан высокоуровневый интерфейс, представляющий собой надстройку над функциями библиотеки SQLite. Интерфейс реализован на языке программирования C++ в объектно-ориентированном стиле с применением идей метапрограммирования и скрывает от пользователей необходимость использования низкоуровневого механизма доступа к базе данных, предоставляемого библиотекой SQLite.

Интерфейс является отдельной библиотекой и может быть использован с другими базами данных. Интерфейс может быть легко адаптирован к изменениям в структуре базы данных САПФОР и позволяет использовать для доступа к информации, хранящейся в базе данных, имена таблиц и содержащихся в них столбцов, не задумываясь об особенностях реализации базы данных. Это позволяет обеспечить обратную совместимость предыдущих версий компонентов САПФОР и внутреннего представления системы при его модификации. Например, добавление новых таблиц или столбцов в таблицы не повлияет на работу компонентов системы. Интерфейс настраивается под особенности компонентов системы.

Все структуры данных разработанного интерфейса сопровождаются комментариями в формате, поддерживаемом средством автоматической генерации документации Doxygen.

Заключение

Для поддержания процесса распараллеливания при активном участии пользователя в системе САПФОР было разработано Универсальное Высокоуровневое Внутреннее Представление (УВВП) единое для языков программирования Fortran и C.

Внутреннее представление реализовано в виде базы данных, что позволяет использовать SQL-запросы для быстрого поиска необходимой информации о программе и поддерживать непротиворечивость структур, описывающих состояние процесса распараллеливания.

Высокий уровень внутреннего представления позволяет реализовать взаимодействие с пользователем в терминах его исходной программы, и обеспечивает удобство использования системы САПФОР при распараллеливании прикладных программ.

Независимость внутреннего представления от языка программирования позволяет использовать общие механизмы анализа, преобразования и распараллеливания для языков программирования Fortran и C. Это необходимо для развития системы САПФОР, изначально ориентированной на язык программирования Fortran, в направлении автоматизированного распараллеливания программ, написанных на языке программирования C.

Литература

1. Intel® Parallel Studio. URL: http://software.intel.com/en-us/intel-parallel-studio-home (дата обращения: 01.12.2012)
2. Система Автоматизированной Параллелизации Фортран Программ. URL: http://www.keldysh.ru/dvm/SAPFOR/ (дата обращения: 01.12.2012).
3. Бахтин В.А., Коновалов Н.А., Крюков В.А., Поддерюгина Н.В. Fortran OpenMP/DVM – язык параллельного программирования для кластеров // Материалы второго Международ-

ного научно-практического семинара “Высокопроизводительные параллельные вычисления на кластерных системах”, г. Нижний Новгород, 26-29 ноября 2002 г., с.28-30.

4. Бахтин В.А., Коновалов Н.А., Поддерюгина Н.В., Устюгов С.Д. Гибридный способ программирования DVM/OpenMP на SMP-кластерах // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования” (сентябрь 2006 г., г. Новороссийск), Изд-во Московского Университета, с.128-130.
5. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: экзафлопсное будущее” (19-24 сентября 2011 г., г. Новороссийск). - М.: Изд-во МГУ, с. 310-315.
6. Крюков В.А., Клинов М.С., Бахтин В.А., Поддерюгина Н.В. Автоматическое распараллеливание последовательных программ для многоядерных кластеров // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, с. 12-15.
7. Катаев Н.А. Статический анализ последовательных программ в системе автоматизированного распараллеливания САПФОР // Труды международной научной конференции “Параллельные вычислительные технологии (ПаВТ’2012)” (Новосибирск, 26 – 30 марта 2012 г.) – Челябинск: Издательский центр ЮУрГУ, 2012, с. 179-190
8. GCC, the GNU Compiler Collection. URL: http://gcc.gnu.org/ (дата обращения: 01.12.2012).
9. The LLVM Compiler Infrastructure. URL: http://llvm.org/ (дата обращения: 01.12.2012).
10. Катаев Н.А. Анализ последовательных программ с помощью средств УБТ // Труды международной научной конференции “Параллельные вычислительные технологии (ПаВТ’2011)” (Москва, 28 марта – 1 апреля 2011 г.) – Челябинск: Издательский центр ЮУрГУ, 2011, с. 697.
11. Optimizing Technologies. URL: http://www.optimitech.com/ (дата обращения: 01.12.2012).
12. pC++/Sage++ Home Page. URL: http://www.extreme.indiana.edu/sage/ (дата обращения: 01.12.2012).
13. Clang. URL: http://clang.llvm.org/ (дата обращения: 01.12.2012).
14. SQLite. URL: http://www.sqlite.org/ (дата обращения: 01.12.2012).

Преобразования последовательных программ при их распараллеливании с помощью системы САПФОР*

Н.А. Катаев, М.С. Клинов, Н.В. Поддерюгина

Институт прикладной математики имени М.В. Келдыша РАН

Есть такие последовательные программы, которые после преобразования некоторых ее фрагментов эффективно отображаются на современные кластеры с помощью автоматически распараллеливающего компилятора системы САПФОР. Такие преобразования не всегда можно сделать автоматически без участия программиста. Использование автоматически распараллеливающего компилятора для этих программ проще написания программ в модели DVM, потому что при преобразовании последовательных программ у этих подходов много общих действий.

1. Введение

Процесс распараллеливания следует понимать не как преобразование последовательной программы в параллельную, а как преобразование последовательной программы в эффективную параллельную. Поэтому и считается, что автоматическое распараллеливание пока недостижимо [1]. Однако процесс распараллеливания можно представить в виде двух процессов: преобразования в рамках последовательной программы и преобразование последовательной программы в эффективную параллельную. В таком случае к выполнению первого процесса можно привлечь программиста, тем самым, с одной стороны снизить уровень автоматизации процесса, но с другой повысить его качество, а второй процесс выполнять полностью автоматически.

Такой подход используется в системе САПФОР [2-4], которая включает в себя диалоговую оболочку и автоматически распараллеливающий компилятор. Система рассчитана на взаимодействие с пользователем, который достаточно хорошо разбирается в последовательной программе, и является в этом смысле программистом. В процессе диалога с системой программист оценивает возможности автоматически распараллеливающего компилятора по распараллеливанию его текущей версии последовательной программы и определяет пути возможных ее преобразований. Когда преобразования над последовательной программой закончены, через запуск автоматически распараллеливающего компилятора он получает параллельную программу. Случай, когда система не может путем автоматического анализа определить свойства некоторых фрагментов последовательной программы, сводится к добавлению программистом этих свойств к программе. Это может быть реализовано через специальные комментарии, вставленные в текст программы, или через подсказки системе в процессе диалога.

2. Преобразования при распараллеливании на кластер

Рассмотрим в этом разделе такие свойства фрагментов последовательных программ, которые требуют их преобразования для получения эффективных параллельных программ. При этом подразумевается, что все преобразования выполняются в рамках последовательной программы, т.е. после преобразований программа остается последовательной и не содержит никаких свойств параллельной программы, указаний по ее генерации, вариантов распределения данных в ней и т.п.

Некоторые преобразования связаны с тем, что система САПФОР направлена на получение программ для кластерных и распределенных систем. В связи с этим она сталкивается с необходимостью выбора вариантов распределения данных между узлами распределенных систем, которые определяют эффективность распараллеливания программы. Поэтому некоторые преобра-

* Работа поддержана грантами Президента РФ МК-6772.2012.9 и НШ-4307.2012.9, грантами РФФИ № 12-01-33003, 12-07-31205 и 12-07-31260.

зования не требуются при автоматическом распараллеливании, например, на мультипроцессоры. При этом многие из приведенных далее преобразований требуются при автоматическом распараллеливании на кластеры при помощи систем, отличных от системы САПФОР, в том числе и при разработке программ с использованием языков параллельного программирования.

Сформулируем для начала набор свойств, которые требуют преобразований последовательных программ, а затем более подробно их рассмотрим:

- Наличие зависимостей в цикле, которые не являются приватными или редукционными:
  - OUTPUT-зависимости,
  - нерегулярные FLOW-зависимости,
  - регулярные FLOW-зависимости;
- Наличие в цикле операторов ввода-вывода;
- Наличие в цикле нескольких входов или нескольких выходов из него;
- Наличие на одном витке цикла присваиваний в разные элементы одного массива;
- Наличие нескольких циклов с разным соответствием на присваивание элементов разных массивов;
- Отсутствие тесно-вложенных циклов во фрагментах программы, для которых их можно организовать;
- Необходимость коммуникаций (обмены значениями редукционных переменных, теневых граней, буферов с данными разных процессов, синхронизации).

Поясним случай наличия OUTPUT-зависимостей в цикле, которые не являются приватными или редукционными, на примере кода на языке Фортран и приведем пример их устранения. Для этого надо сформулировать несколько определений.

Под OUTPUT-зависимостью понимается информационная зависимость между витками цикла, когда имеет место запись на разных витках в одну ячейку памяти.

Под приватной переменной для цикла (в том числе под приватным массивом) понимается такая переменная, для которой на каждом витке цикла сначала выполняется присваивание в переменную (в элементы массива), затем переменная (элементы массива) может быть использована на этом витке, и за пределами цикла также сначала производится присваивание в переменную (в элементы массива), затем использование. Если в цикле есть приватная переменная, то будем говорить, что в цикле есть приватная зависимость по этой переменной.

Под редукционной переменной и соответственно редукционной зависимостью в цикле понимается неприватная переменная, при которой можно преобразовать цикл (при этом результаты его выполнения останутся прежними с точностью до перестановки слагаемых в сумме или множителей в произведении) таким образом:

- завести на каждом витке свою копию этой переменной,
- присвоить ей начальное значения (0 – для суммы, 1 – для произведение, отрицательный максимум при поиске максимума и т.п.),
- после выполнения всех витков скорректировать значения всех копий по единой схеме (например, сумма локальных сумм, максимум среди локальных максимумов). Для всех элементов редукционного массива (когда каждый элемент массива является редукционной переменной) операция коррекции копии является одинаковой.

В данном случае имеет место цикл с шагом 1 (по умолчанию для Фортрана) и выполняется присваивание на витке в соседние элементы одного массива ro1, аналогично и для массива E1. В результате на соседних витках будет выполняться присваивания в один элемент массива (в одну ячейку памяти). Одним из эффективных способов организации параллельного выполнения цикла является выполнение каждого витка целиком некоторым процессом. Это позволяет перед циклом одной операцией распределить витки (и группы витков) по процессам. В противном случае требуется преобразование цикла к такому виду, который подразумевает выполнение каждого витка целиком на процессе, либо преобразование, которое распределяет вычисления по процессам другими способами, и которое представляется более сложным относительно преобразования цикла к нужному виду. Распределение витков цикла целиком является широко распространенным методом среди высокоуровневых языков, использовать которые для автоматического распараллеливания является предпочтительным для уменьшения объема преобрзований кода программы, необходимых для организации параллельного выполнения.

Наличие OUTPUT-зависимостей в цикле	Пример изменения

…
do j = 1,ny
  …
  do i = 0,nx
  …
  Frox = …
  FEx = …
  ro1(i,j) = ro1(i,j) + Frox
  ro1(i+1,j) = ro1(i+1,j) - Frox
  E1(i,j) = E1(i,j) + FEx
  E1(i+1,j) = E1(i+1,j) - FEx
  enddo
enddo
…
	dimension tmp1(0:nx1,0:ny1)
	dimension tmp2(0:nx1,0:ny1)
	…
	do j = 1,ny
	  …
	  do i = 0,nx
	  …
	  Frox = …
	  FEx = …
	  tmp1(i,j) = Frox
	  tmp2(i,j) = FEx
	  ro1(i,j) = ro1(i,j) + Frox
	  E1(i,j) = E1(i,j) + FEx
	  enddo
	enddo
	
	do j = 1,ny
	  do i = 0,nx
	  ro1(i+1,j)= ro1(i+1,j) - tmp1(i,j)
	  E1(i+1,j)= E1(i+1,j) - tmp2(i,j)
	  enddo
	enddo
	…

Рис. 1. Пример устранения OUTPUT-зависимостей в цикле

В случае организации параллельного выполнения цикла в таком виде, что каждый процесс выполняет несколько витков цикла целиком, наличие в цикле OUTPUT-зависимости, которая не является приватной или редукционной, делает невозможным параллельное выполнения цикла. Это требует синхронизации между витками, которая может привести к тому, что некоторый процесс не сможет начать свою работу, пока не закончат свою работу все предыдущие процессы.

В приведенном примере программы выполняется накопление суммы в элементах массива: либо прибавляется некоторая величина, либо вычитается. В этом случае цикл можно разбить на два. В первом сохранять необходимые величины во временный массив с тем, чтобы использовать их для последующих операций над этими ячейками. Полученные циклы не будут содержать OUTPUT-зависимостей и допускают параллельное выполнение в процессе нескольких витков целиком.

Второй случай связан с наличием в цикле нерегулярных FLOW-зависимостей, которые не являются приватными или редукционными.

Под FLOW-зависимостью понимается такая информационная зависимость между витками цикла, когда имеет место запись на более раннем витке (по порядку их выполнения) в ячейку памяти (переменную или элемент массива), а на более позднем витке имеет место чтение из этой ячейки.

Под регулярной зависимостью понимается такая зависимость, при которой зависимые витки в гнезде циклов (несколько циклов, когда телом одного цикла является другой цикл) отстают друг от друга не более чем на константы по каждому из циклов в гнезде. Эти константы называются длинами зависимости. Для гнезда циклов их можно оформить в виде вектора длин. Длина зависимости зависит от шага цикла, например:

- если в цикле шаг равен 2, а один виток (i = 3) зависит от предыдущего (i = 1) – то длина будет 1,
- если шаг был 1, то для витков со значением итерационной переменной 3 и 1 длина равна 2,
- при шаге 3, зависимости может и не быть.

Особое внимание заслуживают регулярные зависимости с малой длиной. Если известны размеры цикла, то всегда можно взять в роли константы, ограничивающей длину зависимости, размер цикла, но она будет большой, и параллельная реализация такого цикла не будет эффек-

тивной. Как правило, если цикл не является циклом с регулярной зависимостью с малой длиной, то для него говорят, что его FLOW-зависимость является нерегулярной.

В случае наличия нерегулярных FLOW-зависимостей, которые не являются приватными или редукционными, для реализации параллельного выполнения цикла требуется рассылка посчитанных данных всем следующим процессам (соответствующим их узлам распределенных систем и областям памяти процессов), а следующих процессов много. Таким образом, имеет место рассылка от одного процессора ко многим, которая не всегда может быть просто организована автоматически, например, в языке Fortran-DVM это невозможно имеющимися в нем средствами, а организация ее другими средствами требует более сложной и низкоуровневой переделки кода программы. К тому же время на подобную рассылку может при значительном количестве процессов превысить выигрыш от распараллеливания этого цикла.

Третье свойство связано с наличием регулярных FLOW-зависимостей, которые не являются приватными или редукционными. Для него требуется посылка посчитанных данных нескольким соседним процессам (это зависит от длины зависимости). Это меньшие затраты по отношению к затратам на нерегулярные FLOW-зависимости. К тому же их можно реализовывать средствами высокоуровневых языков и это достаточно простой метод для автоматического распараллеливания, который связан с минимальным преобразованием текста программы и вставкой директив распараллеливания. Подобные средства и соответствующая поддержка выполнения циклов с регулярными зависимостями реализованы, например, в языке Fortran-DVM. При использовании этого языка, DVM-система автоматически рассчитывает целесообразность и параметры конвейеризации таких циклов. В процессе конвейеризации цикла вычисления разбиваются на кванты, после которых выполняется посылка соседним процессам посчитанных данных, необходимых для их дальнейшей работы. Тем самым, более равномерно загружаются процессы, а цикл выполняется быстрее. Но, конечно, при выполнении циклов с регулярными FLOW-зависимостями все равно есть предел масштабируемости, при котором накладные расходы на организацию конвейера превышают выигрыш от распараллеливания. Это следует учитывать при распараллеливании программ.

Наличие в цикле операторов ввода-вывода препятствует параллельному его выполнению. Дело в том, что для корректной работы порядок операторов ввода-вывода должен соответствовать такому их порядку, который был в исходной последовательной программе. Для обеспечения такого порядка требуются дополнительные синхронизации, которые могут существенно замедлить цикл. В некоторых высокоуровневых языках наличие операторов ввода-вывода в теле параллельного цикла недопустимо из-за того, что в языках не обеспечивается корректная работа с ними с точки зрения алгоритма исходной программы. Можно выносить операторы ввода-вывода из вычислительных циклов и выполнять их в нераспараллеливаемых циклах. Такие преобразования не всегда могут быть полностью автоматизированы, и требуют участие программиста.

Помимо этого некоторые высокоуровневые языки, например Fortran-DVM, накладывают дополнительные ограничения, например, на использование распределенных массивов в операторах ввода-вывода. Это может привести к тому, что без внесения изменений в программу, некоторые массивы не могут быть распределены, что в свою очередь может повлечь за собой требование не распределять и другие массивы и т.д. Требование не распределять другие массивы возникает оттого, что в программе могут быть циклы, которые требуют присваивания на одном витке в элементы массивов, некоторые из которых могут быть распределенными, а некоторые размноженными по процессам. Т.е. с одной стороны виток должен быть выполнен на всех процессах, а с другой не на всех – противоречие, которое можно устранить или преобразованием всех таких циклов, или путем размножения этих массивов.

Следующее свойство связано с наличием в цикле нескольких входов (например, при помощи оператора перехода по метке) или нескольких выходов и приводит к тому, что некоторые витки не надо выполнять. А это сложно обеспечить, потому что условие дополнительного выхода из цикла (а основной выход из цикла – это выход по превышению порогового значения итерационной переменной) можно проверить только в динамике, т.е. в процессе выполнения программы. Автоматически откатить работу нескольких витков непросто, и связано с дополнительными накладными расходами, которые снижают эффективность. Аналогично и для дополнительного входа в цикл.

Наличие на одном витке цикла присваиваний в разные элементы одного массива в случае, если они не находятся в одном процессе из-за соответствующего распределения данных, делает невозможным эффективное выполнение витков целиком в рамках процесса.

Наличие присваиваний в разные элементы одного массива на одном витке цикла	Пример изменения

do i = 1,nx
  ro(i,0) = ro(i,1)
  …
  ro(i,ny1) = ro(i,ny)
  …
enddo
 	
do j = 1,ny
  ro(0,j) = ro(1,j)
  …
  ro(nx1,j) = ro(nx,j)
  …
enddo
	do i = 1,nx
	  ro(i,0) = ro(i,1)
	  …
	enddo
	do i = 1,nx
	  ro(i,ny1) = ro(i,ny)
	  …
	enddo
	
	do j = 1,ny
	  ro(0,j) = ro(1,j)
	  …
	enddo
	do j = 1,ny
	  ro(nx1,j) = ro(nx,j)
	  …
	enddo

Рис. 2. Пример изменения циклов с наличием на одном витке присваиваний в разные элементы одного массива

Такие случаи встречаются достаточно часто в последовательных программах: разработчик прикладной программы не думает о ее распараллеливании или не знает о таких особенностях распараллеливания. Он объединяет похожие операции над массивами и экономит на операциях изменения значений итерационной переменной.

Для приведенных в примере циклов не удается найти такое распределение данных, чтобы элементы массива ro(i,0) и ro(i,ny1) одновременно располагались в одном процессе, и было возможным расположение в одном процессе ro(0,j) и ro(nx1,j). Для получения параллельной программы необходимо разбить хотя бы один из этих циклов на два.

А лучше разбить оба цикла на два, чтобы было возможным использовать двумерное распределение данных.

Наличие нескольких циклов с разным соответствием на присваивание элементов разных массивов	Пример их изменения

do i = 1,nx
  ro(i,0) = …
  …
  E(i+1,0) = …
  …
enddo
 	
do i = 1,nx
  ro(i,1) = …
  …
  E(i,1) = …
  …
enddo
	do i = 1,nx
	  ro(i,0) = ...
	  …
	enddo
	do i = 1,nx
	  E(i+1,0) = …
	  …
	enddo
	
	do i = 1,nx
	  ro(i,1) = …
	  …
	enddo
	do i = 1,nx
	  E(i,1) = …
	  …
	enddo

Рис. 3. Пример изменения циклов с разным соответствием на присваивание элементов разных массивов

Разбиение витка является непростой задачей с точки зрения полной автоматизации этого процесса.

Поясним случай с наличием нескольких циклов с разным соответствием на присваивание элементов разных массивов на примере.

Разное соответствие препятствует способу распределению витков, когда каждый процесс выполняет группу витков целиком. Данные, которые он должен модифицировать, должны быть на нем (он должен иметь их копию). Случай, когда любой процесс должен иметь какой-то определенный элемент и следующий, будет приводить к дублированию элементов между процессами, а значит и к дублированию вычислений, что не очень эффективно.

Следующее свойство связано с тесно-вложенными циклами. Это такие циклы, когда телом одного цикла является другой цикл. Такие многомерные циклы можно эффективно распределять между процессами в отличие от случая, когда циклы не тесно вложены, и требуется повторять в цикле операции распределения внутреннего цикла несколько раз. Тесно-вложенные циклы позволяет снизить накладные расходы на организацию параллельного выполнения.

Наличие нетесно-вложенных циклов	Пример их изменения

`do j = 1,ny`
`  hy4 = …`
`  do i = 0,nx`
`  …`
`  enddo`
`enddo`
	`do j = 1,ny`
	`  do i = 0,nx`
	`  hy4 = …`
	`  …`
	`  enddo`
	`enddo`

Рис. 4. Пример формирования тесно-вложенных циклов

С точки зрения последовательного выполнения преобразованной программы она будет выполняться немного медленнее (крайне незначительно) из-за повтора в цикле одного и того же оператора, однако параллельное ее выполнение значительно эффективнее не преобразованного варианта.

Необходимость разного рода коммуникаций (взаимодействий процессов с целью их синхронизации и обмена данными) замедляет параллельное выполнение программы, что может привести к отсутствию дальнейшего ускорения выполнения программы при увеличении числа вычислительных ядер. Коммуникации являются необходимыми для обеспечения корректной работы параллельных вычислений, их нельзя убрать из программы, но можно оптимизировать: объединять в более крупные обмены, выполнять обмены во время вычислений, разбивать обмены и выполнять некоторые раньше.

Среди операций коммуникаций можно выделить следующие виды:

- обмены значениями редукционных переменных,
- обмены теневыми гранями (теневые грани – это специальные буфера, которые расширяют локальную часть данных процесса за счет элементов с соседних процессов, с целью их своевременной загрузки и удобного к ним доступа),
- пересылка буферов с данными процессов,
- синхронизации.

Алгоритмы автоматической оптимизации коммуникаций зависят от возможностей выходного языка параллельного программирования и в ряде случаев могут потребовать дополнительных модификаций программы, без которых эффективная организация коммуникаций невозможна.

3. Заключение

Для ряда последовательных программ со статическими равномерными прямоугольными сетками удается получать хорошие результаты при помощи системы автоматизированного распараллеливания САПФОР [4]. В процессе распараллеливания программист преобразует по определенной дисциплине последовательную программу. Набор таких преобразований являет-

ся достаточно определенным, что позволяет говорить о накоплении опыта и соответствующих навыков.

Выполнять эти преобразования автоматически без участия программиста не всегда представляется возможным из-за большого разнообразия случаев, хотя некоторая автоматизация этого процесса, безусловно, возможна, и соответствующие методы будут в будущем разработаны и исследованы.

Трудоемкость распараллеливания программ при помощи языков параллельного программирования значительно выше трудоемкости преобразований, которые не выходят за пределы текста последовательной программы, в том числе и из-за того, что наиболее сложные преобразования, такие как устранения зависимостей, одинаковы и при ручном распараллеливании, и при использовании системы САПФОР.

Литература

1. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В. Автоматическое распараллеливание последовательных программ для многоядерных кластеров. // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, Новороссийск, сентябрь 2010 – М.: Изд-во МГУ, 2010, С. 12-15.
2. Бахтин В.А., Катаев Н.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Автоматическое распараллеливание Фортран-программ на кластер с графическими ускорителями // Сборник трудов Международной научной конференции “Параллельные вычислительные технологии” (ПаВТ’2012), Новосибирск, март 2012, С. 373-379.
3. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Супервычисления и математическое моделирование. Труды XIII Международного семинара / Под ред. Р.М. Шагалиева. – Саров: ФГУП “РФЯЦ-ВНИИЭФ”, 2012. – С. 80-84.
4. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Распараллеливание с помощью DVM-системы некоторых приложений гидродинамики для кластеров с графическими процессорами // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: поиск новых решений”, Новороссийск, сентябрь 2012 – М.: Изд-во МГУ, 2012. – С. 444-450.

Параллельный метод Полларда решения задачи дискретного логарифмирования с использованием детерминированной функции разбиения на множества

Е.Г. Качко, К.А. Погребняк

Харьковский национальный университет радиоэлектроники

В работе предлагается усовершенствованный метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой и в мультипликативной группе конечного поля для систем с общей памятью. Усовершенствование метода достигается за счет построения детерминированной функции разбиения на множества. Такая функция позволяет организовать два независимых сбалансированных вычислительных потока построения блока элементов группы фиксированной длины. Далее анализируются известные функции итерирования точек в алгоритме Полларда и строится обобщенная детерминированная функция разбиения на множества.

1. Введение

Сегодня широко используются криптографические системы с открытым ключом, стойкость которых основывается на существовании вычислительно сложных задач. К таким задачам относится нахождение дискретного логарифма в конечной абелевой группе. В практических приложениях используются аддитивная группа точек эллиптической кривой, заданной над конечным полем, и мультипликативная группа элементов поля Галуа.

Особый интерес представляют методы решения задачи дискретного логарифмирования. Для вычисления дискретного логарифма в группе точек эллиптической кривой наиболее эффективным считается ρ-метод Полларда, а для решения задачи логарифмирования в мультипликативной группе поля Галуа – метод решета числового поля [1]. Тем не менее, ввиду простоты и универсальности метода Полларда, его часто используют и в мультипликативной группе поля Галуа при небольшой характеристике поля.

Фактически, ρ-метод Полларда состоит из алгоритма построения псевдослучайной последовательности и алгоритма обнаружения коллизии.

В работах [2–4] предложен параллельный ρ-метод Полларда для систем с общей памятью. Недостатком указанного метода является несбалансированность вычислительных потоков при распараллеливании алгоритма построения псевдослучайной последовательности.

В данной работе предлагается усовершенствованный метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой за счет построения детерминированной функции разбиения на множества. Полученный метод также применяется и для мультипликативной группы поля Галуа.

Отметим, что ρ-метод Полларда решения задачи дискретного логарифмирования не зависит от структуры группы, поэтому, в дальнейшем будет выбрана аддитивная форма записи и описано изложение метода для группы точек эллиптической кривой, которое тривиальным образом переносится на случай мультипликативной группы поля Галуа.

2. Метод Полларда

Пусть задана группа точек эллиптической кривой, которая будет обозначаться как $E(\mathbb{F}_p)$, такая что $\#E(\mathbb{F}_p) = n \cdot cof$, где n – простое число, cof – небольшое натуральное число. Не ограничивая общности, можно предположить, что $p > 3$ и p – простое число. Обозначим

подгруппу $E(\mathbb{F}_p)$ порядка n через G и зафиксируем порождающий элемент P.

Для произвольного элемента группы $Q = xP$ задача дискретного логарифмирования заключается в нахождении элемента $1 < x < n$.

2.1. Последовательный ρ-метод Полларда

Группа G представляется в виде объединения $G = S_1 \cup S_2 \ldots \cup S_N$, где S_i – произвольные множества приблизительно одинаковой мощности, N – натуральное число. Функция итерирования $f : G \to G$ определяется как

$$R_{i+1} = f(R_i) = \begin{cases} Q + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ P + R_i, & R_i \in S_3 \end{cases} \tag{1}$$

Пусть $R_{i+1} = a_i P + b_i Q$, тогда коэффициенты определяются следующим образом

$$a_{i+1} = \begin{cases} a_i \mod n, & R_i \in S_1 \\ 2a_i \mod n, & R_i \in S_2 \\ a_i + 1 \mod n, & R_i \in S_3 \end{cases} \tag{2}$$

$$b_{i+1} = \begin{cases} b_i + 1 \mod n, & R_i \in S_1 \\ 2b_i \mod n, & R_i \in S_2 \\ b_i \mod n, & R_i \in S_3 \end{cases} \tag{3}$$

Так как группа G – конечна, то последовательность $\{R_i\}_{i=0}^{\infty}$ – периодическая. Таким образом, найдутся два наименьших натуральных числа t и l, таких что $R_t = R_{t+l}$. Фактически, l – означает длину периода последовательности.

Идея алгоритма детектирования цикла Флойда заключается в нахождении индекса i_0, такого что $R_{i_0} = R_{2i_0}$, $i_0 \leq t + l$ при произвольно фиксированном начальном значении R_0.

Отметим, что

$$\begin{aligned} R_i &= f\left(R_{i-1}\right) \\ R_{2i} &= f\left(f\left(R_{i-1}\right)\right) \end{aligned} \tag{4}$$

Это означает, что на каждой итерации производится сравнение R_i и R_{2i} для $0 < i \leq t + l$ пока не будет обнаружена коллизия $R_{i_0} = R_{2i_0}$.

Учитывая, что

$$\begin{aligned} R_{i_0} &= a_{i_0} P + b_{i_0} Q \\ R_{2i_0} &= a_{2i_0} P + b_{2i_0} Q \end{aligned} \tag{5}$$

можно вычислить

$$x = \frac{a_{2i_0} - a_{i_0}}{b_{i_0} - b_{2i_0}} \tag{6}$$

Отметим, что i_0 зависит от начального значения R_0 и определяет вычислительную сложность метода Полларда.

Заметим, что принадлежность $R_i = (x_i, y_i)$ к подмножеству S_j, $1 \leq j \leq N$ на практике определяется либо младшими, либо старшими разрядами R_i.

Например:

$$j = \nu(R_i) = x_i \mod N + 1, \tag{7}$$

где $\nu : G \to \{1, \ldots N\}$ – функция разбиения группы на множества.

2.2. Параллельный ρ-метод Полларда с использованием детерминированной функции разбиения на множества

В работе [3] предложен параллельный ρ-метод Полларда для систем с общей памятью. Идея такого метода заключается в распараллеливании отдельно функции итерирования и алгоритма Флойда детектирования цикла.

Учитывая накладные расходы, связанные с использованием потоков, вычисление и сравнение точек R_i и R_{2i} за одну итерацию является неэффективным. Поэтому, было предложено вычислить блок элементов определенной длины для точек R_i и параллельно вычислить блок элементов для точек R_{2i}, после чего параллельно сравнить элементы, содержащиеся в первой и второй половинах блока, как показано на рисунке 1.

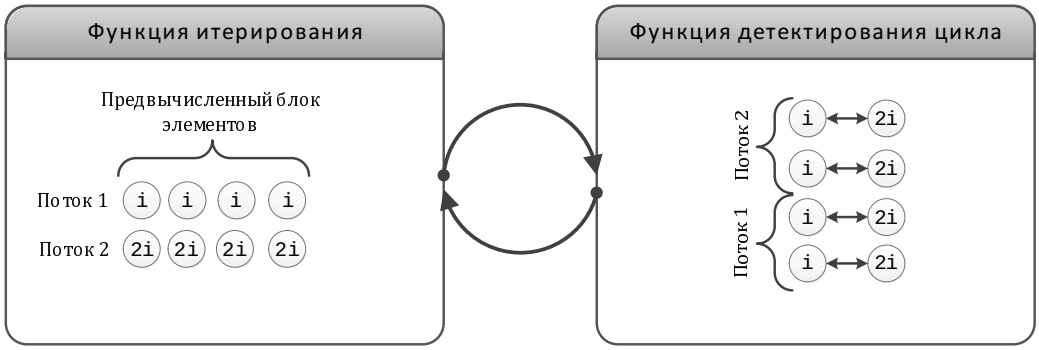


Рис. 1. Параллельный алгоритм Полларда с предвычисленным блоком

Следует отметить, что вычисление точки R_{2i}, согласно формуле (4), требует последовательного вычисления двух образов функции f. Так как функция является итеративной и определяется функцией разбиения группы на множества, то в общем случае нельзя свести ее к явному представлению в виде композиционной функции. Это означает, что вычисление R_i в одном потоке является в два раза медленнее, чем вычисление R_{2i} в другом.

Для балансировки нагрузки между потоками, желательно представить вычисление двух образов функции f в виде вычисления одного образа композиционной функции. Это может быть достигнуто использованием детерминированной функции разбиения абелевой группы на множества.

Определим детерминированную функцию $\xi : G \to \{1, \dots N\}$ разбиения на множества как

$$j = \xi(R_i) = i \mod N + 1. \tag{8}$$

Такая функция разбиения абелевой группы на множества позволяет сделать предсказание траектории перехода и построить композиционную функцию.

Утверждение 1 Пусть задана детерминированная функция разбиения группы на множества $\xi(R_i) = i \mod N + 1$, тогда вычисление R_{2i} может быть представлено как

$$R_{2i} = g(S_{i-1}), \tag{9}$$

где $g : G \to G$, $S_{i-1} = R_{2(i-1)}$.

Докажем утверждение 1. Определим функцию итерирования g следующим образом:

$$S_{i+1} = g(S_i) = \begin{cases} 2(Q + S_i), & S_i \in S_1 \\ S_i + P + Q, & S_i \in S_2 \\ 2S_i + P, & S_i \in S_3 \end{cases} \tag{10}$$

По определению $R_{2i} = f\left(f\left(R_{i-1}\right)\right)$. Так как функция разбиения группы на множества представлена уравнением (8), то

$$R_{2i} = \begin{cases} 2(R_{2(i-1)} + Q), & R_{2(i-1)} \in S_1 \\ R_{2(i-1)} + P + Q, & R_{2(i-1)} \in S_2 \\ 2R_{2(i-1)} + P, & R_{2(i-1)} \in S_3 \end{cases} \quad (11)$$

Следовательно, $R_{2i} = S_i = g(S_{i-1})$.

Таким образом, вычисление двух итераций функции f может быть сведено к вычислению одного образа функции g с использованием детерминированной функции разбиения группы на множества.

Отметим также, что изменение псевдослучайной функции итерирования элементов на детерминированную, с одной стороны, влияет на статистические свойства обнаружения коллизии, предположительно, в худшую сторону, а с другой стороны, ведет к балансировке потоков при построении блока элементов и, следовательно, к уменьшению общей вычислительной нагрузки алгоритма Полларда.

2.3. Обобщение метода Полларда с детерминированной функцией разбиения на множества на произвольную функцию итерирования

Рассмотрим известные модификации [1] функции итерирования для последовательного алгоритма Полларда, а именно, обобщенную функцию итерирования Полларда, функцию итерирования Теске и смешанную функцию итерирования Теске.

Обобщенная функция итерирования Полларда представляется в виде:

$$R_{i+1} = f_{PG}(R_i) = \begin{cases} W_1 + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ W_2 + R_i, & R_i \in S_3 \end{cases} \quad (12)$$

где $W_1 = t_1 P$, $W_2 = t_2 Q$, t_1,t_2 - случайным образом выбранные числа, такие что $0 < t_1, t_2 \leq n$.

Предположим, что предварительно вычислено r значений $W_i = t_i^1 P + t_i^2 Q$, $i = 1..r$. Определим функцию $\nu : G \to \{1..r\}$, тогда функция итерирования Теске может быть описана следующим образом:

$$R_{i+1} = f_{TA}(R_i) = R_i \cdot W_{\nu(R_i)}, \nu(R_i) \in 1..r \quad (13)$$

Смешанная функция итерирования Теске записывается как:

$$R_{i+1} = f_{TM}(R_i) = \begin{cases} R_i \cdot W_{\nu(R_i)}, & \nu(R_i) \in 1..r \\ 2R_i, & \nu(R_i) \notin 1..r \end{cases} \quad (14)$$

Отметим, что исходя из представления функций, тривиальным образом можно распространить идею алгоритма Полларда для систем с общей памятью с использованием детерминированной функции разбиения на множества, а именно, выполнять последовательный переход от одного подмножества к другому.

Такой подход позволит предсказывать траекторию движения функции итерирования, что даст возможность организовывать балансировку потоков более оптимальным способом.

3. Результаты моделирования ρ-метода Полларда с использованием детерминированной функции разбиения на множества

Для реализации описанных выше алгоритмов использовались: процессор Intel (R) Core (TM)2 Duo CPU E6850 3.00GHz, ОЗУ – 2Gb , ОС – Windows 7. Ограничение случаем для

двуядерных процессоров вызвано тем, что итерируются параллельно не более двух последовательностей.

В таблице 1 приведено время выполнения в секундах (δ_t) для последовательного и параллельного методов Полларда для систем с общей памятью. Временная оценка проводится в зависимости от длины блока согласно алгоритму на рисунке 1.

Таблица 1. Временные показатели методов Полларда

№	Методы Полларда	δ_t (сек)	
		21 бит	**27 бит**
1	Классический метод	12.4141	1069.49
2	Параллельный метод (w=1)	9.6856	890.812
3	Параллельный метод (w=4)	8.70438	765.313
4	Параллельный метод (w=8)	8.45161	786.541
5	Параллельный метод (w=128)	8.21501	810.267
6	Параллельный метод (w=512)	8.17755	726.259
7	Параллельный метод (w=1024)	8.1422	706.71
8	Параллельный метод (w=2048)	8.1446	702
9	Параллельный метод (w=4096)	8.21867	706.509

Отметим, что детерминированная функция разбиения группы на множества позволила при программной реализации использовать две независимых функции итерирования для вычисления R_i и R_{2i}, использовать предвычисления в функции итерирования g для значения $P + Q$, а также отказаться от условных операторов при определении принадлежности точки R_i множеству S_i, что позволило минимизировать потери, связанные с неправильным предсказанием переходов и в полной мере использовать параллельные вычисления.

4. Заключение

В работе предложен метод распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой для систем с общей памятью с использованием детерминированной функции разбиения абелевой группы на множества. Такой подход позволяет оптимально использовать преимущества как многопроцессорных, так и многоядерных систем. В работе приведены эмпирические временные показатели для предложенного параллельного метода Полларда. Моделирование проводилось для двуядерных процессоров, что обусловлено наличием двух последовательностей в алгоритме обнаружения цикла. Предложенный подход к распараллеливанию алгоритма Полларда позволил снизить время вычислений на 30%.

Следует отметить, что сравнение производилось для псевдослучайной функции итерирования и детерминированной для нескольких начальных значений. Такое сравнение не позволяет полноценно сделать вывод о степени ухудшения или улучшения статистических характеристик метода Полларда.

В дальнейшем планируется обобщить полученные результаты на случай произвольного

числа ядер и на кривые с большей битовой длиной порядка подгруппы, а также рассмотреть другие варианты детерминированных функций.

Отметим также, что анализировался только один алгоритм обнаружения цикла, а именно алгоритм Флойда [1]. Необходимо также проанализировать альтернативные алгоритмы, например, алгоритм Брента [1]. Структура алгоритма Брента смогла бы позволить построить конвейер вычислений, на котором при вычислении следующего блока точек ЭК на одном ядре, происходит поиск коллизии на другом.

В дальнейшем также необходимо исследовать влияние начального значения для функции итерирования на выбор длины блока. Отметим, что исходя из экспериментальных данных, оптимальной длиной блока является $w = 512$ или $w = 1024$.

Литература

1. Bai S., Brent R. P. On the efficiency of Pollard's rho method for discrete logarithms // Fourteenth Computing: The Australasian Theory Symposium (CATS 2008), January 22–25, 2008, Wollongong, NSW, Australia, Proceedings. CRPIT, 77. Harland J. and Manyem P., Eds. ACS. P. 125–131.

2. Качко Е.Г., Погребняк К.А. Параллельный метод Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой // Параллельные вычислительные технологии (ПАВТ–2012): труды международной научной конференции (Новосибирск, 26–30 марта, 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. – С. 723.

3. Горбенко И.Д., Качко Е.Г., Погребняк К.А. Методы распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования для систем с общей памятью // Высокопродуктивные вычисления (HPC–UA'2012): труды международной научной конференции (Киев, 8–10 октября, 2012 г.). Киев: НАНУ, 2012. – С. 152–157.

4. Горбенко И.Д., Качко Е.Г., Погребняк К.А. Параллельный метод Полларда решения задачи дискретного логарифмирования в мультипликативной группе поля Галуа // Современные проблемы информационной безопасности на транспорте (СПИБТ–2012): материалы всеукраинской научно-технической конференции с международным участием (Николаев, 29–30 ноября, 2012 г.). Николаев: НУК, 2012. – С. 9–11.

Комплексный подход к анализу данных мониторинга высокопроизводительных вычислительных установок

С.С. Конюхов[1], А.А. Московский[1], Е.А. Рябинкин[2], В.Е. Велихов [2]

Группа компаний РСК[1], НИЦ «Курчатовский институт»[2]

Рост масштабов современных высокопроизводительных вычислительных систем предъявляет все более высокие требования к автоматизации управления этими вычислительными комплексами. В данной работе рассматривается подход, основанный на использовании современных методов статистического анализа наборов однотипных данных, временных рядов, к анализу показаний численных сенсоров состояния вычислительной системы. При помощи как стандартных, так и вновь разрабатываемых методик удается автоматически определять режим работы системы (классифицировать поведение), детектировать аномальные состояния, решать задачи визуализации.

1. Введение

Современные высокопроизводительные вычислительные комплексы представляют собой сложные системы. Многолетней тенденцией является устойчивый рост среднего числа узлов в высокопроизводительном вычислительном кластере, наблюдаемый, например, как 38% среднегодовой прирост числа процессорных сокетов для машин в рейтинге суперЭВМ «TOP 500» [1]. В настоящее время эксплуатируются системы с характерным числом узлов 1000-10000. Наращивание числа узлов ведет к усложнению задачи управления (администрирования) вычислительного комплекса.

В настоящее время схема работы службы эксплуатации построена на использовании программных комплексов мониторинга состояния эксплуатируемых установок [2-5]. Программные комплексы собирают и визуализируют информацию о состоянии вычислительного комплекса (ВК).

В большинстве случаев в системах мониторинга кластерных систем отслеживается состояние индивидуальных узлов. Критическое состояние детектируется для индивидуальных компонент либо по превышению пороговых значений (например, перегрев по превышению температуры), либо по достаточно долгому отсутствию обновления данных. В то же время для современного оборудования количество показателей, определяемых на каждом отсчете, может достигать порядка 100, что делает задачу определения пороговых значений и детектированию аномалий достаточно нетривиальной.

Современные системы мониторинга масштабируемы и легко настраиваемы под нужды каждой конкретной установки. В то же время, в случае нештатной ситуации, решение о начале каких-либо действий по исправлению принимается на основе анализа данных мониторинга оператором. При быстром развитии событий скорость реакции человека может оказаться недостаточной, равно как и трудно обеспечить выбор верного варианта поведения в нестандартной ситуации. За счет большой выделяемой тепловой мощности, современные установки могут получить серьезные повреждения при отказе инженерных подсистем. Например, в случае отказа или падения мощности подсистемы охлаждения, на принятие решения об остановке вычислительных узлов остается обычно несколько минут. С наблюдаемым быстрым увеличением масштаба установок необходимость в автоматизации процесса принятия решений по управлению вычислительными комплексами будет только расти.

Систему жизнеобеспечения вычислительного кластера можно рассматривать как сложную динамическую систему, изменение состояний которой можно попытаться описать, если в качестве динамических параметров определить показания сенсорных датчиков, осуществляющих мониторинг вычислительного комплекса.

К настоящему времени сложилось два основных подхода к проблеме описания эволюции и асимптотического поведения динамических систем. Первый подход основан на априорном знании законов, лежащих в основе динамики изучаемого явления и описывающих переходы си-

стемы из одного состояния в другое. Примером такого подхода может служить рассмотрение динамической системы как синонима системе дифференциальных уравнений (автономной или нет, в зависимости от того, является ли изучаемая динамическая система стационарной или нет). Успех этого подхода в большой степени зависит от степени адекватности применяемой аналитической модели рассматриваемому явлению

Второй, вероятностный или статистический подход используется в тех случаях, когда отсутствует точная информация о внутренних или внешних причинах, определяющих динамическое поведение изучаемой системы, или же когда взаимосвязи между динамическими параметрами столь сложны, что не представляется возможным построить более-менее адекватную и пригодную в практических вычислениях аналитическую модель. Этот подход позволяет, не вдаваясь в детали описания динамики системы, тем не менее с той или иной степенью успеха описывать динамику системы в целом.

В настоящей работе, рассматривающей динамическое изменение состояний системы жизнеобеспечения ВК, из-за наличия в ней большого числа динамических параметров с крайне разнообразными характеристиками, которые включают в себя как экзогенные факторы вроде температуры или влажности воздуха на выходе из системы охлаждения, напряжения и силы тока, подаваемого на вычислительные узлы, пользовательской нагрузки на ВК и многое др., так и эндогенные факторы: потребляемая вычислительным кластером мощность, характеристики воздушного потока, создаваемого системой кондиционирования, температура и влажность воздуха в межстоечном пространстве и многое др., представляется разумным применение именно второго, статистического подхода. Конечной целью работы является создание набора методик, позволяющего решать практические задачи по автоматическому выделению аномалий в поведении вычислительных комплексов, визуализации состояния ВК для оператора (администратора) установки.

2. Теоретическая часть - Метод анализа данных

В нашей работе алгоритм анализа данных мониторинга состоит в построении адекватных статистических моделей: модели, описывающей набор собранных ранее данных, т. н. «множество нормальных состояний» и модели, позволяющей вычислять возможные будущие состояния ВК, модели регрессора, и в применении модели «множества нормальных состояний» для классификации состояний, полученных в модели регрессора.

2.1 Представление данных

Состояние вычислительного комплекса в каждый момент времени представляется в виде случайного вектора:

$$S(t) = (r_1(t), r_2(t), \dots , r_N(t)),$$

где $r_i(t)$ – показания сенсоров установки в момент времени t ($i = 1, \dots , N$), N - число сенсоров, осуществляющих мониторинг вычислительного комплекса.

В такой постановке задача анализа данных мониторинга может быть сформулирована как анализ статистических свойств многомерного временного ряда, где под значениями временного ряда в конкретный момент времени понимается вектор состояния, компоненты которого есть набор показаний сенсорных датчиков в данный момент времени.

2.2 Регуляризация набора данных

Перед началом работы всего алгоритма данные требуют процедуры предварительной предобработки, т.н. регуляризации, Это связано с рядом причин: нерегулярность, асинхронность поступления данных измерений от отдельных датчиков и появление отдельных выпадающих значений, что мешает построению адекватных статических моделей

Во многих работах проводится регуляризация на основе интерполяции значений таким образом, чтобы измерения были доступы через одинаковые интервалы – такой подход, например, по умолчанию применяется в популярном средстве хранения данных RRDB [8]. В таком случае

возможно исчезновение «пиковых» значений измерений после регуляризации, что может быть в некоторых ситуациях нежелательным.

Регуляризация в нашем случае проводится по следующей схеме:

1) оценивается интервал времени, содержащий показания со всех сенсорных датчиков;

2) полученный на шаге 1 интервал времени разбивается на заданное пользователем число подинтервалов одинаковой длины;

3) каждому из полученных на шаге 2 значений времени приписывается набор показаний сенсорных датчиков в ближайший слева момент времени, несколько начальных значений времени из исходной нерегуляризованной выборки при этом отбрасываются, если для них не определены все компоненты многомерного вектора состояний ВК.

Дополнительно при построении обеих моделей может проводится отброс выпадающих значений, которыми считаются все значения, не попавшие в интервал допустимый значений, определяемый индивидуально для каждого типа сенсорных датчиков. Если такой интервал неопределен, то отбрасываются те значения, z-координата которых больше 3.29 (5% уровень значимости для нормального распределения).

С одной стороны, такой фильтр значений позволяет снизить влияние отдельных выпадающих значений на качество прогноза, и наш опыт эксплуатации ВК показывает, что наиболее важным фактором является наблюдение за трендами в показаниях сенсорных датчиков. Вариант предподготовки данных с отбрасыванием выпадающих значений это допускает. Но с другой стороны, именно выявление аномального поведения показаний сенсорных датчиков является основной целью данной работы, поэтому этот вопрос требует дальнейшего исследования.

2.3 Анализ временных рядов – методы прогноза

Существует множество методов для представления и анализа временных рядов [6, 7]. Наиболее простыми являются параметрические модели, как линейные, так и нелинейные. К числу первых относятся такие широко используемые методы как векторный метод авторегрессионой модели (VAR) или метод пространства состояний (SS), а также многие другие. К числу вторых можно отнести методы на основе многомерных сплайнов, которые с практической точки зрения представляют определенную сложность как в использовании, так и в трактовке результатов.

В последнее время все больший интерес исследователей стали привлекать т. н. непараметрические методы статистического анализа, наиболее перспективным из которых является построении модели регрессии на основе метода опорных векторов [9].

Мы в своей дальнейшей работе предполагаем исследовать вопрос применимости как параметрического, так и непараметрического подходов. В качестве первого нами был выбрана линейная модель «векторной авторегрессии с экзогенными переменными» (vector-autoregressive model with exogenous variables, VARX), а в качестве второго – метод k-кратной регрессии на основе метода опорных векторов (k-fold SVR).

В общем виде модель VARX можно представить в следующем виде:

$$A(L)y_t = \epsilon_t + B(L)x_t,$$

где $A(L)$, $B(L)$ матричные функции от оператора лага L, x_t и y_t – экзогенные и эндогенные переменные, ϵ_t – вектор ошибок, обладающий определенными заранее заданными свойствами.

Построение модели VARX может осуществляться одним из способов, используемых для построения параметрических моделей, например, либо с помощью стандартного метода наименьших квадратов, либо с помощью метода максимального правдоподобия. На основе модели VARX можно продолжить временной ряд и вычислить прогноз для нескольких следующих временных шагов.

После построения модели VARX происходит оценка ее качества и качества предсказания. Для этих целей используется набор стандартных статистических тестов и информационных критериев.

Основным достоинством непараметрических методов является их универсальность и независимость от априорных знаний. Метод k-fold SVR заключается в последовательном применении метода SVR к каждой из компонент многомерного вектора. Метод SVR состоит в построе-

нии такой линейной функции, которая наилучшим образом приближала бы значения искомой функции. Для поиска такого приближения используется принцип минимизации структурного риска (SRM principle), т.е. минимизация следующего функционала структурного риска:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} L\big(y(i), f(x(i), w)\big),$$

где $L - \epsilon$ – лосс-функция того или иного вида.

2.4 Обнаружение аномального поведения ВК на основе решения задачи об обнаружении выпадающих значений методом SVM

Выявление аномального поведения ВК происходит путем соотнесения результатов прогнозирования или набора данных за последние моменты времени с классом всевозможных состояний ВК, наблюдавшихся в прошлом и признанных экспертами1 как типичные и соответствующие нормальной работе ВК. Данное множество строится с помощью подхода на основе кернел-методов.

Фундаментальной идеей кернел-методов, к которым относится метод опорных векторов (SVM), является отображение входных данных, заданных в произвольном множестве, т.н. входном пространстве (input space), в такое гильбертово пространство, часто называемое характеристческим (feature space), в котором можно использовать хорошо изученные методы линейной алгебры и евклидовой геометрии [10].

Такое неявное преобразование F принято называть кернел-функцией (kernel function). Теоретической основой для подобного подхода является теорема Мерсера, утверждающая, что любая непрерывная симметричная полуположительно определенная кернел-функция $k(x,y)$ может быть выражена как скалярное произведение в некотором евклидовом пространстве подходящей размерностью

$$k(x, y) = \langle F(x), F(y)\rangle.$$

Одной из самых простых операций, которые можно выполнить в евклидовом пространстве, является построение плоскости, которая бы отделяла друг от друга точки двух разных множеств, классов. Другими операциями могут быть кластеризация данных или организация их по какому-либо другому принципу.

Было разработано множество различных вариантов метода SVM с учетом специфики решаемой ими задач. Например, изменив постановку классической задачи о бинарной классификации данных методом SVM, авторам [11] удалось создать простой и надежный алгоритм классификации с одним классом, который можно использовть для решения т.н. задачи об определении новизны (novelty detection) или об определении выпадающих значений (outliers detection). В этом методе тестовое изучаемое множество в характеристическом пространстве отделяется гиперплоскостью от начала координат.

Схожим образом в методе описания областей с помощью метода опорных векторов (support vector density description, SVDD) [12] изучаемое множество в характеристическом пространстве заключается в гиперсферу минимального радиуса. Для наиболее распространенного класса кернел-функций, т.н. RBF кернел-функций (radial-based function RBF). Эти два подхода дают схожие результаты.

Также стоить упомянуть перспективный подход на основе кернел-функций, несвязанный с нахождением опорных вектров, т.н.нелинейный анализ главных компонент (kernel PCA). В работе [13] было показано,что при определенных условиях этот метод может давать лучшие результаты в определении выпадающих значений по сравнению с методами на основе метода опорных векторов. Однако к недостаткам этого метода следует отнести его большую вычислительную стоимость, обусловленную вычислением обратных матриц и собственных значений, а также неустойчивость к помехам при обучении алгоритма.

В данной работе для описания «множества нормальных состояний» X мы использовали классическую задачу о бинарной классификации, где в качестве дополнительного второго

---

1 в их роли, как правило, выступают операторы систем управления ВК или инженерно-технический персонал, обслуживающий работу ВК

множества было взято центрально-симметричное к множеству X множество $-X$. В такой постановке эта задача эквивалентна задаче о классификации с одним классом [11]. Такой подход позволил нам использовать формулы для расчета апостериорных вероятностей, полученных в работах [14] и [15].

3. Программная реализация

Экспериментальное ПО, разработанное и реализованное в рамках данной работы, включает в себя следующие модули:

- модуль сбора данных мониторинга;
- пользовательский интерфейс;
- система обнаружения внештатных ситуаций;
- модуль внешней интеграции.

Модуль сбора данных поддерживает как работу с RRDB из пакета RRDTool, так и средства сбора данных из SNMP источников. Реализована поддержка сбора данных из ПО APC InfrastruXure Central [16] и Netbotz [17], которые осуществляют мониторинг различных инфраструктурных сервисов, таких как прецизионные и межстоечные кондиционеры различных производителей, температурные и влажностные датчики, размещенные в различных точках инфраструктуры залов, распределители питания, индивидуальные компоненты серверной инфраструктуры.

Данные пакеты осуществляют сбор и визуализацию данных как с SNMP-совместимых источников данных, так и с датчиков, доступных по различным индустриальным протоколам: Modbus, one-wire и другим, что позволяет получать информацию от широкого количества датчиков универсальным образом.

Пользовательский интерфейс нашего экспериментального ПО реализован в виде веб-приложения, которое предоставляет пользователю возможность удаленно осуществлять загрузку данных мониторинга ВК и запуск алгоритмов их анализа. Модуль реализован с помощью фреймворка Symfony [18], который представляет собой набор библиотек функций и скриптов на языках программирования php и JavaScript, а также html-шаблонов для создания содержимого веб-сайтов.

Модуль обнаружения внештатных ситуаций реализован при помощи существующих библиотек методов статистического анализа из пакета программ R-software [19, 20].

4. Первые результаты

Для тестирования разработанного ПО нами были собраны данные мониторинга установки - вычислительного кластера. Всего набор включает данные от более чем 1000 сенсоров. Наблюдаемые характеристики включали:

- температуры процессоров и модулей памяти вычислительных узлов;
- силу тока и рабочие напряжения на шкафных блоках распределителей питания;
- влажность воздуха на входе в шкаф;
- температура воздуха на входе в шкаф;
- характеристики кондиционеров: охлаждаемая мощность, параметры теплоносителя: давление, входная и выходная температура.

4.1 Визуализация набора данных

Показания сенсорных датчиков с точки зрения их статистической обработки представляют собой довольно сложный объект. Временные ряды, соответствующие им, как правило, являются нестационарными с резко и хаотически меняющимися статистическими свойствами. На Рис. 1 представлены два типичных примера таких данных, которые условно можно отнести к двум разным классам статистических данных. На этом рисунке приведены как измеренные данные для зависимости сила тока (в А) на шкафных блоках распределителей питания от времени (в мс), так и полученные с помощью модели регрессора k-fold SVR.

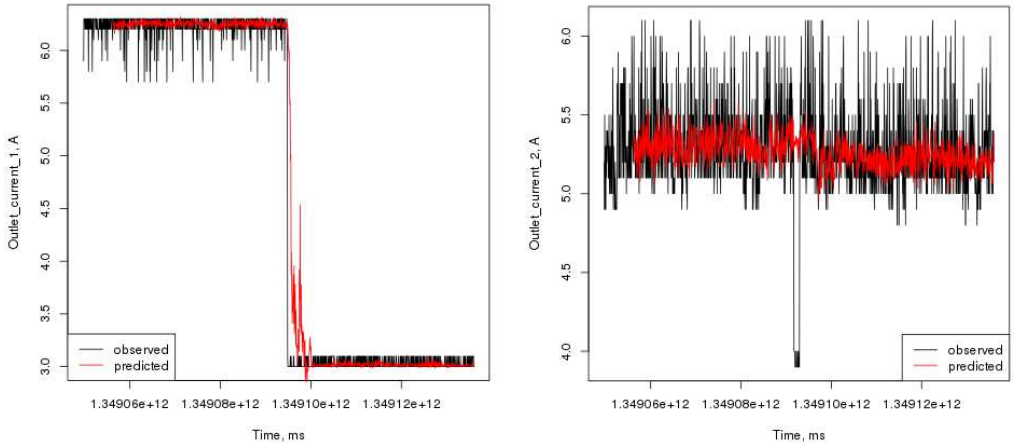


Рис. 1. Типичные зависимости данных мониторинга от времени.

В общем случае для визуализации многомерных данных принято использовать метод анализа главных компонент, имеющий для этого целый ряд оптимальных свойств [21].

4.2 Выделение аномалий

В настоящий момент нами был реализован следующий базовый алгоритм. После регуляризации данных мониторинга происходит построение модели регрессора и осуществление на ее основе прогнозирования будущих возможных состояний ВК. Полученный результат прогноза поступает на вход классификатору, построенному на основе метода SVM, который производит классификацию результата прогноза, т.е. отнесение его к множеству типичных значений, а также вероятности попадания в такое множество, состоящее как правило из нескольких кластеров, каждое из которых обрабатывается отдельно. Обучающими данными этого классификатора служат показания сенсорных датчиков в прошлом.

На Рис. 1-3 приведены результаты моделирования для тестовой системы, состоящей из двух сенсорных датчиков силы тока с использованием метода k-fold SVR в качестве модели регрессора (размерность входного вектора = 20, число обучающих точек = 100).

На Рис. 2 представлена полученная зависимость полной и максимальной вероятности попадания в «множество нормальных состояний». Резкое падение значений вероятностей в середине приведенных графиков обусловлена резким изменением режима работы ВК, о чем также свидетельствуют данные, приведенные на Рис. 1.

На Рис. 3 приведены зависимости нормированной среднеквадратичной ошибки NRMSE [22] для различных способов предподготовки данных. Как видно из этого рисунка довольно высокое значение NRMSE возникает в те моменты времени, когда сильно меняется режим работы ВК. Здесь стоит отметить, что использование процедуры отброса выпадающих значений позволяет уменьшить ошибку прогноза. По-видимому наиболее компромиссным вариантом является отбрасывание отдельных выпадающих значений только при построении модели регрессора, именно для этого варианта предобработки данных приводятся результаты на Рис. 1-2.

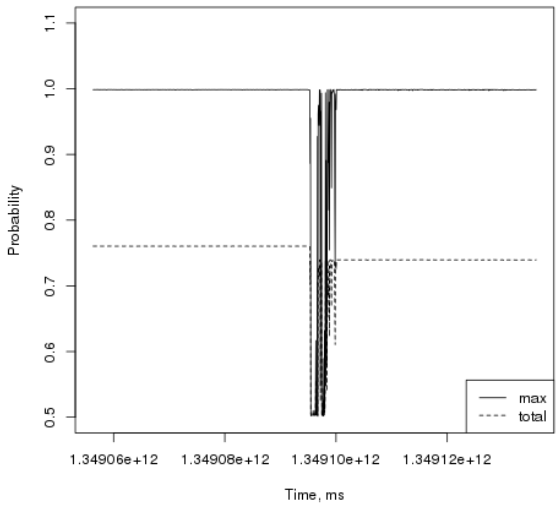


Рис. 2. Зависимости от времени максимальной и полной вероятностей попадания результатов прогноза в «множество нормальных состояний».

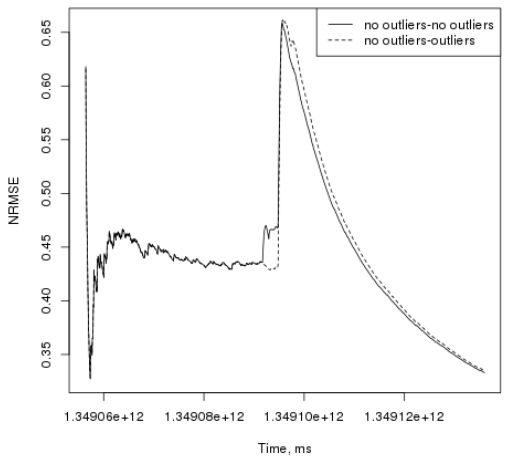


Рис. 3. Зависимости от времени нормированной среднеквадратичной ошибки прогнозирования. Надписи: no outliers - no outliers и no outliers – outliers соответствуют двум вариантам предобработки данных: без отсева выпадающих значений и с отсевом выпадающих значений при построении модели регрессора.

На тестовом примере видно, что применение специальных статистических методов в принципе позволяет правильно классифицировать состояния вычислительной установки. Момент смены режима работы ВК выявляется довольно однозначно.

В общем случае сложность подобной задачи в первую очередь связана с ее высокой размерностью. Типичное число сенсорных датчиков в современных ВК порядка нескольких тысяч, поэтому можно надеяться, что сведение такой задачи как наблюдение за всеми сенсорными датчиками к наблюдению за одним-двумя параметрами (вероятности попадания множество нормальных состояний», ошибки прогноза) позволит решить проблему мониторинга ВК.

5. Заключение

В данном исследовании показано, что ряд стандартных статистических методов анализа и классификации данных в принципе может быть успешно использован для углубленного анализа данных мониторинга высокопроизводительных вычислительных комплексов. В отличии от традиционного подхода, рассматривающего независимо отдельные компоненты системы (вычислительные узлы, агрегаты) в данной работе обоснован комплексный подход, позволяющий

оценить состояние системы в целом с учетом взаимодействия различных компонент друг с другом.

В то же самое время, построенная базовая модель требует дальнейшего тестирования на наборах данных близких к реально используемым в мониторинге ВК с целью определения ее недостатков и их устранения, а также отработки методик работы с ней и трактовки получаемых с ее помощью результатов.

Мы предполагаем, что дальнейшее развитие предложенного подхода позволит разработать эффективные и востребованные средства анализа данных мониторинга ВК. В последующей работе внимание будет уделено не только усовершенствованию моделей анализа данных, но также сбору большего объема данных и развитию программных средств анализа, включая анализ в реальном времени и интерактивные средства взаимодействия с пользоватателем.

Литература

1. Dongarra J. Supercomputers and Clusters and Grids, Oh My!
   URL: http://www.netlib.org/utk/people/JackDongarra/SLIDES/osu-0207.pdf
   (дата обращения: 12.11.2012).
2. Ganglia Monitoring System.
   URL: http://ganglia.sourceforge.net (дата обращения: 12.11.2012).
3. Nagios - The Industry Standard in IT Infrastructure Monitoring.
   URL: http://www.nagios.org (дата обращения: 12.11.2012).
4. Clumon: The Cluster Monitor System.
   URL: http://clumon.ncsa.illinois.edu (дата обращения: 12.11.2012).
5. Supermon.
   URL: http://supermon.sourceforge.net (дата обращения: 12.11.2012).
6. Хенан Э. Многомерные временные ряды. М., Мир, 1974. 576 с.
7. Hannan E. J., Deistler M. The Statistical Theory of Linear System. J. Wiley & Sons, 1988. 380 p.
8. RRDtool.
   URL: http://oss.oetiker.ch/rrdtool (дата обращения: 12.11.2012).
9. Sapankevych N.I. Time Series Prediction using Support Vector Machines: A Survey //
   IEEE Computational Intelligence Magazine. May 2009. P. 25-40.
10. Cristianini N., Schölkopf B. Support Vector Machines and Kernel Methods // AI Magazine. 2002. Vol. 23, No. 3. P. 31-41.
11. Scholkopf B., Platt J.C., Shawe-Taylot J., Smola A.J., Williamson R.C., Estimating the Support of a High-Dimensional Distribution // Neural Computation. 2001. Vol. 13, P. 1443-1471.
12. Tax D.M.J., Duin R.P.W. Support vector domain description // Pattern Recognition Letters. 1999. Vol. 20. P. 1191-1199.
13. Hoffmann H. Kernel PCA for Novelty Detection // Pattern Recognition. 2007. Vol. 40. P. 863-874.
14. Platt J. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods // Smola A.J., Bartlett P.L., Scholkopf B., and Schurmans D., editors. Advances in Large Margin Classifiers. Cambridge, MA. MIT Press, 2000. P. 21-30.
15. Wu T.-F. Probability Estimates for Multi-class Classification by Pairwise Coupling // Journal of Machine Learning Research. 2004. Vol. 5. P. 975-1005.

16. APC's InfraStruXure Central, Change Manager and Capacity Manager Enable Comprehensive Datacenter Management with Predictive Simulation and Modeling.

    URL: http://www.apc.com (дата обращения: 12.11.2012).

17. NetBotz - Remote Environmental Monitoring, Early Detection, Instant Alerting, Temperature Sensors, Humidity Sensors, Fluid Detectors, Chemical Sensors, IT Physical Security, Unified Security, Critical Infrastructure Protection.

    URL: http://www.netbotz.com (дата обращения: 12.11.2012).

18. High Performance PHP Framework for Web Development – Symfony.

    URL: http://symfony.com (дата обращения: 12.11.2012).

19. The R Project for Statistical Computing.

    URL: http://www.r-project.org (дата обращения: 12.11.2012).

20. Chang Ch.-Ch., Lin Ch.-J. LIBSVM – A Library for Support Vector Machines // ACM Transactions on Intelligent Systems and Technology. 2011. Vol. 2, No. 3. P. 27:1–27:27.

21. Айвазян С. А. Прикладная статистика. Основы эконометрики. Том 2. М., Юнити-Дана, 2001. 432 с.

22. Vlachos I., Kugiumtzis D. State Space Reconstruction for Multivariate Time Series Prediction.

    URL: http://arxiv.org/abs/0809.2220 (дата обращения: 12.11.2012).

О решении систем линейных алгебраических уравнений на многоядерных вычислительных системах с графическими ускорителями*

Б.И. Краснопольский[1,2], А.В. Медведев[2]

НИИ механики МГУ[1], ЗАО "Т-Сервисы"[2]

В работе обсуждаются текущие результаты исследования, проводимого авторами с целью поиска и разработки эффективных алгоритмов распараллеливания методов решения больших разреженных систем линейных алгебраических уравнений на современных многопроцессорных системах. Рассматриваются проблемы, возникающие при реализации итерационных методов подпространства Крылова, в частности стабилизированного метода бисопряжённых градиентов (BiCGStab), с алгебраическим многосеточным предобуславливателем на многопроцессорных системах с гибридными узлами, имеющими в своем составе как многоядерные процессоры, так и графические ускорители. Приводятся результаты анализа наиболее эффективных алгоритмов распараллеливания и технологических приёмов реализации для ряда задач, возникающих при аппроксимации эллиптических уравнений на больших сетках.

1. Введение

Разработка и реализация эффективных методов решения эллиптических уравнений является необходимым условием для решения подавляющего большинства задач механики сплошных сред. Зачастую этап решения таких уравнений является одним из наиболее трудоемких и может занимать более 90% от общего времени решения задачи. В зависимости от формы расчетных областей и топологии используемых сеток на практике могут применяться как прямые, так и итерационные методы решения систем уравнений, получаемых при разностной аппроксимации исходных эллиптических уравнений. Прямые быстрые методы, например [1], имеют ограниченную область применимости, однако в ряде случаев оказываются существенно более эффективными при проведении последовательных расчетов. Итерационные методы в свою очередь требуют большего времени расчета, но при этом обладают большей гибкостью относительно вида матриц систем линейных алгебраических уравнений (СЛАУ) и хорошим параллелизмом. Наиболее широко используемыми в настоящее время для решения больших сильно-разреженных СЛАУ для эллиптических уравнений являются итерационные методы подпространства Крылова [2] и/или многосеточные методы [3]. Многосеточные методы могут использоваться как отдельные решатели, либо применяться в качестве предобуславливателей для итерационных методов. С точки зрения эффективности они сравнимы с методами неполной факторизации [2] при последовательных расчетах, но обладают при этом рядом ключевых особенностей, критических при параллельной реализации методов. К таковым можно отнести инвариантность многосеточных методов относительно переупорядочения элементов матрицы, что играет существенную роль при построении псевдо-оптимального разбиения матрицы между вычислительными процессами, а также слабую зависимость скорости сходимости методов от размера задачи (например, [4]).

Современная архитектура вычислительных систем предполагает наличие нескольких многоядерных процессоров внутри каждого узла (в настоящее время, 12-48 ядер в одном узле), которые могут быть дополнены одним или несколькими ускорителями (GPU, FPGA,

*Работа выполнена при поддержке РФФИ (гранты № 11-01-00088-а и 12-01-31002-мол_а) и компании ЗАО "Т-Сервисы".

Intel Xeon Phi). Столь сложная и комплексная архитектура вычислительных узлов приводит к необходимости разработки соответствующих реализаций численных методов, учитывающих все основные особенности имеющейся аппаратуры. Для достижения приемлемого уровня масштабируемости и эффективности методов необходим пересмотр алгоритмов распараллеливания численных методов с целью выделения нескольких иерархических уровней параллелизма, отвечающих аппаратной платформе.

Отдельную обширную тему для исследований представляет эффективное распараллеливание вычислений для задач линейной алгебры на графических ускорителях. В литературе известно большое количество работ в различных областях вычислительной математики, где были опубликованы результаты многократного ускорения вычислений при использовании графических ускорителей. Однако достижения при портировании задач линейной алгебры в большинстве случаев оказываются существенно скромнее. Основная сложность заключается в том, что рассматриваемый класс задач сводится к алгоритмам, в которых преобладают операции чтения из памяти (memory bound), тогда как графические ускорители демонстрируют преимущество на алгоритмах с преобладанием вычислений. Наблюдаемый прирост производительности для таких задач обычно лежит в пределах 1.5-3 по сравнению с одним процессором, и обусловлен в первую очередь более высокой пропускной способностью шины памяти графических ускорителей по сравнению с центральными процессорами.

Несмотря на широкий практический интерес к реализации многосеточных методов на графических ускорителях и большое количество соответствующих проектов, этот вопрос все еще является актуальной темой для исследований. Говоря о текущем состоянии исследований следует упомянуть наиболее интересные работы [5–10], выполняющиеся как крупными компаниями, так и отдельными группами исследователей. Одной из первых библиотек для задач линейной алгебры, распространяемых в исходных кодах, была библиотека CUSP [5]. В ней был реализован набор основных итерационных методов решения СЛАУ, в т.ч. один из семейства многосеточных методов, однако реализация этих методов до сих пор ограничивается возможностью использования только одного графического ускорителя. Схожий функционал реализован в библиотеке CUSPARSE [6], разрабатываемой компанией NVIDIA, и распространяемой только в бинарном виде. По сравнению с CUSP, данная библиотека обладает чуть более высокой производительностью, но не содержит реализации многосеточных методов. Реализация многосеточных методов была официально анонсирована компанией NVIDIA в библиотеке NVAMG [7] в конце 2012 года, но также позволяет проводить расчеты только на одном ускорителе. Релиз следующей версии библиотеки с возможностью проведения расчетов на нескольких ускорителях анонсирован на середину 2013 года и эта библиотека должна войти в состав пакета ANSYS Fluent 15. Схожая ситуация с проектом GAMPACK [8], в котором сделана попытка портировать многосеточный метод из библиотеки hypre: опубликованы результаты для расчетов на одном ускорителе, тогда как распределенная версия, по заявлениям разработчиков, находится на стадии финального тестирования. Проект LibAMA [9] также ориентирован на разработку распределенной версии многосеточных методов, базирующейся на библиотеке hypre, однако в отличие от предыдущих двух библиотек является исследовательской и свободно распространяемой реализацией. В [11] опубликованы результаты исследования масштабируемости реализованных многосеточных методов на нескольких тестовых задачах в простых расчетных областях. Однако проведенные тесты этой библиотеки показали, что эффективность реализации базового блока для итерационных методов, произведения разреженной матрицы на вектор, уступает ряду других реализаций. Еще один проект Cufflink [10] является в некотором смысле промежуточным между последовательной и полностью распределенной реализациями и основан на принципе “domain decomposition”. При этом и в данном случае распределенная версия кода находится в режиме отладки и тестирования, а представленные результаты и отзывы пользователей свидетельствуют о наличии проблем с масштабируемо-

стью методов при решении практических задач на реальных топологиях расчетных сеток.

Приведенный выше список проектов не претендует на полноту обзора состояния вопроса разработки библиотек численных методов для распределенных вычислений на графических ускорителях, однако, по мнению авторов, в нем отражены наиболее интересные проекты. Несмотря на большой практический интерес к распределенной реализации многосеточных методов, стабильными хорошими результатами масштабируемости пока не отличается ни один из заявленных в открытых источниках проектов. Исходя из этого, целью настоящей работы является: совершенствование алгоритмов распараллеливания итерационных методов подпространства Крылова и многосеточных методов для многоядерных процессоров и реализация параллельной версии в рамках гибридной модели программирования, а также проработка базовых алгоритмических вопросов по переносу части вычислений на графические ускорители, и в перспективе - сопроцессоры Intel Xeon Phi.

2. Основные типы вычислительных операций

Наиболее трудоемким этапом реализации итерационных методов является разработка хорошо масштабируемого набора функций для выполнения операции произведения разреженной матрицы на вектор. В общем случае, никакое распределение данных между вычислительными процессами не обеспечивает полной независимости по данным между параллельно выполняющимися частями алгоритма, и требуется обмен элементами вектора-множителя, находящегося на соседних узлах многоузловой вычислительной системы. Использованная в работе схема распределения данных между вычислительными узлами приведена на рис. 1: матрица и вектора распределеяются по узлам полосами построчно. Коммуникация между узлами в таком случае может осуществляться разными способами. На текущем этапе в работе использованы неблокирующие операции типа “точка-точка” из стандарта MPI-1, однако могут быть использованы и альтернативные механизмы передачи данных. В частности, возможно использование односторонних операций чтения из памяти другого узла. Реализация такого рода операций возможна, например, с использованием операций односторонних коммуникаций из MPI-2 (MPI_Create_window, MPI_Put), или с использованием специальных библиотек, реализующих в том или ином виде идеи разделенного глобального адресного пространства (PGAS) [14], например [15, 16]. Исследование таких вариантов реализации может стать одним из направлений дальнейших исследований.

Исходя из того, что передача данных от одного узла к другому в любых современных коммуникационных системах эффективнее выполняется в виде больших блоков, целесообразно построение алгоритма умножения матрицы на вектор в виде последовательного перемножения частей исходной матрицы, соответствующих частям вектора-множителя, расположенного на каждом из узлов. Для реализации такой схемы предлагается разбиение полос матрицы на блоки пропорционально количеству строк, приходящихся на каждый из вычислительных процессов, а результаты умножения каждого из блоков суммируются после обработки последнего блока матрицы. Здесь целесообразна предварительная оптимизация данных, поскольку большая часть таких блоков не содержит ни одного, либо всего несколько ненулевых элементов. Для недиагональных блоков матрицы возможно провести “сжатие” данных таким образом, чтобы в ходе коммуникаций между процессами пересылались только те значения вектора-множителя, которым соответствуют ненулевые элементы в блоках матриц. Указанные преобразования позволяют существенно уменьшить размер пересылаемых векторов и шаблон коммуникаций вычислительных процессов, сократив количество пересылок в среднем до порядка десяти на один MPI-процесс. При этом, за счет использования неблокирующих операций, время на пересылку данных удается эффективно “замаскировать” вычислениями с локальным фрагментом вектора-множителя.

Помимо произведения матрицы на вектор, в используемых в работе итерационных и многосеточных методах фигурируют еще два типа базовых операций. Один из них – сло-

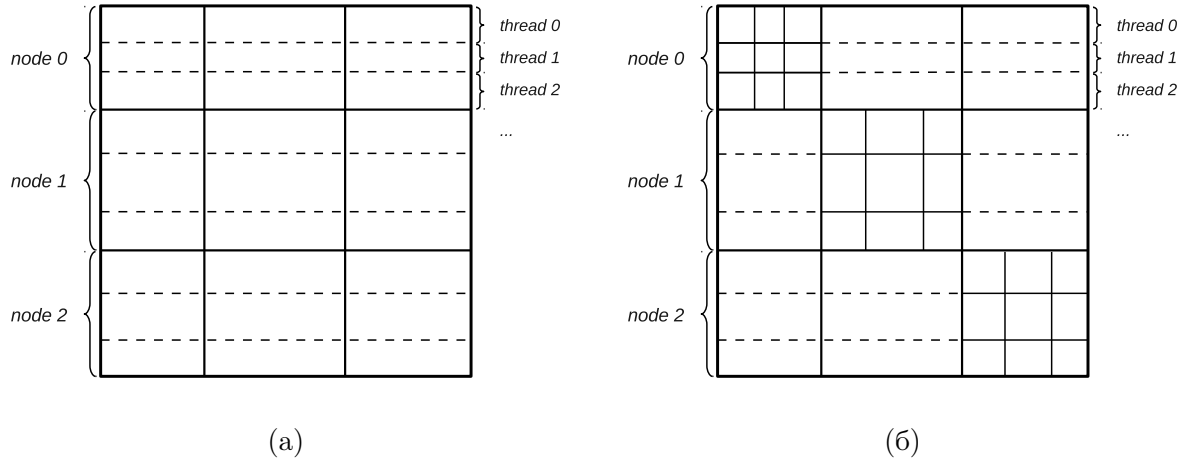
жение векторов/умножения вектора на число. Исходя из принятой схемы разбиения данных между вычислительными процессами, каждый из вычислительных процессов содержит определенные фрагменты векторов, так что указанные операции могут быть выполнены локально без каких-либо коммуникаций с соседними узлами. Еще одной распространенной операцией в итерационных методах является скалярное произведение векторов. Данная операция может быть выполнена локально для имеющихся фрагментов векторов, однако после этого необходимо выполнение глобальной редукции по всем вычислительным процессам, что приводит к возникновению точки глобальной синхронизации всех вычислительных процессов, сказывается на масштабируемости при использовании большого количества вычислительных ядер. Возможные варианты минимизации влияния таких операций на масштабируемость выходят за рамки настоящей статьи, но обсуждались ранее в [4, 13].

3. Гибридная модель MPI+Posix ShM для многоядерных процессоров

"Стандартный" MPI-подход распараллеливания приложения на практике оказывается неприменимым не только при использовании различных ускорителей, но и при проведении вычислений на многоядерных центральных процессорах. При запуске большого количества MPI-процессов на узлах для вычислительно-интенсивных приложений, активно использующих ресурсы коммуникационной сети для пересылки сообщений, возникает перегрузка адаптеров коммуникационной сети, что приводит к увеличению времени передачи сообщений и деградации масштабируемости приложения. Ситуация несколько улучшается с уменьшением количества процессов, распределяемых на каждый из вычислительных узлов, однако это приводит к неэффективной загрузке ресурсов и простою значительной части оборудования. Подробный анализ зависимости эффективности вычислений для задач линейной алгебры от количества вычислительных узлов и используемых ядер обсуждался в одной из предыдущих работ [12].

Решением данной проблемы является использование гибридных моделей параллельного программирования, когда на каждый вычислительный узел приходится только один коммуникационный MPI-процесс, а обмен данными и распараллеливание вычислений внутри узла реализуется тем или иным способом через общую память (OpenMP, Posix Shared Memory, Pthreads и пр.). Такой подход, в зависимости от шаблона коммуникаций приложения, позволяет на один-три порядка сократить общее количество сообщений, пересылаемых через коммуникационную сеть. Первые результаты по разработке двухуровневой схемы распараллеливания в рамках гибридной модели вычислений были представлены авторами в работах [4,12,13]. В настоящей статье обсуждается дальнейшее усовершенствование предложенной схемы многоуровневого распараллеливания вычислений и ее оптимизация с учетом NUMA-архитектуры современных вычислительных систем.

Первая попытка создания алгоритмов распараллеливания выбранных численных методов для решения больших систем линейных алгебраических уравнений в рамках гибридной модели программирования была предпринята в [4]. Изначально разрабатывавшаяся на вычислительной системе СКИФ МГУ "Чебышёв", построенной на базе процессоров с UMA-архитектурой, реализация гибридной модели была достаточно простой. Поскольку все вычислительные ядра имели одинаковую скорость доступа ко всему диапазону оперативной памяти на узле, б*о*льшая часть данных помещалась в область памяти, "видимую" всеми вычислительными процессами (shared memory), а синхронизация и управление доступом к памяти теми или иными вычислительными процессами были реализованы посредством семафоров (рис. 1.а). Разработанная реализация демонстрировала хорошие результаты масштабируемости в сравнении с MPI-версией на вычислительных системах с UMA-архитектурой, однако оказалась не столь эффективной для NUMA-архитектуры [12]: результаты тестов на 24-ядерных узлах на базе процессоров AMD Opteron на одном узле демонстрировали 3-



(а) (б)

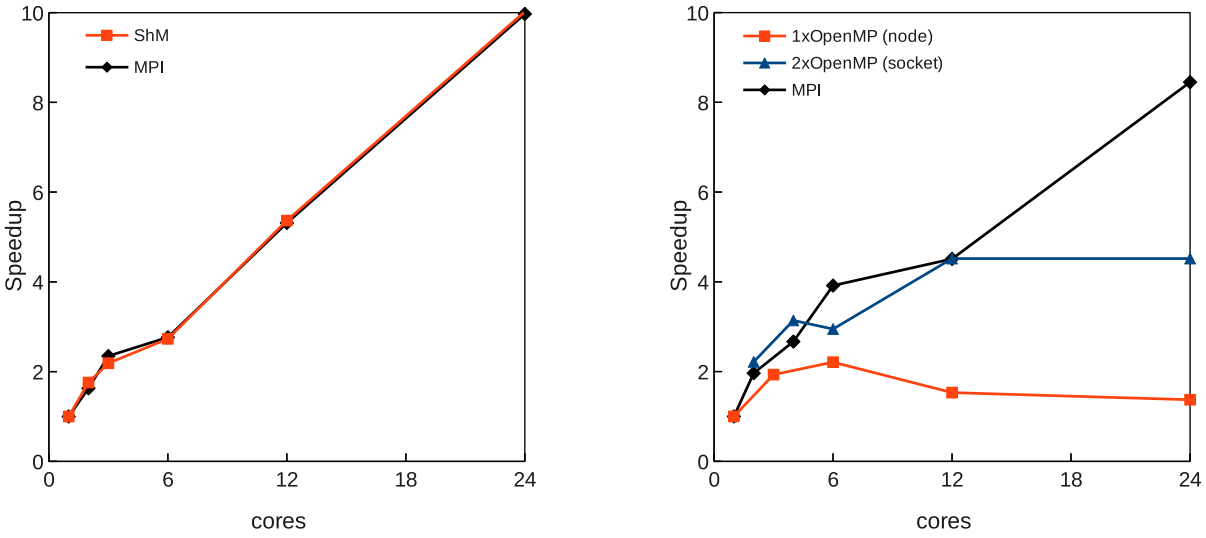
Рис. 1. Схема распределения матрицы между вычислительными процессами: (а) исходная версия для UMA-архитектуры, (б) модифицированная версия для NUMA-архитектуры

кратное превосходство MPI-реализации над гибридной моделью. Данный факт был вызван особенностями организации доступа к памяти. Область общей для всех процессов памяти выделялась одним процессом, в результате чего она физически оказывалась расположенной преимущественно в пределах одного NUMA-узла[1]. Для процессов, запущенных на вычислительных ядрах из других NUMA-узлов, происходило обращение к памяти не локального NUMA-узла через интерфейс HyperTransport, что существенно замедляло скорость доступа к памяти.

Основной целью переработки исходного алгоритма распараллеливания стало изменение схемы распределения данных и их локализация в пределах ближайшего NUMA-узла. Для этого была пересмотрена стратегия разбиения данных между процессами внутри вычислительного узла. Разбиение матриц было реализовано в виде двухэтапной процедуры. Первоначально выполняется разбиение данных между узлами. После этого диагональные блоки матрицы повторно разбиваются на подблоки по количеству вычислительных процессов, запускаемых внутри одного узла (рис. 1.б), но располагаются в ближайших к использующим эти данные ядрам NUMA-банках памяти. При этом, фрагменты вектора-множителя каждого из процессов, как и в предыдущей схеме, остаются видимыми для всех процессов внутри узла, но также располагаются в памяти в локальных для вычислительных процессов NUMA-банках памяти.

Реализованная модификация схемы распределения данных в памяти вычислительного узла позволила существенно улучшить эффективность гибридной версии численных методов. Для тестовых расчетов была выбрана задача решения СЛАУ, полученная при аппроксимации уравнения Пуассона в кубической области на равномерной сетке размером 150^3 ячеек. В качестве решателя использован итерационный метод BiCGStab с алгебраическим многосеточным предобуславливателем. Для построения иерархии матриц многосеточного метода использована библиотека hypre [17]. Разработанная гибридная модель в пределах одного узла демонстрирует результаты масштабируемости, совпадающие с MPI-версией программы и обеспечивает 10-кратное ускорение вычислений (рис. 2.а). Для сравнения на рис. 2.б приведены результаты масштабируемости идентичных методов, реализованных в библиотеке hypre в рамках гибридной модели MPI+OpenMP. При в целом сравнимой производительности MPI-версий решателей (hypre демонстрирует ускорение в 8.6 раза), эффективность реализованной в hypre гибридной модели MPI+OpenMP оказывается существенно хуже, и обеспечивает ускорение всего в 1.4 раза. Несколько лучшие результаты

[1]Использовалась стратегия выделения памяти в максимально близких NUMA-банках памяти.



(а) (б)

Рис. 2. Сравнение масштабируемости различных гибридных моделей в пределах одного узла: (а) разработанные алгоритмы, реализованные в рамках модели MPI+Posix ShM, (б) библиотека hypre, гибридная модель MPI+OpenMP.

наблюдаются при использовании 2 MPI-процессов с 11 OpenMP тредами на каждый (график “2xOpenMP”, ускорение в 4.5 раза), однако и в этом случае они оказываются более чем вдвое хуже результатов предложенной схемы распараллеливания, реализованной в рамках модели MPI+Posix ShM. Наилучшие результаты, по-видимому, могут быть получены для 4 MPI-процессов с 5 OpenMP тредами и с привязкой процессов по NUMA-узлам процессоров, но такая ситуация нивелирует сам эффект от разработки гибридной модели.

4. Применение графических ускорителей для задач линейной алгебры

Вторая часть настоящей работы посвящена проработке основных вопросов использования ускорителей для рассматриваемого класса задач линейной алгебры. Говоря о переносе базовых алгоритмов рассматриваемой задачи, сосредоточимся прежде всего на наиболее трудоёмком алгоритме умножения матрицы на вектор, поскольку на данном этапе исследования эффективная реализация этого алгоритма предоставит прямую возможность для переноса всего программного кода решателя СЛАУ для гибридных вычислительных систем на графические ускорители.

Алгоритм умножения разреженной матрицы на вектор в целом не обладает большими возможностями по поиску вариантов реализации. Соотношение минимального количества чтений из основной памяти, записи в основную память и вычислительных операций выглядит как: $\mathrm{NNZ}+\mathrm{N} : \mathrm{N} : 2\,\mathrm{NNZ}$, где NNZ – количество ненулевых элементов матрицы, N – размер вектора-множителя. Такая оценка предполагает, что все данные, принадлежащие вектору, на который выполняется умножение, могут быть размещены в сверхбыстродействующей памяти (shared memory). Если такое размещение невозможно, или возможно частично, то оценка количества чтений из основной памяти увеличивается, и в предельном случае, когда данные вектора повторно не используются совсем, возрастает до $2\,\mathrm{NNZ}$.

Однако, на практике такое минимально необходимое количество операций чтения из основной памяти достижимо только для разреженных матриц специального вида. Примером такого формата может служить формат DIA [18], предназначенный для хранения диагональных матриц. Для представления разреженных матриц общего вида используют-

ся форматы, хранящие дополнительные массивы, которые позволяют определить значения номера строки и столбца для каждого из элементов матрицы. Вследствие этого, машинное время выполнения операции перемножения разреженной матрицы на вектор может значительно отличаться от теоретического минимума. Простым примером того, насколько существенным может оказаться это отличие, является формат хранения матриц COO [18]. Данный формат предполагает хранение для каждого ненулевого элемента матрицы двух дополнительных индексов: номера строки и столбца элемента. В случае если значения элементов матрицы представлены числами с плавающей точкой одинарной точности, количество требуемых операций чтения возрастает до величины в диапазоне от 3 NNZ+N до 4 NNZ (исходя из тех же соображений о возможном размещении вектора-множителя в сверхбыстродействующей памяти).

Все прочие форматы хранения разреженных матриц также вносят свой дополнительный вклад в объём требуемых операций чтения из основной памяти. Дополнительную трудность в выборе формата хранения и реализации алгоритма умножения матрицы на вектор представляет фактор, специфичный для архитектуры подсистемы доступа к основной памяти графических ускорителей. Для эффективной работы подсистемы памяти графического ускорителя необходимо выполнение требования, предъявляемого к инструкциям чтения из основной памяти. Это требование, называемое coalescing [20], предполагает, что в рамках одной параллельно выполняемой соседними нитями операции чтения из памяти запросы на чтение должны относиться к последовательно расположенным в памяти адресам. Кроме того, необходимо выполнение ряда требований по выравниванию таких адресов. Невыполнение этого требования приводит к снижению эффективности чтения из основной памяти. Ниже кратко описано несколько форматов хранения разреженных матриц, для каждого из которых обсуждаются особенности соблюдения этого условия при построении алгоритма умножения матрицы на вектор.

4.1. CSR формат

Формат хранения CSR [18], помимо основного массива значений, предполагает наличие двух дополнительных массивов. Один из них равен размеру массива значений и содержит соответствующие номера столбцов для ненулевых элементов матрицы. Второй массив хранит индексы начала строк, и его размер меньше или равен размеру вектора-множителя. Общее количество необходимых операций чтения для рассматриваемого алгоритма минимально оценивается как 2 NNZ+2 N. Очевидный способ построения алгоритма умножения матрицы на вектор в массивно-параллельной парадигме ставит в соответствие каждой параллельно выполняющейся нити один элемент массива индексов строк. Таким образом, параллельно выполняется цикл умножения каждой строки на вектор-множитель. Этот подход имеет две принципиальных проблемы при реализации на графических ускорителях. Во-первых, здесь нарушается правило coalesced чтений входных данных, поскольку, в общем случае, нити с соседними номерами не будут читать массивы номеров столбцов и входных значений по последовательным адресам: между адресами, по которым расположены первые элементы каждой строки, есть смещение, равное длине соответствующей строки. Во-вторых, нити могут обладать значительно отличающейся от соседних нитей последовательностью выполнения команд (divergent branching), что также является недостатком реализации, влияющим на общую производительность алгоритма на графическом ускорителе [20].

Алгоритм, который позволяет более эффективно выполнить операцию умножения матрицы, представленной в CSR формате, на вектор [21], предполагает сопоставление одному элементу матрицы индексов строк не одной нити, а группы из 32-х нитей, которые составляют единицу планирования для SIMD-процессоров графических ускорителей. Таким образом, нити входящие в одну 32-нитевую группу будут читать данные, относящиеся к одной и той же строке, а значит, чтение в рамках этого блока будет выполняться по последова-

тельным адресам. Однако, алгоритм работает идеально только в тех случаях, когда размер каждой строки матрицы кратен 32. В остальных же случаях часть пропускной способности шины памяти всё же будет расходоваться впустую, поскольку в блоке будут появляться простаивающие нити из-за того, что им не соответствует ни один ненулевой элемент данной строки. Такой алгоритм широко известен и реализован, в том числе, в рамках открытой библиотеки программ для графических ускорителей CUSP [5].

Существует также описание алгоритма умножения матрицы в формате CSR на строку с "балансировкой" рабочей нагрузки по всем параллельным нитям, предполагающий предварительное применение алгоритма Segmented Scan. Описание алгоритма можно найти на сайте автора [22], однако используется ли данный алгоритм в каких-либо из библиотечных кодах неизвестно. Внедрение такого алгоритма может быть одной из перспективных тем для дальнейшего исследования.

4.2. ELLPACK формат

Формат хранения разреженных матриц ELLPACK исключает массив, хранящий или указывающий на номера строк. Вместо этого, для каждой строки хранится S элементов массива, где S – максимальная длина строки в данной матрице. Каждый элемент содержит либо значение соответствующего по порядковому номеру ненулевого элемента строки матрицы, либо маркер отсутствия элемента, если строка содержит меньше, чем S ненулевых элементов. Дополнительно хранится аналогичный по структуре массив, содержащий номера столбцов для соответствующего ненулевого элемента матрицы.

Общее количество элементов, которые необходимо прочитать из основной памяти, чтобы выполнить операцию умножения матрицы, представленной в формате ELLPACK, на вектор, минимально оценивается как (2 S+1)*N. Если все строки матрицы содержат некоторое фиксированное число ненулевых элементов, то произведение S*N окажется равным NNZ, и требуемое количество операций чтения будет равно 2 NNZ+N, что несколько лучше, чем аналогичный минимальный показатель для CSR-матриц. Однако, если условие равенства количества ненулевых элементов во всех строках не выполняется, то матрица, записанная в ELLPACK-формате будет содержать некоторое количество элементов-маркеров, которые заметно увеличат необходимый объём считываемой информации. Таким образом, эффективность применения формата ELLPACK для описываемого алгоритма существенным образом зависит от такого свойства матрицы, как равномерность распределения числа ненулевых элементов по её строкам.

Оценивая возможность построения алгоритма умножения матрицы на вектор, использующего графические ускорители, можно отметить, что ни проблема, касающаяся нарушения последовательности доступа к памяти (coalescing), ни проблема различных путей выполнения алгоритма для соседних нитей (divergent branching) для данного формата не возникают, или, по крайней мере, в отношении divergent branching, не стоят так остро. Возможна такая реализация массивно-параллельного варианта алгоритма, при котором каждой нити соответствует свой элемент двух матриц, составляющих формат ELLPACK. Адресация считываемых данных соседними нитями при этом будет полностью последовательной.

4.3. HYB формат

Компромиссный формат HYB предполагает устранение принципиальной проблемы формата ELLPACK: для строк матрицы, количество ненулевых элементов в которых превышает некоторое пороговое значение S, используется формат хранения COO. Таким образом, задача умножения матрицы на вектор преобразуется в две такие задачи, выполняемые для двух подматриц. Такой формат расширяет границы возможности эффективного применения формата ELLPACK, реализацию которого на графических ускорителях можно назвать

достаточно выигрышной. Однако, такой формат требует проведения предварительного анализа исходной матрицы с целью определения оптимальной величины S, и реализация которого может потребовать дополнительных усилий, а выполнение – дополнительного машинного времени.

Подводя итог краткому рассмотрению форматов, следует отметить, что здесь описаны не все возможные форматы хранения разреженных матриц. Однако, выбор именно этих форматов для первоочередного исследования продиктован простым практическим соображением – все эти форматы (исключая, в некоторой степени, HYB), требуют минимальных алгоритмических усилий по разбиению матрицы на блоки по столбцам и строкам, и операций сжатия, упомянутых в разделе 2 настоящей статьи. Эти операции являются необходимой алгоритмической составляющей программы решения СЛАУ на параллельных системах с большим количеством узлов, поэтому необходимость их эффективной реализации для выбранного формата следует иметь в виду с самого начала работы над алгоритмом. Форматы CSR и ELLPACK обладают достаточно хорошим потенциалом по созданию эффективных массивно-параллельных алгоритмов для их преобразований.

Как было показано ранее, оценка эффективности реализации алгоритма умножения матрицы на вектор на графических ускорителях, существенным образом зависит от свойств и распределения ненулевых элементов в матрице. Исходя из этого, для экспериментального анализа эффективности операции произведения матрицы на вектор для различных форматов хранения матриц был использован набор тестовых матриц, полученных в ходе построения иерархии матриц алгебраического многосеточного предобуславливателя. Исходный размер матрицы составлял 8 млн. ячеек, основные параметры набора тестовых матриц приведены в табл. 1. В табл. 2 представлено сравнение машинного времени выполнения процедур умножения матрицы на вектор для тестового набора матриц. Для сравнения использовалась реализация алгоритмов из библиотеки CUSP 0.3.1; тестовые расчеты проводились на вычислительной системе “Ломоносов” (IB QDR, NVIDIA X2070, CUDA 4.0).

Анализ результатов показывает, что производительность алгоритма существенным образом зависит от вида матрицы и для матриц разного уровня вложенности нет единого, близкого к оптимальному, формата хранения данных. Это говорит о необходимости поиска альтернативных форматов хранения или альтернативных реализаций алгоритмов умножения матрицы на вектор для имеющихся форматов. Еще одним вариантом может быть выработка каких-либо простых априорных критериев применимости того или иного формата хранения для данной матрицы. Проведя оценку матрицы перед началом расчета, можно выбрать один из форматов её хранения, а значит и соответствующую процедуру умножения.

Поскольку имеет большое значение то, насколько хорошо распараллелен алгоритм в конфигурации вычислительной системы с множеством узлов, приведём графики масштабируемости процедуры выполнения операции умножения матрицы на вектор на вычислительной системе “Ломоносов” (рис. 3). Для этих расчетов использовались тестовые матрицы, сформированные для решения уравнения Пуассона на равномерной декартовой сетке с 27-точечным шаблоном. Операция умножения выполнялась для формата хранения матриц ELLPACK, который наилучшим образом подходит для такого вида матриц. Приведенные графики демонстрируют достаточно хорошие результаты масштабируемости, обеспечивая 20-кратное ускорение для матрицы размером 8 млн. неизвестных на 32 вычислительных узлах. При этом, следует отметить, что на данном этапе для графических ускорителей не были реализованы все перечисленные в разделе 2 варианты оптимизации процесса обмена сообщений между вычислительными процессами, так что приведенные графики масштабируемости могут быть заметно улучшены.

Таблица 1. Параметры матриц из тестового набора.

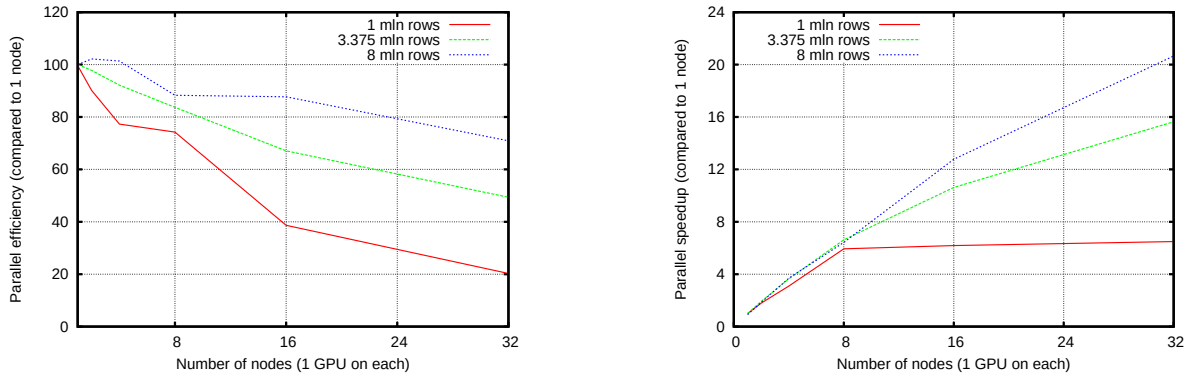
Матрица	**Nrows**	**NNZ**	**NNZ/row, avg**	**NNZ/row, max**
8M_0L	8000000	55760000	7.0	7
8M_1L	4007253	56627171	14.1	27
8M_2L	1752500	36167002	20.6	70
8M_3L	870201	26754707	30.7	178
8M_4L	464875	21050671	45.3	306
8M_5L	239559	15106487	63.1	465
8M_6L	118146	9961480	84.3	877
8M_7L	56793	5962117	105.0	904
8M_8L	26484	3350818	126.5	1058
8M_9L	13258	1965270	148.2	1007

Таблица 2. Сравнение времени выполнения операции SpMV на графическом ускорителе для тестового набора матриц в разных форматах хранения, мсек (GFLOP/s).

Матрица	**COO**	**CSR**	**ELLPACK**	**HYB**
8M_0L	15.06 (7.4)	7.46 (14.9)	4.02 (27.7)	4.02 (27.7)
8M_1L	19.04 (5.9)	8.34 (13.5)	9.24 (12.2)	7.94 (14.2)
8M_2L	16.32 (4.4)	13.32 (5.4)	9.66 (7.4)	7.45 (9.7)
8M_3L	12.26 (4.3)	9.38 (5.7)	11.60 (4.6)	6.69 (7.9)
8M_4L	8.81 (4.7)	7.10 (5.9)	11.66 (3.6)	6.66 (6.3)
8M_5L	5.78 (5.2)	4.95 (6.1)	9.66 (3.1)	5.34 (5.6)
8M_6L	3.71 (5.3)	3.07 (6.4)	7.81 (2.5)	3.81 (5.2)
8M_7L	2.17 (5.4)	1.65 (7.1)	4.51 (2.6)	2.31 (5.1)
8M_8L	1.16 (5.7)	0.83 (8.0)	2.98 (2.2)	1.35 (4.9)
8M_9L	0.68 (5.7)	0.39 (10.0)	1.41 (2.7)	0.78 (4.9)

5. Заключение

На примере метода BiCGStab с алгебраическим многосеточным предобуславливателем разработан двухуровневый алгоритм распараллеливания вычислений. Предложенный алго-



(а) (б)

Рис. 3. Масштабируемость алгоритма умножения матрицы на вектор, выполняемого на узлах с графическими ускорителями (1 ускоритель на узел) для задач разного размера: (а) параллельная эффективность в сравнении с одним узлом, (б) ускорение в сравнении с одним узлом.

ритм реализован в рамках гибридной модели MPI+ShM. Представлены результаты масштабируемости в пределах одного узла, демонстрирующие ускорение, соответствующее ускорению MPI-схемы распараллеливания. Полученные результаты в 7-8 раз превосходят результаты масштабируемости гибридной модели MPI+OpenMP, реализованной в библиотеке hypre.

При распараллеливании ключевого алгоритма умножения матрицы на вектор с использованием графических ускорителей показано, что выбор оптимального формата хранения данных зависит от вида матрицы и существенно влияет на эффективность алгоритма. На выбор формата представления матрицы также влияют соображения по возможности создания эффективных алгоритмов для проведения распределенных расчетов на графических ускорителях. Показана возможность получения приемлемой масштабируемости на многоузловых вычислительных системах для алгоритма умножения матрицы на вектор, выполняемого на графических ускорителях. Дальнейшая работа будет направлена на разработку распределенной реализации итерационных и многосеточных методов на гибридных вычислительных системах и сопряжении этих методов с пакетом OpenFOAM.

Представленные в работе результаты расчетов были получены на вычислительных системах “Ломоносов” суперкомпьютерного комплекса Московского университета и “Зилант” компании ЗАО “Т-Сервисы”.

Литература

1. Swartzreuber P.N. A direct method for the discrete solution of separable elliptic equations // SIAM Journal on Numerical Analysis. 1974. Vol. 11, N. 6. P. 1136–1150.

2. Saad Y. Iterative methods for sparse linear systems, 2-nd edition. SIAM, 2003. 528 p.

3. Trottenberg U., Oosterlee C.W., Schuller A. Multigrid. N.Y.: Academic Press, 2001. 631 p.

4. Krasnopolsky B. The reordered BiCGStab method for distributed memory computer systems // Procedia Computer Science. 2010. Vol. 1. P. 213–218.

5. Bell N., Garland M. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. URL: http://cusp-library.googlecode.com (дата обращения: 02.12.2012).

6. CUSPARSE library. URL: https://developer.nvidia.com/cusparse (дата обращения: 02.12.2012).

7. Strzodka R. Accelerated ANSYS Fluent: Algebraic Multigrid on a GPU. Supercomputing, 2012.
URL: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/RobertStrzodka_Accelerated_ANSYS_Fluent_SC12.pdf (дата обращения: 02.12.2012).

8. Esler K., Natoli V., Samardžić A. Accelerating Reservoir Simulation and Algebraic Multigrid with GPUs. GPU Technology Conference, 2012.
URL: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0140-GTC2012-Reservoir-Simulation-Algebraic.pdf (дата обращения: 02.12.2012).

9. LAMA – Library for Accelerated Math Applications.
URL: http://www.libama.org/overview.html (дата обращения: 02.12.2012).

10. Combest D.P., Day J. Cufflink: a library for linking numerical methods based on CUDA C/C++ with OpenFOAM. URL: http://cufflink-library.googlecode.com (дата обращения: 02.12.2012).

11. Kraus J., Förster M., Brandes T., Soddemann T. Using LAMA for efficient AMG on hybrid clusters // Computer Science - Research and Development. May 2012. P. 1–10.

12. Краснопольский Б.И. Об особенностях решения больших систем линейных алгебраических уравнений на многопроцессорных вычислительных системах различной архитектуры // Вычислительные методы и программирование. 2011. Т. 12, № 1. С. 176–182.

13. Краснопольский Б.И. Исследование эффективности переупорядоченного метода BiCGStab на вычислительных системах СКИФ МГУ «Чебышёв» и «Ломоносов» // Вестник ЮУрГУ. Серия “Математическое моделирование и программирование”. 2011. Т. 7, № 4(221). С. 56–65.

14. PGAS website. URL: http://www.pgas.org/ (дата обращения: 02.12.2012).

15. Bonachea D., Jeong J. GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages. CS258 Parallel Computer Architecture Project. 2002.

16. Корж. А.А. Результаты масштабирования бенчмарка NPB UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP // Вычислительные методы и программирование. 2010. Т. 11, № 1. С. 164–174.

17. Hypre: a library of high performance preconditioners.
URL: https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html (дата обращения: 02.12.2012).

18. Hugues M.R. Sparse Matrix Formats Evaluation and Optimization on a GPU // High Performance Computing and Communications (HPCC), 12th IEEE International Conference. 2010. P. 122–129.

19. CUDA C Programming Guide (PG-028290-001_v5.0). NVIDIA Corporation, 2012.

20. Sandres J., Kandrot E. CUDA by example. An introduction to General-Purpose GPU Programming. Addison-Wesley, 2010. 312 p.

21. Bell N., Garland M. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004. NVIDIA Corporation, 2008.

22. Baxter S. MGPU Sparse Library. URL: http://www.moderngpu.com/sparse/spmxv.html (дата обращения 02.12.2012).

Аксиоматический подход к формальной верификации рекурсивных программ на функционально-потоковом языке параллельного программирования

М.С. Кропачева

Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Сибирский федеральный университет»

Работа посвящена применению дедуктивного анализа для автоматизированного доказательства корректности функционально-потоковых параллельных программ на языке Пифагор. Строится исчисление Хоара для этого языка. Корректность рекурсивных функций доказывается с помощью метода индукции.

1. Введение

В настоящее время происходит повсеместный переход на параллельное программирование, что обусловлено широким применением многоядерных процессоров, кластерных систем и графических ускорителей. Использование традиционных подходов, базирующихся на императивных моделях, для разработки параллельного программного обеспечения достаточно часто ведёт к появлению ошибок. В связи с тем, что в большинстве случаев отказы систем связаны с нештатным функционированием программного обеспечения, актуальны вопросы его надёжности. Для повышения надёжности программного обеспечения всё чаще используется формальная верификация.

Под формальной верификацией понимается доказательство корректности программы, которое заключается в установлении соответствия между программой и её спецификацией, описывающей цель разработки [1]. При этом, соответствие программы её спецификации устанавливается посредством строгого математического доказательства. Главным преимуществом формальной верификации является возможность доказать отсутствие ошибок в программе, тогда как тестирование позволяет лишь выявлять отдельные ошибки.

Один из методов формальной верификации был предложен Хоаром [2]. В его основе лежит аксиоматический подход на основе исчисления Хоара — расширения какой-либо формальной теории $\mathfrak{I}$ введением в неё формул специального вида, называемых тройками Хоара. Тройка Хоара — это аннотированная программа, то есть программа, к которой приписаны две формулы теории $\mathfrak{I}$, описывающие ограничения на входные переменные и требования к результату выполнения программы. Эти формулы называются предусловие и постусловие соответственно. Тройки Хоара обычно записывается в виде $\{\varphi\}$ `Prog` $\{\psi\}$, где `Prog` — программа, а φ и ψ — предусловие и постусловия для `Prog`. Также для расширения теории $\mathfrak{I}$ вводится набор аксиом для операторов языка и правила вывода, с помощью которых из аксиом можно выводить утверждения о свойствах программ, в том числе о свойствах корректности. При этом, расширение теории $\mathfrak{I}$ строится таким образом, чтобы корректность программы соответствовала истинности тройки Хоара для этой программы. Основная идея данного подхода заключается в том, чтобы на базе аксиом, с помощью последовательных применений правил вывода, преобразовать тройку Хоара в формулу теории $\mathfrak{I}$, и доказать истинность формулы в этой теории.

Основным преимуществом аксиоматический подхода на основе исчисления Хоара, в отличие от менее формализованных методов дедуктивного анализа, является возможность его частичной автоматизации. Кроме того, данный подход является универсальным методом и применим для произвольных языков программирования, однако аксиомы и правила

вывода не универсальны, так как строятся на основе семантики языка программирования. Следовательно, различна и сложность доказательства корректности программ в разных аксиоматических системах.

В настоящее время достигнуты определённые успехи в практическом применении данного подхода для императивных языков программирования [3], однако, в целом, проблема не решена. Стоит отметить, что для параллельных императивных программ сложность формальной верификации сильно возрастает. Главной проблемой являются конфликты из-за ресурсов. Например, неправильное совместное использовании общей памяти или взаимные блокировки процессов в случае работы с распределённой памятью. Альтернативным вариантом императивного подхода является функционально-потоковая параллельная (ФПП) парадигма и предложенный для её поддержки язык программирования Пифагор [4]. Этот язык позволяет избежать конфликтов из-за ресурсов. Каждая программа — это функция, поэтому в языке отсутствуют переменные и циклы, а операции выполняются по готовности данных. В результате, сложность формальной верификации ФПП программ сравнима со сложностью верификации последовательных программ. Другая важная особенность языка — возможность достичь максимального параллелизма программы за счёт того, что параллелизм реализуется на уровне операций. Поэтому после доказательства корректности такая программа может быть перенесена на конкретную архитектуру с конечными ресурсами, при необходимости, с ограничением её параллелизма.

На данный момент, вопросы формальной верификации для языка программирования Пифагор проработаны недостаточно, поэтому актуальным является развитие методов анализа и формальной верификации ФПП программ.

2. Описание аксиоматической теории

Для выполнения формальной верификации на основе исчисления Хоара, для ФПП языка Пифагор разработана аксиоматическая теория, в рамках которой проводятся формальные доказательства корректности программ.

Основными объектами аксиоматической теории являются тройки Хоара.

Для описания предусловий и постусловий используется язык логической спецификации на базе логики предикатов первого порядка, которая достаточна для описания большинства требований к программе. Для описания выражений языка логической спецификации используется алфавит исчисления предикатов с функциональными и предикатными символами, которые соответствуют функциям языка программирования. Переменные языка логической спецификации могут быть различных типов:

$$T \quad = \quad \{signal, int, float, char, bool, func, error, datalist, delaylist, parlist, asynclist, \text{user\_type}\},$$

где user_type - соответствует множеству пользовательских типов. Фигурные скобки в выражениях языка логической спецификации используются для обозначения множества.

В исчислении предикатов различают функции, которые формируют термы, и предикаты, которые формируют формулы. В данном случае можно не выделять предикаты в отдельную группу, а считать их подмножеством функций с областью значений *bool*. Введем следующие функциональные и предикатные символы, и кванторы:

1. Арифметические операции $(+, -, *, /)$.
2. Знаки сравнения $(=, \neq, >, <, \geqslant, \leqslant)$.
3. Логические операции и кванторы $(\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow, \forall, \exists)$.
4. Функция длины списка (len), функция выбора элемента из списка $(select)$, функция принадлежности к типу $(\in)$.

Функции len, $select$, $\in$ эквивалентны соответствующим функциям языка Пифагор. Схемы для перечисленных функций будут совпадать со схемами функций из исчисления предикатов и арифметики.

Определим понятие элементарного терма с помощью индукции:

1. Каждая предметная переменная является элементарным термом.
2. Если $t_1, t_2, \ldots, t_n$ — элементарные термы, f — n-местная функция, то выражение $f(t_1, t_2, \ldots, t_n)$ является элементарным термом, при этом все константы считаются нуль-местными функциями.
3. Других элементарных термов нет.

Элементарная формула языка — это элементарный терм типа *bool*. Тогда произвольную формулу (выражение) можно определить по индукции:

1. Каждая элементарная формула является формулой.
2. Если A — формула, то $\forall x A(x)$ и $\exists x A(x)$ тоже является формулой.
3. Других формул нет.

В традиционном виде тройка Хоара записывается в виде $\{\varphi\}$ `Prog` $\{\psi\}$, где `Prog` — программа, а φ и ψ — предусловия и постусловия. Для того, чтобы фигурные скобки тройки не совпадали с фигурными скобками языка программирования или языка логической спецификации, для тройки Хоара вводится следующее обозначение:

$$\boxed{\varphi(x)}\ \texttt{Prog(x)}\rightarrow r\ \boxed{\psi(r)}\ ,$$

где `Prog(x)` — функция с входным аргументом x, r — результат работы программы, $\varphi(x)$ — предусловие для `Prog`, зависящее от аргумента x, $\psi(r)$ — постусловия для `Prog`, зависящее от результата выполнения функции.

Например, рассмотрим следующую функцию на языке Пифагор:

```
Fun << funcdef arg {
   arg:F >>  return
}
```

Пусть P и Q — предусловия и постусловия для этой функции. Тогда тройка Хоара будет иметь вид:

$$\boxed{P(arg)}\ \texttt{arg:F}\rightarrow r\ \boxed{Q(r)}\ .$$

В связи с тем, что во многих ситуациях программу на языке Пифагор удобно отображать в виде информационного графа, возможна непосредственная привязка предусловия и постусловия к дугам этого графа. Пример подобной привязки представлен на рисунке 2.

Аксиомами рассматриваемой аксиоматической теории языка программирования Пифагор служат тройки Хоара для встроенных функций. Возможны различные алгоритмы вычисления встроенной функции в зависимости от некоторых условий на аргументы [5]. Каждому такому алгоритму соответствует отдельная аксиома. Использование нескольких аксиом для одной встроенной функции, по сравнению с использованием одной сложной аксиомы, упрощает вид итоговых формул, являющихся условиями корректности программы.

Ниже приведены аксиомы для функции создания списка из одинаковых элементов, которая в языке Пифагор задаётся следующим образом: `p:dup`. Предполагается, что `p` — это список, состоящий из двух элементов, где первый элемент — это произвольный аргумент, а второй — целое число, определяющее количество повторений первого элемента. При использовании данной функции могут возникать различные ошибки. Тогда аксиомы для функции дублирования будут следующими:

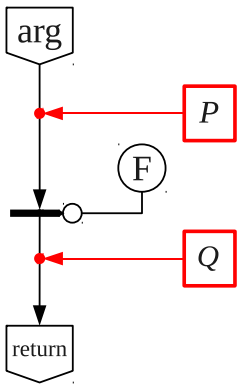


Рис. 1. Тройка Хоара для программы `arg:F`

$$\boxed{\begin{array}{l}(p \in datalist)\wedge\\(len(p) = 2)\wedge\\(select(p,2) \in int)\end{array}}\ \texttt{p:dup}\ \to r\ \boxed{\begin{array}{l}(r \in datalist)\wedge\\(len(r) = select(p,2))\wedge\\\forall k\,(k \in \{1,2,\ldots,select(p,2)\}\wedge\\\qquad select(r,k) = select(p,1))\end{array}},$$

$$\boxed{\begin{array}{l}\neg(p \in datalist)\vee\\\neg(len(p) = 2)\vee\\\neg(select(p,2) \in int)\end{array}}\ \texttt{p:dup}\ \to r\ \boxed{r = \texttt{BASEFUNCERROR}}.$$

Правила вывода описывают схему преобразования композиций операторов и позволяют выводить теоремы из имеющихся аксиом и уже доказанных теорем. Для того, чтобы процесс преобразования троек Хоара в формулу исчисления предикатов являлся однозначным, используются правила прямого прослеживания, ориентированные на вывод от аргумента функции к её результату. В случае ФПП языка Пифагор таких функций может быть несколько, тогда из них выбирается произвольная. При этом, произвольный порядок применения правила прямого прослеживания для параллельно выполняющихся функций не окажет влияние на результирующую формулу с точностью до порядка операндов конъюнкции.

В общем случае, «правило прямого прослеживания» для некоторой функции с кодом «$\texttt{x:F}_1\texttt{:F}$» имеет следующий вид:

$$\boxed{P_1(x_1)}\ \texttt{x}_1\texttt{:F} \to r\ \boxed{Q(r)},\ A_1,\ A_2 \quad \vdash \quad \boxed{P(x)}\ \texttt{x:F}_1\texttt{:F} \to r\ \boxed{Q(r)}, \tag{1}$$

$$A_1 := \boxed{\varphi(x)}\ \texttt{x:F}_1 \to x_1\ \boxed{\psi(x_1)},\ A_2 := P(x) \Rightarrow \varphi(x),\ P_1(x_1) := P(x) \wedge \varphi(x) \wedge \psi(x_1),$$

где `x` — входной аргумент, $\texttt{F}_1$ — функция, применяемая непосредственно к входному аргументу, `F` — остальная часть программы, которая может содержать другие функции, применяемые непосредственно к аргументу `x`, $\vdash$ — символ выводимости, который свидетельствует о том, что при истинности выражений в левой части, истинны и выражения в правой части. Согласно этому правилу, на основе аксиомы A_1 для функции $\texttt{F}_1$ и при условии A_2, тройка Хоара $\boxed{P(x)}\ \texttt{x:F}_1\texttt{:F} \to r\ \boxed{Q(r)}$ преобразуется в тройку с более «короткой» программой $\boxed{P_1(x_1)}\ \texttt{x}_1\texttt{:F} \to r\ \boxed{Q(r)}$, при этом предусловие $P(x)$ изменяется на $P_1(x_1)$, а исходный аргумент x заменяется на x_1 — результат применения функции $\texttt{F}_1$ к x.

При последовательном применении правила прямого прослеживания происходит «сокращение» или «свёртка» программы, и по завершению анализа всех операторов получаем

тройку Хоара с «пустой» программой: $\boxed{\text{P}}\ \boxed{\text{Q}}$. Это означает, что предусловие и постусловие приписаны к одной дуге информационного графа. Для завершения преобразования тройки Хоара к формуле исчисления предикатов вводится следующее правило:

$$P \Rightarrow Q \vdash \boxed{\text{P}}\ \boxed{\text{Q}}. \tag{2}$$

Таким образом, с помощью преобразований произвольных троек Хоара на основе правил вывода можно получить формулу исчисления предикатов, истинность которой доказывается в рамках исчисления предикатов. Если эта формула истинна, то истинна и исходная тройка Хоара, откуда следует корректность программы.

В зависимости от входных данных программа может иметь различные пути выполнения. Аксиомы встроенных функций полностью определяют дерево $\mathfrak{T}_0$ всех путей выполнения программы. Количество различных вариантов выполнения программы для каждой функции равно количеству аксиом этой функции. Если на входные данные наложены ограничения (предусловие программы), то часть ветвей в дереве становятся недостижимыми. Эти ветви соответствуют тем аксиомам, предусловие которых не следует из предусловия программы. Откидывая недостижимые пути, получаем новое дерево $\mathfrak{T}_1$, которое является поддеревом $\mathfrak{T}_0$. Каждый путь дерева $\mathfrak{T}_1$ можно рассмотреть отдельно и преобразовать в формулу исчисление предикатов по правилам вывода. В результате таких преобразований получается k формул, где k соответствует числу листьев в дереве $\mathfrak{T}_1$. Если каждая из полученных формул истинна, то программа корректна. В противном случае программа некорректна, а ошибки присутствуют в тех ветвях, которым соответствуют не тождественно истинные формулы. Таким образом, доказательство корректности программы сводится к доказательству истинности нескольких формул.

3. Анализ корректности рекурсий

Основная проблема верификации программ связана с циклическими конструкциями, которые могут, в случае неправильной программы, привести к зацикливанию, при этом информационный граф программы становится бесконечным. В языке Пифагор отсутствуют операторы цикла, и все повторяющиеся конструкции реализуются через рекурсии. Для того, чтобы программа была корректной, рекурсия должна, во-первых, завершаться, а, во-вторых, возвращать правильный результат.

Проанализируем особенности рекурсивной функции. Если в программе присутствует рекурсия, то один и тот же код вызывается несколько раз, а различия будут касаться только аргументов. Тогда необходимым условием завершения рекурсии является то, что аргумент пробегает неповторяющуюся последовательность значений.

Большинство рекурсивных функций достаточно просто представить в следующем виде: в корректной рекурсивной функции обязательно должна присутствовать «точка ветвления», в которой происходит выбор между возможными путями выполнения программы. Одна часть путей приводит к завершению работы и возвращению результата вычислений, другая — к рекурсивному вызову функции. По какому пути пойдут дальнейшие вычисления, определяется значением некоторой функции от входных аргументов. Эту функцию можно рассматривать как некий аналог оператора `if-else` в том смысле, что значения выражения, определяющего переход к следующей итерации, и выражения завершения итерации рекурсии, являются взаимно исключающими, то есть $\neg(\textit{условие перехода} = \textit{true}) \Leftrightarrow (\textit{условие завершения} = \textit{true})$.

Для доказательства корректности рекурсивной программы необходимо выделить «точку ветвления», ветвь, соответствующую рекурсивному пути и ветвь, приводящую к завершению программы, а также условия, которые определяют какой путь будет выбран в «точке ветвления». Назовём эти условия «условием рекурсивного вызова» и «условием завершения функции». Также введём два термина: «текущий аргумент» — входной аргумент функции

и «аргумент рекурсивного вызова» — аргумент для рекурсивного вызова функции.

Правильность рекурсии можно доказать по индукции (доказательство завершения рекурсии проводится аналогично). Введём понятие ограничивающей функции. *Ограничивающая функция* — это ограниченная снизу функция, переводящая аргумент рекурсивной функции во множество натуральных чисел $\mathbb{N}$, при этом аргументы, для которых выполняется условие завершения, отображаются в единицу.

Рассмотрим схему доказательства корректности рекурсивной программы Rec, которое проводится индукцией по значениям ограничивающей функции f.

База индукции: проверяем корректность программы для аргумента $x = p_0$, такого что $f(p_0) = 1$.

Шаг индукции: предположим, что программа корректна для всех аргументов, для которых значение ограничивающей функции меньше N. Числу N соответствует параметр p_N, такой что $f(p_N) = N$. Тогда, для того, чтобы рекурсивная функция была корректна, достаточно, чтобы одновременно выполнялись следующие условия:

1. При выполнении функции $Rec(p_N)$ рекурсивно могут быть вызваны только $Rec(p_i), i = \overline{1,n}$, для которых значения ограничивающей функции $f(p_i)$ меньше N.

2. Параметры $p_i, i = \overline{1,n}$ являются допустимыми аргументами функции Rec. Данное условие не зависит от ограничивающей функции, поэтому его можно использовать, как первоначальную проверку корректности функции.

3. Функция $Rec(p_N)$ вернёт верный результат, если все рекурсивные вызовы заменить на результат выполнения $Rec(p_i), i = \overline{1,n}$, то есть верно, что из корректности функции Rec для аргументов p_i с меньшим значением ограничивающей функции следует корректности функции Rec для текущего значения аргумента p_N.

Описанный алгоритм является достаточным условием корректности рекурсии. В случае, если не удаётся доказать корректность, то это может свидетельствовать о том, что программа некорректна, или неправильно выбрана ограничивающая функция. В последнем случае придется искать другую ограничивающую функцию.

4. Пример доказательства корректности программы с рекурсией

Докажем корректность рекурсивной функции `fact`, вычисляющей факториал x. Код программы на языке Пифагор приведён на рисунке 2.

```
fact << funcdef x {
    fl << ((x,1):[<=,>]):?;
    act << (1,
          { (x, (x,1):-:fact ):*  } );
    return << act:fl:.;
}
```

Рис. 2. Код функции `fact`, вычисляющей факториал числа x

Пользователь задаёт для программы следующую тройку Хоара (дописывает предусловие и постусловие):

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge \\ \prod\limits_{i=1}^{x} i \leqslant \texttt{INT\_MAX}\end{array}} \ \texttt{x:fact} \rightarrow r \ \boxed{\begin{array}{l}((r = \prod\limits_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}, \tag{3}$$

где `INT_MAX` — максимальное целое, которое допускает тип int. Данная тройка Хоара содержит в себе следующую спецификацию программы: если входное значение x — целое число

больше нуля и произведение чисел от 1 до x не превышает максимального целого, которое допускает тип int, то после выполнения функция `fact` возвращает результат перемножения чисел от 1 до x, если $x > 0$ или 1, если $x = 0$.

В виде информационного графа данная тройка Хоара приведена на рисунке 3.В серых кружках указаны идентификаторы дуг, все идентификаторы должны быть различны.

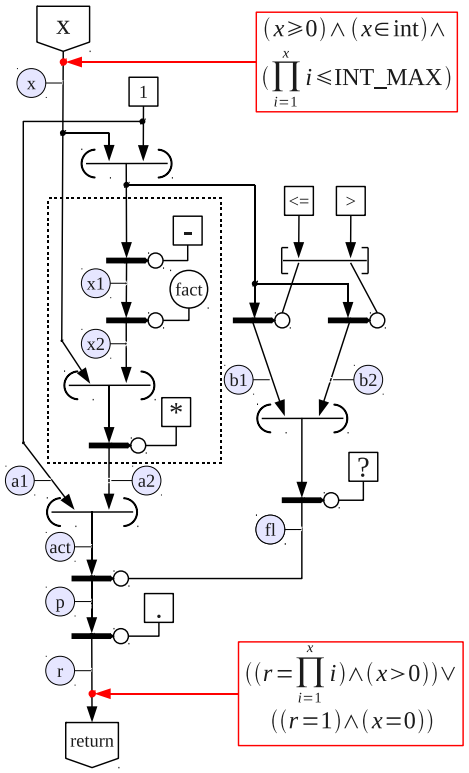


Рис. 3. Информационный граф тройки Хоара для функции `fact`

Рассмотрим процесс вычисления функции `fact`. При $x = 0$ или 1 должно быть возвращено значение 1, при $x > 1$ должен быть совершен рекурсивный вызов `fact` с аргументом $x - 1$. Проверка истинности одного из условий $x \leqslant 1$ или $x > 1$ выполняется в правой ветви, содержащей функцию «?». Параллельно, в левой ветви формируется список p из двух элементов: «1» и «задержанный список» (на рисунке он выделен пунктиром), содержащий подграф с рекурсивным вызовом функции `fact`. Операторы задержанного списка выполняются только после снятия задержки, даже если все аргументы готовы. Выбор одного из элементов в списке p осуществляется в зависимости от того, какое из условий $x \leqslant 1$ или $x > 1$ верно. Во втором случае выбирается задержанный список, с него снимается задержка функцией «.», и происходит рекурсивный вызов. В случае $x \leqslant 1$ из списка p выбирается константа «1», а вычисление функций задержанного списка вообще не происходит.

Для доказательства корректности программы необходимо доказать истинность тройки (3). Преобразуем тройку в формулу, последовательным применением «правила прямого прослеживания» (1) на основе аксиом для встроенных функций. Выполнение функции `fact` начинается с того, что входной аргумент `x` и константа «`1`» формируют список `(x,1)`. К этому списку применяются функции «`<=`» и «`>`». Это встроенные функции, их аксиомы одинаковые и отличаются лишь знаком операции. Для функции «`<=`» аксиомы имеют вид:

$$\boxed{(p_1, p_2 \in \{int, float\}) \vee (p_1, p_2 \in char) \vee (p_1, p_2 \in bool)} \ \texttt{(p1,p2):<=} \ \rightarrow r_1 \ \boxed{(r_1 \in bool) \wedge (r_1 = (p_1 \leqslant p_2))}, \qquad (4)$$

$$\boxed{\begin{array}{l}\neg((p_1,p_2\in\{int,float\})\vee\\(p_1,p_2\in char)\vee(p_1,p_2\in bool))\end{array}}\ \texttt{(p1,p2):<=}\ \to r_1\ \boxed{\begin{array}{l}(r_1\in error)\wedge\\(r_1=\texttt{BASEFUNCERROR})\end{array}}. \qquad (5)$$

Перед применением правила прямого прослеживания необходимо выбрать те аксиомы, у которых предусловие может следовать из предусловия тройки Хоара (3). Все остальные аксиомы отбрасываются. Формируем две формулы, описывающие «условие невыводимости» предусловия аксиом (4) и (5) из предусловия тройки (3), и проверяем их истинность; аксиомы, которым соответствуют истинные формулы — отбрасываем. После согласования обозначений аргументов аксиом (4) и (5) с рассматриваемой тройкой (3) и введения идентификатора b_1 для результата выполнения функции «<=», получаем что $p_1 := x$, а $p_2 := 1$, тогда условия невыводимости опишутся следующими формулами:

$$\neg\left(\left((x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\right)\to \right.$$
$$\left.\to\big((x,1\in\{int,float\})\vee(x,1\in char)\vee(x,1\in bool)\big)\right)$$

и

$$\neg\left(\left((x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\right)\to \right.$$
$$\left.\to\neg\big((x,1\in\{int,float\})\vee(x,1\in char)\vee(x,1\in bool)\big)\right).$$

Первая формула является тождественно ложной, вторая — тождественно истинной, поэтому вторая аксиома отбрасывается, так как выполнение программы не сможет пойти по этому пути.

В результате применения «правила прямого прослеживания» к тройке (3) на основе аксиомы (4), получаем следующую тройку Хоара:

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\(b_1\in bool)\wedge(b_1=(x\leqslant 1))\end{array}}\ \texttt{fact1}\ \to r\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}, \qquad (6)$$

где `fact1` соответствует код:

```
[( 1,
   { (x, (x,1):-:fact ):*  }
):(b1,(x,1):>):?]:.
```

Далее последовательно применяем «правило прямого прослеживания» на основе аксиом для следующих встроенных функций: «>», «?», функции выбора элемента из списка и «.». В результате преобразований получаются следующие тройки:

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\((x\leqslant 1)=true)\wedge((x>1)=false)\wedge(r=1)\end{array}}\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}, \qquad (7)$$

$$\boxed{\begin{array}{l}(x\in int)\wedge(x\geqslant 0)\wedge\\(\prod_{i=1}^{x} i\leqslant \texttt{INT\_MAX})\wedge\\((x\leqslant 1)=false)\wedge\\((x>1)=true)\end{array}}\ \texttt{(x, (x,1):-:fact):*}\ \to r\ \boxed{\begin{array}{l}((r=\prod_{i=1}^{x} i)\wedge(x>0))\vee\\((r=1)\wedge(x=0))\end{array}}. \qquad (8)$$

Тройка (7) является тройкой с «пустой» программой, поэтому её можно преобразовать в формулу по правилу (2):

$$\left((x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge ((x \leqslant 1) = true) \wedge ((x > 1) = false) \wedge (r = 1)\right) \rightarrow$$
$$\rightarrow \left(((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee ((r = 1) \wedge (x = 0))\right).$$

Очевидно, что данная формула истинна, так как из $(x \geqslant 0)$ и $((x \leqslant 1) = true)$, следует, что $x = 0$ или $x = 1$, а $r = 1$.

Теперь рассмотрим тройку (8). Функция «-» — ближайшая к входному аргументу, поэтому выполняется первой. При применении «правила прямого прослеживания» на основе аксиом функции «-» к тройке (8) получаем тройку:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ ((x \leqslant 1) = false) \wedge ((x > 1) = true) \wedge \\ (x_1 \in int) \wedge (x_1 = x - 1)\end{array}} \text{ (x, x1:fact):*} \rightarrow r \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}. \qquad (9)$$

Далее должен выполняться рекурсивный вызов функции `fact`. Поэтому использовать «правило прямого прослеживания» напрямую нельзя, так как корректность функция `fact` не доказана.

Отметим, что в рассматриваемой функции «текущий аргумент» — это входной аргумент «`x`», «аргумент рекурсивного вызова» — аргумент для рекурсивного вызова функции «`(x,1):-`» (обозначен x_1); ветвь a_1 информационного графа не содержит рекурсию, а ветвь a_2 — содержит, тогда «условие завершения функции» и «условие рекурсивного вызова» определяют выражения дуг $b_1 := (x \leqslant 1)$ и $b_2 := (x > 1)$ (обозначения приведены на рис. 3).

В начале, необходимо показать, что при рекурсивном вызове функции `fact` ей передаётся корректный аргумент. Это эквивалентно тому, что из предусловия тройки (9) выводимо предусловие функции `fact` (3), в котором «текущий аргумент» x заменяется на выражение, описывающее «аргумент рекурсивного вызова» x_1, тогда «условие невыводимости» примет вид:

$$\neg \left(\left((x \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge (x_1 \in int) \wedge (x_1 = x - 1)\right) \rightarrow \right.$$
$$\left. \rightarrow \left((x_1 \in int) \wedge (x_1 \geqslant 0) \wedge (\prod_{i=1}^{x_1} i \leqslant \texttt{INT\_MAX})\right)\right).$$

Исходя из того что $(x > 1)$, а $(x_1 = x - 1)$, то $(x_1 > 0)$. Если $(\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX})$, то $(\prod_{i=1}^{x_1} i = \prod_{i=1}^{(x-1)} i < \texttt{INT\_MAX})$. Значит формула тождественно ложна, и «аргумент рекурсивного вызова» всегда удовлетворяет предусловию функции `fact`. В противном случае программа сразу бы считалась некорректной.

Зададим ограничивающую функцию для `fact`:

$$f(x) = \begin{cases} 1, & x = 0; \\ x, & x > 0. \end{cases}$$

Докажем корректность функции `fact` индукцией по значению ограничивающей функции.

База индукции. Если аргумент x удовлетворяет предусловию (3) $(x \in int) \wedge (x \geqslant 0) \wedge \prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}$ и «условию завершения программы» $x \leqslant 1$, то $x = 0$ или $x = 1$. Для этих значений $f(x) = f(0) = f(1) = 1$. Очевидно, что для указанных значений аргументов функция `fact` завершается без рекурсивного вызова, а результат в выполнения функции равен единице ($r = 1$) и удовлетворяет постусловию (3). Формальная проверка этого утверждения реализуется с помощью доказательства корректности тройки Хоара (7), что было сделано ранее.

Шаг индукции. Возьмём аргумент x, удовлетворяющий (3), для которого выполнено «условие рекурсивного вызова» $x > 0$ и значение ограничивающей функции равно N: $f(x) = N$. Предположим, что функция корректна для всех значений аргумента, для которых значение ограничивающей функции меньше N. Покажем, что значение ограничивающей функции «аргумента рекурсивного вызова» всегда меньше N:

$$f(x_1) = f(x-1) = f(N-1) = N - 1 < N.$$

Тогда по индуктивному предположению будет верна тройка Хоара для функция `fact`, применённая к «аргументу рекурсивного вызова»:

$$\boxed{\begin{array}{l}(x_1 = x - 1) \wedge (x_1 \in int) \wedge \\ (x_1 \geqslant 0) \wedge (\prod_{i=1}^{x_1} i \leqslant \texttt{INT\_MAX}\ )\end{array}}\ \texttt{x1:fact}\ \rightarrow x_2\ \boxed{\begin{array}{l}((x_2 = \prod_{i=1}^{x_1} i) \wedge (x_1 > 0)) \vee \\ ((x_2 = 1) \wedge (x_1 = 0))\end{array}}. \tag{10}$$

Полученную тройку можно использовать в качестве теоремы и доказывать корректность программы `fact` с помощью «правила прямого прослеживания», считая функцию нерекурсивной.

Тогда при применении «правила прямого прослеживания» на основе теоремы (10) к тройке (9) получаем:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x \geqslant 0) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ ((x \leqslant 1) = false) \wedge ((x > 1) = true) \wedge \\ (x_1 \in int) \wedge (x_1 = x - 1)\end{array}}\ (\text{x, x2 }){:}* \rightarrow r\ \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}. \tag{11}$$

В коде тройки (11) осталась только функция умножения «`*`». Применяя «правило прямого прослеживания» на основе аксиом для этой функции получаем тройку с «пустой» программой:

$$\boxed{\begin{array}{l}(x \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge \\ (x_2 = \prod_{i=1}^{(x-1)} i) \wedge (x, x_2 \in int) \wedge (x * x_2 \in int) \wedge (r = x * x_2)\end{array}}\ \boxed{\begin{array}{l}((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee \\ ((r = 1) \wedge (x = 0))\end{array}}.$$

Далее применяем правило преобразования тройки в формулу (2), и после упрощения получаем:

$$\left((x, x_2 \in int) \wedge (x > 1) \wedge (\prod_{i=1}^{x} i \leqslant \texttt{INT\_MAX}) \wedge (x_2 = \prod_{i=1}^{(x-1)} i) \wedge (r = x * x_2)\right) \rightarrow$$
$$\rightarrow \left(((r = \prod_{i=1}^{x} i) \wedge (x > 0)) \vee ((r = 1) \wedge (x = 0))\right).$$

Полученная формула истинна, а значит истинна исходная тройка Хоара (3), из истинности которой следует корректность программы.

Таким образом, корректность функции `fact` доказана.

5. Заключение

В работе рассмотрены особенности формальной верификация ФПП программ, выстраивается аксиоматическая теория для языка Пифагор, а также показано использование этой теории для доказательства корректности рекурсивных программ. Благодаря тому, что язык программирования Пифагор не ограничивает параллелизм разрабатываемых на нём программ, он может быть использован в качестве инструмента для обобщённой спецификации параллельных программ, обеспечивая их более простую формальную верификацию, тестирование и отладку, а затем — формальное преобразование (автоматическое или ручное) программы в представление, предназначенное для выполнения на конкретной (целевой) архитектуре. В дальнейшем, разработанный подход может быть положен в основу среды инструментальной поддержки доказательства корректности, так как подвергается автоматизации на многих этапах.

Литература

1. Непомнящий В.А., Рякин О.М. Прикладные методы верификации программ. М.: Радио и связь, 1988. 255 с.

2. Hoare C. A. R. An axiomatic basis for computer programming // Communications of the ACM. 1969. Vol. 10, No. 12. P. 576–585.

3. Ануреев И. С., Марьясов И. В., Непомнящий В. А. Верификация С-программ на основе смешанной аксиоматической семантики // Моделирование и анализ информационных систем. 2010. Т. 17, № 3. С. 5–28.

4. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1 (10). С. 71–89.

5. Кропачева М.С. Формализация семантики функционально-потокового языка параллельного программирования Пифагор // Материалы XII Всероссийской научно-практической конференции «Проблемы информатизации региона» (ПИР-2011). Красноярск: Сибирский федеральный университет. 2011. С. 144–148.

Высокопроизводительные вычисления как облачный сервис: ключевые проблемы*

А.О. Кудрявцев, В.К. Кошелев, А.О. Избышев, А.И. Аветисян

Институт системного программирования Российской академии наук (ИСП РАН)

В настоящее время облачные технологии получают все большее распространение. При этом возможности применения концепции облачных сервисов в области высокопроизводительных вычислений существенно ограничены. Ключевая проблема — накладные расходы, возникающие при использовании технологий виртуализации. В данной работе рассматриваются ключевые источники накладных расходов при виртуализации высокопроизводительного кластера. Приводятся результаты исследований накладных расходов и методы повышения производительности параллельных приложений в виртуальных машинах. Подробно рассматривается проблема виртуализации высокопроизводительного ввода-вывода на примере сети Infiniband.

1. Введение

В настоящее время облачные технологии получают все большее распространение. Облачные системы широко применяются на индустриальном уровне, в науке и образовании. При этом возможности применения концепции облачных сервисов в области высокопроизводительных вычислений существенно ограничены. Ключевая проблема — накладные расходы, возникающие при использовании технологий виртуализации. Технологии виртуализации играют ключевую роль в реализации облачных систем, обеспечивая возможность предоставления вычислительных ресурсов по запросу.

Возможности технологий виртуализации платформы x86_64 в последние годы растут высокими темпами. Технологии аппаратной и контейнерной виртуализации уже позволяют обеспечить производительность отдельных классов приложений в виртуальных средах, практически не отличимую от производительности на реальном оборудовании. В результате исследователи по всему миру начали изучать возможности и ограничения виртуализации высокопроизводительных вычислений (HPC, High Performance Computing).

Достоинства применения виртуализации в области высокопроизводительных вычислений широко обсуждаются [1–3]. Среди них устойчивость к сбоям, совместимость и гибкость при работе с виртуальными машинами (ВМ). Особенно интересна идея применения концепции облачных вычислений для создания высокопроизводительных облачных систем, предоставляющих масштабируемость, эффективность использования ресурсов и удобство доступа. Последние исследования показывают, что виртуализация HPC имеет смысл по крайней мере для отдельных классов приложений [1, 4].

Основной целью проводимых работ является оценка накладных расходов при виртуализации высокопроизводительных вычислений, а также выявление причин и минимизация этих расходов. Исследования проводятся с использованием системы виртуализации KVM (Kernel-based Virtual Machine) [5].

Для достижения наилучшей производительности (относительно случая запуска на реальном оборудовании) виртуальным машинам выделяется максимально возможное количество ресурсов, включая все процессорные ядра. Кроме того, виртуальным машинам предоставляется реальное коммуникационное устройство, с использованием технологий виртуализации ввода-вывода Intel VT-d [6]. При анализе производительности были обнаружены накладные расходы, связанные с использованием технологии Intel VT-d. В данной работе подробно рассматриваются причины этих накладных расходов и методы повышения про-

*Работа выполнена при финансовой поддержке РФФИ, грант № 12-07-00726-а.

изводительности.

В качестве тестового пакета используется пакет SPEC MPI2007 [7]. Данный пакет достаточно широко распространен в области высокопроизводительных вычислений. Исследование производительности выполнялось на современном кластере, с использованием до 16 узлов, построенных на базе процессоров Intel Xeon (в сумме до 192 процессорных ядер). Узлы кластера связаны высокоскоростной сетью Infiniband 40 Гбит/сек. Стоит отметить, что многие исследования в области виртуализации HPC, представленные ранее, проводились на системах из одного узла, что не позволяло полноценно оценить накладные расходы, вызываемые виртуализацией.

В ходе выполнения данной работы достигнуты следующие результаты:

- Выявлены основные источники накладных расходов, возникающих при виртуализации.
- Проведена оценка накладных расходов виртуализации высокопроизводительной системы ввода-вывода для современного HPC кластера, построенного на базе высокоскоростной сети Infiniband. Узлы кластера — двухпроцессорные 12-ядерные серверы, используется 16 узлов.
- Проанализированы полученные экспериментальные данные и накладные расходы, вызванные применением технологий виртуализации ввода-вывода Intel VT-d. В рамках ядра ОС Linux и системы виртуализации KVM/QEMU разработан и реализован механизм, позволяющий предоставлять реальное коммуникационное устройство доверенной виртуальной машине без применения технологии Intel VT-d. Данный механизм позволил существенно снизить накладные расходы на отдельных тестах не повлияв на другие.
- В целом, на рассматриваемом стенде достигнуто снижение накладных расходов на виртуализацию до уровня 1-10% на тестах пакета SPEC MPI2007.

2. Ключевые источники накладных расходов при виртуализации

Можно выделить несколько ключевых источников накладных расходов, возникающих при виртуализации. Часть проблем связана с использованием технологий аппаратной виртуализации процессора, памяти, устройств. В настоящее время считается, что вопросы виртуализации процессора тщательно изучены, и различные подходы к реализации гипервизоров позволяют достичь производительности, близкой к производительности реального процессора при запуске приложения, требовательного к вычислительной мощности.

При виртуализации памяти стоит цель разделения и изоляции памяти виртуальных машин и основной системы (системы, в рамках которой выполняется гипервизор). При этом решается задача отображения гостевого виртуального адреса в реальный физический адрес. На платформе x86 применяется два подхода: теневая страничная трансляция (shadow paging), основанная на традиционном механизме пейджинга x86, и вложенная страничная трансляция (nested paging), основанная на аппаратных расширениях виртуализации памяти (более подробно см. [3]). Метод вложенной трансляции дает лучшие результаты для гостевых систем с частым переключением таблиц страниц, например, для ОС Linux. В нашей работе мы также использовали вложенную трансляцию, поскольку в качестве гостевой ОС используется Linux.

В рамках виртуализации HPC-вычислений, необходимо предоставлять виртуальной машине все доступные на узле ресурсы. С этой точки зрения актуальным становится соответствие архитектуры основной системы и гостевой системы. Многие современные серверы имеют архитектуру NUMA (Non-Uniform Memory Access, архитектура с неравномерным

доступом к памяти), при наличии двух и более процессоров. Проблемы виртуализации таких серверов рассматривались нами в работе [4]. В системе виртуализации KVM/QEMU не предусмотрена возможность полносистемной эмуляции архитектуры NUMA в соответствии с реальной топологией узла, такая возможность была реализована нами ранее и используется в данной работе. Корректная эмуляция архитектуры NUMA позволила снизить накладные расходы с 50-60% до 1-10% на многих тестах.

Отдельной проблемой, особенно актуальной при виртуализации HPC, является виртуализация высокопроизводительного ввода-вывода. Данная проблема подробно рассматривается в разделе 3.

При увеличении масштаба вычислений (числа задействованных узлов в кластере), начинают влиять незначительные факторы отклонения производительности, вызванные работой основной ОС, гипервизора и гостевой ОС. Такие факторы принято называть шумом. Среди таких факторов — работа программ-демонов, обработчиков прерываний, других программ в системе. Влияние шума на производительность приложений широко изучается [8]. Для ряда приложений даже минимальный уровень шума может привести к значительному ухудшению производительности и масштабируемости. Таким образом, при виртуализации высокопроизводительной системы, особенно с большим числом узлов, необходимо учитывать дополнительный шум гипервизора и основной ОС.

3. Проблема виртуализации ввода-вывода

Существенной проблемой при виртуализации HPC-задач является виртуализация высокопроизводительных систем ввода-вывода, в частности, коммуникационных сред. В настоящее время для эффективной работы устройства необходимо предоставлять контроль над этим устройством гостевой системе (при серверной виртуализации все устройства в виртуальной машине, как правило, реализованы программно в гипервизоре).

На платформе x86 для проброса устройств возможно применение устройства аппаратной виртуализации ввода-вывода (IOMMU, I/O Memory Management Unit). Основной задачей IOMMU является отображение адресного пространства пробрасываемого устройства в адресное пространство физической памяти гостевой системы. При отсутствии такого устройства, гостевая система должна быть осведомлена о наличии гипервизора и положении гостевой физической памяти в реальной физической памяти, что позволит выполнять запросы на прямой доступ к памяти (DMA, Direct Memory Access) корректно.

Отдельной проблемой виртуализации устройств являются накладные расходы на обработку прерываний устройства, вызванные ограничениями технологий аппаратной виртуализации процессора. При каждом прерывании процессор должен передать управление от виртуальной машины гипервизору. Далее, гипервизор передает управление основной ОС, которая, в свою очередь, определяет, что прерывание принадлежит гипервизору. После этого, гипервизор производит «вброс» прерывания в гостевую систему. Эта цепочка вызовов может замедлить доставку прерываний и серьезно увеличить задержку коммуникаций.

В работе [9] описанное выше предположение подтверждается. Авторы предложили систему ELI (ExitLess Interrupts), которая позволяет обрабатывать часть прерываний в гостевой системе без необходимости выходов в гипервизор. Результаты тестов показали, что система ELI может уменьшить накладные расходы KVM/QEMU с 40% до 1-2% при работе ВМ на одном вычислительном ядре. В данном случае ухудшение производительности было связано с большим числом прерываний, генерируемых проброшенной внутрь ВМ сетевой картой Ethernet 10 Гбит/сек.

В рамках данной работы применялось коммуникационное устройство на базе сети Infiniband, при этом были выявлены накладные расходы, связанные с применением IOMMU. Описание проведенного исследования приведено в следующих подразделах.

3.1. Экспериментальная среда

В качестве стенда используется кластер фирмы HP, узлы HP ProLiant SL390s G7, используется до 16 узлов. Каждый узел имеет архитектуру NUMA, содержит по 2 процессора Intel Xeon X5650 (6 ядер на процессор) и 24 Гб ОЗУ. Взаимодействие узлов осуществляется с использованием 1 Гбит/сек сервисной сети Ethernet и 40 Гбит/сек коммуникационной сети Infiniband. Используются адаптеры Mellanox ConnectX PCIe 2.0.

В качестве основной ОС и гостевой ОС применяется дистрибутив CentOS 6.3 (версия ядра `2.6.32-279.5.2.el6.x86_64`). В качестве библиотеки MPI применяется MVAPICH версии `1.2.0`. Для сборки тестов применяется компилятор GCC версии `4.4.6`.

Гипервизор KVM встроен в ядро ОС Linux, его версия соответствует версии ядра ОС. Версия эмулятора QEMU — `1.0.1` с изменениями, необходимыми для корректной эмуляции архитектуры NUMA. Гостевая система запускается с 16 ГБ ОЗУ, все ядра процессоров назначены ВМ с учетом топологии NUMA. Адаптер Infiniband проброшен в ВМ с использованием технологии Intel VT-d. Память ВМ выделяется с использованием вложенной страничной адресации, страницами размером 2 МБ.

Для оценки накладных расходов на виртуализацию применяется тестовый пакет SPEC MPI2007 версии 2.0. Подробное описание тестов пакета приведено в документации по адресу `http://www.spec.org/mpi/Docs/`. Тесты запускаются по 10 раз.

3.2. Оценка накладных расходов

Результаты проведенных экспериментов представлены на Рисунке 1. Столбец «Native» соответствует конфигурации при запуске без гипервизора, «KVM» — базовой рассматриваемой конфигурации, столбец «KVM, no IOMMU» рассматривается в следующем разделе. Тонкие черные столбцы обозначают среднеквадратичное отклонение для соответствующих средних значений. Данные изображены относительно случая Native, которому соответствует среднее значение 0 в каждой группе столбцов. Каждая группа столбцов соответствует конкретному тесту пакета, название теста приведено под группой.

В целом, на большинстве тестов пакета падение производительности не превышает 4%. Существенные накладные расходы возникают в случае тестов leslie3d (8%) и pop2 (23%).

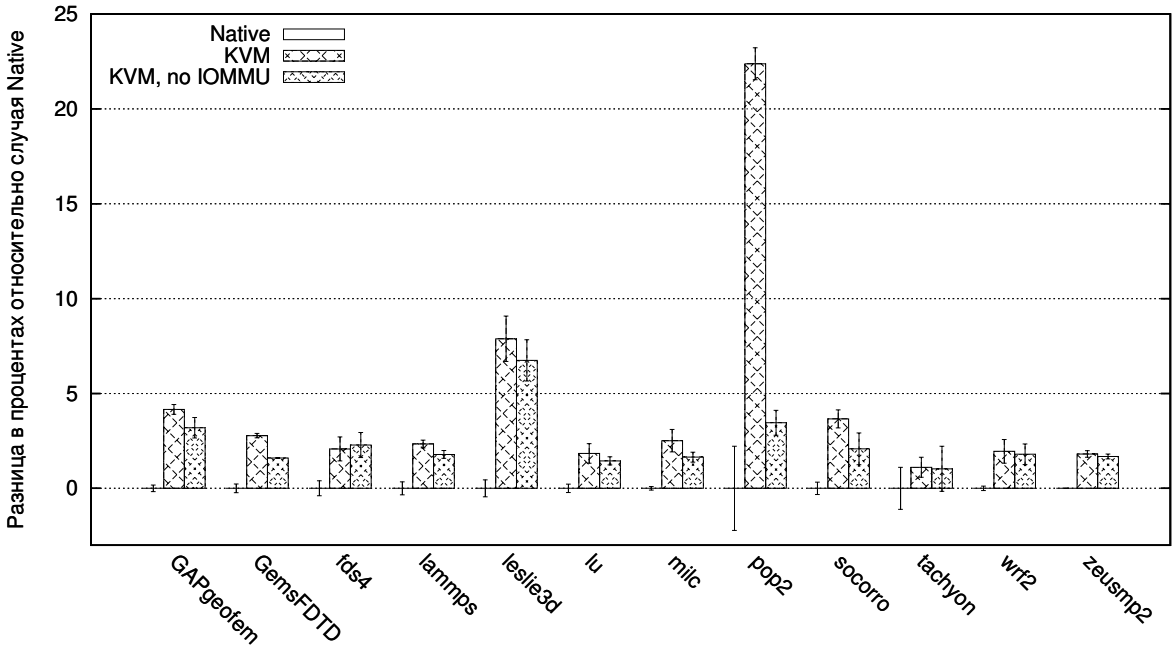


Рис. 1. Результаты экспериментов SPEC MPI2007 с использованием 192 ядер.

3.3. Метод обхода аппаратного IOMMU

В ходе анализа производительности теста pop2 было выдвинуто предположение, что накладные расходы вызваны работой устройства IOMMU. Основной операцией MPI, выполняемой в тесте pop2, является функция Allreduce с небольшим объемом посылаемых данных (8 байт). В ходе тестирования производительности синтетического теста, реализующего такой характер коммуникаций, было получено снижение производительности в ВМ в 2 раза.

Основная задача, которую выполняет IOMMU — трансляция адреса устройства в физический адрес ВМ при выполнении DMA-транзакций. При трансляции используется таблица страниц, предоставленная основной ОС по запросу гипервизора. Так же, как и в случае классической страничной трансляции, используется буфер трансляции, в котором сохраняются последние оттранслированные адреса. Данный буфер имеет ограниченный размер. В соответствии со спецификацией Intel VT-d, в таблице страниц могут поддерживаться различные размеры страниц (включая 4 КБ, 2 МБ, 1 ГБ). Размер страниц непосредственно влияет на объем участка памяти, трансляции адресов для которого могут одновременно находиться в буфере трансляции.

В ходе проведенного исследования было установлено, что в ядре ОС Linux дистрибутива CentOS 6.3 драйвер Intel VT-d не поддерживает размеры страниц, отличные от 4 КБ. При этом в основной ветке ядра Linux (см. `http://kernel.org`) такая поддержка есть начиная с версии 3.0. Несмотря на это, применение обновленного ядра показало отсутствие на тестовом стенде поддержки страниц размером более 4 КБ.

Для проверки предположения о влиянии IOMMU на производительность был разработан и реализован метод обхода аппаратного IOMMU.

Проброс устройства в ВМ без использования IOMMU осуществляется с использованием паравиртуального интерфейса между гостевой ОС и гипервизором. Взаимодействие организуется между драйвером устройства, установленным в гостевой ОС, и гипервизором. Архитектура системы приведена на Рисунке 2. В драйвер пробрасываемого устройства включается специальный модуль, обеспечивающий трансляцию адресов при выполнении DMA-запросов. При этом, как правило, не требуется значительных модификаций драйвера, за счет наличия стандартного API для работы с DMA в ядре Linux. При инициализации драйвера модуль устанавливает связь с гипервизором с использованием паравиртуального интерфейса, реализуемого компонентом обхода IOMMU, входящим в состав гипервизора. Модуль получает таблицу отображений гостевых физических адресов в реальные физические адреса, после чего с использованием этой таблицы осуществляется трансляция адресов DMA-запросов перед передачей команд устройству.

Реализация модуля обхода IOMMU как части драйвера позволяет избежать изменения кода гостевой ОС. Недостатком данного подхода является необходимость незначительной модификации драйвера. Описанный метод также имеет недостатки с точки зрения безопасности (возможно получение доступа к памяти основной системы путем выполнения DMA-транзакций проброшенного устройства), однако в случае HPC-вычислений зачастую гостевая система может считаться доверенной.

Описанный метод обхода IOMMU был реализован для ОС Linux и системы виртуализации KVM/QEMU. Результаты тестирования производительности приведены на Рисунке 1, столбец «KVM, no IOMMU». Применение описанного метода позволило снизить накладные расходы теста pop2 с 23% до 4%. При этом, для теста leslie3d уменьшение накладных расходов незначительно. Необходимо также отметить, что число прерываний, генерируемых адаптером Infiniband, невелико (около 3000 прерываний в секунду), в связи с чем применение подхода ExitLess Interrupts в данном случае не целесообразно.

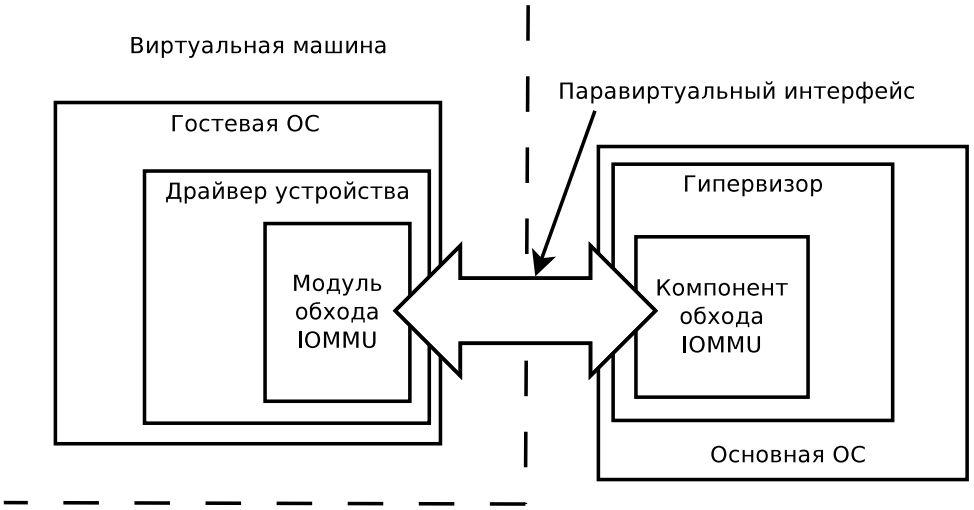


Рис. 2. Архитектура системы обхода IOMMU.

4. Заключение

Основным вкладом данной работы является демонстрация возможности эффективной виртуализации достаточно широкого класса HPC-приложений на Linux-кластерах небольшого масштаба. Для большинства тестов пакета SPEC MPI2007 накладные расходы на виртуализацию составили не более 5% при использовании 192 процессорных ядер (16 узлов). При этом были использованы наработки в области улучшения производительности гипервизора KVM/QEMU, в частности эмуляция архитектуры NUMA в соответствии с архитектурой реального сервера.

Продемонстрировано влияние аппаратной виртуализации ввода-вывода на производительность на примере адаптера Infiniband. В рамках ядра ОС Linux и системы виртуализации KVM/QEMU разработан и реализован механизм, позволяющий предоставлять реальное коммуникационное устройство доверенной виртуальной машине без применения технологии Intel VT-d. Данный механизм позволил существенно снизить накладные расходы на тесте pop2 (с 23% до 4%).

В рамках дальнейших работ планируется увеличение масштаба экспериментов до 1000 ядер и последующее изучение накладных расходов. При увеличении масштаба расчетов на производительность HPC-приложений начинает оказывать влияние шум как гостевой ОС, так и основной.

Литература

1. Younge A.J., Henschel R., Brown J., G. von Laszewski, Qiu J., Fox G.C. Analysis of Virtualization Technologies for High Performance Computing Environments // 4th International Conference on Cloud Computing (IEEE CLOUD 2011), July 4-9, 2011, Washington DC, USA.

2. A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya. Abstract High-Performance Hypervisor Architectures: Virtualization in HPC Systems // 1st Workshop on System-level Virtualization for High Performance Computing, (HPCVirt), in conjunction with EuroSys 2007, March 20, Lisbon, Portugal.

3. J. R. Lange, K. Pedretti, P. Dinda, P. G. Bridges, C. Bae, P. Soltero, and A. Merritt. Minimal-overhead virtualization of a large scale supercomputer // 7th ACM

SIGPLAN/SIGOPS international conference on Virtual execution environments, VEE '11, March 09-11, 2011, Newport Beach, CA, USA, Proceedings. ACM New York. 2011. P. 169–180.

4. А. О. Кудрявцев, В. К. Кошелев, А. И. Аветисян. Перспективы виртуализации высокопроизводительных систем архитектуры x64 // Труды Института системного программирования РАН, том 22, 2012. Стр. 189-209.

5. A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux virtual machine monitor // OLS '07: The 2007 Ottawa Linux Symposium, July, 2007, P. 225–230.

6. D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, J. Wiegert. Intel® Virtualization Technology for Directed I/O // Intel Technology Journal.
URL: http://www.intel.com/technology/itj/2006/v10i3/2-io/1-abstract.htm (дата обращения: 22.11.2012).

7. M. S. Muller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, H. Feng, C. Ponder. SPEC MPI2007—an application benchmark suite for parallel systems using MPI // Concurrency and Computation: Practice and Experience, February 2010, Vol. 22, N. 2, P. 191–205.

8. K. Ferreira, P. Bridges, R. Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection // International Conference for High Performance Computing, Networking, Storage and Analysis, November 2008, P. 1-12.

9. A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, D. Tsafrir. ELI: Bare-Metal Performance for I/O Virtualization // Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2012), March 03-07, 2012, London, UK, Proceedings. ACM New York. 2012. P. 411-422.

Моделирование нестационарного процесса сопряженного теплообмена горного массива и рудничного воздуха с применением высокопроизводительных вычислительных систем

А.Р. Куцев

Пермский государственный университет, 614990, Пермь, ул. Букирева, 15

В данной работе рассматривается нестационарный процесс сопряженного теплообмена между горным массивом и рудничным воздухом. Для численного моделирования используются вычисления на графических процессорах NVIDIA CUDA суперкомпьютера «ПГУ-Тесла». Полученное в ходе расчетов ускорение составляет 45 раз.

1. Введение

Интерес к исследованию теплообменных процессов в рудничной вентиляции вызван, прежде всего, их влиянием на изменение основных характеристик вентиляционного воздуха – температуры и влажности, формирующих рудничный микроклимат.

В данной работе предполагается следующий подход к построению физико-математической модели теплообмена. Суть подхода заключается в постановке и решении сопряженной задачи нестационарного теплообмена двух сред. Ставится задача рассчитать численно температуру воздуха $T(z,t)$ в горизонтальной цилиндрической выработке. Задача является принципиально нестационарной, поскольку интенсивность теплообмена воздуха со стенками зависит от глубины прогревания (охлаждения) массива, которая будет тем больше, чем больше пройдет времени (рис. 1).

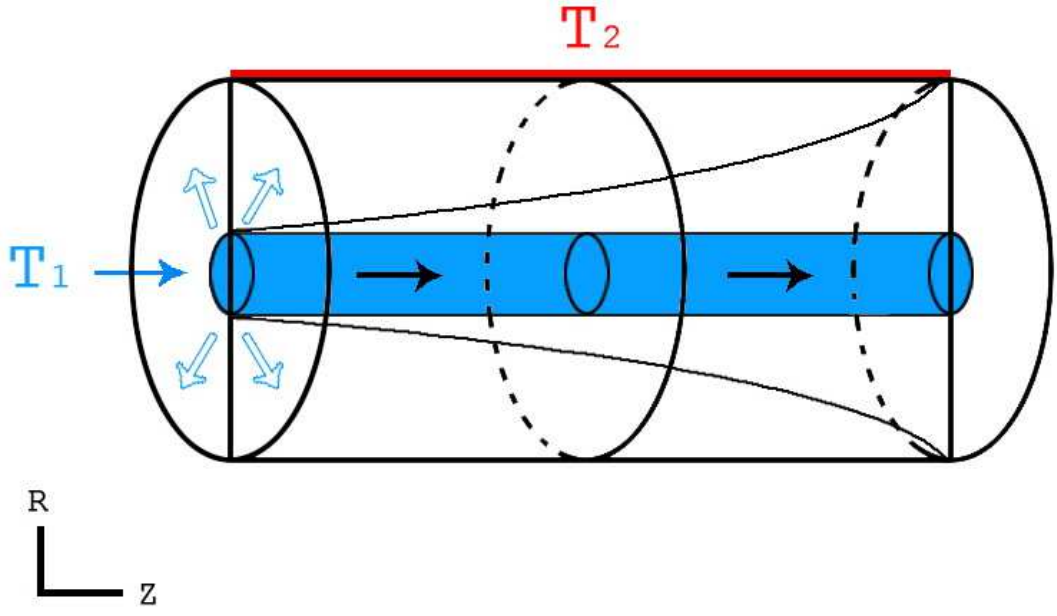


Рис. 1. Цилиндрическая расчетная область для отдельной горной выработки

Помимо теплопереноса в рудничном воздухе мы должны рассматривать теплоперенос в горном массиве. В данном случае существенно не только распределение температуры вдоль выработки, но и радиальное распределение температуры вглубь горного массива.

Задача о теплопереносе нас интересует, прежде всего, в сетевой постановке, когда температурное поле моделируется не в отдельной выработке, а множестве выработок с учетом согласования температур и тепловых потоков в разветвлениях выработок.

2. Постановка задачи (одномерный вариант)

Пусть по цилиндрическому каналу радиуса *R1* движется однородный однонаправленный поток газа (воздуха) с постоянной скоростью *v*. Толщина стенки канала *R2 - R1*, причем *R2 >> R1* (рис. 2). Температура газа в начальный момент времени $\tau=0$ во всем канале равна *T1*, температура стенки в начальный момент времени равна *T2*, причем $T2 \neq T1$. Температура на внешней поверхности стенки канала в любой момент времени $\tau>0$ остается постоянной и равной *T2*.

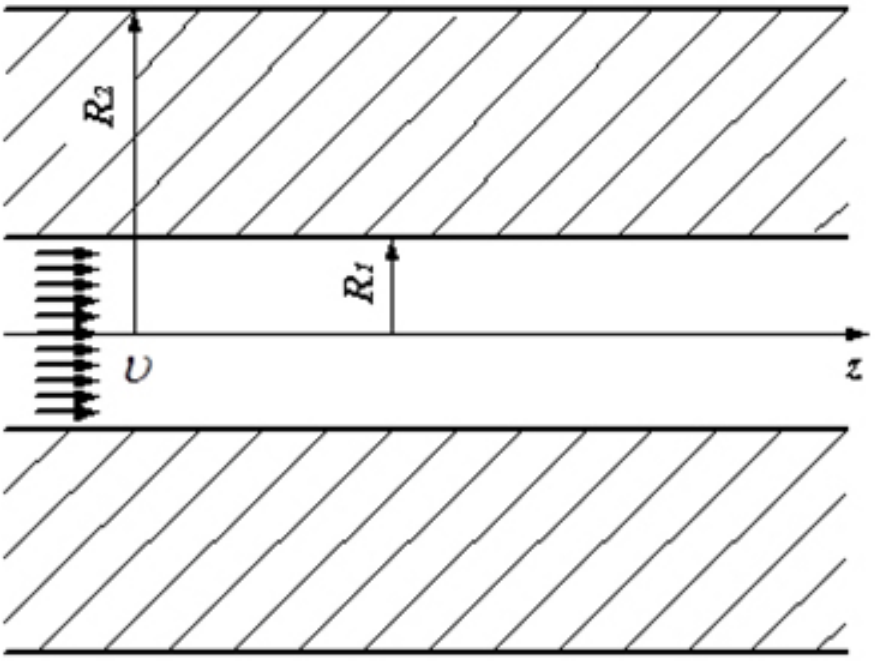


Рис. 2. Схема расчетной области

С течением времени в газе и в стенке канала установится стационарное температурное поле, которое будет зависеть только от радиальной координаты r. Необходимо найти это температурное поле.

Так как расчетная область обладает осевой симметрией, задачу будем решать в цилиндрических координатах. Ось Oz цилиндрической системы координат направим по оси канала (рис. 2). Поперечными течениями в газе пренебрегаем, тогда, так как $T=T(r,\tau)$, уравнения, описывающие изменение температуры в газе и в стенке примут вид [1]:

$$\frac{\partial T}{\partial \tau} = a\left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r}\right). \tag{1}$$

В уравнении (1) коэффициент температуропроводности при $r \le R_1$

$$a = \frac{\lambda_г}{c_{pг}\rho_г}, \tag{2}$$

а при $R_1 \le r \le R_2$

$$a = \frac{\lambda_c}{c_c \rho_c}. \tag{3}$$

Здесь $\lambda_г$, $c_{pг}$ и $\rho_г$ – коэффициент теплопроводности, изобарная удельная теплоемкость и плотность газа, а λ_c, c_c и ρ_c – коэффициент теплопроводности, удельная теплоемкость и плотность материала стенки.

Уравнение (1) дополним начальными условиями

$$T(r,0)\big|_{r\le R_1} = T_1,\ T(r,0)\big|_{R_1\le r\le R_2} = T_2, \tag{4}$$

граничным условием на оси канала (условие симметрии)

$$\left.\frac{\partial T}{\partial r}\right|_{r=0} = 0, \tag{5}$$

граничным условием на внутренней поверхности стенки канала

$$\alpha_г\left(T_г - T_c\right)\big|_{r=R_1} = -\lambda_c \left.\frac{\partial T}{\partial r}\right|_{r=R_1}, \tag{6}$$

где $T_г$ и T_c температуры газа и стенки и граничным условием на внешней поверхности стенки канала

$$T\big|_{r=R_2} = T_2 \cdot \tag{7}$$

Граничные условия (5) – (7) должны выполняться в любой момент времени τ .

3. Численное решение задачи

Для численного решения уравнения (1) построим сетку с шагом Δr_1 по пространственной координате r в области $0 \le r \le R_1$, с шагом Δr_2 по пространственной координате r в области $R_1 \le r \le R_2$ и с шагом $\Delta\tau$ по времени (рис. 3). Причем при $r = R_1$ создается двойная точка. Координаты узлов сетки

$$r_k = (k-1)\Delta r_1,\ k = \overline{1, N_1}, \tag{8}$$

где $\Delta r_1 = \frac{R_1}{N_1 - 1}$ относятся к газовой среде, а узлы с координатами

$$r_k = R_1 + (k - N_1 - 1)\Delta r_2,\ k = \overline{N_1 + 1, N_1 + N_2}, \tag{9}$$

где $\Delta r_2 = \frac{R_2 - R_1}{N_2 - 1}$ находятся в стенке канала.

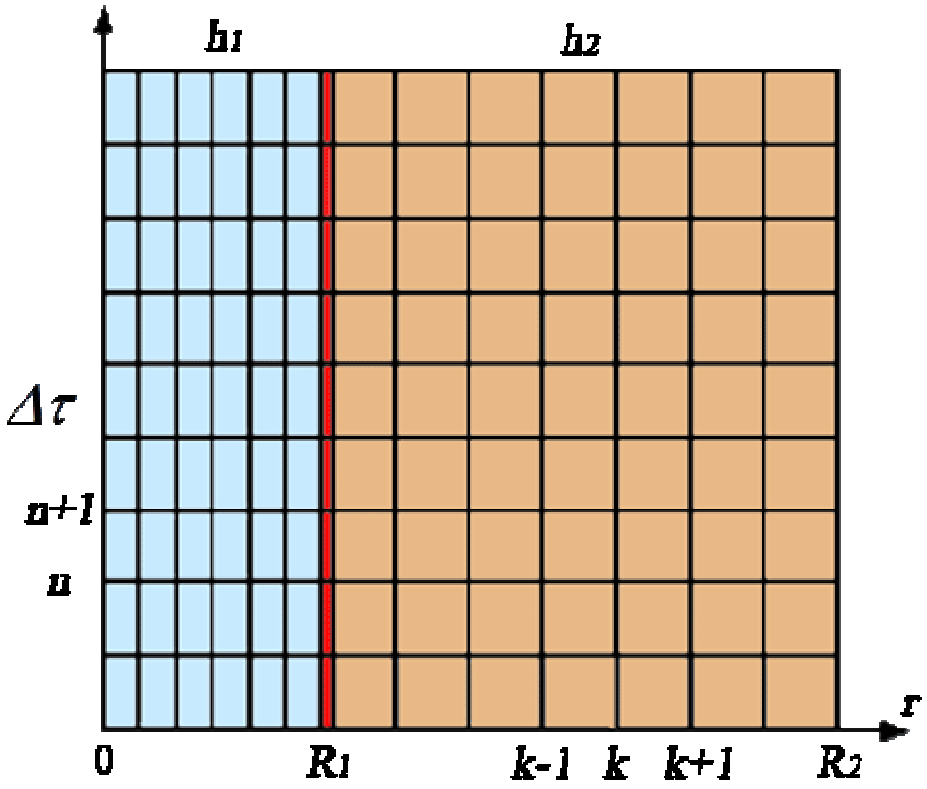


Рис. 3. Разностная сетка

Значение температуры $T(r,\tau)$ в точке r_k в момент времени $n\Delta\tau$, $n=0,1,2,...$ будем обозначать через T_k^n, тогда конечно-разностный аналог уравнения (1) будет иметь вид

$$\frac{T_k^{n+1}-T_k^n}{\Delta\tau}=a\left(\frac{T_{k+1}^n-2T_k^n+T_{k-1}^n}{(\Delta r)^2}+\frac{1}{r_k}\frac{T_{k+1}^n-T_{k-1}^n}{2\Delta r}\right). \tag{10}$$

Таким образом, уравнение, позволяющее по известным значениям температуры на i-ном временном слое определить температуру в узлах сетки на $i+1$ временном слое, будет иметь вид

$$\begin{aligned}T_k^{n+1}&=T_k^n+\frac{a\Delta\tau}{(\Delta r)^2}\left(T_{k+1}^n-2T_k^n+T_{k-1}^n\right)\\&+\frac{a\Delta\tau}{2r_k\Delta r}\left(T_{k+1}^n-T_{k-1}^n\right)\end{aligned} \tag{11}$$

Уравнение (11) применимо только для внутренних узлов сетки, т. е. для $k=\overline{2,N_1-1}$ и для $k=\overline{N_1+2,N_1+N_2-1}$. Причем при $k=\overline{2,N_1-1}$ в этом уравнении $\Delta r=\Delta r_1$, r_k и коэффициент температуропроводности a вычисляются по формулам (8) и (2), а при $k=\overline{N_1+2,N_1+N_2-1}$ в уравнении (6.4) $\Delta r=\Delta r_2$, r_k вычисляется по формуле (9), а коэффициент температуропроводности – по формуле (3).

При $k=1$ на оси симметрии, с учетом граничного условия (5), уравнение (1) принимает вид

$$\frac{\partial T}{\partial\tau}=a\left(\frac{\partial^2 T}{\partial r^2}\right).$$

Если за пределами расчетной области ввести на расстоянии Δr_1 фиктивную «нулевую» точку, то получим разностное уравнение

$$T_1^{n+1}=T_1^n+\frac{a\Delta\tau}{(\Delta r_1)^2}\left(T_2^n-2T_1^n+T_0^n\right),$$

Из условия симметрии $T_0^n=T_2^n$, и для определения температуры в точке $k=1$ получаем

$$T_1^{n+1}=T_1^n+\frac{2a\Delta\tau}{(\Delta r_1)^2}\left(T_2^n-T_1^n\right). \tag{12}$$

Для определения температуры в точках $k=N_1$ и $k=N_1+1$ запишем граничное условие (5.6) в конечно-разностном виде

$$\alpha_c\left(T_{N_1}^{n+1}-T_{N_1+1}^{n+1}\right)=-\lambda_c\frac{T_{N_1+2}^{n+1}-T_{N_1+1}^{n+1}}{\Delta r_2}. \tag{13}$$

Уравнения (13) недостаточно для определения двух неизвестных величин, поэтому поступим следующим образом [2]. Представим значение температуры в узле сетки N_1 через разложение в ряд Тейлора в окрестности точки N_1-1

$$T_{N_1}=T_{N_1-1}+\Delta r_1\left(\frac{\partial T}{\partial r}\right)_{N_1-1}+\frac{(\Delta r_1)^2}{2}\left(\frac{\partial^2 T}{\partial r^2}\right)_{N_1-1}.$$

но из уравнения (1)

$$\frac{\partial^2 T}{\partial r^2}=\frac{1}{a}\frac{\partial T}{\partial\tau}-\frac{1}{r}\frac{\partial T}{\partial r},$$

тогда

$$T_{N_1}=T_{N_1-1}+\Delta r_1\left(\frac{\partial T}{\partial r}\right)_{N_1-1}+\frac{(\Delta r_1)^2}{2}\left(\frac{1}{a}\frac{\partial T}{\partial\tau}-\frac{1}{r}\frac{\partial T}{\partial r}\right)_{N_1-1},$$

Запишем полученное уравнение в конечных разностях

$$\frac{T_{N_1-1}^{n+1}-T_{N_1-1}^{n}}{\Delta\tau}=\frac{2a}{(\Delta r_1)^2}\left(T_{N_1}^{n+1}-T_{N_1-1}^{n+1}\right)+\left(\frac{a}{r_{N_1-1}}-\frac{2a}{\Delta r_1}\right)\left(\frac{T_{N_1}^{n+1}-T_{N_1-2}^{n+1}}{2\Delta r_1}\right). \quad (14)$$

Из уравнений (13) и (14) получим

$$T_{N_1}^{n+1}=\frac{\Delta r_1-2r_{N_1-1}}{\Delta r_1+2r_{N_1-1}}T_{N_1-2}^{n+1}+\frac{2r_{N_1-1}(\Delta r_1)^2+4ar_{N_1-1}\Delta\tau}{a\Delta\tau(\Delta r_1+2r_{N_1-1})}T_{N_1-1}^{n+1}-\frac{2r_{N_1-1}(\Delta r_1)^2}{a\Delta\tau(\Delta r_1+2r_{N_1-1})}T_{N_1-1}^{n} \quad (15)$$

$$T_{N_1+1}^{n+1}=\frac{\alpha_г\Delta r_2}{\alpha_г\Delta r_2+\lambda_c}T_{N_1}^{n+1}+\frac{\lambda_c}{\alpha_г\Delta r_2+\lambda_c}T_{N_1+2}^{n+1}. \quad (16)$$

Для определения температуры в узле N_1+N_2 воспользуемся граничным условием (5.7)

$$T_{N_1+N_2}^{n+1}=T_2. \quad (17)$$

Для определения шага по времени $\Delta\tau$ исследуем уравнение на устойчивость методом гармоник [3]. Представим решение разностной задачи в узле сетки в виде $T_k^n=\lambda^n e^{ik\varphi}$ и подставим в уравнение (10). Получим, что схема устойчива, при:

$$\Delta\tau\le\min\left(\frac{(\Delta r_1)^2}{2a_г},\frac{(\Delta r_2)^2}{2a_c}\right) \quad (18)$$

Для решения поставленной задачи была написана программа на языке C# для расчета распределения температуры в стенке массива и воздухе.

В результате для следующих входных данных (*R1*, *R2*, υ, T_{01}, T_{02}, $\lambda_г$, $c_{pг}$, $\rho_г$, λ_c, c_c, ρ_c), мы получили следующее графическое решение:

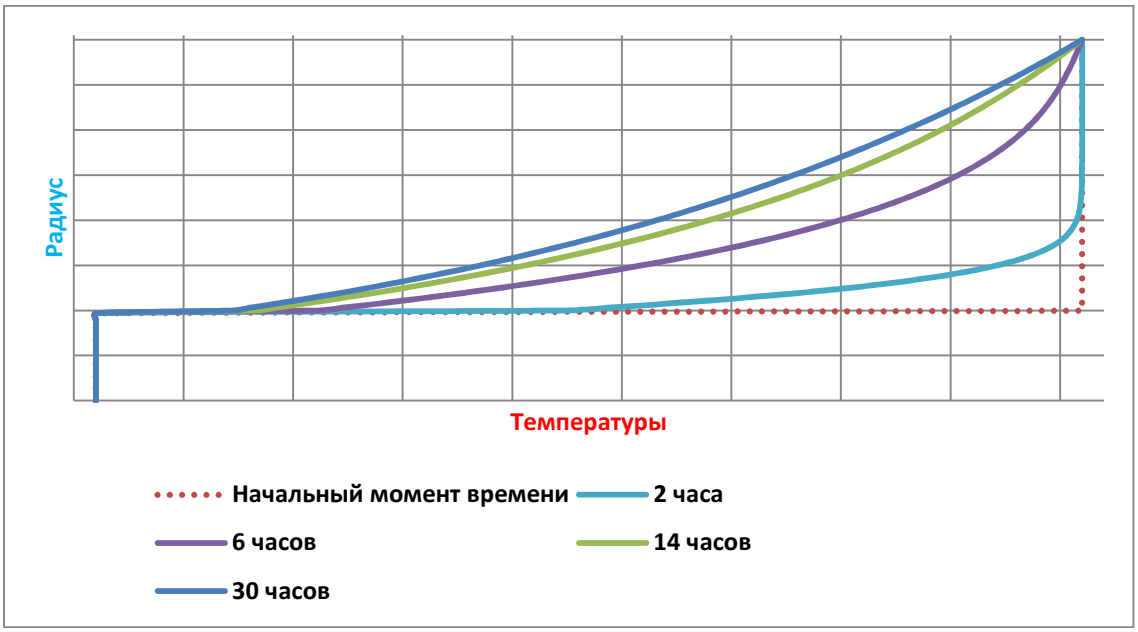


Рис. 4. График распределения температуры в цилиндрической горной выработке в зависимости от времени

Приведенные результаты (рис. 4) и результаты других расчетов показали, что температура воздуха практически не меняется по поперечному сечению канала. Поток воздуха в любом сечении канала можно считать однородным. В реальных течениях температура, скорость и давление воздуха по длине канала изменяются. Поэтому в дальнейшем будем считать поток воздуха одномерным (зависящим от продольной координаты z) нестационарным. Поле температур в массиве зависит как от времени, так и от координат r и z. Причем при каждом фиксированном z, зависимость температуры от радиальной координаты при постоянных граничных условиях стремится к стационару.

4. Решение задачи нестационарного процесса сопряженного теплообмена между рудничным воздухом и горным массивом в двумерной постановке

Пусть по цилиндрическому каналу радиуса R_1 и длиной l движется однородный однонаправленный поток газа (воздуха) со скоростью $u=u_z(z,\tau)$. Толщина стенки канала R_2-R_1 , причем $R_2>>R_1$. Температура газа в начальный момент времени $\tau=0$ во всем канале равна T_1, температура стенки в начальный момент времени равна T_2, причем $T_2 \neq T_1$. Температура на внешней поверхности стенки канала в любой момент времени $\tau=0$ остается постоянной и равной T_2.

Так как расчетная область обладает осевой симметрией, задачу будем решать в цилиндрических координатах. Ось Oz цилиндрической системы координат направим по оси канала. Поперечными течениями изменениями температуры в газе пренебрегаем, тогда, так как $T=T(z,\tau)$, уравнение, описывающие изменение температуры в газе примет вид

$$\frac{\partial T}{\partial \tau}+\frac{\partial}{\partial z}\left(uT\right)=a_z\frac{\partial^2 T}{\partial z^2}+\frac{Q_V}{c\rho} \tag{19}$$

Для определения объемной плотности внутренних источников теплоты Qv выделим участок канала длиной l, пусть d – диаметр канала, тогда тепловой поток от стенки канала к воздуху будет равен

$$q=\alpha\left(T_{cm}-T\right)\pi dl$$

Эквивалентный тепловой поток от объемных источников тепла

$$q=Q_V\frac{\pi d^2}{4}l$$

Из этих равенств получаем, что в одномерной постановке

$$Q_V=\frac{2\alpha\left(T_{cm}-T\right)}{R_1}$$

Дополним уравнение (19) уравнением неразрывности

$$\frac{\partial \rho}{\partial \tau}+\frac{\partial}{\partial z}\left(u\rho\right)=0, \tag{20}$$

уравнением движения

$$\frac{\partial u}{\partial \tau}+u\frac{\partial u}{\partial z}+\frac{1}{\rho}\frac{\partial p}{\partial z}=0 \tag{21}$$

и уравнением состояния

$$p=\rho RT \tag{22}$$

Здесь $R=287\frac{Дж}{кг\cdot К}$ – газовая постоянная воздуха, T – абсолютная температура (в $К$).

Изменение нестационарного поля температур в стенке $Tc=Tc(r,z,\tau)$ описывается уравнением

$$\frac{\partial Tc}{\partial \tau}=a_c\left(\frac{\partial^2 Tc}{\partial r^2}+\frac{1}{r}\frac{\partial Tc}{\partial r}+\frac{\partial^2 Tc}{\partial z^2}\right) \tag{22}$$

Приведем систему уравнений (19) –(22) к безразмерному виду. В качестве обезразмеривающих параметров возьмем характерную плотность ρ^*, характерное давление p^*, характерную длину l^*, характерную температуру T^*, характерное время $\tau^* = \frac{l^*}{u^*}$, характерную скорость $u^* = \sqrt{\frac{p^*}{\rho^*}}$. При таком выборе обезразмеривающих параметров, уравнения (20) и (21) сохраняют свой вид. Уравнение (19) примет вид

$$\frac{\partial T}{\partial \tau} + \frac{\partial}{\partial z}(uT) = \overline{a}_г \frac{\partial^2 T}{\partial z^2} + \frac{\overline{Q}_V}{\rho} \tag{23}$$

уравнение (22) примет вид

$$\frac{\partial Tc}{\partial \tau} = \overline{a}_c \left(\frac{\partial^2 Tc}{\partial r^2} + \frac{1}{r}\frac{\partial Tc}{\partial r} + \frac{\partial^2 Tc}{\partial z^2} \right) \tag{24}$$

а уравнение (21) примет вид

$$p = \rho \overline{R} T \tag{25}$$

Здесь $\overline{R} = \frac{\rho^* T^*}{p^*} R$ безразмерная газовая постоянная, $\overline{a}_г = \frac{a_г}{u^* l^*}$ безразмерная температуропроводность газа, $\overline{a}_c = \frac{a_c}{u^* l^*}$ безразмерная температуропроводность стенки, $\overline{Q}_V = \frac{l^* Q_V}{T^* u^* \rho^* c_p}$ безразмерная объемная плотность внутренних источников теплоты.

При расчетах принималось $p^* = 101325 Па$, $T^* = 273K$, $l^* = 2R_1$, $\rho^* = \frac{p^*}{RT^*}$.

Систему уравнений (20), (21), (24), (25) дополним начальными условиями

$$T(z,0)\big|_{0 \le z \le l} = T_1(z),$$
$$u(z,0)\big|_{0 \le z \le l} = u_1(z),$$
$$p(z,0)\big|_{0 \le z \le l} = p_1(z),$$
$$Tc(r,z,0)\big|_{R_1 \le r \le R_2, 0 \le z \le l} = Tc_1(r,z),$$

и граничными условиями:

$$Tc(r,z,\tau)\big|_{r=R_2, 0 \le z \le l} = Tc_2(z,\tau),$$
$$Tc(r,0,\tau)\big|_{R_1 \le r \le R_2} = Tc_3(r,\tau),$$
$$Tc(r,l,\tau)\big|_{R_1 \le r \le R_2} = Tc_4(r,\tau),$$
$$\alpha(T - Tc)\big|_{r=R_1, 0 \le z \le l} = \lambda \frac{\partial Tc}{\partial r}\Big|_{r=R_1, 0 \le z \le l},$$
$$u(0,\tau) = u_2(\tau),\ u(l,\tau) = u_3(\tau),$$
$$p(0,\tau) = p_2(\tau),\ p(l,\tau) = p_3(\tau),$$
$$T(0,\tau) = T_2(\tau),\ T(l,\tau) = T_3(\tau).$$

Исследование на устойчивость выбранной разностной схемы показала, что шаг по времени $\Delta\tau$ необходимо выбирать из следующего условия

$$\Delta\tau = min\left(\min_j \frac{2\Delta z^2}{u_j^n \Delta z - 2a_г}, \frac{\Delta z^2 \cdot \Delta r^2}{2a_c\left(\Delta z^2 + \Delta r^2\right)} \right). \tag{26}$$

Для реализации двумерного алгоритма была выбрана программно-аппаратная архитектура NVIDIA CUDA. Вычисления проводились на суперкомпьютере «ПГУ-Тесла» - высокопроизводительном многопроцессорном вычислительном комплексе с гибридной архитектурой [6].

При разработке параллельных алгоритмов решения задач вычислительной математики принципиальным моментом является анализ эффективности использования параллелизма, состоящий обычно в оценке получаемого ускорения процесса вычисления (сокращения времени решения задачи). Формирование подобных оценок ускорения может осуществляться применительно к выбранному вычислительному алгоритму. При этом упор сделан на выбор архитектуры ГПУ для решения поставленной задачи, т.к. предполагается независимая параллельная однотипная обработка большого количества данных.

В таблице 1, рисунке 5 приведены результаты решения задачи на стационарном ПК (Intel Dual-Core T4200), процессоре Intel Xeon 5670 (ПГУ-Тесла), на видеокарте Nvidia Tesla S2050 (непосредственный алгоритм на CUDA C, а так же последовательный код с использованием директивы OpenACC). Задача решалась на сетке 1024 × 1025 в области [0, 100] × [0, 20] , было выполнено 50000 шагов по времени.

Таблица 1. Результаты решения нестационарного процесса сопряженного теплообмена между горным массивом и рудничным воздухом

Устройство	Время решения	Ускорение
Intel Dual-Core T4200	34 ч. 28 мин. 11 сек.	–
Intel Xeon 5670	21 ч. 09 мин. 44 сек.	1.65x
Nvidia Tesla S2050 (OpenACC)	03 ч. 37 мин. 48 сек.	9.4x
Nvidia Tesla S2050 (CUDA C)	00 ч. 46 мин. 35 сек.	45x

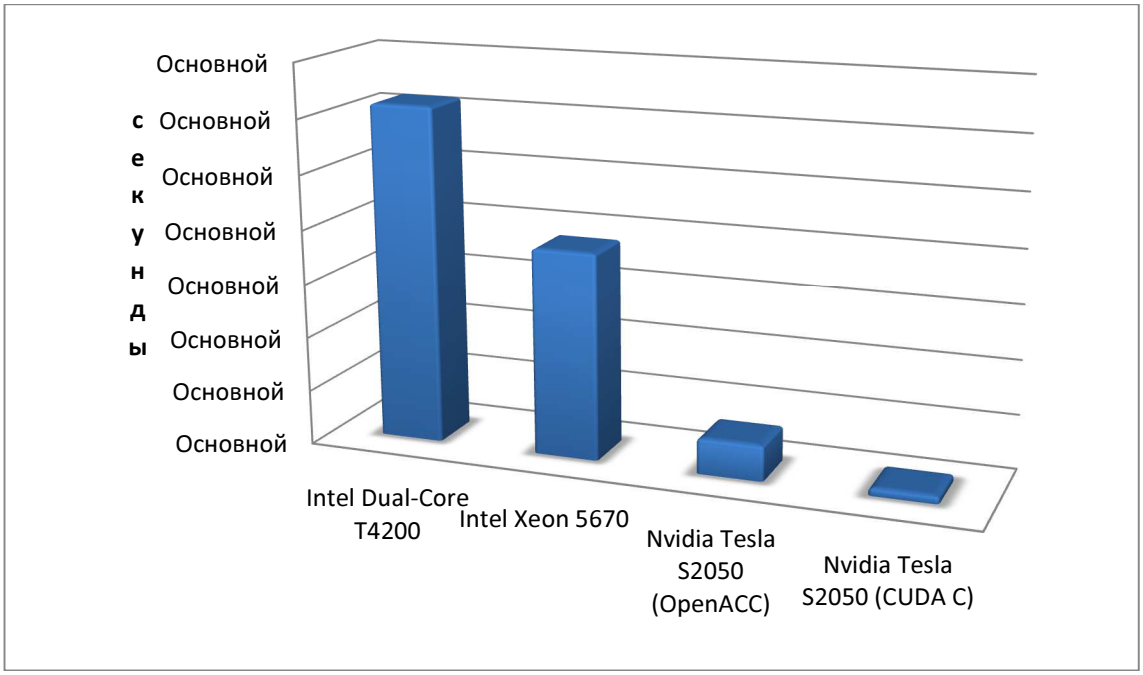


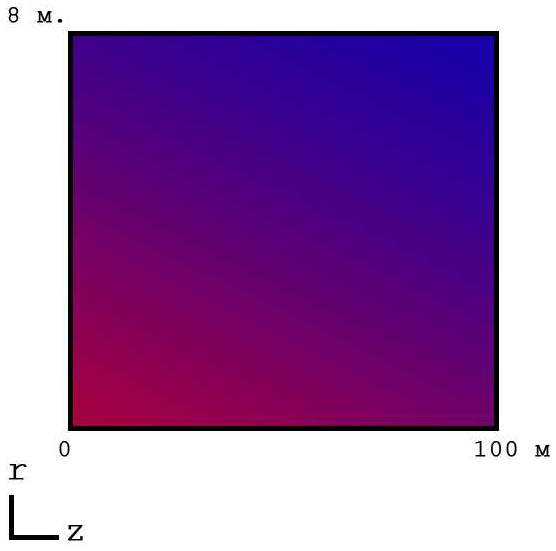
Рис. 5. Сравнение времени решения задачи на CPU, GPU

Проведены многочисленные расчеты с различными параметрами.

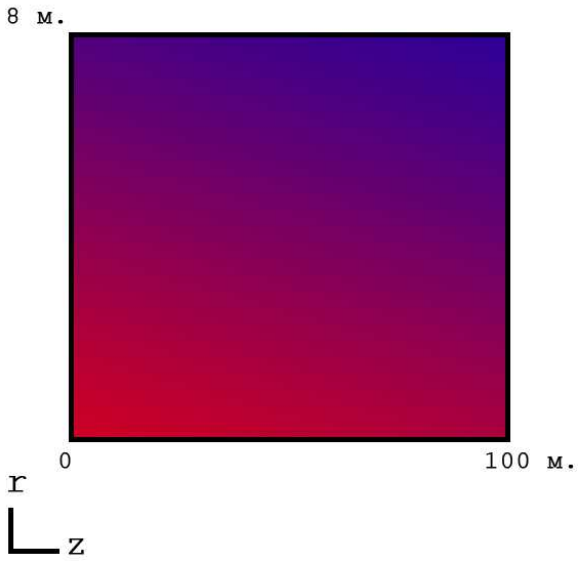
Для примера, приведем результаты расчета программы на сетке 1024 × 1025 в области [0, 100] × [0, 8]**.**

Прошло времени:

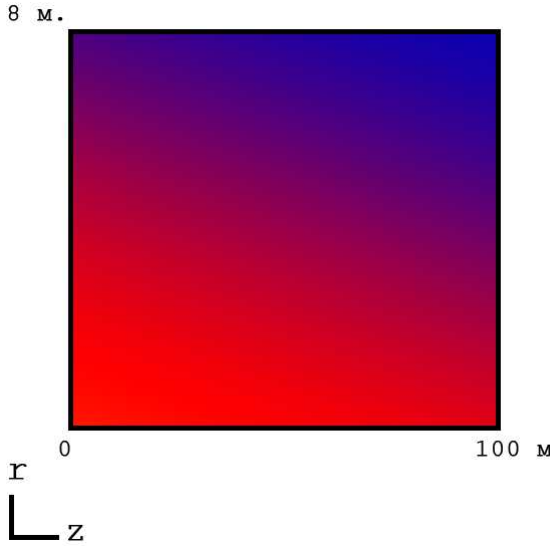
56 минут:



2 часа 57 минут:



6 часа 51 минута:



13 часов 37 минут:

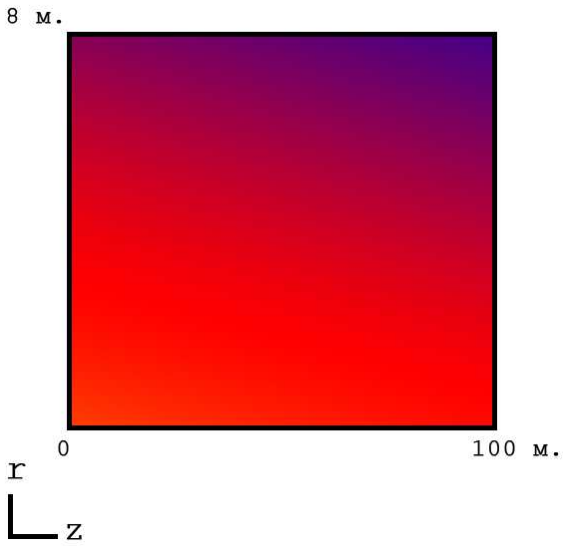


Рис.6. Распределение температуры в горном массиве

Данная работа направлена на численное моделирование нестационарного процесса сопряженного теплообмена между горным массивом и рудничным воздухом с использованием высокопроизводительной вычислительной системы «ПГУ-Тесла» (Рис. 6).

В процессе решения поставленной задачи была реализована программа для моделирования теплопереноса в горном массиве и рудничном воздухе с применением технологии NVIDIA CUDA, а также с использованием директивы OpenACC.

Результаты работы вошли в программный комплекс «Аэросеть» Горного института УрО РАН.

Литература

1. Теплотехника:Учебник для вузов/ В.Н.Луканин, М.Г.Шатров, Г.М.Камфер и др.; Под ред. В.Н.Луканина – М.: Высшая школа,2000.
2. Юдаев Б. Н. Теплопередача. Учебник для вузов. – М.: Высшая школа, 1973.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. Михеев М. А. Основы теплопередачи. М.: Госэнергоиздат, 1956.
4. Мак-Адамс В. Ч. Теплопередача. М.:Металлургиздат, 1961.
5. Арнольд Л. В., Михайловский Г. А., Селиверстов В. М. Техническая термодинамика и теплопередача. М.: Высш.шк., 1979.
6. Гергель В. П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем: Учебник – М.: Издательство Московского университета, 2010.

Высокопроизводительная реконфигурируемая вычислительная система РВС-7 на основе ПЛИС VIRTEX-7*

И.И. Левин[1], И.А. Каляев[1], А.И. Дордопуло[2], Е.А. Семерников[2]

НИИ многопроцессорных вычислительных систем имени академика А.В. Каляева
Южного федерального университета, г. Таганрог, Россия[1]
Южный научный центр Российской академии наук, г. Ростов-на-Дону, Россия[2]

В статье рассматриваются конструктивные особенности и характеристики реконфигурируемой вычислительной системы РВС-7, построенной на основе программируемых логических интегральных схем (ПЛИС) семейства Xilinx Virtex-7. Отличительной характеристикой РВС-7 являются высокая удельная производительность и энергоэффективность при решении прикладных задач, наличие высокоскоростных интерфейсов, а также близкий к линейному рост производительности при увеличении аппаратного ресурса. В статье приводятся технические характеристики РВС-7 и описывается разрабатываемый комплекс системного программного обеспечения.

1. Введение

Поиск новых решений в области архитектурных принципов построения суперкомпьютеров, использующихся для решения прикладных задач в различных областях науки и техники, подтвердил высокую эффективность реконфигурируемых вычислительных систем при решении вычислительно трудоемких задач. В полной мере преимущества от использования реконфигурируемых вычислительных систем (РВС) достигаются при использовании в качестве основного вычислительного элемента аппаратного ресурса программируемых логических интегральных схем (ПЛИС) [1], объединенных в единое вычислительное поле высокоскоростными каналами передачи данных.

Методы разработки и создания таких систем успешно развиваются в НИИ многопроцессорных вычислительных систем Южного федерального университета (г. Таганрог). Концепция построения РВС [2] позволила создать целый ряд высокопроизводительных систем различных архитектур и конфигураций, предназначенных для решения вычислительно трудоемких задач различных предметных областей, успешно эксплуатируемых организациями и ведомствами Российской Федерации. В качестве элементной базы для построения таких РВС использовались ПЛИС Xilinx семейств Virtex-5 (семейство РВС, разработанное по госконтракту №02.524.12.4002 от 20.04.2007) [3] и Virtex-6 большой интеграции, соединенные в единый вычислительный ресурс высокоскоростными каналами передачи данных – LVDS и Rocket GTX [4].

2. Состав аппаратного обеспечения РВС-7

Перспективная реконфигурируемая вычислительная система РВС-7 на основе ПЛИС Virtex-7, разработанная по государственному контракту №14.527.12.0004 от 03.10.2011 и изготавливаемая в настоящее время, содержит вычислительное поле из 576 микросхем ПЛИС Virtex-7 XC7V585T-FFG1761, каждая из которых содержит 58 миллионов эквивалентных вентилей, конструктивно объединенных в один вычислительный шкаф высотой 47U с пиковой производительностью 10^{15} операций с фиксированной запятой в секунду.

Основным структурным компонентом РВС-7, предназначенным для установки в стандартную 19" вычислительную стойку, является вычислительный модуль (ВМ) 24V7-750, компоновка которого представлена на рис. 1, а фотография – на рис. 2. В состав ВМ 24V7-750 входят: четыре платы вычислительного модуля 6V7-180; управляющий модуль УМ-7; подсистема питания; подсистема охлаждения и другие подсистемы.

* Исследования выполнены при финансовой поддержке Министерства образования и науки РФ.

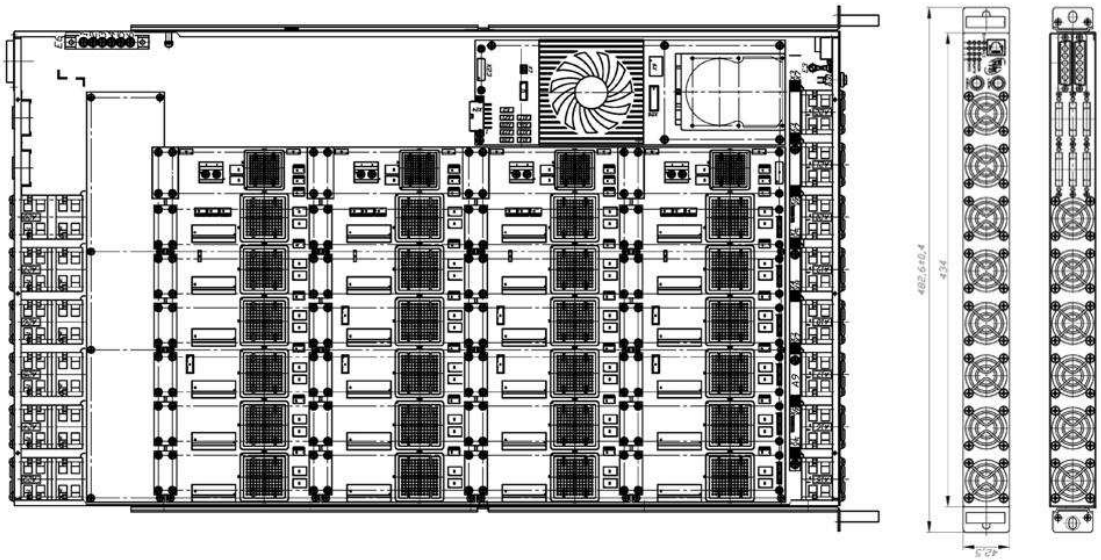
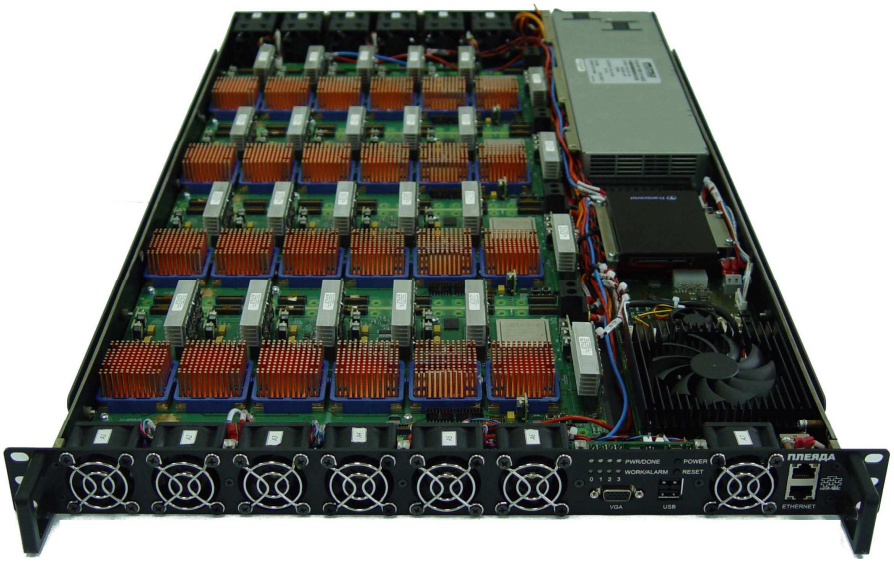


Рис. 1. Компоновка ВМ 24V7-750



а) со снятой верхней крышкой

б) с установленной верхней крышкой

Рис. 2. Фотография ВМ 24V7-750

На рис. 3 представлена структурная схема платы вычислительного модуля (ПВМ) 6V7-180, являющейся основой для построения ВМ 24V7-750. Вычислительное поле ПВМ 6V7-180 выполнено на микросхемах XC7V585T-FFG1761, содержащих около 58 миллионов эквивалентных вентилей.

В состав ПВМ 6V7-180 входят:

- контроллер ПВМ, выполненный на ПЛИС XC6V130T-1FFG1156C производства Xilinx;
- вычислительное поле, состоящее из 6-ти ПЛИС XC7V585T-1FFG1761 семейства Virtex-7 производства фирмы Xilinx. Между собой ПЛИС вычислительного поля соединены последовательно, передача данных осуществляется по 144 дифференциальным линиям LVDS-интерфейса на частоте 800 МГц;
- 12 каналов интерфейса LVDS на частоте 800 МГц по 25 дифференциальных пар каждый (разъёмы типа SS4) для связи с другими вычислительными модулями;
- узлы основной и резервной загрузки ПЛИС по интерфейсам JTAG-1 и JTAG-2;
- подсистема синхронизации (генераторы ECS-2033-250-BN и распределители тактовых импульсов IDT5T9316NLI;
- распределённая память в составе 12-ти микросхем динамической памяти (MT47H128M16HR-25E с организацией 128 М*16 и частотой записи/чтения до 400 МГц). К ПЛИС вычислительного поля, а также к ПЛИС контроллера базового модуля, подключено по две микросхемы памяти DDR2. Объем оперативной памяти на ПВМ 3 Гбайта;
- 2 канала интерфейса LVDS по 20 дифференциальных пар для связи с персональным компьютером и внешней аппаратурой;
- подсистема загрузки ПЛИС;
- подсистема питания, в состав которой входят DC-DC преобразователи напряжения, вырабатывающие напряжения питания: +1 В – питание ядер ПЛИС; +2,5 В – питание узла тактирования; +1,8 В – питание микросхем памяти DDR2, +3,3 В – буферных каскадов ПЛИС.

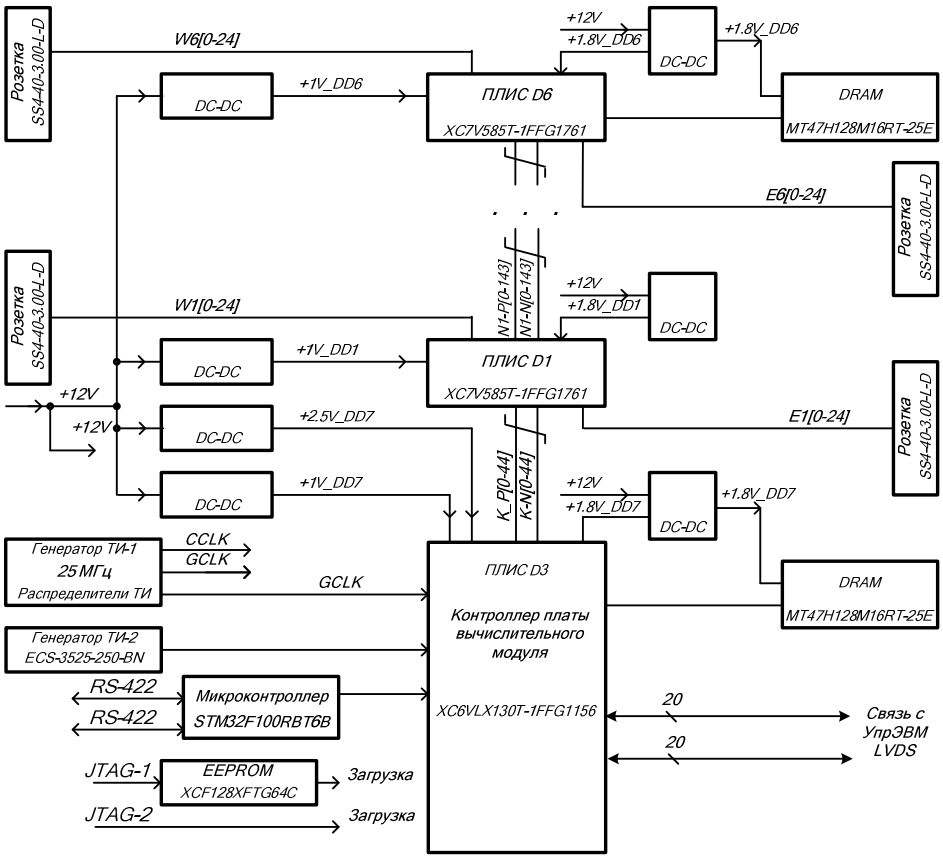


Рис. 3. Структурная схема ПВМ 6V7-180

Технические характеристики ПВМ 6V7-180 представлены в таблице 1, в таблице 2 приведены технические характеристики ВМ 24V7-750.

Таблица 1. Технические характеристики ПВМ 6V7-180

Технический параметр		Значение
Число ПЛИС XC7V585T-FFG1761 (вычислительная ПЛИС) (58 млн. экв. вент.), шт.		6
Число ПЛИС XC6V130T-FFG1156 (контроллер ПВМ) (13 млн. экв. вент.), шт.		1
Число м/с памяти DDR2 MT47H128M16HR-25E (128 М * 16 = 2048 Мбит), шт.		12
Объем памяти, Гбайт		3
Частота обработки информации ПЛИС, МГц		до 400
Тактовая частота каналов между соседними ПЛИС, МГц (не менее)		800
Производительность вычислений, приведённых операций в секунду		$1{,}05 \cdot 10^{13}$ оп/с
Интерфейсы	Каналы LVDS для связи с управляющей ЭВМ, дифф. пар	20
	Разъемы SS4, шт.	12
	Каналы LVDS для обмена со смежными ПЛИС, дифф. пар	144
Потребляемая мощность, не более, Вт		300
Габариты ПВМ, мм		140 х 325

Таблица 2. Технические характеристики ВМ 24V7-750

Технический параметр	Значение
Число вычислительных модулей в 19 " стойке	24-36
Производительность вычислительного модуля Pi_{32}/Pi_{64} (Гфлопс)	2600/820
Производительность при решении задач символьной обработки данных (Топ/с)	42
Производительность при решении задач математической физики, арифметики с плавающей запятой (Тфлопс)	2,2/0,8
Скорость передачи данных с блоками распределенной памяти (Гбит/с)	16,4
Скорость передачи данных между ПЛИС вычислительного поля (Тбит/с)	2,0
Скорость передачи данных с другими вычислительными модулями (Тбит/с)	0,5

Таким образом, производительность РВС-7 при комплектации от 24 до 36 ВМ 24V7-750 составит от 62 до 93,0 Тфлопс при обработке 32-разрядных данных с плавающей запятой и 19,4 – 29,4 ТФлопс при обработке 64-разрядных данных с плавающей запятой.

Применение ПЛИС семейства Virtex-7 в качестве элементной базы для ВМ 24V7-750 позволяет при сохранении стоимости поставки вычислительного модуля увеличить производительность в 1,7 раза по сравнению с аналогичным решением на основе ПЛИС семейства Virtex-6 [5]. Этот факт позволяет рассматривать созданные вычислительные модули нового поколения как наиболее перспективные варианты для построения на основе РВС-7 высокопроизводительных вычислительных комплексов различных архитектур и конфигураций и обеспечивает им существенное конкурентное преимущество по большинству технико-экономических параметров: удельной производительности, энергоэффективности и др.

3. Комплекс программного обеспечения РВС-7

Для создаваемой вычислительной системы РВС-7 сохраняется преемственность принципов программирования: программирование всех рассмотренных вычислительных модулей осуществляется с помощью единого комплекса системного программного обеспечения, поддерживающего структурно-процедурные методы организации вычислений и определяющие не только организацию параллельных процессов и потоков данных, но и структуру вычислительной си-

стемы в поле логических ячеек ПЛИС. Наиболее характерной отличительной особенностью создаваемого комплекса программного обеспечения РВС-7 является поддержка проблемно-ориентированных софт-архитектур, позволяющих создавать и программировать макрообъекты, представляющие собой совокупность вычислительных устройств, выполняющих определенную группу команд и соединенных между собой коммутационной системой. Это обеспечивает при тех же принципах программирования возможность простой адаптации программных компонентов средств разработки для РВС при переходе на новые топологии ПВМ без внесения существенных изменений в код программных компонентов комплекса, а также позволяет сократить время решения прикладных задач.

Для поддержки проблемно-ориентированных софт-архитектур разрабатывается комплекс программного обеспечения (КПО) РВС-7, включающий новые программы-синтезаторы параллельно-конвейерных вычислительных структур из макрообъектов и обеспечивающий поддержку вводимых расширений всеми средствами разработки прикладных программ на всех необходимых для этого уровнях.

Структура разрабатываемого комплекса программного обеспечения РВС-7 представлена рис.4.

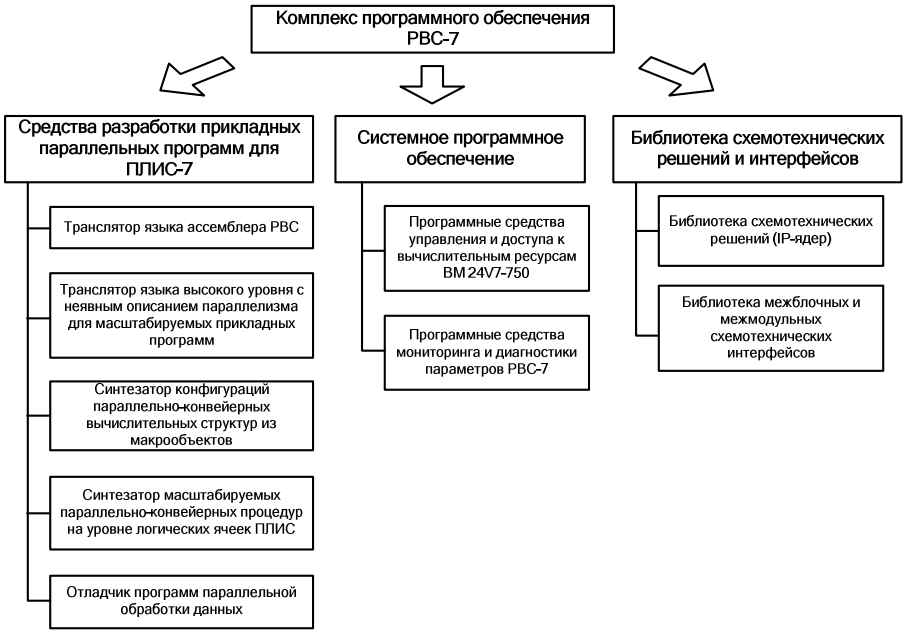


Рис. 4. Структура КПО РВС-7

КПО РВС-7 будет обеспечивать:

- рациональную реализацию прикладных задача различных областей на произвольном количестве взаимосвязанных кристаллов ПЛИС в составе вычислительных блоков ВБР-7 и вычислительных модулей (ВМ) 24V7-750 для любых допустимых конфигураций РВС-7;
- разработку прикладных масштабируемых программ на языке высокого уровня с вызовом библиотечных функций, которые будут настраивать архитектуру системы и реализовывать необходимые вычислительные структуры на множестве ПЛИС;
- тестирование и контроль эксплуатационных параметров составных частей РВС-7;
- управление и администрирование оборудования, в том числе удаленное, включение, выключение, остановку и запуск как отдельных ВМ, так и стоек РВС-7.

Языковые средства программирования прикладных задач должны содержать:

- транслятор языка ассемблера для программирования на уровнях унифицированного макрообъекта и структурно- и процедурно-программируемого макрообъекта;
- транслятор языка программирования РВС высокого уровня с неявным описанием параллелизма для трансляции в логические ячейки ПЛИС и связи между ними;

- среду разработки прикладных программ, поддерживающую языки ассемблера и высокого уровня для РВС;

- среду синтеза масштабируемых параллельно-конвейерных процедур для трансляции структурной составляющей с языка высокого уровня в конфигурацию ПЛИС;

- библиотеку функционально-законченных структурно-реализованных аппаратных устройств (IP-ядер) для различных предметных областей и интерфейсов для согласования скорости обработки информации и связи в единую вычислительную структуру;

- библиотеку программных функций доступа к аппаратным ресурсам базовых модулей РВС для программирования на уровне использования функционально законченных фрагментов задачи.

Разрабатываемый комплекс программного обеспечения позволит создавать эффективные прикладные программы для РВС при решении задач различных предметных областей, обеспечивая удобство программирования и сокращая время разработки прикладного решения.

4. Заключение

Проведенные исследования производительности созданных аппаратных средств РВС-7 – ВМ 24V7-750 при решении прикладных задач многоканальной цифровой фильтрации показывают, что реальная производительность ВМ 24V7-750 составляет 25 Топ/с, что позволяет достичь реальной производительности в $0{,}6 \cdot 10^{15}$ оп/с при решении прикладных задач на РВС-7.

Таким образом, конструктивные решения, положенные в основу создаваемой РВС-7, позволяют сосредоточить в пределах одной вычислительной стойки высотой 47U мощный вычислительный ресурс на основе ПЛИС, обеспечивает удельную производительность РВС-7 на уровне лучших мировых показателей для суперЭВМ с кластерной архитектурой. Поскольку РВС по сравнению с кластерными ЭВМ обладают до 10 раз превосходящей удельной производительностью на широком классе задач, можно сделать вывод о том, что РВС-7 может являться основой для создания высокопроизводительных вычислительных комплексов нового поколения, обеспечивающих высокую эффективность вычислений и близкий к линейному рост производительности при наращивании вычислительного ресурса.

Литература

1. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. М.: Янус-К, 2003. 380 с.
2. 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.
3. 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.
4. Дордопуло А.И., Каляев И.А., Левин И.И., Семерников Е.А. Высокопроизводительные реконфигурируемые вычислительные системы нового поколения // Труды Международной суперкомпьютерной конференции с элементами научной школы для молодежи «Научный сервис в сети Интернет: экзафлопсное будущее». М.: Изд-во МГУ, 2011. С. 42-49.
5. Каляев И.А., Левин И.И., Семерников Е.А., Дордопуло А.И. Реконфигурируемые вычислительные системы на основе ПЛИС семейства Virtex-6 // Сборник трудов Международной научной конференции «Параллельные вычислительные технологии 2011» (ПАВТ 2011). Челябинск-М.: Издательский центр ЮУрГУ [Электронный ресурс], 2011. С. 203–210.

Определение физико-химических параметров процесса анодного растворения железа в кислых средах с использованием технологий параллельных вычислений

М.А. Малеева[1], А.И. Маршаков[1], А.Р. Еникеев[2], И.М. Губайдуллин[2]

Институт физической химии и электрохимии РАН [1],
Институт нефтехимии и катализа РАН[2]

Рассмотрена математическая модель активного растворения железа в сульфатном электролите. Разработаны методы поиска кинетических параметров процесса на основе технологии параллельных вычислений. С использованием реализованных методов произведен вычислительный эксперимент и решена задача поиска кинетических параметров реакции анодного растворения железа в сульфатном электролите.

1. Введение

Изучение закономерностей анодного растворения железа в водных электролитах, является одной из задач исследования коррозионных электрохимических процессов [1]. Это обусловлено тем, что в природных электролитах коррозия стальных конструкций протекает, как правило, при электродных потенциалах активного растворения металла. Поэтому изучение кинетики растворения железа представляет практический интерес.

2. Постановка задачи

В настоящее время изучение механизмов сложных химических реакции остается актуальной проблемой современной химии. Установление механизмов химических процессов является сложной физико-химической задачей, решение которой опирается на основные положения теоретической химической кинетики и экспериментальные исследования.

Основные задачи настоящей работы: рассмотреть математическое описание процесса коррозии и анодного растворения железа в виде систем алгебраических уравнений, выражающих основные законы сохранения; провести расчет передаточной функции системы “металл-электролит” для определения кинетических констант последовательных и параллельных стадий анодного растворения железа. Задача сводится к решению обратной задачи химической кинетики, связанной с минимизацией целевой функции.

2.1 Математическая модель процесса анодного растворения железа в сульфатном электролите

Рассмотрим модель процесса анодного растворения железа (Рис. 1). Данная модель предполагает наличие трех адсорбированных интермедиатов и описывается совокупностью кинетических уравнений, где V_i и k_i – скорость и константа скорости i - й стадии.

$$V_1 = k_1(1-\theta_1-\theta_2-\theta_3),$$
$$V_2 = k_2\theta_2,$$
$$V_3 = k_{31}\theta_1 - k_{32}\theta_2,$$
$$V_4 = k_4\theta_2,$$
$$V_5 = k_{51}\theta_1 - k_{52}\theta_3,$$
$$V_6 = k_6\theta_3;$$

Предполагается, что адсорбция-десорбция компонентов происходит по закону Ленгмюра, $\theta_1 = \tilde{A}_1 / \tilde{A}_{\max}$ обозначает степень заполнения поверхности частицами типа Fe(I), степень запол-

нения $\theta_2 = \tilde{A}_2 / \tilde{A}_{\max}$ - частицами Fe*(I), а $\theta_3 = \tilde{A}_3 / \tilde{A}_{\max}$ - частицами Fe*(II). Принимается, что $\tilde{A}_{\max}$ - предельная поверхностная концентрация ($\theta_j = 1$) одинаковая для всех типов адсорбированных частиц. При этом скорость элементарных стадий экспоненциально зависит от потенциала электрода, причем знак «+» соответствует скорости j –той стадии в анодном направлении.

$$k_{j1} = k_{j1}^0 \, \mathrm{e}^{(n_j b_j E)}, j = 1,2...6;$$

где $b_j = \dfrac{\beta_j F}{RT}$, n_j – число электронов, переносимых на стадии j, β_j - коэффициент переноса этой стадии.

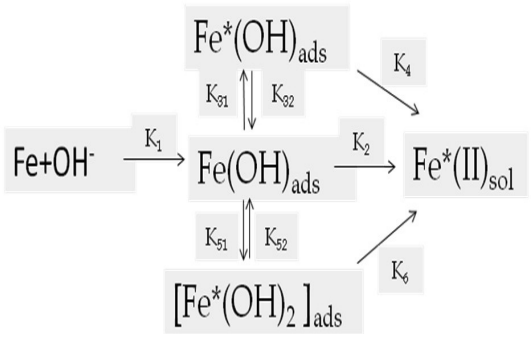


Рис. 1. Модель растворения железа

Экспериментальные исследования проводились с помощью метода электрохимической импедансной спектроскопии. Это подход исследования двойного электрического слоя, механизма и кинетики процессов на границе электрод - электролит.

При классическом подходе, на основе принятой гипотезы о характере исследуемых процессов, составляется исходная система дифференциальных уравнений, полностью описывающая эти процессы. Но, в соответствии с гипотезой о линейности импедансного метода, эта исходная система нелинейных уравнений линеаризуется путем разложения каждого члена уравнений в ряд Тейлора с исключением всех членов разложения, кроме линейного. Далее эти зависимости подвергаются преобразованию Лапласа, которое переводит систему в операторную область. В соответствии с другой основной гипотезой о стационарности оператор Лапласа p заменяется комплексным оператором jω, в результате чего получается система уравнений в частотной области. Необходимо найти конечное решение этой системы уравнений, которое дало бы в явном аналитическом виде зависимость вещественной и мнимой составляющих импеданса от частоты[2]. Данную проблему позволяет решить метод направленных графов. На основе работ[3] находим выражение для адмиттанса кинетики в виде дробно-рационального выражения:

$$Y_k = a_0 + \frac{E_2 p^2 + E_4 p + E_6}{p^3 + T_2 p^2 + T_4 p + T_6}; \qquad (1)$$

где $p = i\omega$, ω-угловая частота, a_0 – величина адмиттанса при бесконечно большой частоте

Таким образом, полученное выражение адмиттанса кинетики анодного растворения железа позволяет рассчитать кинетические константы (константы скорости реакций и коэффициенты переноса) элементарных стадий этого процесса, исходя из набора экспериментальных спектров, $E_{2,4,6}$– коэффициенты знаменателя передаточной функции, $T_{2,4,6}$– коэффициенты числителя передаточной функции.

Ниже представлена подробная расшифровка величин, содержащихся в выражении (1):

$$E_2 = w_1 + w_2 + w_3 + w_5,$$
$$E_4 = w_1(w_3 + w_5) + w_2(w_{32} + w_{52}) + w_{52}w_3 + w_{51}w_{32},$$
$$E_6 = w_1(w_{52}w_3 + w_{51}w_{32}) + w_2w_{32}w_{52},$$
$$a_0 = F(n_1V_{1E} + n_2V_{2E} + n_3V_{3E} + n_4V_{4E} + n_5V_{5E} + n_6V_{6E}),$$
$$T_2 = F[(-n_1w_1 + n_2w_2 + n_3w_{31} + n_5w_{51}) \cdot (V_{1E} - V_{2E}) + V_{3E} \cdot (-n_2w_2 + n_4w_4 - n_3w_3 - n_5w_{51}) +$$
$$+V_{5E} \cdot (-n_2w_2 - n_3w_{31} - n_5w_5 + n_6w_6)],$$
$$T_4 = [[-n_1w_1(w_3 + w_5) + n_2w_2(w_{32} + w_{52}) + -w_{31}(n_3w_{52} + n_4w_4) + w_{51}(n_5w_{32} + n_6w_6)] \cdot$$
$$\cdot(V_{1E} - V_{2E}) + V_{3E}[-w_2[(n_1 + n_2)w_1 + n_2w_{52}] - n_3[w_3(w_1 + w_{52}) + w_{32}(w_2 + w_{51})] +$$
$$+n_4w_4(w_1 + w_2 + w_5) - n_5w_1w_{51} - n_6w_6w_{51}] + V_{5E}[-w_2[(n_1 + n_2)w_1 + n_2w_{32}] - n_5[w_5(w_1 + w_{32}) + \quad (2)$$
$$+w_{52}(w_2 + w_{31})] + n_6w_6(w_1 + w_2 + w_3) - n_3w_1w_{31} - n_4w_4w_{31}]],$$
$$T_6 = F[[-n_1w_1(w_{32}w_5 + w_{31}w_{52}) + n_2w_2w_{32}w_{52} + n_4w_4w_{31}w_{52} + n_6w_6w_{51}w_{32}](V_{1E} - V_{2E}) +$$
$$[V_{3E}[-(n_1 + n_2)w_1w_{52}w_{52} - n_3E_6 + n_4w_4(w_1w_5 + w_2w_5) - n_6w_6w_1w_{51})] +$$
$$+V_{5E}[-(n_1 + n_2)w_1w_{32}w_2 - n_5E_6 + n_6w_6(w_1w_3 + w_2w_{32}) - n_4w_4w_1w_{31}]],$$
$$i_a = F\tilde{A}_{1ss}\left[(n_1 + n_2)w_2 + n_4(w_4K_2) + n_6(w_6K_3)\right],$$
$$w_i = w_{i0}e^{b_iE}.$$

. где w_i - константы скорости i стадии, b_i - Тафелевские коэффициенты i стадии, F - постоянная Фарадея, (Кл·моль)$^{-1}$, i_a – стационарный ток (скорость реакции при данном потенциале E).

2.2 Связь параметров эквивалентной схемы и метода импедансной спектроскопии

Для количественного описания экспериментальных годографов использовали эквивалентную схему (**Рис. 2)** , которая содержит элементы, моделирующие сопротивления раствора (R_s), емкость электрода (элемент постоянной фазы CPE) и трехмаршрутный процесс растворения металла (R_t и совокупность цепочек R и L).

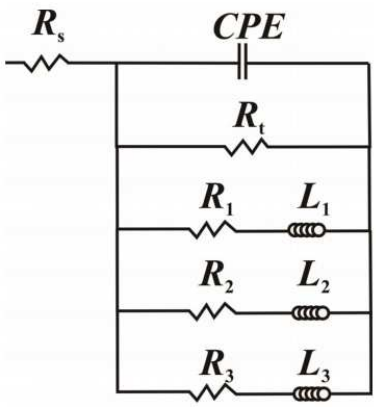


Рис. 2. Модель растворения железа

Импеданс такой схемы рассматривается в виде

$$Z = R_s + 1/(C_p + Y_F);$$	(3)

, где фарадеевский адмиттанс эквивалентной схемы имеет вид:

$$Y_F = \frac{1}{R_t} + \frac{1}{R_1 + j\omega L_1} + \frac{1}{R_2 + j\omega L_2} + \frac{1}{R_3 + i\omega L_3};$$	(4)

Экспериментальные значения элементов эквивалентной схемы использовали при расчете кинетических констант элементарных стадий активного растворения железа, полагая, что реак-

ционную схему этого процесса можно отождествить со схемой на рис. 1. Существует связь между коэффициентами уравнения (1) и параметрами электрической эквивалентной схемы:

$$
\begin{aligned}
a_0 &= \frac{1}{R_t}, \\
E_2 &= \frac{1}{L_1} + \frac{1}{L_2} + \frac{1}{L_3}, \\
E_4 &= \frac{R_1 + R_2}{L_1 L_2} + \frac{R_1 + R_3}{L_1 L_3} + \frac{R_3 + R_2}{L_2 L_3}, \\
E_6 &= \frac{R_1 R_2 + R_1 R_3 + R_3 R_2}{L_1 L_2 L_3}, \\
B_2 &= \frac{R_1}{L_1} + \frac{R_2}{L_2} + \frac{R_3}{L_3}, \\
B_4 &= \frac{R_1 R_2}{L_1 L_2} + \frac{R_2}{L_1 L_3} + \frac{R_3}{L_3 L_2}, \\
B_6 &= \frac{R_1 R_2 R_3}{L_1 L_2 L_3}.
\end{aligned}
\qquad (5)
$$

3 Алгоритм

Решение обратной задачи поиска кинетических параметров реакции анодного растворения железа сводится к нахождению решения системы уравнений (2). В нашем случае в системе (3) 8 уравнений и 16 неизвестных (w_i и b_i) и следовательно она имеет бесконечное число решений. Для решения данной системы уравнений был выбран генетический алгоритм [4].

3.1 Генетический алгоритм

Генетические алгоритмы оперируют совокупностью особей (популяцией), которые представляют собой строки, кодирующие одно из решений задачи. Этим ГА отличается от большинства других алгоритмов оптимизации, которые оперируют лишь с одним решением, улучшая его. Генетический алгоритм успешно применяется в расчете обратных задач химической кинетики [5-6]. Поэтому в качестве метода был выбран именно он. Была использована схема ГА со следующими генетическими операциями: скрещиванием, мутациями, кроссинговером мутацией, так же в наборах использовался элитизм (сохранение части лучших особей).

Вычисления происходят следующим образом. Сначала находим параметры уравнения (3), в качестве критерия оптимизации рассматриваем условие минимизации расчетных и экспериментальных значений импеданса Z.

1) Генерация начальной популяции. Случайно создаются наборы (R_t, R_1, R_2, R_3, L_1, L_2, L_3) для поиска значений элементов эквивалентной схемы.

2) Для выбора родительской пары был использован элитный отбор, то есть используем K особей с минимальными значениями функции минимизации F и составляем из них все возможные пары $\frac{K*(K-1)}{2}$ наборов решений.

3) Кроссинговер и мутация. Скрещиваем хромосомы(решения) «одного вида» со случайными коэффициентами, но так, чтобы не выйти за рамки ограничений наложенные на параметры. Вероятность мутации полагаем 3%.

4) Полученные особи-потомки добавляются в популяцию после переоценки. Новую особь добавляем взамен самой плохой старой особи, при условии, что значение функции на новой особи меньше значения функции на старой особи. Если самое лучшее решение в популяции нас не удовлетворяет, то снова переходим на шаг 2.

Далее по этой же схеме рассчитываем значения констант скоростей реакции, только в качестве критерия оптимизации рассматривается условие минимизации экспериментальных и расчётных данных ($E_2, E_4, E_6, T_2, T_4, T_6, i_0, i_a$), а в качестве начальной популяции рассматриваются наборы w_i и b_i.

3.2 Параллельная реализация генетического алгоритма. Результаты.

Возможность распараллеливания алгоритма основа на том, что вычисление функции соответствия, включающие вычисления параметров эквивалентной схемы и кинетических параметров системы может быть вычислена независимо для особей в популяции на каждом шаге эволюционного процесса. Решение задачи строится по схеме «мастер-рабочий». За составление начальной популяции отвечает «мастер», а за выполнение генетических операций и поиск в новом поколении наиболее жизнеспособных особей – рабочие. Каждый рабочий по истечении лимита итераций передаёт информацию о найденной наиболее жизнеспособной особи мастеру. Мастер анализирует все полученные особи и выбирает среди них наиболее приспособленные.

Также рассматривался процесс растворения железа в сульфатном электролите при различных значениях потенциала E. Значит, кроме распараллеливания метода решения системы, можно использовать параллелизм по экспериментальной базе. В этом случае по схеме «мастер-подчиненные» один процессор распределяет эксперименты по отдельным узлам. Потом на каждом узле ПГА «вычисляет параметры популяции», пока не будет найдено решение, после чего на узел поступает следующий эксперимент.

Вычислительные эксперименты проводились на кластере БашГУ (32 процессора AMD Opteron). Рассмотрим оценку эффективности распараллеливания представленной схемы решения (рис. 3).

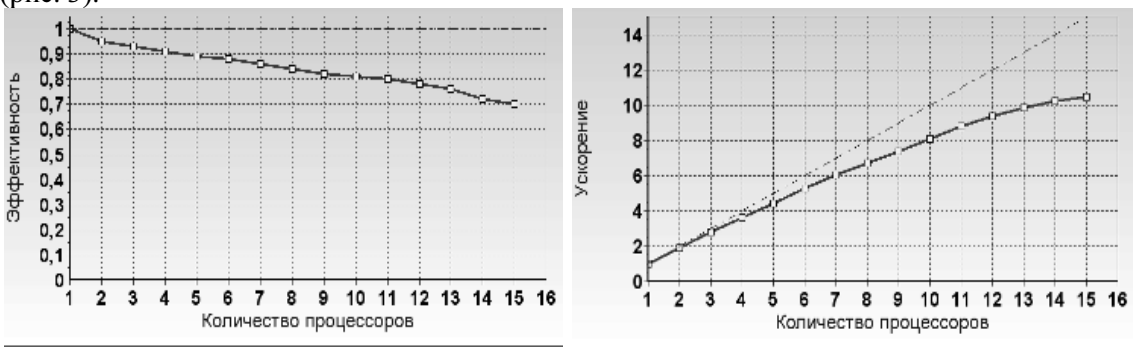


Рис. 3. Эффективность и ускорение работы программы

Как видно из графиков, параллельный алгоритм работает быстрее последовательного варианта. Показатель эффективности выше 0.7, что говорит о хорошей параллельности программы.

Анализ полученных зависимостей показывает, что для расчета кинетических параметров анодного растворения железа наиболее эффективным является использование 12-15 процессоров суперкомпьютера.

4.Результаты расчетов

Измерения импеданса проводили в кислом (pH=1.3) сульфатном растворе в области потенциалов активного растворения железа -0.26 / -0.18 В, что соответствовало плотностям тока в интервале 10^{-3} – $7*10^{-3}$. Методическая часть, использованные реактивы и основные результаты экспериментальной части работы рассмотрены в [7].

Рассчитанные величины элементов, моделирующих фарадеевский процесс растворения железа, приведены в табл. 1. Несмотря на некоторый разброс данных, в целом, они монотонно уменьшаются со сдвигом потенциала в положительную сторону.

Используя результаты вычислений параметров эквивалентной схемы (таблица 1), были найдены кинетические характеристики элементарных стадий растворения железа, которые представлены в табл. 2. Значение целевой функции для представленных данных F= 0.874.

Таблица 1. Средние значения элементов в эквивалентной схеме.

E, В	R_t, Ом*см^2	R_1, Ом*см^2	L_1, Гн*см^2	R_2, Ом*см^2	L_2, Гн*см^2	R_3, Ом*см^2	L_3, Гн*см^2
-0.26	26.82	117.21	5.89	223.78	63.67	75.495	139.8
-0.25	23.5	99.3	2.9	93.3	19.3	102.5	237
-0.24	24.3	90.77	2.7	99.62	18.88	103.2	303.1
-0.23	11.47	46.67	0.72	86	7.5	82.34	284.7
-0.22	8.9	22.9	0.3	28.4	2.1	59.55	156.2
-0.21	7.4	17.88	0.1	20.88	2.1	59.5	156.2
-0.2	3.5	11.6	0.06	12.5	0.67	17	65

Таблица 2. Значения тафелевских коэффициентов b_j скорости элементарных стадий w_j процесса анодного растворения железа в кислом сульфатном растворе (pH 1.3).

Номер стадии	b_j,В-1	W_j 1/сек
1	36.3794	50582.1
2	14.246	1593.7
31	33.5923	377.808
32	4.80768	0.251
4	30.71	42090.1
51	19.6109	47.0205
52	14.6842	0.99227
6	22.2	100988

Как известно, из способов проверки на адекватность модели, является сравнение рассчитанных и экспериментальных характеристических частот. Если предположить, что в процессе изменения количества одного из адсорбатов, величина остальных не изменяется, можно получить значения характеристических частот:

$$W_1^{\text{ðàñ÷}} = w_1 + w_2 + w_{31} + w_{51}$$
$$W_2^{\text{ðàñ÷}} = w_{32}$$
$$W_3^{\text{ðàñ÷}} = w_{52}$$

Сравнение полученных экспериментальных характеристических частот с рассчитанными представлено в табл. 3. При данном допущении значения характеристических частот в целом удовлетворяют эксперименту.

Таблица 3. Рассчитанные и экспериментальные характеристические частоты E=-0.1926.

	Эксперимент	Расчёт
$W_1^{\text{ðàñ÷}}$	135.01	148.24
$W_2^{\text{ðàñ÷}}$	14.07	13.34
$W_3^{\text{ðàñ÷}}$	0.944	0.911

Однако, как показала практика, найденные кинетические параметры с удовлетворительной точностью описывают поведение импеданса системы, в целом моделируя его поведение. Это, возможно, связано с выбранной моделью расчёта и с большой областью поиска решения.

В результате исследовательской работы найдены кинетических констант последовательных и параллельных стадий анодного растворения железа в сульфатном растворе. В дальнейшем, планируется использование других эвристических алгоритмов для поиска решений с целью сглаживания годографа, получившегося в результате вычислительного эксперимента.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 12-07-00324).

Литература

1. Флорианович Г.М. Механизм активного растворения металлов группы железа. // Итоги науки и техники. Сер. "Коррозия и защита от коррозии". –М.:ВИНИТИ, 1978. Т.6. – 136.
2. Mason S.J. Feedback theory - some properties of signal flowgraphs. // Proceedings of the IRE. – 1953. - V. 41. - № 9. - P. 1144-1156.
3. Keddam M., Mattos O.R., Takenouti H. Reaction model for iron dissolution studied by electrode impedance. I. Experimental results and reaction model. // J. Electrochem. Soc. – 1981. - V. 128. - № 2, - P. 257-266
4. Васильев Ф.П. Численные методы решения экстремальных задач. М.: Наука. Гл. ред. физ.-мат. лит. – 1988. – 552 с.
5. Губайдуллин И.М., Линд Ю.Б., Коледина К.Ф. Методология распараллеливания при решении многопараметрических задач химической кинетики // Вычислительные методы и программирование. 2012. Т. 13, №1. С. 236-244
6. Губайдуллин И. М., Коледина К. Ф., Линд Ю. Б. Современные технологии высокопроизводительных вычислений при моделировании детального механизма реакции каталитического гидроалюминирования олефинов // Наука и образование. - № 6, июнь 2011 г. http://technomag.edu.ru/doc/187631.html
7. Малеева М.А., Рыбкина А.А., Маршаков А.И., Ёлкин В.В. Изучение влияния атомарного водорода на анодное растворение железа в сернокислом электролите методом импедансной спектроскопии. \\ Физикохимия поверхности и защита материалов. – 2008. - № 6. - С. 587-595.

Моделирование образования протопланет в околозвездном диске на суперкомпьютерах с распределенной памятью*

Т.В. Маркелова, В.Н. Снытников

Институт катализа им. Г.К. Борескова СО РАН

Разработан параллельный код для решения одной из актуальных задач астрофизики - образования крупных тел в массивном газо-пылевом околозвездном диске. Проведены численные эксперименты, показывающие динамику роста твердых тел, а также изменение их тепловой энергии во времени. Изучено влияние изменения спектра масс на возникновение гравитационной неустойчивости в диске. Применение суперкомпьютера позволило существенно увеличить численное разрешение.

1. Введение

За последние 20 лет найдено около 550 звезд с планетами и имеется еще 1790 кандидатов в подобные звезды. На сегодняшний день поиски таких звезд продолжаются. Хотя и существуют несколько конкурирующих гипотез, ответ на вопрос, каким образом сформировались твердые планеты, остается актуальным.

Для отдельной планетарной системы мы наблюдательно видим лишь краткое мгновение в, примерно, 100 миллионлетней общей истории зарождения. Поэтому воспроизвести основные этапы формирования звезд с планетами можно методами математического моделирования.

Энеевым Т.М. и Козловым Н.Н. была предложена “капельная” модель образования планет [1, 2]. В основу модели положены упрощающие предположения: масса тел много меньше массы центрального тела и рассматривались только парные столкновения сгустков из тел, движущихся по одной траектории. Эти сгустки представляют собой эффективные тела. Радиус эффективного тела значительно больше, чем радиус физического отдельного тела.

Модель Энеева хорошо подходит для роя первичных тел с зародышами планет, т.е. для последней стадии эволюции протопланетного диска. Для расчетов массивного диска в модели Энеева-Козлова необходимо дополнительно учитывать динамику газа. В этом случае могут изучаться гравитационная неустойчивость и другие коллективные процессы, влияющие на дальнейшую эволюцию диска, а также рассчитываться величины тепловой энергии новых тел. На начальном этапе создания модели мы пренебрегли моментами вращения твердых тел. Однако в столкновениях проводится расчет величины тепловой энергии тел.

2. Математическая постановка задачи

Эволюцию газо-пылевого протопланетного диска можно описать с помощью системы уравнений, приведенной ниже.

$$\frac{\partial f}{\partial t} + \vec{u}\frac{\partial f}{\partial \vec{x}} + \vec{a}\frac{\partial f}{\partial \vec{v}} = St(f),$$
$$\vec{a} = -\nabla\Phi, \quad \sigma_{par} = \int f d\vec{u} dz. \qquad (1)$$

Где *f(t,r,u,m)* — одночастичная функция распределения по координатам *r*, скоростям *u* и массам *m*, а *St(f)* — член, через который учитываются неупругие столкновения.

*Работа была поддержана интеграционным проектом СО РАН (координатор ак. Михайленко Б.Г.), программами Президиума РАН (№ 21, 22, 28).

$$\begin{cases} \dfrac{\partial \sigma}{\partial t} + div(\sigma \vec{v}) = 0, \\ \sigma \dfrac{\partial \vec{v}}{\partial t} + \sigma(\vec{v}, \nabla)\vec{v} = -\nabla p^* - \sigma \nabla \Phi, \\ p^* = C\sigma_{gas}^{\gamma}, \quad \sigma_{gas} = \int_{-\infty}^{+\infty} \rho_{gas} dz; \end{cases} \tag{2}$$

$$\Phi = \Phi_1 + \Phi_2, \Phi_1 = -\frac{M_c}{r}, \quad \Delta\Phi_2 = 0, \Phi_2 \xrightarrow[r\to\infty]{} 0, \tag{3}$$

Где p^* — поверхностное давление газа, давление является функцией от плотности газа, v – скорость газа,σ — поверхностная плотность газа, γ – показатель адиабаты, Φ — суммарный гравитационный потенциал центрального тела и газо-пылевого диска.

Уравнение Больцмана (1) описывает движение частиц пыли в диске с учетом парных неупругих столкновений. Для моделирования газовой динамики взята система (2), описывающая динамику газового диска в экваториальной плоскости. Уравнение (3) описывает самосогласованное гравитационное поле газа, твердой фазы и центрального тела.

Система уравнений была решена методом расщепления по физическим процессам, причем первое уравнение (1) расщепляется на два: уравнение Власова и уравнение Смолуховского (4). Бесстолкновительная модель, содержащая только уравнение Власова и уравнения (2)-(3), подробно описана в работе [3]. Уравнение Смолуховского (4) описывает процесс слияния частиц.

$$\frac{df(m_1,t)}{dt} = \frac{1}{2}\int_0^{m_1} \Phi(m_1 - m, m) f(m_1 - m, t) dm - f(m_1, t) \int_0^{\infty} \Phi(m_1, m) f(m, t) dm, \tag{4}$$

где f — функция распределения частиц, $f(m,t)dm$ — средняя концентрация частиц физической системы, массы которых в момент времени t лежат в интервале $(m,m+dm)$. Ядро $\Phi(m_1,m_2)$ уравнения Смолуховского считается известной функцией слияния частиц с массами m_1 и m_2, а ее численное значение пропорционально частоте слияния таких частиц в единице объема системы, то есть величине, обратной среднему времени жизни частиц с указанными массами. Конкретный вид ядра Φ получается на основании анализа явлений, обуславливающих взаимодействие частиц моделируемой физической системы.

Начальное распределение плотности частиц и газа отвечает модели твердотельного вращения:

$$\sigma_{par,gas}(r) = \begin{cases} \dfrac{3M_{par,gas}}{2\pi R^2}\sqrt{1 - \left(\dfrac{r}{R}\right)^2}, r < R, \\ 0, \quad r \geq R. \end{cases}$$

Начальные скорости частиц определяются максвелловским распределением. Объемная плотность среды вне бесконечно тонкого диска равна нулю. На самом диске происходит разрыв нормальной производной потенциала, который позволяет получить граничное условие:

$$\frac{v_\varphi^2}{r} = \frac{1}{\rho(r)}\frac{\partial p}{\partial r} + \frac{\partial \Phi}{\partial r}, \quad v_r = 0,$$

$$\frac{u_\varphi'^2}{r} = \frac{\partial \Phi}{\partial r}, u_r' = 0. \ \vec{u} = \vec{u}' + \vec{u}'', \text{ где } \vec{u}' \text{ - регулярная скорость частиц, } \vec{u}'' \text{ - хаотическая.}$$

$$\Phi_2(r) = -\frac{M_{par} + M_{gas}}{r} - \frac{1}{r^3}\left(I_x + I_y + I_z - 3I_0\right)$$

$$\left.\frac{\partial \Phi_2}{\partial z}\right|_{z=0} = 2\pi(\sigma_{par} + \sigma_{gas})$$

Чтобы получить безразмерные переменные, в качестве основных характерных параметров выбраны диаметр диска $R_0 = 10^{10}$ км, масса протозвезды $M_0 = 2*10^{30}$ кг, гравитационная постоянная $G = 6.672*10^{-11}$ Н*м^2/кг^2. Соответствующие характерные величины скорости частиц (V_0), времени (t_0), потенциала (Φ_0) и поверхностной плотности (σ_0) записаны как $V_0 = \sqrt{GM_0 / R_0}$, $t_0 = R_0/V_0$, $\Phi_0 = V_0^2 = GM_0/R_0$, $\sigma_0 = M_0/R_0^2$.

В качестве вспомогательных параметров задаются радиус $r_{\max}$ и высота $z_{\max}$ расчетной области, имеющей форму цилиндра; радиус диска R, его масса, дисперсия радиального разброса частиц по скоростям (при задании функции нормального распределения по скоростям). Кроме того, задаются временной шаг τ, количество временных шагов, полное число модельных частиц, итерационный и другие “технические” параметры.

3. Численные методы

На каждом шаге по времени последовательно решаются все уравнения системы. Уравнения Власова и газовой динамики решаются в полярных координатах. Уравнение Пуассона (3) — в цилиндрической области. Для моделирования слияний частиц введена отдельная расчетная сетка в декартовых координатах. Для решения кинетического уравнения Власова используется метод частиц в ячейках. В начальный момент времени модельные частицы одинаковой массы размещаются в области решения так, чтобы их количество было пропорционально плотности в ней и ее размеру. Частицы имеют скорость, равную скорости вещества в соответствующей точке. Для решения газодинамических уравнений использован метод крупных частиц Белоцерковского-Давыдова. Этот метод наиболее хорошо согласуется с методом частиц для решения уравнения Власова-Лиувилля и позволяет отслеживать границы газ-вакуум. Для решения уравнения Пуассона используется комбинированная схема с преобразованием Фурье по углу, последовательной верхней релаксацией по радиусу и прогонками по z. Применяемые методы описаны в работе [3]. Уравнение Смолуховского решалось методом прямого моделирования [5]. По модели неупругих столкновений могут слипаться частицы, лежащие в одной ячейке расчетной области. Размер ячейки для неупругих столкновений является входным параметром модели.

4. Параллельная реализация программы

При тестировании метода прямого моделирования столкновений установлено, что для погрешности вычислений менее 2 %, необходимо более 1000 частиц в ячейке. Для сетки 500x512 в плоскости диска для расчетов на персональной ЭВМ максимальное число частиц составляет примерно 10^7. При таких параметрах в одной ячейке находится около 800 частиц, что недостаточно для расчета неупругих столкновений. Чтобы увеличить число частиц, а так же сократить время расчета, необходимо использовать суперкомпьютеры.

Так как доля машинного времени, которая приходится на расчет газовой компоненты, невелика (около 1%), он выполняется на каждом из процессоров. Параллельная реализация решения уравнения Пуассона (3) осуществляется через распределение по процессорам гармоник потенциала, полученных в результате дискретного преобразования Фурье. Основная проблема, создать эффективный параллельный алгоритм для расчета столкновений частиц. В каждой ячейке расчет производится автономно, но для этого все частицы ячейки должны находиться на одном процессоре. Поэтому для решения уравнения Власова и уравнения Смолуховского (4) применяется эйлерова декомпозиция области. После вычисления координат частиц требуется пересылка массивов плотности, а также обмен частицами, которые переместились в другие ячейки, между процессорами. На каждом шаге необходимо определять пересылаемые частицы. Для этого разработан алгоритм неполной сортировки массивов [4].

Алгоритмы были реализованы с использованием библиотеки MPI для ЭВМ с распределенной памятью. Расчеты проводились на базе Сибирского суперкомпьютерного центра СО РАН и на кластерах ИК СО РАН.

5. Численные эксперименты

Для тестовых расчетов был выбран такой набор параметров, при которых в диске образуются сгущения в виде концентрических колец плотности. Были проведены тестовые расчеты с учетом неупругих столкновений и без учета. Начальные данные приведены в таблице 1.

Таблица 1. Начальные безразмерные параметры для расчета

Шаг по времени	0.00025	Радиус расчетной области	4.0
Радиус диска частиц	2.0	Масса частиц	0.01
Радиальная дисперсия частиц	0.1	Радиус газового диска	2.0
Масса газа	0.5	Давление в центре	0.001
Показатель адиабаты	1	Коэффициент трения	0.1
Начальная масса звезды	1.0	Начальное число частиц	10^8
Сетка в цилиндрических координатах	500x512x500	Сетка для столкновений	100x100

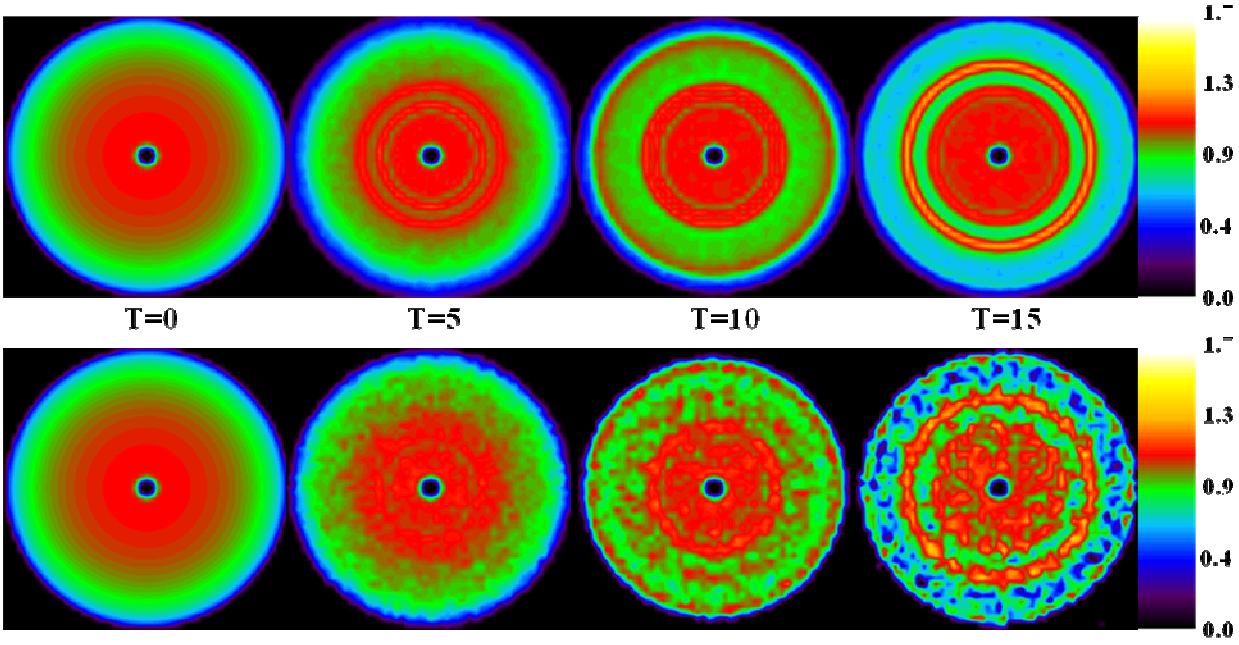


Рис. 1. Логарифм плотности частиц без учета (верхний ряд)
и с учетом неупругих столкновений (нижний ряд) в безразмерных единицах

В три момента времени через равные интервалы сравнивалась плотность пылевой компоненты (рис. 1), а также в определенные моменты времени сравнивалась плотность вдоль оси Ox. Время T=5 в безразмерных единицах соответствует одному обороту диска вокруг оси по начальному заданию расчетных параметров. Как видно из рис. 1, в момент времени T=5 в пылевом диске образовались два кольца плотности, которые в дальнейшем оставались практически на тех же местах. Для расчета динамики диска с учетом неупругих столкновений было выбрано ядро, независящее от массы частиц. При сравнении результатов этих двух расчетов видно, что появление более тяжелых частиц не повлияло на общую динамику пылевого диска. Новые частицы продолжают двигаться по близким к старым траекториям.

На рис. 2 представлена зависимость тепловой энергии частиц от массы в три момента времени. Каждая точка на графике изображает одну частицу. Из рисунка видно, что происходит

процесс укрупнения частиц и тела, имеющие максимальную массу, не приобретают высокой тепловой энергии на протяжении их формирования и роста. Однако в спектре масс всегда присутствуют тела, тепловая энергия которых в среднем выше в пять и более раз, чем у остальных частиц.

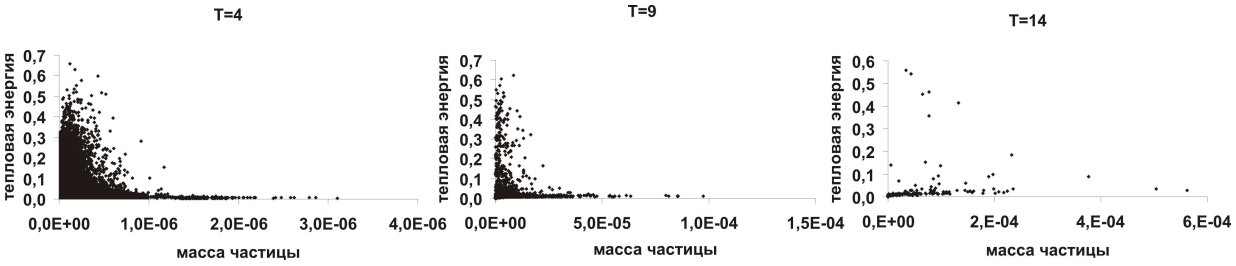


Рис. 2. Зависимость температуры частиц от их массы в три момента времени в безразмерных единицах

6. Заключение

Параллельная версия программы позволяет производить расчеты с точностью, определяемой числом частиц в ячейке, в основном, при более 10 000 частиц в одной ячейке сетки для столкновений. Программа будет использоваться в дальнейших расчетах физических процессов в массивном диске, а также для изучения влияния коллективных процессов на развитие гравитационной неустойчивости в протопланетных дисках. В первую очередь будет изучено влияние процесса коагуляции на развитие различных структур в диске, таких как сгустки и рукава плотности.

Литература

1. Энеев Т.М., Козлов Н.Н. Модель аккумуляционного процесса формирования планетных систем // Астрономический вестник. 1981. Т.XV, №2. С.80-94.

2. Ле-Захаров А.А., Кривцов А.М. Разработка алгоритмов расчета столкновительной динамики гравитирующих частиц ля моделирования образования системы Земля-Луна в результате гравитационного коллапса пылевого облака // Проблемы происхождения биосферы. Под ред. Галимова Е.М. 2012. УРСС. Москва, 613 с.

3. Снытников В.Н., Пармон В.Н., Вшивков В.А., Дудникова Г.И., Никитин С.А., Снытников А.В. Численное моделирование гравитационных систем многих тел с газом // Вычислительные технологии. 2002. Т.7, № 3. С. 72-84.

4. Вшивков В.А., Маркелова Т.В., Шелехов В.И. Об алгоритмах сортировки в методе частиц в ячейках // Научный вестник Новосибирского государственного технического университета. 2008. № 4. С. 79-92.

5. Галкин В.А. Уравнение Смолуховского. М: ФИЗМАТЛИТ. 2001. 336 с.

Сферическая блоковая модель: оптимизация вычислительной нагрузки и новые результаты моделирования*

Л.А. Мельникова, В.Л. Розенберг

Институт математики и механики УрО РАН, Екатеринбург

Представлено краткое описание последней версии сферической блоковой модели динамики и сейсмичности литосферы. В модификации определяется механизм модельного землетрясения и реализована попытка учета случайных факторов, влияющих на величины порогов прочности среды. Для проведения новых расчетов разработана методика динамического перераспределения вычислительной нагрузки между процессорами МВС в предположении о стационарности модельного потока событий. Обсуждаются результаты некоторых численных экспериментов с блоковой структурой, аппроксимирующей глобальную систему тектонических плит.

1. Последняя версия сферической блоковой модели

В настоящей работе представлены новые результаты, полученные в задаче численного моделирования динамики и сейсмичности литосферы `1, 2] посредством разработанной авторами сферической блоковой модели. Фактически работа является продолжением исследований [3–5], в которых, с одной стороны, достаточно подробно изложено обоснование актуальности моделирования сейсмичности по причине крайне короткой истории надежных инструментальных наблюдений, а с другой, приведен обзор различных подходов к изучению литосферных процессов с указанием соответствующих ссылок (отметим работу [1] и библиографию к ней). Необходимость получения искусственного каталога землетрясений, в котором каждое событие характеризуется моментом времени, координатами эпицентра, глубиной и магнитудой, на достаточно большом временном интервале объясняется возможностью анализа по такому каталогу статистической значимости некоторых закономерностей, предшествующих сильным толчкам в реальном сейсмическом потоке. Новизна данной работы состоит в описании дополнительных возможностей математической модели и программной реализации, а также результатов инспирированных ими вычислительных экспериментов. Приведем краткое описание последней версии сферической блоковой модели.

Блоковая структура в сферической модели является ограниченной и односвязной частью шарового слоя глубиной H, заключенного между двумя концентрическими сферами, одна из которых (внешняя) интерпретируется как поверхность Земли, другая (внутренняя) ─ как нижняя граница упругой литосферы. Разделение структуры на блоки определяется пересекающими этот слой бесконечно тонкими разломами, каждый из которых представляет собой коническую поверхность, наклоненную под определенным углом к внешней сфере. Общие точки двух разломов на внешней и внутренней сферах называются вершинами. Участки разломов, ограниченные соответствующими парами соседних вершин, называются сегментами. Пересечения блока с ограничивающими сферами представляют собой сферические многоугольники, при этом пересечение с нижней (для блока) сферой называется подошвой. Предполагается, что вне блоковой структуры могут находиться граничные блоки, примыкающие к внешним сегментам. Другая возможность состоит в рассмотрении блоковой структуры, замкнутой на сфере. Для учета неоднородности литосферы предусмотрена возможность задания различных глубин (в пределах H) для разных блоков и учета зависимости вязко-упругих свойств разлома от его глубины. Все блоки считаются абсолютно жесткими, а их смещения ─ бесконечно малыми по сравнению с линейными размерами, поэтому геометрия блоковой структуры не меняется в процессе моде-

* Работа выполнена в рамках программы Президиума РАН № 15 «Информационные, управляющие и интеллектуальные технологии и системы» при поддержке УрО РАН (проект 12-П-1-1023) и Программы государственной поддержки ведущих научных школ (проект НШ-6512.2012.1).

лирования, и структура не движется как единое целое. Блоки имеют шесть степеней свободы; смещение каждого блока состоит из поступательной и вращательной компонент. Предполагается, что законы движения граничных блоков и подстилающей среды известны, при этом движение описывается как вращение на сфере, т.е. задаются положение оси вращения и угловая скорость.

Жесткость блоков приводит к тому, что деформации имеют место только на разломах и подошвах блоков; силы возникают на подошвах из-за смещения блоков относительно подстилающей среды и на поверхностях ограничивающих их разломов из-за смещений соседних блоков или их подстилающей среды. Приведем формулы для определения упругой силы (f_t, f_l, f_n), действующей на единицу площади разлома:

$$f_t = K_t(\Delta_t - \delta_t), \quad f_l = K_l(\Delta_l - \delta_l), \quad f_n = K_n(\Delta_n - \delta_n). \tag{1}$$

Здесь (t, l, n) — система координат, связанная с точкой приложения силы (оси t, l лежат в плоскости, касательной к поверхности разлома, ось n ей перпендикулярна); Δ_t, Δ_l, Δ_n — компоненты относительного смещения в системе (t, l, n) соседних блоков в случае, если точка принадлежит части разлома, разделяющей блоки, или блока и подстилающей среды соседнего блока в случае, если точка принадлежит части разлома, отделяющей блок от подстилающей среды соседнего блока; δ_t, δ_l, δ_n — соответствующие неупругие смещения, зависимость от времени которых описывается уравнениями

$$\frac{d\delta_t}{dt} = W_t f_t, \quad \frac{d\delta_l}{dt} = W_l f_l, \quad \frac{d\delta_n}{dt} = W_n f_n. \tag{2}$$

Коэффициенты K_t, K_l, K_n (1), характеризующие упругие свойства разлома, и коэффициенты W_t, W_l, W_n (2), характеризующие вязкие свойства разлома, могут быть различными для разных разломов и, кроме того, могут изменяться в зависимости от глубины.

Аналогично выглядят формулы для вычисления сил и неупругих смещений на подошвах блоков. Смещения любого внутреннего блока и углы его поворотов находятся из условия равенства нулю суммы всех сил, действующих на блок, и суммарного момента этих сил. Это условие обеспечивает состояние квазистатического равновесия системы и одновременно является условием минимума энергии. Поскольку в рассматриваемой модели зависимость сил от смещений и поворотов блоков является линейной (явные формулы опущены ввиду их громоздкости, см. [3]), то система уравнений для определения этих величин также линейна и имеет вид

$$Aw = b. \tag{3}$$

Компонентами неизвестного вектора $w = (w_1, w_2, \ldots, w_{6n})$ являются смещения и углы поворота внутренних блоков (n — число таких блоков). Элементы матрицы A (размерности $6n \times 6n$) не зависят от времени и могут быть вычислены один раз в начале процесса. Для подсчета различных криволинейных интегралов выполняется дискретизация (разбиение на ячейки) сферической поверхности подошв блоков и сегментов разломов, при этом предполагается, что значения сил и неупругих смещений совпадают для всех точек ячейки. Система (3) решается в дискретные моменты времени t_i.

В каждый момент t_i при вычислении компонент силы, действующей на разломе, определяется безразмерная величина κ, трактуемая как модельное напряжение:

$$\kappa = \frac{\sqrt{f_t^2 + f_l^2}}{P - f_n}. \tag{4}$$

Здесь P — одинаковый для всех разломов параметр, который может интерпретироваться как разность между литостатическим и гидростатическим давлением. Таким образом, фактически величина κ является отношением модуля силы, стремящейся сдвинуть блоки вдоль разлома, к модулю силы, прижимающей блоки друг к другу. Для каждого разлома задаются значения трех порогов прочности, вообще говоря, зависящие от времени:

$$B > H_f \geq H_s, \; B = B(t_i) = B_0(t_i) + \sigma X(t_i), \; H_f = H_f(t_i) = aB(t_i), \; H_s = H_s(t_i) = bB(t_i). \tag{5}$$

Для всех i выполняется: $0 < B_0(t_i) < 1$, $0 < \sigma << 1$, $X(t_i)$ — случайная величина, распределенная по стандартному нормальному закону $N(0;1)$, $0 < a < 1$, $0 < b \le a$. Предполагается, что начальные условия таковы, что неравенство $\kappa < B$ имеет место во всех ячейках структуры. Взаимодействие между блоками (между блоком и соседней подстилающей средой) полагается вязкоупругим (нормальное состояние) до тех пор, пока величина κ (4) на части разлома, разделяющего элементы структуры, не достигает заданного порога B. Такая ситуация интерпретируется как землетрясение. В ячейках, попавших в «критическое» состояние, в соответствии с законом сухого трения, происходит резкий сброс напряжения посредством изменения значений неупругих смещений δ_t, δ_l, δ_n по формулам:

$$\delta_t^e = \delta_t + \gamma_t^e f_t, \quad \delta_l^e = \delta_l + \gamma_l^e f_l, \quad \delta_n^e = \delta_n + \gamma_n^e f_n, \tag{6}$$

где $\delta_t, f_t, \delta_l, f_l, \delta_n, f_n$ — значения неупругих смещений и компонент вектора силы непосредственно перед землетрясением. Коэффициенты $\gamma_t^e, \gamma_l^e, \gamma_n^e$ вычисляются таким образом, чтобы для нового значения модельного напряжения κ было справедливо равенство $\kappa = H_f$. После описанных выше пересчетов находится правая часть системы (3), затем определяются векторы сдвига и углы поворота блоков. Если вновь в какой-либо ячейке $\kappa \ge B$, то вся процедура повторяется. Когда во всех ячейках на разломах $\kappa < B$, вычисления продолжаются по обычной схеме. Считается, что ячейки, в которых произошли землетрясения, находятся в состоянии крипа. Это означает, что для них в уравнениях (2) для вычисления значений неупругих смещений используются параметры W_t^s ($W_t^s >> W_t$), W_l^s ($W_l^s >> W_l$) и W_n^s ($W_n^s >> W_n$), обеспечивающие значительно более быстрый, по сравнению с нормальным состоянием, рост неупругих смещений и, следовательно, уменьшение значений сил и напряжений. Состояние крипа продолжается до тех пор, пока $\kappa > H_s$, после чего ячейка возвращается в нормальное состояние с использованием при расчетах W_t, W_l и W_n.

Основным результатом процесса моделирования является искусственный каталог землетрясений. Принадлежащие одному разлому ячейки, в которых произошло землетрясение в момент времени t_i, объединяются в одно событие. Географические координаты его эпицентра и глубина вычисляются как взвешенные суммы координат и глубин ячеек (вес ячейки определяется как отношение ее площади к сумме площадей всех ячеек, вовлеченных в землетрясение). Взвешенная сумма векторов $(\gamma_t^e f_t, \gamma_l^e f_l)$ добавок к неупругим смещениям δ_t и δ_l при пересчете по формулам (6) аппроксимирует случившуюся подвижку блоков вдоль разлома, и позволяет определить механизм модельного события. Механизм землетрясения — важная его характеристика, информирующая о процессе распространения различных сейсмических волн от очага. В зависимости от направления подвижки и угла наклона разлома принято выделять следующие основные механизмы: сдвиг, сброс и взброс; в новой версии модели магнитуда землетрясения вычисляется в зависимости от его механизма с использованием известных в сейсмологии эмпирических формул

$$M = D \lg S + E, \tag{7}$$

где S — сумма площадей ячеек (в км^2), $D = 1.02$, $E = 3.98$ для сдвига, $D = 1.02$, $E = 3.93$ для сброса, $D = 0.90$, $E = 4.33$ для взброса.

Отметим, что модель дополнительно позволяет получить картину мгновенной кинематики блоков и информацию о характере их взаимодействия вдоль границ.

2. Динамическое перераспределение нагрузки

Вычислительные эксперименты показали, что сферическая блоковая модель допускает эффективное распараллеливание на основе стандартной схемы «мастер-рабочий» [3] с единым загрузочным MPI-модулем. На каждом шаге дискретного времени наиболее трудоемкими процедурами являются сохранение информации о модельных событиях и, главным образом, определение значений сил, неупругих смещений и напряжений (1), (2), (4)–(6) во всех ячейках структуры (так, в расчетном типовом варианте для глобальной системы тектонических плит,

покрывающих всю поверхность Земли [3–5], имеем около 200000 ячеек на подошвах 15 блоков и около 3500000 ячеек на сегментах 199 разломов). Основные вычисления могут быть проведены независимо друг от друга, поэтому они должны быть равномерно распределены между процессорами. Подробное описание параллельного алгоритма и некоторых его характеристик (ускорения, эффективности, масштабируемости, характера межпроцессорных коммуникаций) приведено в [4, 5]. Отметим, что основным фактором, негативно влияющим на качество распараллеливания, является то, что равные порции ячеек на всех процессорах обеспечивают равные объемы вычислений только при отсутствии землетрясений. В случае интенсивного потока модельных землетрясений нагрузка на процессоры становится существенно неравномерной (ввиду неравномерного распределения сейсмичности по разломам), что уменьшает эффективность.

Серия новых вычислительных экспериментов по изучению влияния случайных факторов в порогах прочности (5) на модельную сейсмичность подразумевает расчет большого числа однотипных вариантов для последующего усреднения результатов, а также увеличение объема передаваемой рабочими процессорами мастеру информации в связи с определением механизма модельного события. По этой причине задача уменьшения времени счета выходит на первый план. Предлагается ее решать с помощью динамической оптимизации вычислительной нагрузки процессоров непосредственно на стадии моделирования. Фактически речь идет о перераспределении порций ячеек пространственной дискретизации сегментов, поскольку объем вычислений на подошвах блоков от времени не зависит, так как модельные землетрясения происходят только на разломах. Перераспределение будем осуществлять после выхода модельного потока событий на так называемый стационарный режим, когда стабилизируется среднее количество землетрясений на единицу времени в большинстве сейсмоактивных зон (обычно это происходит по истечении первых 50–70 единиц модельного времени). Основным входом алгоритма служит информация о распределении ячеек, вовлеченных в землетрясения (будем говорить далее «разорванных»), по сегментам на некотором тестовом временном интервале, длина которого зависит от варианта. Опираясь на стационарность потока, мы предполагаем, что распределение таких ячеек на следующих промежутках той же длины будет аналогичным. Идея алгоритма состоит в том, чтобы перераспределить ячейки между процессорами с учетом количества разорванных и нормальных ячеек на тестовом промежутке и соотношения времен обработки ячеек разных типов (очевидно, разорванная ячейка обрабатывается дольше, чем нормальная, ввиду пересчетов (1), (4), (6).

Формализуем указанную эмпирическую процедуру. Введем величины p_1 и p_2 ($p_2 \ge p_1$) — времена обработки одной ячейки, соответственно, в нормальном состоянии и разорванной. Для каждого сегмента рассмотрим взвешенную сумму

$$S_i = p_1 N_i + (p_2 - p_1)\overline{N}_i / m, \tag{8}$$

где i — номер сегмента, $i=1,\ldots,k$ (k — количество сегментов в структуре), N_i — количество ячеек дискретизации на i-м сегменте, $\overline{N}_i$ — количество ячеек на i-м сегменте, разорванных в течении m временных шагов, составляющих тестовый промежуток. Эта сумма является аппроксимацией среднего времени обработки всех ячеек сегмента на одном шаге модельного времени. Тогда величину

$$S_* = \sum_{i=1}^{k} S_i / n, \tag{9}$$

можно трактовать как идеальное время обработки ячеек на каждом из n процессоров при равномерной вычислительной нагрузке. Теперь остается разбить все ячейки структуры на одинаковые в смысле времени обработки порции между процессорами. Накопление ячеек в порции для процессора j происходит с учетом величин из (8) до тех пор, пока сумма типа (8) на процессоре не достигнет S_*, после чего происходит переход к формированию следующей, $(j+1)$-й, порции. Отметим, что, поскольку удобнее делить ячейки на порции без разрыва слоя по глубине, то возможны незначительные отклонения от S_*.

Приведем пример, иллюстрирующий описанный метод. Моделирование сейсмичности глобальной системы тектонических плит проводилось в Институте математики и механики УрО РАН (г. Екатеринбург) на гибридном вычислителе кластерного типа «Уран» (состоит из 208 вычислительных узлов, каждый из которых оснащен двумя четырехядерными процессорами

Intel Xeon, работающими на частоте 3.0 ГГц, и 16–32 Гб оперативной памяти; имеет пиковую производительность порядка 20 Тфлопс). Вычислительные эксперименты показали, что отношение p_2 / p_1, характеризующее длительность обработки разорванной ячейки по сравнению с нормальной, колеблется от 1.2 до 1.5 в зависимости от процессора и варианта. Оказалось, что информации о состоянии ячеек сегментов структуры за 500 шагов модельного времени после выхода потока событий на стационарный режим (интервал [100, 105], шаг дискретизации 0.01) достаточно для того, чтобы алгоритм (8), (9) перераспределил ячейки на новые порции с уменьшением времени счета на последующих интервалах. При равномерном распределении по 16 процессорам (на каждый попадает около 217750 ячеек) на тестовом интервале результат перераспределения для $p_2 / p_1 = 1.25$ схематично показан на Рис. 1.

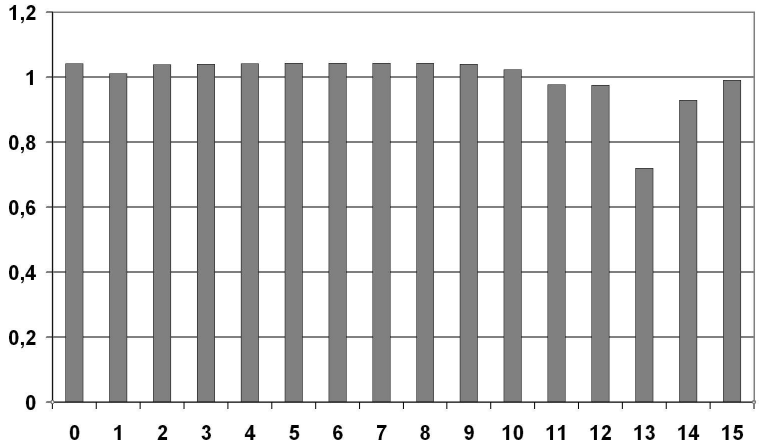


Рис. 1. Результат перераспределения ячеек между 16 процессорами для $p_2 / p_1 = 1.25$; горизонтальная ось ─ номер процессора, вертикальная ось ─ величина порции, попавшей на процессор; единица соответствует порции при равномерном распределении

В Табл. 1 приведены усредненные времена счета с равномерным ($p_2 / p_1 = 1$) и неравномерным ($p_2 / p_1 = 1.25$) распределением ячеек на разном количестве процессоров как для самого тестового интервала [100, 105], так и для следующего [105, 110]. Очевидно, что именно данные для второго интервала являются информативными для оценивания качества работы алгоритма. Отметим, что несмотря на незначительное уменьшение времени счета согласно Табл. 1 (до 10%), в целом имеем позитивный результат, так как, во-первых, и такое улучшение важно при расчетах больших вариантов (до 1000 единиц модельного времени), а во-вторых, других резервов для оптимизации распараллеливания блоковой модели не просматривается.

Таблица 1. Времена счета (сек) с равномерным и неравномерным распределением ячеек по n процессорам

n	$T^{1}_{[100,105]}$	$T^{1.25}_{[100,105]}$	$T^{1}_{[105,110]}$	$T^{1.25}_{[105,110]}$
16	1172	1054	971	939
32	589	539	573	536
64	347	341	340	334
96	296	268	300	267
128	238	219	230	213

3. «Большой» вычислительный эксперимент

Благодаря новым возможностям распараллеливания, описанным в предыдущем разделе, проведено несколько достаточно времяемких экспериментов с аппроксимацией глобальной системы тектонических плит: 1) по использованию случайных значений для порогов прочности

(5); 2) по получению модельного каталога на длинном (по сравнению со всеми предыдущими) временном интервале. Представим некоторые результаты 2-го эксперимента.

Время счета одного варианта до 1000 единиц модельного времени на 64 процессорах МВС «Уран» составило 24 часа, размер файла с информацией о землетрясениях ─ 320 Гб. Анализировались прежде всего параметры закона Гутенберга–Рихтера, характеризующего распределение землетрясений по магнитуде.

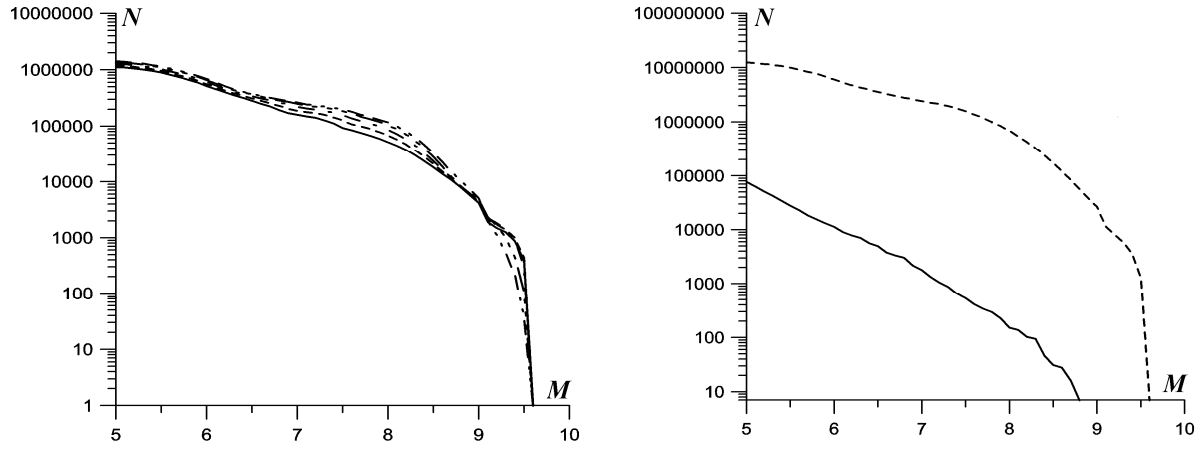


Рис. 2. Графики повторяемости, построенные по пяти модельным каталогам длиной 100 единиц (левый рисунок), по реальному (NEIC [6], события с магнитудой не менее 5.0, период 01.01.1900–31.12.2011, сплошная линия) и модельному (1000 единиц безразмерного времени, штриховая линия) каталогам (правый рисунок); N ─ аккумулированное число землетрясений, M ─ магнитуда

Почти совпадающие графики на левой части Рис. 2 свидетельствуют об установившейся стационарности потока событий, графики на правой части ─ о хорошем приближении угла наклона графика реальной глобальной сейсмичности, что подтверждает гипотезу об улучшении свойств искусственной сейсмичности на длительном интервале времени вследствие выхода модельного потока событий на стационарный режим. Кроме того, «почти параллельность» реального и модельного графиков повторяемости в области больших магнитуд говорит об идентичности распределений землетрясений по магнитуде в данном интервале и, следовательно, о возможности изучения реальных закономерностей по «длинному» модельному каталогу.

Литература

1. Gabrielov A.M., Newman W.I. Seismicity Modeling and Earthquake Prediction: a Review // Geophysical Monograph 83, IUGG. 1994. Vol. 18. P. 7-13.
2. Keilis-Borok V.I., Soloviev A.A. (Eds.) Nonlinear Dynamics of the Lithosphere and Earthquake Prediction. Springer, 2003. 337 p.
3. Мельникова Л.А., Розенберг В.Л. Сферическая блоковая модель динамики и сейсмичности литосферы: различные модификации и вычислительные эксперименты // Труды ИММ УрО РАН. 2007. Т. 13. №3. С. 95-120.
4. Розенберг В.Л., Мельникова Л.А. Применение параллельных технологий к моделированию глобальной сейсмичности // Параллельные вычислительные технологии (ПаВТ-2010): Тр. междунар. конф. Челябинск, 2010. ISBN 978-5-696-03987-9. С. 311-321.
5. Мельникова Л.А., Розенберг В.Л. Высокопроизводительные вычисления в моделировании динамики и сейсмичности систем тектонических плит. // Параллельные вычислительные технологии (ПаВТ-2012): Тр. междунар. конф. Новосибирск, 2012. ISBN 978-5-696-04237-4. С. 248-259.
6. Global Hypocenters Data Base, NEIC/USGS. Denver, CO. URL: http://earthquake.usgs.gov/regional/neic/ (дата обращения: 1.09.2012).

Гибридный алгоритм решения задачи 3-ВЫПОЛНИМОСТЬ ассоциированной с задачей факторизации

Ю.Ю. Огородников, Р.Т. Файзуллин

Омский государственный технический университет, г. Омск

В данной статье рассматривается гибридный алгоритм поиска приближённого решения задачи ВЫПОЛНИМОСТЬ. Предлагается комплексный подход – сегментный генетический алгоритм и метод последовательных приближений.

1. Введение

Задача ВЫПОЛНИМОСТЬ – одна из наиболее интересных задач теории сложности [1]. Широкое применение эта задача находит в таких сферах, как искусственный интеллект, проектирование компьютерных систем, криптография. В частности, в криптографии часть выполняющего набора для КНФ и 3-КНФ, ассоциированных с задачами криптографического анализа можно рассматривать как ключ шифрования.

Одним из перспективных направлением в поиске алгоритмов решения задачи ВЫПОЛНИМОСТЬ представляется сведение КНФ к непрерывному аналогу [2], т.е. к задаче поиска точек глобального минимума ассоциированной функции и гибридизация с дискретными методами.

Другим перспективным направлением поиска решений являются генетические алгоритмы - стохастические, эвристические оптимизационные методы, впервые предложенные Холландом[3]. Они основываются на идее эволюции с помощью естественного отбора. На основе генетических алгоритмов разработаны такие методы, как, например, VEGAS[4].

В поисках более эффективных методов решения происходит разработка комплексных подходов к решению данной задачи. Одним из таких подходов является гибридный алгоритм поиска приближённого решения, состоящий из двух стадий: сегментного генетического алгоритма и метода последовательных приближений. Этот метод и рассматривается в данной статье.

2. Этап работы сегментного генетического алгоритма

Эволюционные (или генетические) алгоритмы – это эвристические методы поиска, используемые для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию [3], [5].

Данные алгоритмы оперируют такими «биологическими» понятиями, как наследование, отбор, мутация, кроссинговер, индивид. Под индивидом понимается строка битов – геном. Для каждого индивида в каждом поколении вычисляется значение фитнесс-функции – его приспособленность к ареалу.

Перед непосредственным применением генетического алгоритма для задачи ВЫПОЛНИМОСТЬ следует описать задачу в терминах, соответствующих генетическому методу. Исходную 3-ДНФ можно представить в виде среды обитания (ареала). Геномом будет служить строка размерности N (где N – число переменных, задействованных в 3-ДНФ), состоящая из чисел 0 и 1. Исходной популяцией будет служить набор из L случайно сформированных геномов. Приспособленностью генома будем называть число скобок, которые при подстановке данного генома в 3-КНФ обращаются в значение ИСТИНА. В целях повышения эффективности процесса поиска решения будем использовать несколько параллельно работающих генетических алгоритмов. В основе такого подхода лежит разделение множества скобок исходной 3-КНФ на подмножества с последовательным распределением по генетическим алгоритмам с различными функциями приспособленности, т.е. один и тот же геном длины N будет иметь различную приспособленность в разных подмножествах. Для удобства дальнейшего изложения будем также

называть данные подмножества скобок *подзадачами* или *сегментами.* Формирование подзадач следует проводить в соответствии с разбиением *множества индексов* исходной 3-КНФ. Каждой подзадаче соответствует своё подмножество индексов, на основании которого происходит распределение скобок. Подмножества индексов попарно не пересекаются. Также отметим, что разбиение, полученное для 3-КНФ, также применимо и для эквивалентной ей 3-ДНФ.

Таким образом, задача ВЫПОЛНИМОСТЬ сводится к задаче минимизации функционала вида

$$F(x) = \sum_{i=1}^{M} \sum_{j=1}^{Qi} C_j(x), \text{ где } C_j \text{ - произведения вида } C_j(x) = \prod_{k=1}^{N} q_{j,k}(x), \quad (1.5.1)$$

$$q_{j,k}(x) = \begin{cases} x_k, \textit{если переменная } x_k \textit{ входит в } j-\textit{й конъюнкт непосредственно}, \\ \overline{x_k}, \textit{если переменная } x_k \textit{ входит в } j-\textit{й конъюнкт с отрицанием}, \\ 1, \textit{иначе} \end{cases}$$

Здесь M – число подзадач, Q_i – число конъюнктов в подзадаче i, x_k - литерал, N – общее число индексов.

По окончании работы всех генетических алгоритмов из каждого подмножества скобок будет браться геном с наилучшей из найденных приспособленностей. Данные результаты объединяются в главный геном. Отметим, что каждая подзадача оперирует геномами длины N, но при объединении достигнутых результатов в главный геном будут включаться только биты с индексами, на которых основана подзадача.

Существует несколько способов формирования подзадач. Ниже представлен один из таких методов. Пусть изначально имеется множество индексов $\Xi_0 = \{1,2,..., N\}$. Разделим множество индексов Ξ_0 = (1, 2,…, N) на два равных по мощности множества Ξ_1 = (1, 2,…,N/2), Ξ_2 = (N/2 + 1,…,N). Выберем произвольный индекс i из Ξ_1 для которого существует слагаемое вида $x_i \vee x_j \vee y$ (индекс j, также как и индекс i, принадлежит Ξ_1, y – произвольный литерал) в рассматриваемой 3-КНФ. Если такого слагаемого не существует, то мы переносим индекс i в множество индексов Ξ_2. В этом множестве искомое слагаемое по необходимости найдётся, иначе x_i не присутствует ни в одном из слагаемых. Перебирая все i в Ξ_1, мы получим два непересекающихся подмножеств индексов, в каждом из которых любые два индекса литералов (с отрицаниями или без) входят в Ξ_1 либо в Ξ_2.

Разделив далее большее по мощности из Ξ_s (или оба, если мощности сравнимы) на два подмножества Ξ_{s1} и Ξ_{s2} повторим процедуру и т.д.

Полностью выполнить такое требование для всех скобок невозможно, однако можно значительно уменьшить число скобок вида $x_i \vee x_j \vee x_k$, где i, j, k принадлежат разным подмножествам индексов. Такие слагаемые следует равномерно распределить по имеющимся подзадачам.

Рассмотрим пример разбиения на ниши:

Имеется следующая 3-КНФ:

$$(x_1 \vee \overline{x_9} \vee x_{10})(\overline{x_8} \vee x_9 \vee x_{10})(x_2 \vee \overline{x_3} \vee x_8)(x_4 \vee x_6 \vee \overline{x_{10}})(\overline{x_2} \vee x_5 \vee \overline{x_7})(x_4 \vee x_8 \vee \overline{x_9})$$

Литералом называется переменная x_1, индексом литерала называется в данном случае 1.

Стартовое множество индексов выглядит следующим образом:

$$\Xi_0 = \{1,2,3,4,5,6,7,8,9,10\}$$

I. Разобьём множество индексов Ξ_0 на 2 подмножества:

$$\Xi_1 = \{1,2,3,4,5\} \quad \text{и} \quad \Xi_2 = \{6,7,8,9,10\}$$

В соответствии с алгоритмом берём множество наибольшей мощности.

Если таких множеств несколько, то берём множество с наименьшим номером – в данном случае выбираем множество Ξ_1.

II. Перебирая множество индексов из Ξ_1 замечаем, что для индексов $i = 1$ и $i = 4$ не существует слагаемых вида $x_i \vee x_j \vee y$ в рассматриваемой 3-КНФ (x_i - литерал или его отрицание, индекс j принадлежит множеству Ξ_1, y – произвольный литерал). Для индексов $i = 2, 3, 5$ такие слагаемые существуют (для $i = 2$ это будет индекс $j = 3$ и дизъюнкт $x_2 \vee \overline{x_3} \vee x_8$, для $i = 3$ индекс j = 2 и дизъюнкт $x_2 \vee \overline{x_3} \vee x_8$, для $i = 5$ индекс $j = 2$ и дизъюнкт $\overline{x_2} \vee x_5 \vee \overline{x_7}$). Переносим индексы $i = 2$ и $i = 4$ во множество Ξ_2. Переходим на следующий шаг алгоритма разбиения.

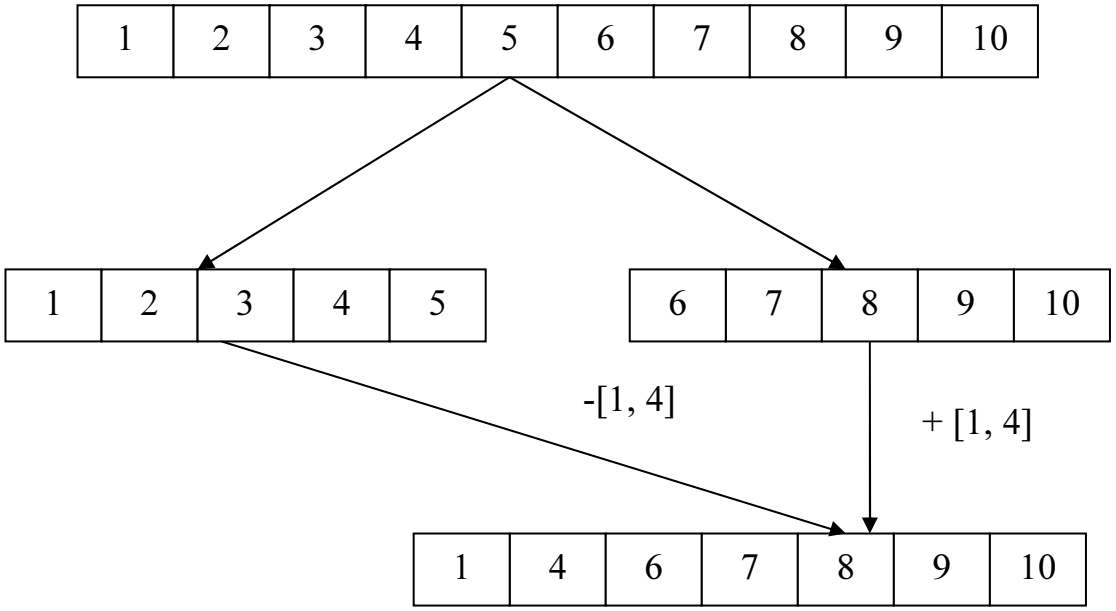


Рис. 1. Первая и вторая итерации разбиения множества индексов

Ш. Рассматриваем большее по мощности множество Ξ_2, которое теперь имеет следующий вид: $\Xi_2 = \{1,4,6,7,8,9,10\}$. Разделим его на два подмножества $\Xi_3 = \{1,4,6\}$ и $\Xi_4 = \{7,8,9,10\}$. Рассмотрим большее по мощности из этих множеств, а именно: Ξ_4. Перебираем все его элементы и замечаем, что только для $i = 7$ нет индекса j во множестве Ξ_4 такого, что $x_i \vee x_j \vee y$ присутствует в рассматриваемой 3-КНФ. Для остальных же индексов условие выполняется (для $i = 8$ индекс $j = 9$ и дизъюнкт $x_4 \vee x_8 \vee \overline{x_9}$, для $i = 9$ индекс $j = 8$ и тот же дизъюнкт $x_4 \vee x_8 \vee \overline{x_9}$, для $i = 9$ индекс $j = 8$ и тот же дизъюнкт $x_4 \vee x_8 \vee \overline{x_9}$, для $i = 10$ индекс $j = 8$ и дизъюнкт $\overline{x_8} \vee x_9 \vee x_{10}$). Для индекса $i = 7$ условие не выполняется, однако в силу того что он единственный такой индекс, то его можно оставить в множестве Ξ_4.

IV. Берём множество Ξ_3 – следующее по мощности за Ξ_4. Замечаем, что для i = 1 рассматриваемая 3-КНФ не содержит дизъюнктов вида $x_i \vee x_j \vee y$. Для i = 4 и i = 6 такие дизъюнкты существуют (для i = 4 индекс j = 6 и дизъюнкт $x_4 \vee x_6 \vee \overline{x_{10}}$, а для i = 6 индекс j = 4 и тот же дизъюнкт $x_4 \vee x_6 \vee \overline{x_{10}}$). Так как индекс i = 1 – единственный, для которого не существует дизъюнкта вида $x_i \vee x_j \vee y$, то его можно оставить в множестве Ξ_3.

Дальнейшее проведение итераций не имеет смысла, так как далее в полученных подмножествах не будет существовать дизъюнктов вида $x_i \vee x_j \vee y$.

РАЗБИЕНИЕ ОКОНЧЕНО.

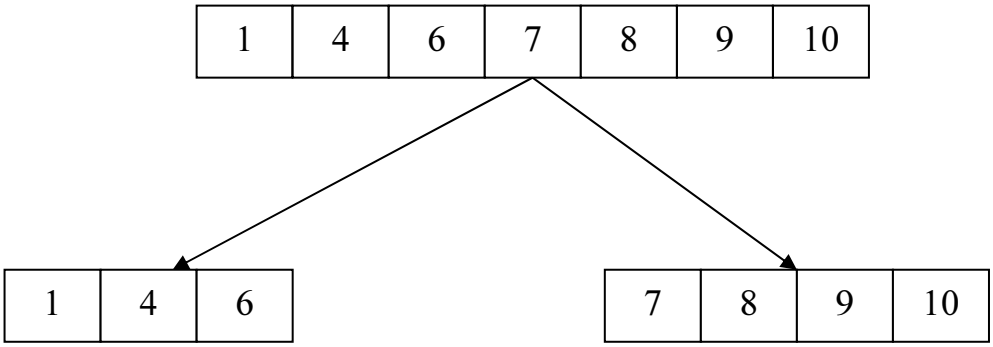


Рис. 2. Третья и четвёртая итерации разбиения множества индексов

Таким образом, скобки будут разделены следующим образом:

- на основании Ξ_1: $(x_2 \vee \overline{x_3} \vee x_8)(\overline{x_2} \vee x_5 \vee \overline{x_7})$
- на основании Ξ_3: $(x_4 \vee x_6 \vee \overline{x_{10}})$
- на основании Ξ_4: $(x_1 \vee \overline{x_9} \vee x_{10})(\overline{x_8} \vee x_9 \vee x_{10})(x_4 \vee x_8 \vee \overline{x_9})$

Множества Ξ_0 и Ξ_2 не учитывались при разбиении, так как из них были получены меньшие по мощности множества: Ξ_1 и Ξ_2 из Ξ_0, Ξ_3 и Ξ_4 из Ξ_2 соответственно.

Родители выбираются с близкими значениями функции приспособленности, и передаются функции скрещивания. В качестве оператора скрещивания выбирается случайная точка в геноме, так называемая точка разрыва. Геном потомка состоит двух частей: части генома первого родителя до точки разрыва и части генома второго родителя после точки разрыва. После создания генома производится его мутация: значения нескольких бит заменяются противоположными[6].

После выполнения процедуры создания следующего поколения вычисляются значения фитнесс для всех потомков, и новая популяция строится путём объединения старого и нового поколения, причём размер популяции остаётся тем же, но туда вбираются индивиды с наилучшей приспособленностью.

По окончании работы всех генетических алгоритмов из каждого подмножества скобок будет браться геном с наилучшей из найденных приспособленностей. Данные результаты объединяются в главный геном. Отметим, что каждая подзадача оперирует геномами длины *N*, но при объединении достигнутых результатов в главный геном будут включаться только биты с индексами, на которых основана подзадача.

3. Этап работы метода последовательных приближений

Осуществим переход от задачи ВЫПОЛНИМОСТЬ к задаче поиска экстремума в подмножестве $[0, 1]^N$ евклидова пространства. Переход основан на построении функционала специального вида, глобальный минимум которого соответствует решению исходной задачи[2].

Имеется КНФ на множестве переменных $(y_1, \ldots. y_n) \in B^N$:

$$L^*(y) = \bigwedge_{i=1}^{M} G_i^*(y), \text{ где}$$

$$G_i^*(y) = \bigvee_{j=1}^{N} q_{i,j}^*(y),$$

$$q_{i,j}^*(y) = \begin{cases} y_j, \text{если} \quad y_j \in \{G_i^*\} \\ \overline{y}_j, \text{если} \quad \overline{y}_j \in \{G_i^*\}, \\ 0, \text{иначе} \end{cases}$$

N - число переменных , M - число дизъюнктов

Преобразуем исходную КНФ к ДНФ следующим образом:

$$L(y) = \overline{L^*(y)} = \bigvee_{i=1}^{M} G_i(y), \text{ где}$$

$$G_i(y) = \bigwedge_{j=1}^{N} q_{i,j}(y),$$

$$q_{i,j}(y) = \overline{q_{i,j}^*(y)}$$

Сопоставим булевым переменным $y = (y_1, \ldots y_N) \in B^N$ исходной КНФ переменные пространства вещественных чисел $x = (x_1, \ldots x_N) \in R^N$. Для этого рассмотрим функционал, на множестве переменных $x \in R^N$:

$$F(x) = \sum_{i=1}^{M} C_i(x), \text{ где}$$

C_i - произведения вида $C_i(x) = \prod_{j=1}^{N} p_{i,j}(x),$

$$p_{i,j}(x) = \begin{cases} x_j^2, & \text{если } \overline{y}_j \in \{G_i^*\} \\ (1-x_j)^2, & \text{если } y_j \in \{G_i^*\} \\ 1, & \text{иначе} \end{cases}$$

Суммирование ведётся по всем M конъюнктам ДНФ, эквивалентной исходной КНФ.

Соответствие между булевыми (y_i) и вещественными (x_i) переменными следующее:

$$\begin{cases} y_i = 1 \text{ (ИСТИНА)} \quad \Rightarrow x_i = 1 \\ y_i = 0 \text{ (ЛОЖЬ)} \quad \Rightarrow x_i = 0 \end{cases}$$

В обратную сторону

$$\begin{cases} x_i = 1 \Rightarrow y_i = 1 \text{ (ИСТИНА)} \\ x_i \neq 1 \Rightarrow y_i = 0 \text{ (ЛОЖЬ)} \end{cases}$$

Переход от булевой формулы к вещественной основан на использовании литерной функции вида:

$$\begin{cases} T(y_i \vee y_j) = x_i + x_j \\ T(y_i \wedge y_j) = x_i^2 \cdot x_j^2 \text{ , где } y_i \in B,\ x_i \in B,\ i = 1 \ldots N \\ T(\overline{y}_i) = 1 - x_i \end{cases}$$

Функционал F имеет единственный глобальный экстремум, соответствующий решению исходной задачи. Но вместо минимизации F будем производить поиск стационарных точек. Для этого дифференцируем функционал по всем переменным и приравниваем к нулю полученные выражения. В результате получается система уравнений с n неизвестными. Применяя к полученной системе метод последовательных приближений, получим приближённое решение функционала. В качестве начального приближения следует брать значение, полученное в результате выполнения сегментного генетического алгоритма[2].

После окончания работы этапа полученное приближение добавляется в стартовую популяцию генетического алгоритма.

4. Интеграция генетического алгоритма и метода последовательных приближений

С целью получения комплексной процедуры решения задачи ВЫПОЛНИМОСТЬ, объединим этапы работы генетического алгоритма и метода последовательных приближений.

Связь между двумя этапами будет осуществляться подстановкой в выходные данные одного этапа входные данные другого. Так, геном, соответствующий минимуму в части генетического алгоритма, служит начальным приближением для метода последовательных приближений. В свою очередь, конечный вектор $x(W)$, где W – число итераций в методе последовательных приближений, добавляется в популяцию для генетического алгоритма.

Полученная процедура повторяется заданное число итераций C.

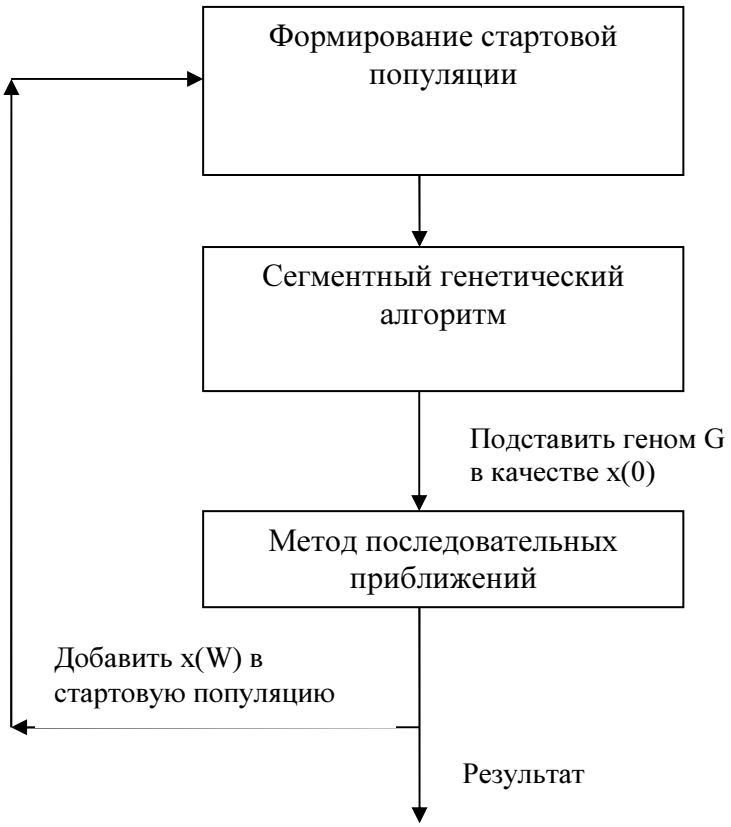


Рис. 3. Графическое представление двухэтапного метода

5. Способы распараллеливания алгоритма

Главный поток выполняет чтение входных данных и инициализацию структур программы. Он же производит запуск вспомогательных потоков для сегментов. Каждому сегменту соответствует отдельный поток. Объединение результатов подзадач происходит также в главном потоке.

6. Промежуточные результаты

В таблице 1 представлены результаты вычислений для числа итераций $C = 100$. Вычисления проводились для разных параметров N и M, где N – число переменных, M – общее число скобок. Остальные параметры фиксированы и равны следующим значениям:

- численность популяции $L = 2000$
- число подзадач $Q = 10$
- число поколений $G = 500$

Под числом верных скобок T подразумевается количество скобок в исходной 3-КНФ, которое обращается в значение ИСТИНА при подстановке решения, полученного в результате выполнения алгоритма.

Таблица 1. Результаты вычислений

Число переменных N	Общее число скобок M	Число верных скобок T	Отношение T / M
20	68	68	1
120	440	440	1
252	952	951	0,9989
540	2080	2075	0,9975
1260	4920	4920	1
3600	14200	14200	1
5220	20640	20640	1
9360	37120	37120	1
14700	58400	58396	0,9999
33300	132600	132591	0,9999
59400	236800	236787	0,9999
97536	389120	389038	0,9999

Из полученных результатов видно, что число найденных верных скобок довольно велико, и к неверным (т.е. принимающим значение ЛОЖЬ) скобкам можно применить различные методы для поиска точного решения. В следующей теореме показано, что существует предположительная область сходимости для общей 3-КНФ с единственным решением при выполнении определённых условий на приближении.

Теорема 1.

Пусть задана 3-КНФ содержащая N литералов, M скобок, и имеющая единственный решающий набор. Предположим, что выполнено следующее условие: пусть задано приближение к решению в виде целочисленной точки с компонентами (0,1) такое, что среди невыполняющихся при подстановке приближения скобках имеются L скобок, невыполнение которых зависит только от одного литерала, а остальные два верные, и $L \geq M/8$. В этом случае сужение функции, ассоциированной с 3-КНФ, на луч, соединяющий решение и данную целочисленную точку, с координатами в вершинах гиперкуба, отличающуюся от решения является строго монотонной убывающей, а если $L \geq M/3$, то и выпуклой функцией. Заметим, что из теоремы не следует выпуклость графика самой функции, но численные эксперименты показали, что данная теорема имеет практическую ценность. Оказывается, что при числе верных бит N (т.е. совпадающих с битами точного решения), большем 6/7 от общего количества, метод последовательных приближений, применённый к задаче ВЫПОЛНИМОСТЬ, всего за несколько итераций сходится к решению. Причём смещение идёт практически прямо по отрезку, соединяющему приближение и точное решение.

Основываясь на результатах теоремы, можно попытаться построить методику поиска наиболее вероятных бит, что позволит нам выбрать приближение из области сходимости к точке глобального экстремума.

Методика может состоять из нескольких независимых тестов. Например, по содержанию ненулевых скобок можно оценить, у скольких бит и примерно каких следует изменить значение.

Другой способ может быть основан на таблице частот устойчивости бит, полученной в результате применения метода последовательных приближений к поиску выполняющего набора для 3-ДНФ, эквивалентной 3-КНФ. Чем частоты более удалены от значения 0.5, тем выше вероятность верно определить бит. К примеру, если частота равна 1, то это означает, что во всех случаях бит точно угадан. Напротив же, если частота равна 0, то, следовательно, во всех случаях бит неверно угадан, и можно гарантированно произвести его инвертирование. Если частота равна 0.5, то ничего определённого сказать нельзя.

В таблицах 2 – 4 приведены данные исследования устойчивости бит для задачи ВЫПОЛНИМОСТЬ, эквивалентной задаче ФАКТОРИЗАЦИИ.

Частоты получены на основе исследований 200 различных исходных данных для задачи ФАКТОРИЗАЦИИ, которые получены качественным источником генерации случайных значений[7]. В каждой таблице указано количество бит, для которых частота устойчивости удовлетворяет некоторым условиям, и средняя ошибка.

Также эксперименты показали, что для крайних бит сомножителей (например, для сомножителей размерности 600 это 1, 300, 301 и 600 биты), частота устойчивости равна 1. Это означает, что биты, полученные на таких позициях, гарантированно являются верными.

Таблица 2. Частота устойчивости бит удовлетворяющих условию > 0.7 или < 0.3

Размерность сомножителя	Общее число бит M	Число нулевых бит T	Отношение T / M	Средняя ошибка
100	14700	9331	0,6347	0,1803
200	59400	38912	0,655	0,1871
300	134100	86174	0,6462	0,1821
400	238800	153336	0,6421	0,1836
500	373500	239339	0,6408	0,1832
600	538200	345203	0,6414	0,1834

Таблица 3. Частота устойчивости бит удовлетворяющих условию > 0.8 или < 0.2

Размерность сомножителя	Общее число бит M	Число нулевых бит T	Отношение T / M	Средняя ошибка
100	14700	4924	0,3349	0,1237
200	59400	19329	0,3254	0,1241
300	134100	46081	0,3436	0,1259
400	238800	81710	0,3421	0,1268
500	373500	128226	0,3433	0,127
600	538200	186164	0,3459	0,1275

Таблица 4. Частота устойчивости бит удовлетворяющих условию > 0.9 или < 0.1

Размерность сомножителя	Общее число бит M	Число нулевых бит T	Отношение T / M	Средняя ошибка
100	14700	757	0,0514	0,0844
200	59400	441	0,0074	0,0879
300	134100	6014	0,0448	0,0858
400	238800	9524	0,0398	0,0864
500	373500	14973	0,04008	0,0866
600	538200	27003	0,0501	0,0883

Как видно из полученных результатов, отношение числа нулевых бит к общему числу бит в целом остаётся постоянным при увеличении размерности задачи. Средняя ошибка также стабилизирована возле определённого значения.

7. Заключение

В работе представлен двухэтапный метод поиска решения задачи ВЫПОЛНИМОСТЬ. Первый этап заключается в сегментом генетическом алгоритме, второй – в методе последовательных приближений. Произведено тестирование метода для различных размерностей. Выяснено, что при применении двухэтапного метода число верных скобок достаточно велико. В таком случае для оставшихся скобок возможно применение перебора с использованием таблицы частот устойчивости бит.

Литература

1. Cook S.A., Mitchel D., G. Finding hard instances for the satisfiability problem: A survey. DIMACS series in discrete mathematics and theoretical computer science. V. 5. 1997.
2. Хныкин И.Г. Минимизация функционалов, ассоциированных с задачей ВЫПОЛНИМОСТЬ: Дис. канд. техн. наук // Омск, 2009. – С.38-42.
3. Holland J.H. Adaptation in natural and artificial systems / The University of Michigan Press, 1975.
4. Скворцов Е.С. VEGAS – новый генетический алгоритм для решения задачи ВЫПОЛНИМОСТЬ. Известия Уральского Государственного Университета №74, 2010.
5. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. – Reading: Addison Wesley, 1989.
6. Еремеев А.В., Генетические алгоритмы и оптимизация. // Учебное пособие. – Омск: Изд-во Омского гос. университета, 2008. С.16-24.
7. Дулькейт В.И. КНФ представления для задач факторизации, дискретного логарифмирования и логарифмирования на эллиптической кривой: Дис. канд. физ.-мат. наук // Омск, 2010. с.27-71.

Параллельный метод LU-SGS для трехмерных задач газовой динамики со сложной геометрией на вычислительных системах с многими графическими ускорителями

П.В. Павлухин[1], И.С. Меньшов[1,2]

Московский Государственный Университет им М.В. Ломоносова[1], Институт прикладной математики им М.В. Келдыша Российской академии наук[2]

В статье представлен эффективный алгоритм параллельного расчета трехмерных газодинамических течений со сложной геометрией на вычислительных системах с многими графическими ускорителями. Неявная схема, лежащая в основе алгоритма, приводит к большим линейным системам с сильно разреженной матрицей, которые решаются методом приближенной факторизации LU-SGS (Lower-Upper Symmetric Gauss-Seidel). Параллельный алгоритм точно воспроизводит работу последовательного и обладает высокой степенью масштабируемости. Для описания геометрии расчетной области используется метод VOF (Volume Of Fluid).

1. Введение

В механике жидкостей и газов в рамках модели уравнений Навье-Стокса задачи прямого численного моделирования требуют для решения значительных вычислительных и временных ресурсов. Экстенсивное наращивание производительности вычислительных систем за счет повышения тактовой частоты процессоров закончилось несколько лет тому назад, когда она вплотную приблизилась к отметке в 4 Ггц, поэтому единственной возможностью вести расчеты за приемлемое время осталось одновременное использование большого числа вычислительных ядер. Но здесь возникают две фундаментальные проблемы. Первая из них – это построение корректного параллельного алгоритма: результат его работы должен быть идентичен работе последовательного. Многие параллельные реализации численных методов (как правило, неявных) соблюдают эту идентичность лишь на определенных подобластях исходной расчетной области. Как результат, в процессе вычислений накапливаются ошибки, связанные с различиями в последовательной и параллельной версиях алгоритма. Дополнительно, сама архитектура вычислительной системы накладывает ограничения на структуру параллельного алгоритма. Вторая проблема – масштабируемость параллельных программ. Как правило, они могут эффективно использовать лишь порядка сотни процессорных ядер и 2 – 3 графических ускорителя, установленных внутри одного узла. Падение эффективности при задействовании бóльшего числа ресурсов связано как со скоростными характеристиками сетей передачи данных вычислительной системы, так и с ограниченной масштабируемостью самого параллельного алгоритма (безотносительно его конкретных реализаций).

Все чаще современные суперкомпьютеры используют гибридную структуру – в дополнение к «классическим» процессорным ядрам устанавливаются графические ускорители, которые и дают основной вклад в суммарную производительность системы (на многих приложениях производительность GPU на 1 – 2 порядка превосходит производительность многоядерных центральных процессоров). Однако сложность массивно-параллельной архитектуры графических плат значительно затрудняет написание программ для них. В абсолютном большинстве случаев программы для GPU используют только явные схемы с простыми вычислительными ядрами и способны задействовать лишь несколько ускорителей, находящихся внутри одного узла. Помимо сложностей программной модели, архитектура мультитредовых графических процессоров обуславливает трудности при построении алгоритмов для них – в частности, отсутствие механизма глобальной синхронизации потоков является одной из основной причин отсутствия реализаций неявных методов для задач механики жидкостей и газов на GPU.

Проблема построения алгоритмов одновременного счета на узлах системы с распределенной памятью без единого адресного пространства уже является общей для CPU и GPU архитек-

тур. Для неявных схем она решается, как правило, лишь частичным соблюдением работы последовательного алгоритма на подобластях расчетной области, т.е., вообще говоря, параллельный алгоритм нарушает работу последовательного. Версии параллельных алгоритмов с идентичной последовательной работой, как, например, в [1], имеют уже теоретическое ограничение по масштабируемости, которая еще более ухудшается при программной реализации под конкретную вычислительную систему.

Наряду с рассмотренной выше проблемой особую важность имеет также подготовка исходных данных задачи, а именно построение сеточной геометрии расчетной области. В задачах со сложной геометрией этот процесс трудно автоматизируется, поэтому «ручное» построение высококачественных расчетных сеток по времени зачастую сравнивается с полным счетом всей задачи. Альтернативный предлагаемый подход основан на методе Volume of fluid (VOF) [2 – 4], в котором используются простые структурированные сетки, а вся информация о геометрии (поверхностях) расчетной области описывается т.н. характеристической функцией. Данный подход позволяет вести эффективный расчет трехмерных задач со сложной геометрией на GPU без затрат времени на построение сеток.

Большинство программных реализаций численных методов, во-первых, ограничивают размер расчетной области суммарной памятью доступных графических плат, и во-вторых, для запуска задач с начальными данными объемом более десятка гигабайт минимально требуется от двух и более ускорителей. Эти два ограничения связаны с тем, что для хранения данных в процессе счета (за исключением пересылаемых между ускорителями т.н. ghost-ячеек) используется только видеопамять GPU, составляющая в последних моделях не более 5 - 6 ГБ, что более чем на порядок меньше оперативной памяти в вычислительных узлах современных кластеров. Между тем, карты серии Nvidia Tesla поддерживают возможность одновременного выполнения трех операций: счет вычислительного ядра и копирование данных между оперативной памятью и памятью GPU в обоих направлениях. Благодаря этому можно снять ограничение по объему видеопамяти, выполняя одновременно *выгрузку* из GPU уже обработанных данных, *расчет* над предварительно загруженными данными и *загрузку* в GPU для последующего расчета новой порции. Таким образом, если в этой схеме время перемещения данных по шине pci-express будет меньше соответствующего времени расчета, то без потерь производительности можно задействовать для хранения данных всю доступную оперативную память, не ограничиваясь лишь объемом видеопамяти.

В настоящей работе будет рассмотрен параллельный алгоритм неявного метода LU-SGS (Lower-Upper Symmetric Gauss-Seidel) [5 – 7] для трехмерных задач газовой динамики в областях со сложной геометрией, описываемой VOF-методом. Его реализация выполняется с помощью технологии CUDA и MPI для multi-GPU кластеров с возможностью использования всей доступной оперативной памяти.

2. Численный метод

Рассматривается модель сжимаемой жидкости, определяемая системой уравнений Навье-Стокса в декартовых координатах, которые дискретизируются по пространственным переменным методом конечного объема. Применяемая явно-неявная схема интегрирования по времени приводит к системе дискретных уравнений, решаемой методом установления по псевдовременной переменной с использованием неявной дискретизации и ньютоновских итераций. Таким образом, для определения итерационного инкремента $\delta^s \vec{q}$ получается следующая линейная система:

$$(D+L+U)\delta^s \vec{q}_i = -\vec{R}_i^{n+1,s} \tag{1}$$

где D – блочно-диагональная матрица, а L и U - блочно-диагональные нижняя и верхняя треугольные. Уравнение (1) преобразуется к следующему виду:

$$(D+L)D^{-1}(D+U)\delta^s \vec{q}_i = -\vec{R}_i^{n+1,s} - LD^{-1}U \tag{2}$$

Для систем такого вида в методе LU-SGS выполняется приближенная факторизация левой части уравнений (1), которая получается, если в уравнениях (2) пренебречь последним слагае-

мым правой части (которое фактически пропорционально Δt^2). После факторизации уравнения распадаются на две подсистемы:

$$\begin{aligned}(D+L)\delta^s\vec{q}_i^{\,*} &= -\vec{R}_i^{n+1,s}\\ (D+U)\delta^s\vec{q}_i &= D\delta^s\vec{q}_i^{\,*}\end{aligned} \tag{3}$$

Первая система уравнений в (3) имеет нижне-треугольную блочную матрицу, каждый элемент которой представляется блоком размерностью (5 x 5), а вторая – соответственно верхне-треугольную. Такая структура матриц систем позволяет эффективно вычислить их решения за два расчетных цикла по ячейкам: первый - в прямом направлении (от первой ячейки к последней), а второй - в обратном. При этом необходимо обращать только диагональные блоки, т. е., матрицы размером (5 x 5). Получающаяся при этом итерационная невязка $\delta^s\vec{q}$ служит для обновления итерационного вектора, после чего процедура (3) повторяется. Более подробное описание метода приведено в [5 – 7]

Для счета задач со сложной геометрией применяется VOF-метод [2 – 4], который, в частности, используется для отслеживания поверхностей-границ в многокомпонентных течениях. В расчетной области предполагается наличие непроницаемых твердых тел со сложной геометрией. Изначально вся область погружается в т.н. расчетный куб, для которого строится декартова сетка (со сгущением координатных линий там, где это необходимо). Затем вычисляется характеристическая функция, определяющая для каждой ячейки куба объемную долю твердого тела, содержащегося в ней:

$$f_i = \frac{V(c_i \cap G)}{V(c_i)} \tag{4}$$

где c_i - i-*ая* ячейка, G - область твердого тела, V - объем. Следовательно, $0 \le f_i \le 1$, и $f_i = 1$ только для ячеек, полностью находящихся в области твердого тела, а $f_i = 0$ - для ячеек внутри расчетной жидкости. Непосредственно в расчетном цикле исходная информация о геометрии уже не используется. Она восстанавливается для ячеек с $0 < f_i < 1$ по локальным значениям характеристической функции в текущей и соседних ячейках. В результате процедуры восстановления исходная поверхность твердого тела, пересекающего ячейку, приближается проходящей через ячейку плоскостью. Восстановление поверхности выполняется одним из ранее разработанных методов - SLIC, CM (центральных масс), CD (центральных разностей), LVIRA/ELVIRA [4].

Таким образом, выполняется сквозной однородный расчет методом LU-SGS по всем ячейкам куба с последующей коррекцией значений вектора состояния в тех ячейках, где $0 < f_i < 1$, методом внутренней границы [8]. Данный подход эффективно реализуем для работы на массивно-параллельных архитектурах при условии корректного счета предлагаемой параллельной версией метода LU-SGS.

3. Параллельная версия метода LU-SGS

Если рассматривать алгоритм LU-SGS как объект для распараллеливания, то его последовательный вариант выглядит как два прохода (прямой и обратный) по ячейкам расчётной области. Оба прохода осуществляются по одной последовательности ячеек, но в разных направлениях. В общем случае значения искомых функций в ячейке на следующем временном слое зависят от всех ячеек расчётной области на текущем временном слое, что затрудняет построение параллельного алгоритма.

При обращении к геометрически соседней ячейке вычислительные операции в текущей зависят от того, где располагается в выполняемом обходе эта соседняя ячейка. Если она находится до текущей (т.е. уже была обсчитана), то выполняется один блок вычислений, если после текущей (т.е. еще не была обсчитана), то выполняется другой блок вычислений, и при этом происходит обновление данных в соседней ячейке. При построении параллельного алгоритма необходимо сохранить эту линейную упорядоченность ячеек, т.е. гарантировать, чтобы каждая соседняя ячейка была всегда уже обсчитанной или еще не обсчитанной при обращении к ней.

Особенностью рассматриваемого метода является возможность произвольного выбора порядка обхода ячеек – не обязательно, чтобы две последовательные ячейки в обходе были бы геометрически соседними. Этим мы и воспользуемся для построения специального обхода. Будем рассматривать трехмерные расчетные области с структурированными сетками, где каждая расчетная ячейка может иметь 6 ячеек-соседей – по числу своих граней.

Расчетная область разбивается на блоки с примерно равным числом ячеек, каждый из которых будет считаться на отдельном GPU. Их взаимное расположение аналогично структуре ячеек в сеточном разбиении – «стык в стык». Затем выполняем раскраску блоков двумя цветами так, чтобы два блока с общей гранью были разных цветов. В итоге получаем трехмерное «шахматное» разбиение блоков на два типа – «черные» и «белые».

В каждом блоке выделим две части ячеек – граничные, прилегающие к граням блока и остальные - внутренние. Построим глобальный обход ячеек в блоках следующим образом. Сначала обойдем все внутренние в каждом из «черных» блоков, затем половину внутренних и граничные в «белых», потом граничные в «черных» и, наконец оставшуюся вторую половину в «белых» (схема обхода показана на Рис. 1). Данный обход позволяет вести параллельный счет в блоках идентично соответствующему последовательному прототипу. При этом передача ghost-ячеек между соседними блоками совмещается со счетом. Доказательство корректности такой параллельной схемы представлено в [9].

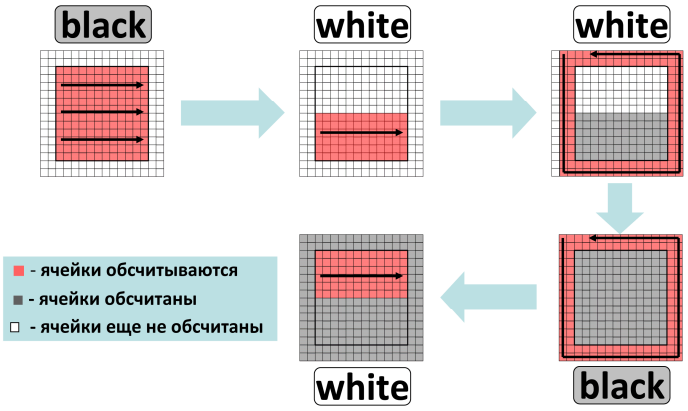


Рис. 1. Порядок обхода ячеек в «черных» и «белых» блоках

Представленный выше алгоритм описывает корректный счет на уровне разделения блоков по нескольким GPU, дополнительно необходимо сохранить корректность при счете и внутри каждого GPU. Средств глобальной синхронизации, с помощью которых это можно было бы достичь, в современных графических ускорителях нет. Однако есть способ обойтись без нее – разбить граничные и внутренние части блоков на несколько подмножеств, внутри каждого из которых ячейки считаются независимо.

Разбиение ячеек на подмножества основано на их раскраске разными цветами так, чтобы соседние по общей грани ячейки всегда были разных цветов. В случае трехмерных структурированных сеток в такой раскраске достаточно использовать только два цвета, при этом получается т.н. red-black разбиение [10], образующее, как и в случае разбиения блоков, «объемную шахматную» доску.

Будем выполнять последовательный обход сначала всех ячеек из black, затем – всех оставшихся из red. Тогда эквивалентный параллельный счет будет состоять из двух этапов – сначала одновременный счет всех ячеек из black и после – из red. Нетрудно видеть, что в этом случае сохраняется линейная упорядоченность между всеми соседними ячейками, поэтому результат работы построенного параллельного алгоритма будет совпадать с результатом работы последовательного.

Таким образом, корректный параллельный алгоритм метода LU-SGS для систем с многими графическими ускорителями полностью построен.

4. Реализация и результаты

Программный код для расчета трехмерных задач реализуется с помощью библиотеки MPI и CUDA ToolKit. Из-за вычислительной сложности ядра его полная сборка возможна только с версиями 4.2 и 5.0 компилятора nvcc, при использовании более старых версий компиляция завершалась с ошибкой на этапе конечной оптимизации ассемблерного кода. Red и black ячейки хранятся в отдельных массивах для более эффективного обращения к ним. Для совмещения счета и передачи данных ячеек во времени использовались неблокирующие функции приема/передачи MPI и несколько CUDA-«потоков» (Stream). В одном из них выполнялось вычислительное ядро (kernel), в других параллельно шло копирование данных. Для синхронизации вычислений использовались функции MPI_Wait и функции работы с событиями (event) в API CUDA. Все вычисления проводились на GPU, ядро центрального процессора использовалось лишь для копирования данных между оперативной памятью и памятью ускорителя, а также для обмена данными между процессами посредством MPI.

Чтобы не ограничиваться только памятью GPU, реализуется следующая схема счета. Блок ячеек делится на части, по объему занимающие 1/3 памяти одного GPU. Ячейки последовательно обрабатываются на GPU: предыдущая (обсчитанная) часть ячеек копируется из видеопамяти в память CPU, текущая часть используется вычислительным ядром на GPU, и из памяти CPU в видеопамять копируется следующая часть ячеек. Эти три операции выполняются одновременно. Таким образом, для счета задач на GPU возможно использовать всю доступную оперативную память, не ограничиваясь лишь малым объемом видеопамяти.

Представленная на конференции ПАВТ-2012 реализация двумерного кода [11] показала эффективность предлагаемых алгоритмов и подходов при расчете задач с использованием большого числа GPU (Рис. 2). Реализация трехмерного кода ведется в настоящее время. Результаты multi-GPU расчетов будут доложены на конференции. Полученные предварительные результаты показывают, что время счета типовой задачи газодинамики (трехмерная фокусировка ударной волны) с разрешением 100x100x100 оказалось на одном GPU (Nvidia Tesla C2070) в 30 раз меньше по сравнению с одним ядром CPU (Intel Xeon X5670).

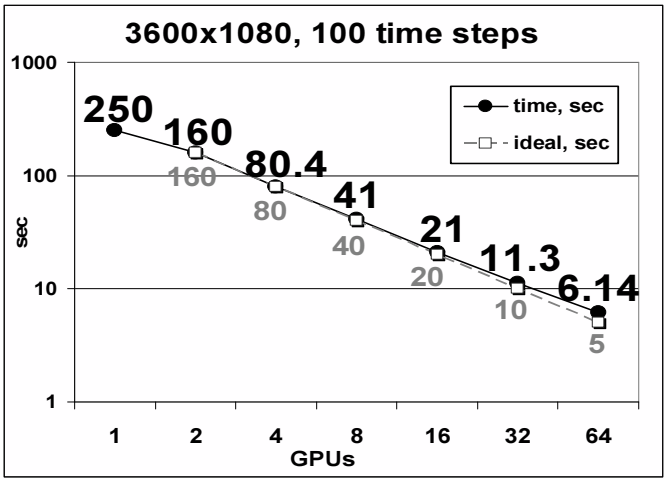


Рис. 2. Время счета двумерной задачи с расчетной сеткой 3600x1080 с разным числом GPU

5. Заключение

В работе представлен параллельный алгоритм, реализующий метод LU-SGS для трехмерных задач газовой динамики со сложной геометрией, описываемой VOF-методом, на кластерной вычислительной системе распределенных графических ускорителей. Показана корректность этого алгоритма, описана его реализация с помощью технологий MPI и CUDA, позволяющая задействовать всю доступную оперативную память, не ограничиваясь объемом видеопа-

мяти. Двумерная версия кода показала масштабируемость, близкую к линейной. Предварительные результаты для трехмерной версии дают 30-кратный рост производительности относительно одного процессорного ядра.

Литература

1. И. В. Семенов, И. Ф. Ахмедьянов. Разработка параллельного алгоритма LU-SGS для решения многомерных задач вычислительной газодинамики // Материалы Четвертой Сибирской школы-семинара по параллельным и высокопроизводительным вычислениям. Изд-во Томск: Дельтаплан, 2008. -- С. 122-129, 2008.
2. Hirt, C.W.; Nichols, B.D. (1981), "Volume of fluid (VOF) method for the dynamics of free boundaries", Journal of Computational Physics 39 (1): 201–225
3. W.J. Rider, D.B. Kothe, Reconstructing volume tracking, J. Comput. Phys. 141 (2) (1998) 112–152.
4. J.E. Pilliod Jr., E.G. Puckett, Second-order accurate volume-of-fluid algorithms for tracking material interfaces // Journal of Computational Physics 199 (2004) 465–502
5. A. Jameson, E. Turkel Implicit schemes and LU decomposition // Math.of Comp., v.37, № 156, pp.385-397, 1981.
6. I. Menshov, Y. Nakamura On implicit Godunov's method with exactly linearized numerical flux //Computers & Fluids, 29 (6), pp. 595 – 616, 2000.
7. I. Menshov, Y. Nakamura Hybrid Explicit-Implicit, Unconditionally Stable Scheme for Unsteady Compressible Flows //AIAA Journal,vol. 42, № 3, pp. 551-559, 2004.
8. И. С. Меньшов. Метод свободной границы для расчета задач динамики газа и твердого тела. Материалы XXII Научно-технической конференции по аэродинамики ЦАГИ, 3-4.03.2011, п. Володарского МО, с.108-109, 2011.
9. П.В. Павлухин, И.С. Меньшов. Эффективная реализация метода LU-SGS для задач газовой динамики. //Научный вестник МГТУ ГА, Т.165, 3, с.46 - 55, 2011
10. T. Iwashita, M. Shimasaki, Block Red-Black Ordering Method for Parallel Processing of ICCG Solver // 4th International Symposium, ISHPC 2002 Kansai Science City, Japan, May 15–17, 2002 Proceedings, pp 175-189, 2006
11. П. В. Павлухин. Реализация параллельного метода LU-SGS для задач газовой динамики на кластерных системах с графическими ускорителями //Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ’2012, Новосибирск, 26 – 30 марта 2012 г.), с. 627-634, 2012

Применение параллельных и распределенных вычислительных систем в радиометеорологии

А.В. Панюков

Южно-Уральский государственный университет

В связи с внедрением новых технологий грозорегистрации и накоплением разнообразных данных о грозовой активности облаков возможным оказывается исследование микрофизических, динамических и электрических свойств электроактивных зон и их взаимосвязи с опасными явлениями, возникающими в наблюдаемых облаках. Обоснование предикторов опасных явлений, связанных с мощной конвекцией, имеет большое практическое значение при их реализации в национальной грозопеленгациоиной сети. Внедрение новых технологий грозорегистрации и сравнение получаемых при этом данных с данными существующих многоволновых активно-пассивных комплексов дает возможность исследовать микрофизические, динамические и электрические свойства электроактивных зон и их взаимосвязь с опасными явлениями в облачной атмосфере. Накопление данных и обоснование предикторов явлений, связанных с мощной конвекцией и имеющих электрическую природу, приобретает большое практическое значение для их реализации в национальной грозорегистрациопной сети. Получаемый при этом колоссальный объем информации требует, для ее обработки и применения ультрасовременной вычислительной техники и современного программного обеспечения. Особенность применения вычислительной техники в радиометеорологии состоит в том, что первичная обработка и возможность её экспресс анализа особенно в оперативной работе, должны осуществляться непосредственно в информационной радиоизмерительной системе, причем в реальном масштабе времени. В докладе отражены некоторые достижения в развитии аппаратного и программного обеспечения радиометеорологических систем.

1. Введение

Применение в метеорологии радиолокаторов работающих в пассивном режиме, принцип действия которых основан на приеме собственного излучения атмосферы, относится к концу 50-х началу 60-х годов прошлого столетия [1], [2]. Особую значимость для исследования облачности и осадков в метеорологии приобрела разработка методов, основанных на совместном применении активно-пассивных комплексов, в состав которых входят метеорологический радиолокатор, работающий в активном режиме, и микроволновая радиометрическая аппаратура. Создание таких комплексов было осуществлено совместно ИРЭ АН СССР и ЦАО в середине 60-х годов прошлого столетия [3].

За прошедшие 60-65 лет радиометеорология в мире получила бурное развитие. Достаточно указать, что в США каждые 1,5-2 года, регулярно проводятся радиометеорологические конференции. В настоящее время резко расширился круг задач, который пытаются решить при помощи радиолокации. Так помимо традиционных направлений, связанных с изучением процессов протекающих в различного типа и масштаба облачных образованиях и системах, появилось направление, связанное с изучением «тонкой структуры» и безоблачной атмосферы, недоступной для исследования при помощи иных средств и методов.

Расширение направлений, по которым развивается современная радиометеорология, стало возможным за счет резкого увеличения потенциала радиолокационных станций, их установки не только на Земле, но и на самолетах, спутниках.

2. Оперативное гидрометеобеспечение

В области оперативного гидрометобеспечения в последние 10-15 лет все большее внимание фокусируется на практических вопросах грозового электричества: предупреждение об опасных явлениях погоды, связанных с мощными конвективными облаками (грозы, смерчи-торнадо, град, шквалы, избыточные осадки); прогноз поражения самолета молниями; защиты линий электропередач от попадания молний; уточнение количества выпадающих осадков при наблюдениях зон грозовой активности с ИСЗ; предупреждение о возможности возникновения лесных пожаров; прогноз эволюции тропических циклонов и другие [4] – [6]. Этому способствует развитие и внедрение в оперативную практику инструментальных методов обнаружения гроз - систем и датчиков типа ALDF, LDAR, SAF1R, OLS, US и т.д.

В результате начали накапливаться фактические данные об особенностях электрической активности мощных конвективных облаков, которые в силу определенных причинно-следственных связей и с некоторой заблаговременностью дают информацию о существенной микрофизической перестройке облака и возможных опасных явлениях, т.е. на их основе становится возможным формировать информационные признаки (предикторы) опасных явлений. К таким предикторам можно отнести резкое увеличение числа внутриоблачных молний (60 разрядов в минуту и более) за 10-15 мин до появления торнадо или за 5-10 мин до формирования опасных для авиации шквалов (микробарстов). В градовых облаках в период зарождения и выпадения града часто происходит реверс полярности молний с преимущественно отрицательной на положительную с возвращением обратно после окончания града. Очевидно, что исследование физических механизмов, временных параметров и устойчивости таких связей в различных синоптических и физико-географических условиях способствует повышению качества диагноза и прогноза степени опасности облаков и атмосферных процессов в целом.

В работе [7] приведены результаты наблюдений за изменениями электромагнитного излучения молниевых разрядов при развитии грозового очага в Подмосковье и эволюции глубокого циклона с конечными координатами центра вблизи побережья Кольского полуострова. Предложены методика обработки и статистические оценки полученных результатов по характеристикам потока атмосфериков в виде разности и отношения между числами сигналов с разной полярностью. Установлено соответствие между аномальными изменениями этих характеристик при вихревых явлениях и генерацией атмосфериков преимущественно с отрицательной полярностью.

В работах [5] – [10] рассмотрены методические вопросы дистанционного зондирования грозовых облаков применительно к решению задач оперативного гидрометобеспечеиия, включая активные воздействия иа гидрометеорологические процессы. В связи с внедрением новых технологий грозорегистрации и накоплением разнообразных данных о грозовой активности облаков формулируется цель: исследование микрофизических, динамических и электрических свойств электроактивных зон и их взаимосвязи с опасными явлениями, возникающими в таких облаках. Обоснование предикторов опасных явлений, связанных с мощной конвекцией, имеет большое практическое значение при их реализации в национальной грозопеленгациоиией сети.

Таким образом, внедрение новых технологий грозорегистрации и сравнение получаемых при этом данных с данными существующих многоволновых активно-пассивных комплексов дает возможность исследовать микрофизические, динамические и электрические свойства электроактивных зон и их взаимосвязь с опасными явлениями в облачной атмосфере. Накопление данных и обоснование предикторов явлений, связанных с мощной конвекцией и имеющих электрическую природу, приобретает большое практическое значение для их реализации в национальной грозорегистрациопной сети.

3. Системы пассивного мониторинга грозовой деятельности

Обзор состояния систем пассивного мониторинга грозовой деятельности к концу 2003 г. и демострация возможности использования систем местоопределения молниевых разрядов для пассивной радиолокации опасных метеорологических явлений представлены в работе [11]. Современные методы анализа поля, позволяющие определить параметры источника ЭМИ, характеризующие его размещение и ориентацию представлены в работах [12] – [22]. Математической моделью, адекватно представляющей внутриоблачные молниевые разряды, является электрический диполь над плоскостью с бесконечной проводимостью. 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 [13], [14], [16], [21], [22].

В рамках проекта МНТЦ №1822 (http://www.istc.ru/istc/db/projects.nsf) разработан образец однопунктового грозопеленгатора-дальномера по патенту РФ №2230336 [20]. В результате проведенных полевых испытаний данного образца в период май – август 2004 г. было зарегистрировано более 2,5 млн. атмосфериков. Из них не более 10% были классифицированы как излучение от молниевого разряда, остальные были классифицированы как предгрозовое излучение.

Регистрация предгрозового излучения облаков (т.е. до первой вспышки молнии) однопунктовой системой грозолокации, для целей прогнозирования развития грозы, было обеспечено расширением динамического диапазона приемной аппаратуры и дальнейшим развитием математического и программного обеспечения [18]. Ранее подобное предгрозовое излучение либо отфильтровывалось, либо отрабатывалось некорректно однопунктовыми системами местоопределения гроз.

База данных полевых испытаний дает большой экспериментальный материал для проверки адекватности моделей грозового очага и тестирования разрабатываемых программного обеспечения и устройства.

В целом результаты проекта продемонстрировали возможность и необходимость создания нового поколения систем местоопределения гроз и расширения круга решаемых ими задач [23]. Это ведет не только к существенному пересмотру требований к их техническим характеристикам, но и к разработке новых математических моделей и алгоритмов анализа грозовых явлений, их трассирования и отображения, а также ведения архивов и их использования специалистами разных предметных областей.

3.1. Однопунктовые системы

При использовании однопунктовых систем пассивного мониторинга грозовой активности возникает неустранимая ошибка [12] – [15] обнаружения пеленга на разряд, эквивалентный дипольный момент которого не является вертикальным. Отличие ориентации эквивалентного источника от вертикали наиболее характерно как раз для внутри облачных и межоблачных разрядов, образующих предгрозовое излучение.

В работах [24] – [28] предложены методы снятия данной неопределенности с помощью карты плотности вероятности, для построения которой используется все множество зарегистрированных разрядов. Дополнительным ресурсом повышения качества решения данной задачи, является получение более точных значений параметров определяющих местоположение и ориентацию эквивалентного источника каждого разряда.

Другим способом устранения возникающей неопределенности, являются методы кластеризации [29], [30]. Подобный подход позволяет определить количество грозовых ячеек, их центры, выявить разряды входящие в каждую из ячеек и определить возможное размещение для каждого разряда. Область, которая покрыта кластером, является потенциально опасной.

3.2. Многопунктовые системы

Из требований надежности и живучести грозопеленгационной сети следует необходимость использования многопунктовых систем автономных грозопеленгаторов-дальномеров. Алгоритмы многопунктового определения параметров положения дипольного источника излучения описаны в работах [21], [31]. В патенте [32] предложены способ применимый для многопунктового анализа предгрозового излучения и архитектура многопунктовой системы местоопределения молниевых разрядов на основе сети автономных грозопеленгаторов-дальномеров [20]. В работе [33] рассмотрен круг вопросов, связанный с программным обеспечением [34] пространственно распределенной вычислительной системы на примере геоинформационной сети анализа и прогнозирования грозовой активности. Приведено описание и функции отдельных блоков, рассмотрены связи между ними. Очерчены перспективы развития системы и возможности интеграции с существующими аналогами.

Главной задачей решаемой многопунктовой географически распределенной системой является определение параметров положения дипольного источника излучения в реальном времени. Потребителями информации могут быть: метеослужбы, службы обеспечения безопасности полетов, электроэнергетические, строительные и страховые компании, службы по борьбе с лесными пожарами, научные организации по исследованию физики атмосферы. Основная цель системы – создание системы мониторинга электромагнитного поля Земли в реальном времени, предназначенной для определения координат молниевых разрядов, происходящих в наблюдаемом районе. Вторичными целями являются: (1) повышение уровня автоматизации сбора данных со станций наблюдения; (2) предоставление доступа пользователям как к оперативной информации о грозовой обстановке, так и к архиву наблюдений; (3) повышение вероятности обнаружения молниевых разрядов; (4) ведение базы данных о грозовых явлениях.

4. Развитие информационного обеспечения радиометеорологии

Таким образом, радиолокационная метеорология подошла к этапу, характеризующемуся тем, что она может и должна не только выполнять «наблюдательные функции», но и превращается в настоящую измерительную систему. Измерения того или иного метеорологического параметра осуществляется с указанием достоверности и точности его измерения. И в конечном итоге полученные результаты предоставляются в величинах, которыми обычно пользуются метеорологи или специалисты занимающиеся изучением строения и динамических процессов, протекающих в атмосфере. Во всех эмпирических измерениях, существует определенная вероятность погрешности между измеренным и истинным значением. На результаты измерения могут оказать влияние многие факторы. Для исследования этих факторов производятся многочисленные замеры. Затем эти данные можно представить в виде гистограммы и получить эмпирическое распределение вероятности данной величины. Исходя из центральной предельной теоремы, можно предположить, что величина может быть распределена по нормальному закону и тогда самой лучшей оценкой измеренной величины будет среднее значение, которое находиться в центре распределения. Заметим, что для повышения статистической значимости оценки параметров следует использовать по возможности большее число алгоритмов.

Среди методов определения дальности до молниевого разряда, использующие в качестве математической модели молниевого разряда произвольно ориентированный электрический диполь, можно выделить прямой [13], [16], [11] и экстремальный методы [18], которые обладают достаточно высокими точностными характеристиками определения дальности и дают гарантированные оценки угловых координат эквивалентного дипольного источника, а при принятии дополнительных гипотез – их распределение вероятности. Увеличить число используемых алгоритмов для получения статистически значимых оценок можно за счет применения параметризации. Одним из способов такой параметризации [35], [36] является

применение параметризованного семейства полосовых фильтров для предобработки входных сигналов $E_z(t)$, $H_x(t)$, $H_y(t)$ от электромагнитного излучения молниевого разряда, зарегистрированных с помощью антенной системы, содержащей вертикальную электрическую антенну и пару ортогональных рамочных магнитных антенн. В работах [35], [36], [37] описаны способы применения многопроцессорных систем для эффективного выполнения множества параметризованных алгоритмов.

Появление технологий вычисления в гетерогенных средах, а также соответствующего оборудования позволило значительно расширить область применения статистических методов [38], [39] для решения обратных задач. Появилась возможность решать в реальном времени неустойчивые задачи для которых потребовалось бы значительно больше времени в случае использования только ресурсов CPU мобильных систем. Техника параллельной реализации на гетерогенной вычислительной системе параметрических алгоритмов вычисления параметров электрического диполя и статистического оценивания множества значений этих параметров изложена в работе [40].

5. Заключение

В связи с внедрением новых технологий грозорегистрации и накоплением разнообразных данных о грозовой активности облаков потенциально возможным становится исследование микрофизических, динамических и электрических свойств электроактивных зон и их взаимосвязи с опасными явлениями, возникающими в таких облаках. Обоснование предикторов опасных явлений, связанных с мощной конвекцией, имеет большое практическое значение при их реализации в национальной грозопеленгациоиной сети. Сложность решения задачи в такой постановке состоит в том, что информация, полученная при помощи метеорологических радаров настолько объемна и неоднозначна, что ее очень сложно сопоставить с данными, которые получены при помощи традиционных метеорологических приборов и обычно применяемой методики измерений. Поэтому проблемы постановки таких сравнительных измерений требуют к себе особого внимания и в конечном итоге приводят к новым открытиям.

Литература

1. Жевакин С. А., Троицкий В. С. Радиоизлучение атмосферы и исследование поглощения сантиметровых волн. // Изв. ВУЗов. Радиофизика. – 1958. – №2.

2. Жевакин С. А., Наумов А. П. Поглощение сантиметровых и миллиметровых радиоволн атмосферными парами воды. // Радиотехника и электротехника. – 1964. – №8.

3. Башаринов А.Е., Горелик А.Г., Калашников В.В., Кутуза Б.Г. Определение параметров облаков и дождя по радиоизлучению на волне 0,8 см. // Тр. ЦАО, вып. 86. – 1969.

4. Мареев Е. А., Стасенко В. Н. Российские исследования в области атмосферного электричества в 2003-2007 гг. // Известия РАН. Физика атмосферы и океана. – Т. 45, № 55. – 2009. – С. 709-720.

5. Горелик А. Г. Радиолокационная метеорология и перспективы ее развития. // III Всероссийская конференция «Радиолокация и радиосвязь» (Москва, 26–30 октября 2009 г.). – М: ИРЭ РАН. – 2009. – С. 400 – 404.

6. Стасенко В. Н., Гальперин С. М., Степаненко В. Д., Щукин Г. Г. Методология исследования грозовых облаков и активных воздействий на них // Проектирование и технология электронных средств. Спец. вып. – 2004. – С. 2–6.

7. Мигунов Н. И., Московенко В. М., Росанов Н. И. Наблюдения за молниевыми разрядами при вихревых явлениях // Метеорология и гидрология. – 2005. – № 1.

8. Кононов И. И., Ришар Ф. Методы пассивной локации гроз. / Тр. V Всерос. Конф. по атмосферному электричеству. – Владимир, 2003. – Т. 1. – С. 8-12.

9. Горбатенко В. П., Ипполитов И. И., Кабанов М.В. и др. Анализ структуры временных рядов повторяемости форм атмосферной циркуляции и грозовой активности // Оптика атмосферы и океана. – 2002. – Т. 15. № 8. – С. 693-697.

10. Кононов И.И., Петренко И.А., Снегуров В.С. Радиотехнические методы местоопределения грозовых очагов. – Л.:Гидрометеоиздат, 1986.

11. 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.

12. Panyukov A. V. Analytical and Computational Study of the Lightning Location under Single-point Observation of Electromagnetic Field // Proceedings 23rd International Conference on Lightning Protection . Vol. 1. (Firenze, Italy, September 23 – 27, 1996). – AEI. – P. 252 – 257.

13. Panyukov A.V. Estimation of the Location of an Arbitrary Oriented Dipole under Single-point Direction Finding // Journal of Geophysical Research. – 1996, June 27. – Vol. 101. – №D10.

14. 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.

15. Panyukov A. V. Lightning Detection and Mapping Algorithms // Proceedings 24th International Conference on Lightning Protection . Vol. 1. (Birmingham, United Kingdom, September 14–18, 1998). – Staffordshire university. – P. 227–231.

16. 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.

17. Панюков А.В., Будуев Д.В. Алгоритм определения расстояния до местоположения молниевого разряда // Электричество – 2001, апрель. – Т. 4. – С. 10–14.

18. 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.

19. Панюков А. В., Будуев Д. В. Аналитическое и численное исследование устойчивости и точности алгоритмов определения дальности до дипольного источника СДВ излучения. // Известия Челябинского научного центра, вып. 4(28), 2004. – С. 15 – 20.

20. Панюков А.В., Файзулин Н.А., Будуев Д.В. Однопунктовая система местоопределения гроз в ближней зоне. – Патент РФ на изобретение №2230336. – 2004.

21. 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.

22. B.Z. Taibin. An Approach to Define Parameters for Localization of Thunderstorm // IEEE Antennas and Propagation Magazine. – 2006. – Vol. 48. – P.48–54.

23. Панюков А.В. Математическое и программное обеспечение распределенной сети грозопеленгаторов-дальномеров. // VI Российская конференция по атмосферному электричеству (Нижний Новгород, 1–7 октября 2007). – Нижний Новгород: ИПФ РАН. – С. 255 – 256.

24. Panyukov A.V., Avramenko A.G. Increasing Accuracy of Single Point Thunderstorm Locating System. // International Conference ≪Electrical and Control Technologies - 2006≫. Selected papers of conference – Kaunas. – 2006. – P. 63 – 65.

25. Богушов А.К., Панюков А.В. Применение методов вторичной обработки информации о грозовой активности для предупреждения аварий // Безопасность критичных инфраструктур и территорий: Материалы III Всероссийской конференции и XIII Школы молодых ученых. - Екатеринбург: УрО Ран. – 2009. – С.117 – 118.

26. Богушов А.К., Панюков А.В. Методы вторичной обработки результатов мониторинга грозоопастности // VII Российская конференция по атмосферному электричеству (Санкт-Петербург, 24–28 сентября 2012). – СПб: ГГО им. Воейкова. – С. 40 – 42.

27. Богушов А.К., Панюков А.В. Вторичная обработка результатов пассивного мониторинга грозовой деятельности // Сборник трудов 40-й молодежной школы-конференции. 2009. Екатеринбург. С. 286–290.

28. Богушов А.К. Построение карты плотности вероятности по результатам пассивного мониторинга грозовой активности // I Научная конференция аспирантов и докторантов. Материалы конференции. Апрель, 2009, Челябинск. С. 281 – 284.

29. Богушов А.К., Панюков А.В. Размещение взаимосвязанных объектов в условиях неопределенности // IV Всероссийская конференция ≪Проблемы оптимизации и экономические приложения≫: Материалы конференции (Омск, 29 июня – 4 июля, 2009) / Омский филиал Института математики СО РАН. – Омск: Полиграф. Центр КАН, 2009. – С.113.

30. Кононов И.И., Юсупов И.Е. Кластерный анализ грозовой активности.// Радиотехника и электроника, 2004, том 49, № 3, с.283–291.

31. He S., Popov M., Romanov V. Explicit Full Identification of a Transient Dipole Source in the Atmosphere from Measurement of the Electromagnetic Fields at Several Points at Ground Level // Radio Science. Vol. 35. No 1. 2000. P. 107–117.

32. Панюков А.В., Малов Д.Н. Способ определения местоположения молниевого разряда и многопунктовая система для его реализации. – Патент РФ на изобретение № 2253133. – 2005.

33. Панюков А.В., Малов Д.Н. Математическое и программное обеспечение распределенной сети грозопеленгаторов-дальномеров // Параллельные вычислительные технологии (ПАВТ'2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮурГУ, 2010. – С. 572–583. – Режим доступа http://omega.sp.susu.ac.ru/books/conference/PaVT2010

34. Панюков А.В., Малов Д.Н. Комплекс программ для сети автономных грозопеленгаторов-дальномеров. Свидетельство РосАПО об официальной регистрации программы для ЭВМ № 2002611854. // Программы для ЭВМ, базы данных, топологии

интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам № 1(42). – 2003. – С.57–58.

35. Панюков А.В., Богушов А.К. Параметризация алгоритмов идентификации электрического диполя. // Вестник Южно-Уральского государственного университета, №18 (277), серия: «Математическое моделирование и программирование», вып. 12. – С. 32 – 43.

36. Панюков А.В., Богушов А.К. Спектрально-статистический подход к проблеме идентификации параметров положения дипольного источника электромагнитного излучения // VII Российская конференция по атмосферному электричеству (Санкт-Петербург, 24–28 сентября 2012). Сборник трудов. – СПб: ГГО им. А.И. Воейкова. – С. 179 – 181.

37. Богушов А.К., Панюков А.В. Параллельная реализация комплекса программ для задачи определения параметров электрического диполя. // Параллельные вычислительные технологии (ПаВТ́2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.) – Челябинск: Издательский центр ЮУрГУ, 2011. – 730 с. С. 427 – 432. – URL: http://omega.sp.susu.ac.ru/books/conference/PaVT2011

38. Peredo O., Ortiz Ju. Parallel implementation of simulated annealing to reproduce multiple-point statistics // Computers & Geosciences. – Vol. 37, Issue 8. – August 2011, P. 1110 - 1121.

39. Suchard M.A., et all. Understanding GPU Programming for Statistical Computation: Studies in Massively Parallel Massive Mixtures / M.A.Suchard, Q.Wang, C.Chan, J.Frelinger, A.Cron, M.West // Journal of Computational and Graphical Statistics. – Vol. 19(2). – 2010 http://ftp.stat.duke.edu/WorkingPapers/10-02.pdf .

40. Богушов А.К., Панюков А.В.Применение гетерогенных вычислительных систем для решения задачи идентификации параметров положения дипольного источника излучения // Вестник Пермского университета. Математика. Механика. Информатика. – Вып. 3(11). – С.17–22.

Численное моделирование колебательных процессов центробежного насоса на кластере ПНИПУ

П.В. Писарев, В.Я. Модорский

ФГБОУ ВПО «Пермский национальный исследовательский политехнический университет»

В рамках данной работы произведены численные эксперименты с использованием высокопроизводительной вычислительной техники, по моделированию потока в соединительном канале с учетом прилегающих свободных объемов. Выявлены особенности усиления колебаний в модельных каналах при варьировании параметров потока и геометрии канала. Учитывается работа крыльчатки первой ступени центробежного насоса.

Цель исследования заключается в изучении влияния конструктивных и технологических параметров на характеристики потока жидкости в каналах центробежного насоса. На первом этапе исследования произведены численные эксперименты по моделированию потока в соединительном канале с учетом прилегающих свободных объемов. Выявлены условия усиления колебаний в модельных каналах.

В рамках данной работы, на втором этапе исследований производилось уточнение твердотельной модели и произведен учет особенностей поступления и отвода рабочего тела в полость первой ступени модельного двухступенчатого насоса. В данной работе учитывается работа крыльчатки первой ступени. Учет данных факторов вызвал необходимость использования многопроцессорной вычислительной техники.

Для решения задач по расчету гидродинамических характеристик в свободном объеме первой ступени центробежного насоса, с учетом допущений была принята следующая расчетная модель:

- конструкция полагается трехмерной (x, y, z);
- расчет нестационарный;
- стенки конструкции не поглощают и не выделяют тепло;
- сила трения между стенками и рабочей жидкостью не учитывается;
- рабочее тело представляет собой вязкую сжимаемую жидкость(вода);
- стенки конструкции не деформируются;
- сила тяжести не учитывается;
- давление на входе подается через осевой двусторонний вход;
- реализуется Лагранжев подход. При этом кинетическая энергия вращения колеса передается жидкости и приводит ее в движение;
- колесо представляет собой вращающееся тело, состоящее из втулки и 12 одинаковых непрофилированных лопастей.

Математическое описание гидродинамического процесса в указанной постановке включает в себя следующие соотношения:

-уравнение сохранения массы:

$$\frac{\partial \rho}{\partial t} + \nabla \bullet \left(\rho \vec{V}\right) = 0 \quad ; \qquad (1)$$

где ρ – плотность жидкости; $\vec{V}$ – вектор скорости жидкости; t – время;

∇ - оператор Гамильтона.

-уравнение сохранения импульса:

$$\frac{\partial(\rho\vec{V})}{\partial t}+\nabla\bullet\left(\rho\vec{V}\otimes\vec{V}\right)=-\nabla p+\nabla\bullet\tau+\vec{S}_M \quad ; \qquad (2)$$

где P-давление; $\vec{S}_M$ - источниковый член для импульса; τ- тензор напряжений, записываемый в виде:

$$\tau=\mu_e\left[\nabla\vec{V}+\left(\nabla\vec{V}\right)^T-\frac{2}{3}\delta\nabla\bullet\vec{V}\right] \qquad (3)$$

где δ - дельта-функция Кронекера, μ_e- эффективная вязкость.

$$\mu_e=\mu+\mu_t \qquad (4)$$

где μ - вязкость, μ_t - турбулентная вязкость.

-уравнение сохранения энергии:

$$\frac{\partial(\rho h_{\text{tot}})}{\partial t}-\frac{\partial p}{\partial t}+\nabla\bullet\left(\rho\vec{V}h_{\text{tot}}\right)=\nabla\bullet\left(\lambda\nabla T\right)+\nabla\bullet\left(\vec{V}\bullet\vec{\tau}\right)+\vec{V}\bullet\vec{S}_M+S_E \quad ; \qquad (5)$$

где h_{tot} - полная энтальпия; h_{stat} - статическая энтальпия; S_E - источниковый член для энергии; λ - коэффициент теплопроводности жидкости.

$$h_{\text{tot}}=h_{\text{stat}}+\frac{\vec{V}^2}{2} \; ; \qquad (6)$$

$$h_{\text{stat}}=c_p(T-T_0) \; ; \qquad (7)$$

где c_p-теплоемкость жидкости; T – текущая температура; T_0 - начальная температура.

-уравнение состояния сжимаемой жидкости:

$$\rho=\rho(T,P) \qquad (8)$$

-уравнение турбулентной энергии:

$$\frac{\partial(\rho k)}{\partial t}+\nabla\bullet(\rho\vec{V}\,k)=\nabla\bullet((\mu+\frac{\mu_t}{\sigma_k})\nabla\bullet k)+\mu_t G-\rho\varepsilon \; ; \qquad (9)$$

где k-турбулентная энергия; $\sigma_к$-константа; G- определяет скорость генерации турбулентной энергии; ε-скорость диссипации турбулентной энергии.

$$G=D_{ij}\frac{\partial V_i}{\partial x_j} \; ; \qquad (10)$$

$$D_{ij}=S_{ij}-\frac{2}{3}(\nabla\bullet\vec{V}+\frac{\rho k}{\mu_t})\delta_{ij} \; ; \qquad (11)$$

где S_{ij}-удвоенный тензор скоростей деформации,

$$S_{ij}=\frac{\partial w_i}{\partial x_j}+\frac{\partial w_j}{\partial x_i}; \qquad (12)$$

-уравнение скорости диссипации турбулентной энергии:

$$\frac{\partial(\rho\varepsilon)}{\partial t} + \nabla\bullet(\rho\vec{V}\,\varepsilon) = \nabla\bullet((\mu + \frac{\mu_t}{\sigma_\varepsilon})\bullet\nabla\varepsilon) + C_1\frac{\varepsilon}{k}\mu_t G - C_2 f_1 \rho\frac{\varepsilon^2}{k}; \quad (13)$$

где ε-скорость диссипации турбулентной энергии,σ_ε , C_1, C_2-константы.

-уравнение турбулентной вязкости вычисляется по формуле Колмогорова-Прандтля:

$$\mu_t = C_\mu \rho\frac{k^2}{\varepsilon}, \quad (14)$$

где C_μ-константа.

Система уравнений (1-14) замыкается начальными и граничными условиями. При этом жидкость в момент времени t=0 полагается невозмущенной и ее параметры соответствуют нормальным условиям для воды.

Для проведения вычислительного эксперимента была построена, геометрическая модель, которая состоит из свободного объема первой ступени (жидкостной регион) и крыльчатки (подвижное твердое тело).

Свободный объем первой ступени имеет два входа и два выхода. На боковых поверхностях модели расположены элементы осевого двустороннего входа А (рис. 1). Первый выход В – выход из соединительного канала (рис. 1). Второй выход С имеет большее сечение (рис. 1). Крыльчатка, представляет из себя твердое тело D (рис. 1). Геометрическая модель построена с учетом тангенциального расположения выходов первой ступени. Учитывалось геометрическое расширение каналов.

Твердотельная модель для данной задачи была построена в инженерном пакете *Solid Works*, для проведения вычислительного эксперимента было необходимо построить геометрическую модель, состоящую из двух регионов:

1) Жидкостной регион, представляет собой свободный объем первой ступени.

2) Твердотельный регион, представляет собой крыльчатку первой ступени, крыльчатка помещена в свободный объем первой ступени, сносно.

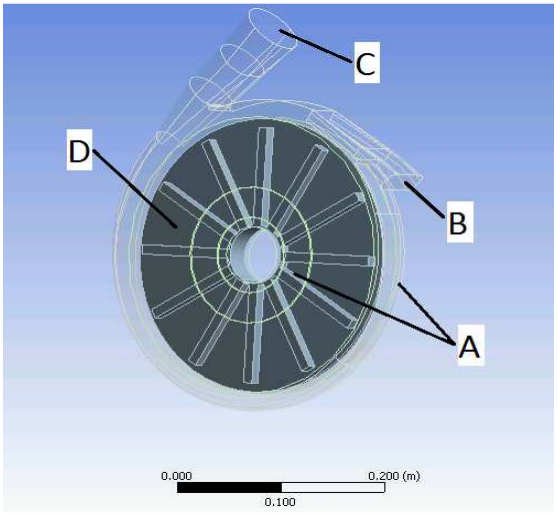


Рис. 1. Общий вид геометрической модели.

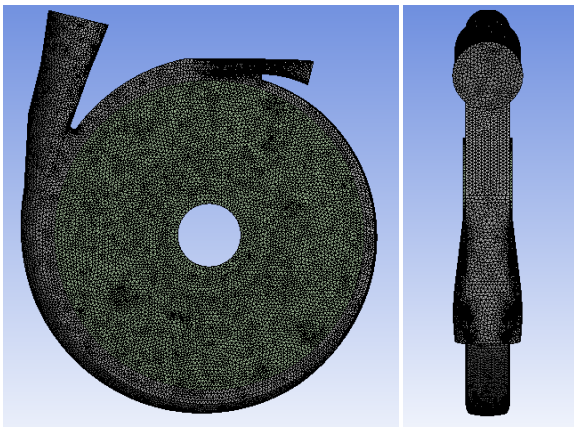
Для проведения численного эксперимента были заданы следующие граничные условия:

- движение крыльчатки задавалось через постоянную скорость вращения n= 8500 об/мин.
- для граничного условия входа в первую ступень задавалось постоянное давление 0,05, 0,3 и 0,5 МПа.

- для граничного условия выхода из первой ступени задавалось граничное условие «свободный выход».

Структура расчетной сетки следующая. Для лучшей сходимости решения и снижения погрешностей получаемых результатов необходимо применить сетку, ячейки которой имеют равномерную форму, близкую к форме тетраэдра.

Вместе с тем, при измельчении сетки желательно избегать резких отличий геометрических размеров соседних ячеек – линейные размеры не должны отличаться более, чем в 2 раза. На рисунке 2 представлена тетраэдральная расчетная сетка, примененная для решения поставленной задачи.



а б

Рис. 2. Расчетная сетка: а – вид спереди; б – вид справа.

Расчетная сетка имеет равномерную структуру. Количество элементов расчетной сетки 20 млн. В области взаимодействия крыльчатки и жидкости была проведена адаптация расчетной сетки первого уровня.

В рамках вычислительных экспериментов исследовалось влияние давления на входе в первую ступень центробежного насоса, на колебательные процессы в соединительном канале. Давление на входе в первую ступень задавалось равным 0,05, 0,3 и 0,5 МПа.

Для данного класса задач так же проводилось исследование масштабируемости. Количество используемых ядер варьировалось от 2 до 64.

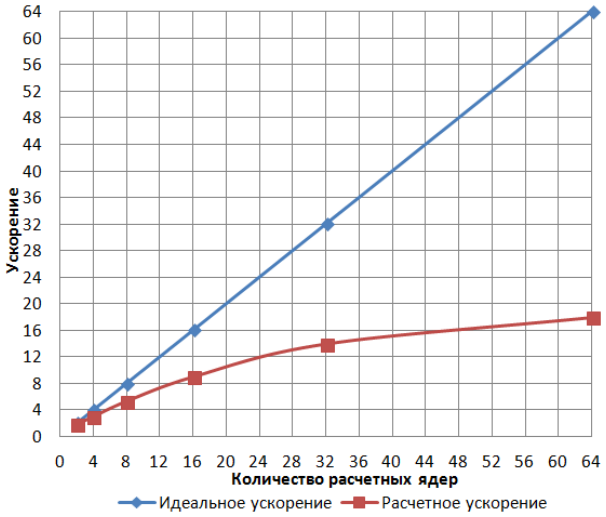


Рис. 3. График масштабируемости.

Из графика видно, что эффективное ускорение при решении данной задачи, для данного количества расчетных узлов и используемого ПО наблюдается при использовании от 2 до 16 ядер, при последующем увеличении количества ядер до 32 ускорение резко снижается.

На рисунке 4 изображена схема расположения математического «датчика давления». «Датчик давления» располагается в центре поперечного сечения.

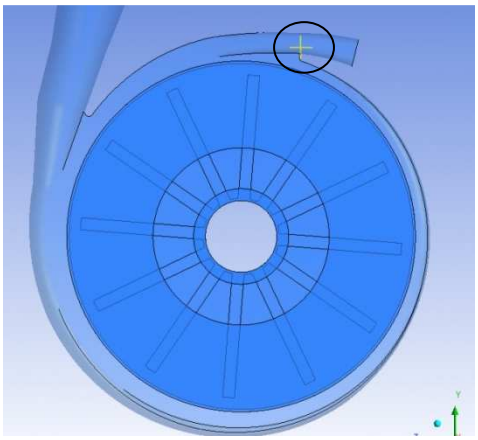
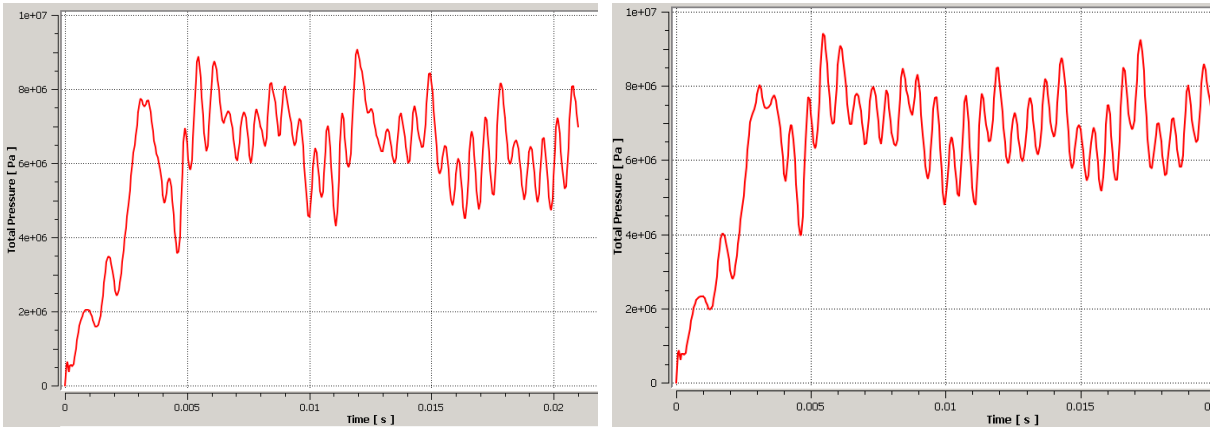


Рис. 4. Схема расположения «датчика давления».

Графики зависимости давления от времени в контрольной точке представлены на рисунке 5(а,б).



а б

Рис. 5. График зависимости полного давления от времени
$P_{вх}$: а – $P_{вх}$= 0,05 МПа.; б – $P_{вх}$= 0,5 МПа.

Анализ полученных зависимостей показал, что в модельной конструкции обнаружены высокочастотные колебания (≈1700 Гц), модулированные низкочастотными колебаниями (≈ 222 – 333 Гц). По результатам вычислительных экспериментов выявлено, что с увеличением давления на входе частота модулирующего сигнала повышается (≈ 100 Гц).
Выводы:

1. Обнаружена модуляция высокочастотных колебаний давления в области входа в соединительный канал.
2. Модуляция обусловлена формированием пульсирующей области высокого давления в межлопаточном пространстве. Частота модуляции близка к частоте вращения крыльчатки.
3. Обнаружена прямая зависимость частоты модуляции от давления на входе.
4. Проведен анализ масштабируемости данного класса задач.

Литература

1. Модорский В.Я., Бульбович Р.В., Бутымова Л.Н., Зимин Д.В., П.В.Писарев. Численное моделирование напряженно-деформированного состояния активной заглушки резонатора Научно-технический вестник Поволжья №1, 2012, Казань, с.95-100.
2. Модорский В.Я., Бутымова Л.Н., Зимин Д.В., Писарев П.В., Соколкин Ю.В. Численное моделирование деформированного состояния крупногабаритной конструкции из композиционного материала. Научно-технический вестник Поволжья №1, 2012, Казань, с.116-120.
3. Модорский В.Я., Бутымова Л.Н., Зимин Д.В., Петров В.Ю., Писарев П.В. Разработка экспериментального комплекса для анализа резонансных процессов в энергетических установках. Научно-технический вестник Поволжья №1, 2012, Казань, с.111-115.
4. Модорский В.Я., Щенятский Д.В., Арбузов И.А., Бульбович Р.В., Кириевский Б.Е., Писарев П.В., А.А. Ташкинов. Численное моделирование колебательных процессов в центробежном насосе. Научно-технический вестник Поволжья №3, 2012, Казань, с.44-49.
5. Модорский В.Я., Козлова А.В., Петров В.Ю., Поник А.Н. Инженерная методика определения настроек газохода переменного сечения для отвода и охлаждения горячих газов энергетических установок. Научно-технический вестник Поволжья №2, 2012, Казань, с. 216-219.

Программная и аппаратная архитектура сервера визуализации сеточных данных программного комплекса GIMM_NANO*

А.И. Плотников

Институт математического моделирования РАН, Москва

В работе рассматривается принципиальная схема архитектуры сервера визуализации сеточных данных программного комплекса GIMM_NANO, включая интерфейс клиент-серверного взаимодействия. Также рассмотрен принцип работы данной архитектуры на примере разрабатываемой программы визуализации сеток, которая будет включена в программный комплекс сервера визуализации.

1. Постановка задачи

Разработать аппаратную и программную архитектуру сервера визуализации сеточных данных. При этом сервер визуализации должен выполнять следующие функции:

1. Обеспечение интерактивного взаимодействия с ПО клиентской части программы визуализации (получение параметров визуализации и отправка обработанных данных, корректное завершение работы по требованию клиента)
2. Работа с файловым хранилищем комплекса GIMM_NANO для получения входных данных для анализа.
3. Осуществлять параллельную обработку сеточных данных, с учетом полученных от пользователя параметров.

Для обеспечения масштабируемости программного обеспечения сервера визуализации сеточных данных, необходимо было разработать единый программный интерфейс для всех используемых процедур визуализации (изоповерхностей, сеток, изолиний и т.д.)

Далее в работе рассматривается архитектура программы визуализации сеточных данных и реализация программного интерфейса на примере разрабатываемого модуля визуализации сеток.

2. Архитектура сервера визуализации

Принципиальная схема разработанной архитектуры сервера визуализации сеточных данных проекта GIMM NANO представлена на диаграмме (рис.1).

* Работа выполнена при поддержке РФФИ, проект 11.07.00.779_а

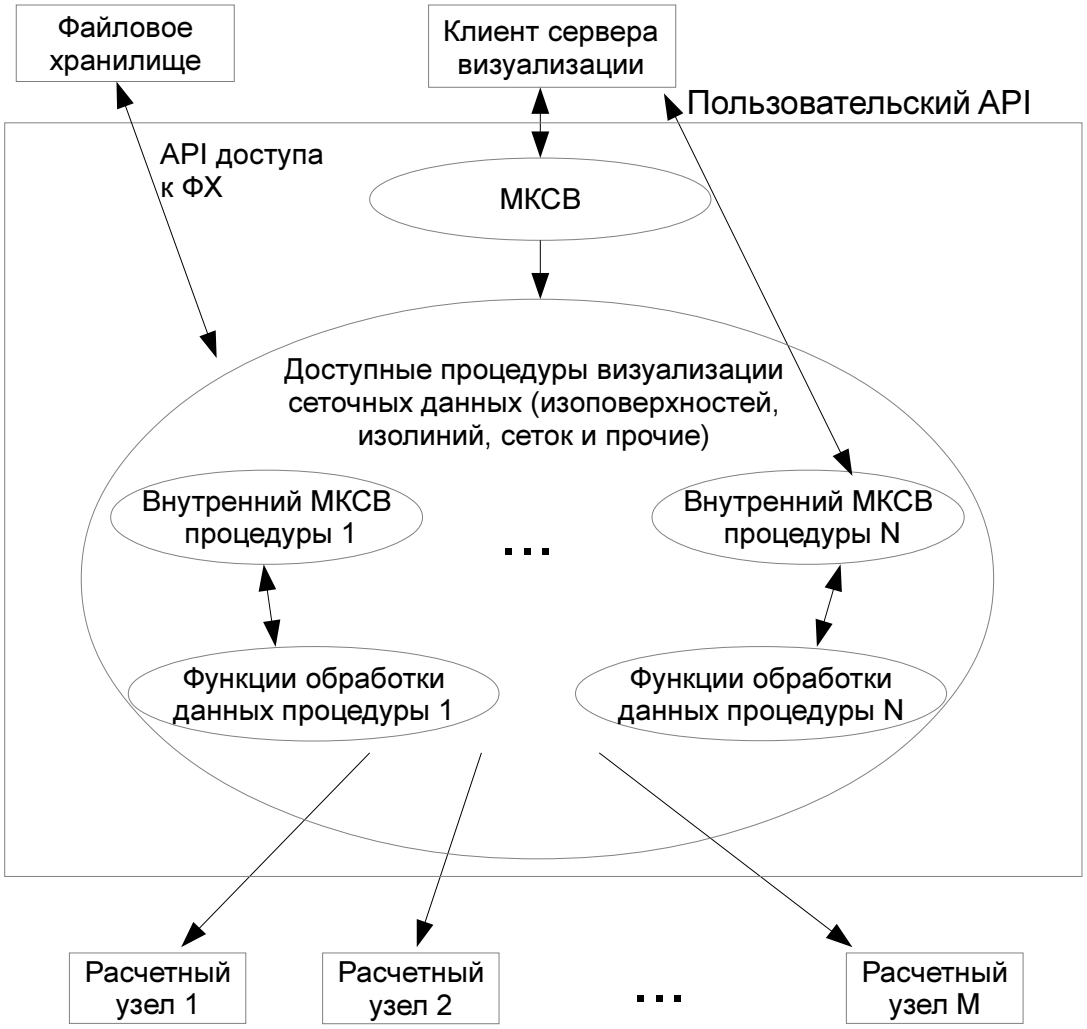


Рис. 1. Принципиальная схема архитектуры сервера визуализации. МКСВ – модуль клиент-серверного взаимодействия. Элементы программной архитектуры изображены в форме эллипса, аппаратной – в форме прямоугольника

Таким образом, можно выделить два основных модуля данного сервера:

- Модуль клиент-серверного взаимодействия (МКСВ)
- Процедуры обработки данных (каждая процедура состоит из внутреннего МКСВ и функций обработки данных)

Основная задача модуля клиент-серверного взаимодействия – предоставить программе-клиенту интерфейс (далее API) для доступа к функциям сервера визуализации, объединив API различных программных компонентов последнего в единый интерфейс. Данный интерфейс описан ниже. Также модуль осуществляет соединение с клиентом, корректное завершение работы сервера визуализации и разрыв соединения по требованию. В качестве клиента могут выступать и две сущности (один на прием данных, другой на передачу). Это полезно в случае действия на клиента определённых лицензионных ограничений. В случае с GIMM_NANO, в качестве клиента, осуществляющего передачу данных серверу визуализации, выступает управляющая программа, которая контролирует права доступа пользователей всего программного пакета.

Каждая процедура обработки данных состоит из внутреннего МКСВ и набора функций обработки данных. Внутренний МКСВ процедуры получает на вход параметры соединения и имя файла сетки, считывает данную сетку из файлового хранилища, получает от пользователя все

необходимые параметры и запускает функции обработки сетки, возвращая результат пользователю. Требования к данному модулю описаны в следующем разделе. Функции обработки данных используют для проведения необходимых вычислений расчётные узлы сервера визуализации, осуществляя распределение данных между выделенными узлами и балансировку нагрузки.

2.1 API модуля клиент-серверного взаимодействия

Каждая процедура визуализации должна иметь функцию с унифицированным именем, которая получает на вход параметры соединения и осуществляет взаимодействие с клиентом, получая от него необходимые параметры, проверяя их, вызывая функции параллельной обработки данных и отправляя результат клиенту. Возможные возвращаемые значения функции описаны ниже. В терминологии данной статьи описываемая функция называется внутренним модулем клиент-серверного взаимодействия (внутренний МКСВ). Далее представлен алгоритм взаимодействия клиента программы визуализации с сервером и описаны основные требования к внутреннему МКСВ

Взаимодействие клиента программы визуализации с сервером осуществляется посредством МКСВ следующим образом:

1. Клиент запрашивает соединение. В случае успеха, сервер высылает приветственное сообщение и ждет команды.
2. Клиент передаёт название процедуры визуализации, которой собирается воспользоваться и имя файла сетки, на которой будет производиться работа. Сервер анализирует файл сетки на корректность и отправляет результат пользователю.
3. Далее вызывается внутренний МКСВ выбранной процедуры. Возвращает функция одно из следующих значений:
    a. Флаг успешного выполнения всех инструкций
    b. Флаг ошибки выполнения и её код, позволяющий определить тип ошибки, среди которых: неверные входные параметры, ошибка самой программы обработки данных, потеря связи с клиентом.
    c. Сигнал завершения работы процедуры по требованию клиента. В самой же процедуре визуализации реализован функционал максимально быстрой остановки обработки данных по требованию клиента. Команда завершения работы универсальная для всех процедур.
    d. Сигнал завершения работы сервера.
4. Соответственно, в зависимости от результата вызова предыдущей функции, МКСВ делает одно из следующих действий:
    a. В случае успешного выполнения, переходит к п.5.
    b. В случае возникновения ошибки, сообщает об этом пользователю и переходит к п.5.
    c. В случае сигнала о завершении работы процедуры, сообщает пользователю, что процедура успешно остановлена, переходит к п.5.
    d. Закрывает соединение, сообщая об этом клиенту, и завершает свою работу.
5. Ожидание одной из следующих команд от пользователя:
    a. Изменение файла сетки и/или процедуры обработки (визуализации). При этом клиент передает оба параметра, а на стороне сервера осуществляется проверка того, какой параметр изменился. Далее идет переход к п.3.
    b. Передача новых параметров. В данном случае программа переходит к п.3.
    c. Выход из программы.

Далее рассмотрена архитектура процедуры визуализации сеток, разрабатываемой на основе данного архитектурного шаблона.

3. Архитектура процедуры визуализации сеток

Программное обеспечение визуализатора сеточных данных позволяет анализировать сетки большого объема, осуществляя огрубление данных удаленно на сервере визуализации (в качестве которого может выступать многопроцессорная система). Система визуализации позволяет перемещаться по трехмерной сетке большого объема по принципу карт, позволяя выделять и уточнять необходимые участки, изображая лишние участки лишь в общих чертах. Данная программа хорошо приспособлена к решению таких задач, как поиск элементов сетки, удовлетворяющих (или не удовлетворяющих) определенным условиям; проверка качества алгоритмов декомпозиции и балансировки нагрузки при решении сеточных задач.

Полученные сеточные данные программа приводит к внутреннему формату, в котором структура доступа к доменам сетки имеет вид дерева и который позволяет максимально быстро получать различные области сетки с заданным уровнем детализации, существенно экономя, благодаря этому, время и вычислительные ресурсы аппаратного обеспечения сервера визуализации. Недостатком такого подхода является относительно долгое время первоначального анализа сетки. Алгоритм хорошо адаптирован для применения на многопроцессорных системах, что позволяет решить описанную проблему при условии наличия достаточных вычислительных ресурсов.

Программное обеспечение данной системы визуализации состоит из 4-х базовых модулей:

1. Внутренний МКСВ. Функциональное назначение описано в разделе 2. Описание API приведено ниже.
2. Средство декомпозиции сеток. Данный модуль позволяет агрегировать узлы сетки в домены с приблизительно одинаковым количеством узлов внутри каждого домена, благодаря чему можно оценивать плотность сетки в наблюдаемой точке, не отображая всю сетку целиком. Также, цветовая маркировка домена может показывать интересующие интегральные характеристики сетки внутри данного домена (количество не удовлетворяющих определенному условию элементов и т.д.). Данный модуль имеет как последовательную, так и параллельную реализации, осуществляя максимально равномерное разбиение между доменами по количеству узлов.
3. Модуль огрубления поверхностей. Позволяет осуществлять огрубление поверхностей доменов для сокращения объема данных, хранящихся на сервере и пересылаемых клиенту. Модуль имеет параллельную реализацию и осуществляет огрубление с заданной точностью.
4. Ядро программы — процедура построения дерева иерархии доменов. Данный модуль рекурсивно применяет процедуру декомпозиции сеток (сначала параллельно, разбивая исходный объект на домены, а потом и каждый домен в отдельности, уже последовательно на вычислительных узлах многопроцессорной системы). После этого осуществляется параллельное огрубление получившихся поверхностей и объединение доменов в дерево по принципу вложенности.

Блок-схема работы сервера визуализации имеет вид, изображенный на рис. 2.

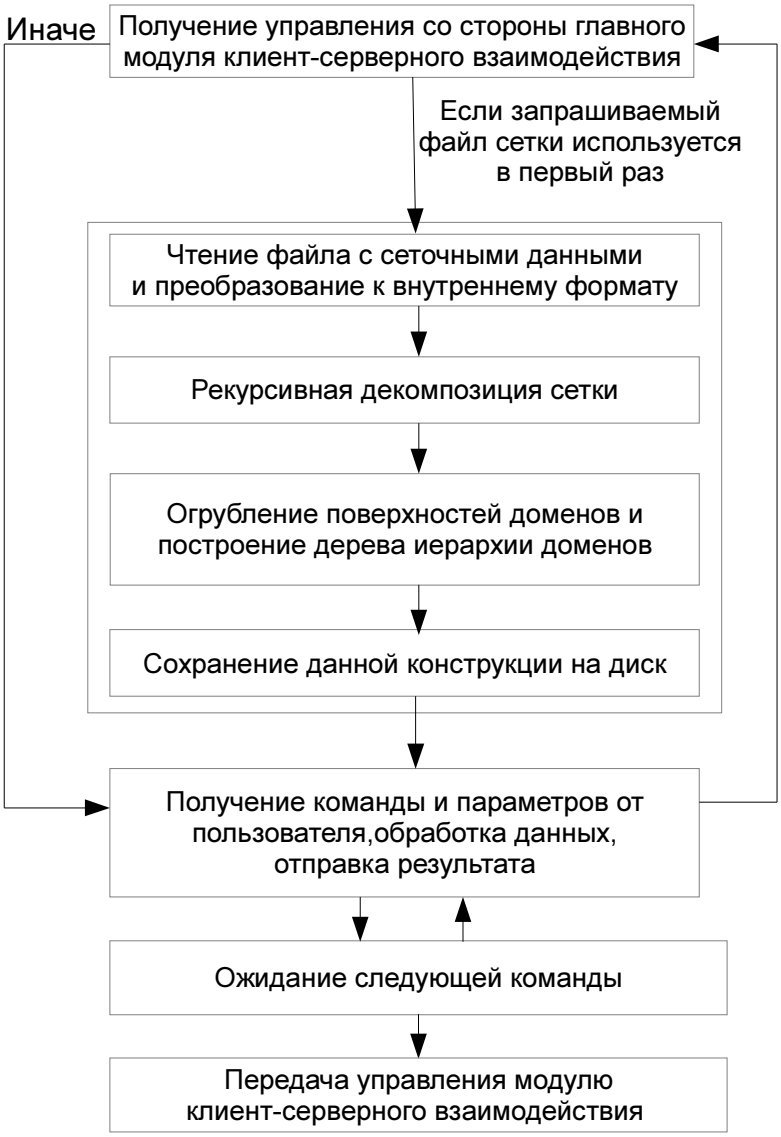


Рис. 2. Блок-схема работы процедуры визуализации сеток

3.1 API внутреннего МКСВ процедуры визуализации сеток

На основе разработанного архитектурного шаблона, в соответствии с описанием процедуры визуализации сеток, приведённым выше, API внутреннего МКСВ данной процедуры имеет следующий вид:

1. Получение управления от МКСВ, ожидание параметров от клиента
2. Клиент отправляет параметры отображаемой области сетки (прямоугольная область, задающаяся двумя точками). В случае наличия ошибок в данных, в МКСВ передается флаг ошибки входных данных (всю остальную работу МКСВ делает самостоятельно)
3. Клиент отправляет характерный размер поддомена (эффективный диаметр). Осуществляется проверка, как в п. 2.
4. Получение со стороны клиента дополнительных параметров отображения (если используются функции выделения цветом и т.п.)
5. Осуществляется обработка данных сетки по алгоритму, описанному выше. В случае возникновения ошибки в МКСВ передается её флаг в соответствие с п 3.b раздела 2.1 данной статьи. Аналогично реализованы функции пунктов 3.c и 3.d.

6. Передача управления МКСВ с флагом успешного выполнения процедуры

Заключение

Таким образом, разработана программная и аппаратная архитектура сервера визуализации сеточных данных, удовлетворяющая следующим требованиям:

1. Система реализует все функции, заявленные в п.1 «Постановка задачи»
2. Обладает хорошим потенциалом масштабируемости за счет единого интерфейса всех программных модулей

Данная архитектура легла в основу программной реализации визуализатора GIMM_Data_Visualizer, являющегося частью комплекса GIMM_NANO, разрабатываемого в институте математического моделирования РАН, Москва.

Литература

1. Б. Н. Четверушкин, В. А. Гасилов, С. В. Поляков, Е. Л. Карташева, М. В. Якобовский, И. В. Абалакин, В. Г. Бобков, А. С. Болдарев, С. Н. Болдырев, С. В. Дьяченко, П. С. Кринов, А. С. Минкин, И. А. Нестеров, О. Г. Ольховская, И. В. Попов, С. А. Суков. Пакет прикладных программ GIMM для решения задач гидродинамики на многопроцессорных вычислительных системах // Матем. моделирование, 17:6 (2005), 58–74.
2. С. В. Поляков, Т. А. Кудряшова, А. А. Свердлин, Э. М. Кононов, О. А. Косолапов. Параллельный программный комплекс для решения задач механики сплошной среды на современных многопроцессорных системах // Матем. моделирование, 22:6 (2010), 132–146.
3. Якобовский М.В. Обработка сеточных данных на распределенных вычислительных системах // Вопросы атомной науки и техники, Сер. «Математическое моделирование физических процессов», 2 (2004), 40-53.
4. S.Polyakov, M. Iakobovski. GEOMETRICAL SIMULATION AND VIZUALIZATION IN NANOELECTRONICS PROBLEMS // COMPUTER GRAPHICS & GEOMETRY, 11:1 (2009), 46-68.
5. M. Iakobovski, I. Nesterov, P. Krinov. LARGE DISTRIBUTED DATASETS VISUALIZATION SOFTWARE, PROGRESS AND OPPORTUNITIES // COMPUTER GRAPHICS & GEOMETRY, 9:2 (2007), 1-19.

Высокопроизводительные вычисления в облачных системах с использованием OpenFlow*

П.Н. Полежаев

Оренбургский государственный университет

В настоящей работе описывается подход к созданию грид-системы, функционирующей поверх вычислительной облачной системы, построенной на базе OpenStack и программно-конфигурируемых OpenFlow сетей. Предложена архитектура облачного вычислительного центра обработки данных с поддержкой OpenFlow, разработаны алгоритмы назначения групп виртуальных машин с учетом состояния физической сети облачного центра обработки данных.

1. Введение

Облачные вычислительные системы стали стандартом де факто для многих прогрессивных областей сферы информационных технологий. Российские компании и университеты используют их с целью размещения своих научных и бизнес-приложений, что позволяет им избежать расходов на создание и поддержание функционирования собственных центров обработки данных (ЦОД). С другой стороны, владельцы облачных вычислительных ЦОД путем консолидации вычислительных ресурсов и систем хранения данных (СХД) способны снизить совокупную стоимость владения ИТ-инфраструктурой за счет обслуживания значительного количества клиентов, а также использования эффективных технических средств планирования и балансировки нагрузки, управления пересылкой данных в сети, интеграции нескольких территориально разрозненных сегментов ЦОД.

В последнее время облачные системы стали использоваться в качестве основы для высокопроизводительных систем (High performance computing, HPC), включая вычислительные кластеры и грид-системы. Это стало возможным за счет развития средств виртуализации ресурсов процессора, памяти, а также ввода-вывода.

В настоящей работе предлагается подход к улучшению существующей облачной системы OpenStack [1] за счет интеграции инфраструктурных средств грид-системы, облачной системы OpenStack и программно-конфигурируемых сетей (ПКС) сегментов ЦОД на основе OpenFlow [2].

2. Облачная система OpenStack

Наиболее распространенным открытым программным обеспечением для управления облачными вычислительными ЦОД является OpenStack. Его популярность, прежде всего, обусловлена, открытостью, документированностью, наличием исходных кодов, а, следовательно, возможностью адаптации его компонент под произвольные архитектуры вычислительных систем.

OpenStack позволяет создавать многоарендные архитектуры IaaS облачных вычислительных ЦОД, когда вычислительные и сетевые ресурсы разделяются между несколькими пользователями-арендаторами. Каждому арендатору выделяется виртуализованные ресурсы в виде группы виртуальных машин, связанных виртуальной сетью. Пользователь вправе сам настраивать конфигурацию виртуальных машин, включая параметры процессора, памяти, загружаемый

* Исследования поддержаны федеральной целевой программой «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы» (госконтракт №07.514.11.4153), федеральной целевой программой «Научные и научно-педагогические кадры инновационной России на 2009-2013 годы» (соглашение №14.B37.21.1881) и РФФИ (проект №12-07-31089).

образ с операционной системой, программами и данными. Хранение образов виртуальных машин обеспечивается компонентой Glance, а данных – облачным хранилищем Swift.

OpenStack имеет следующие недостатки:

– Отсутствие эффективных средств управления потоками данных в используемой компьютерной сети. Как правило, применяются стандартные протоколы маршрутизации, такие, как RIP, OSPF, EIGRP и др., которые опираются на распределенные алгоритмы выбора маршрута, без учета логических путей передачи (потоков) данных, существующих на уровне приложений (в том числе параллельных программ, выполняющихся в HPC системе). Данные протоколы поддерживаются существующими коммутаторами без ПКС.

– Использование неэффективного алгоритма планирования экземпляров виртуальных машин. Он заключается в следующем: для каждой планируемой виртуальной машины из нескольких одинаковых экземпляров, сначала вычислительные узлы фильтруются в соответствии с ресурсными требованиями виртуальной машины, затем они сортируются в порядке возрастания значения интегрального критерия, представляющего собой линейную свертку характеристик вычислительного узла, после чего выбирается необходимое количество узлов. Данный алгоритм не учитывает топологию сети, ее состояние в текущей момент времени, а также коммуникационные схемы передачи данных между виртуальными машинами.

– Отсутствие встроенных средств для эффективной реализации HPC поверх облака. OpenStack позволяет размещать виртуальные кластеры и грид-системы в группах экземпляров виртуальных машин, функционирующих в облаке. При этом облако не имеет информации о том, что запущено внутри него, включая сведения о выполняемых параллельных программах, а HPC-система не обладает сведениями о состоянии физической сети облака. С одной стороны, это обеспечивает абстракцию и универсальность облака, а с другой, отрицательно сказывается на производительности HPC-системы, т.к. планировщик задач в HPC-системе способен эффективнее назначать вычислительные задачи (особенно коммуникационно-интенсивные) при наличии информации о топологии и состоянии сети облака.

OpenStack был выбран в качестве основы для построения HPC систем, функционирующих поверх облака. Разделы 3 и 4 раскрывают предложенные автором подходы к устранению данных недостатков. В разделе 5 описывается предложенная архитектура облачного вычислительного ЦОД с поддержкой OpenFlow.

3. Разработанные алгоритмы назначения групп виртуальных машин

Обозначим в качестве Φ_{assign} функцию, назначающую каждой группе виртуальных машин $VMGroup_j$ набор вычислительных узлов

$$\Phi_{assign}(VMGroup_j) = (Node_{k_1 i_1}, Node_{k_2 i_2}, ..., Node_{k_{g_j} i_{g_j}}),$$

где $Node_{k_r i_r}$ – узел для назначения виртуальной машины VM_{jr}. Условно зададим $Node_{k_r i_r} = \varphi_{assign}(VM_{jr})$.

Для оценки качества алгоритма назначения групп экземпляров виртуальных машин Φ_{assign} может использоваться интегральный критерий оценки размещения отдельной виртуальной машины $VMGroup_j$:

$$I(\Phi_{assign}, VMGroup_j, DataCenter) \to min,$$

где $DataCenter$ – мультиграф, описывающий весь вычислительный ЦОД.

Пусть $dist_{phys}(d_1, d_2)$ – топологическое расстояние между физическими сетевыми устройствами $d_1 \in Devices$ и $d_2 \in Devices$ в облачной системе. Как правило, в качестве топологического расстояние берется оценка суммарной задержки времени передачи пакета с учетом состояния сети на текущий момент времени. Для двух физических узлов внутри одного сегмента топологическое расстояние будет гораздо меньше, чем расстояние для двух узлов внутри разных, территориально удаленных сегментов.

В качестве интегрального критерия могут быть использованы:

1) суммарное попарное топологическое расстояние между назначенными узлами для группы экземпляров виртуальных машин

$$I_{SDMA}(\Phi_{assign}, VMGroup_j, DataCenter) = \sum_{VM_{jr_1}, VM_{jr2} \in VMs_j} dist_{phys}(\varphi_{assign}(VM_{jr_1}), \varphi_{assign}(VM_{jr_2})) \to min;$$

2) максимальное топологическое расстояние между двумя назначенными узлами для группы экземпляров виртуальных машин

$$I_{MDMA}(\Phi_{assign}, VMGroup_j, DataCenter) =$$

$$= \max_{VM_{jr_1}, VM_{jr2} \in VMs_j} dist_{phys}(\varphi_{assign}(VM_{jr_1}), \varphi_{assign}(VM_{jr_2})) \to min.$$

Опишем общую схему предложенных алгоритмов назначения групп экземпляров виртуальных машин.

Входной параметр: $VMGroup_j$ – назначаемая группа экземпляров виртуальных машин.

Выходной параметр: $R = (Node_{k_1 i_1}, Node_{k_2 i_2}, ..., Node_{k_{g_j} i_{g_j}})$ – набор назначенных физических вычислительных узлов.

1. С помощью алгоритма Флойда-Уоршелла вычисляется топологическое расстояние $dist_{phys}(d_1, d_2)$ между всеми парами физических сетевых устройств $d_1, d_2 \in Devices$ облачной системы.

2. Для каждого физического сетевого устройства $d \in Devices$ облачной системы выполняются следующие действия:

2.1. Пока имеются не назначенные экземпляры виртуальных машин, перечисляются все физические узлы $Node_{k_r i_r} \in Devices$ в порядке возрастания величины $dist_{phys}(d, Node_{k_r i_r})$, при равных значениях расстояния – упорядочиваются по производительности.

2.2. Формируется список экземпляров виртуальных машин L группы $VMGroup_j$, ресурсные требования которых способен удовлетворить узел $Node_{k_r i_r}$.

2.3. Из списка L выбирается экземпляр виртуальной машины VM_{jr}, требования которого наиболее близки к характеристикам физического узла:

$$VM_{jr} = arg \min_{VM_{jr} \in L} Similarty(VM_{jr}, Node_{k_r i_r}).$$

Функция схожести представляет собой линейную свертку разностей характеристик узла $Node_{k_r i_r}$ и соответствующих величин запрашиваемых VM_{jr} ресурсов, взятых с некоторыми весовыми значениями.

2.4. За экземпляром виртуальной машины VM_{jr} закрепляется физический вычислительный узел $Node_{k_r i_r}$, задается $\varphi^d_{assign}(VM_{jr}) = Node_{k_r i_r}$.

2.5. В конце для сформированного назначения вычисляется значение интегрального критерия I^d.

3. В качестве результата R выбирается набор $(\varphi^{d_{min}}_{assign}(VM_{j1}), \varphi^{d_{min}}_{assign}(VM_{j2}), ..., \varphi^{d_{min}}_{assign}(VM_{jg_j}))$, который для первоначального физического сетевого устройства d_{min} обеспечивает минимум $I^{d_{min}}$ интегрального критерия:

$$I^{d_{min}} = \min_{d \in Devices} I^d.$$

4. Набор назначенных физических вычислительных узлов R передается подсистеме маршрутизации трафика облачной системы для реализации схемы проактивной маршрутизации всевозможными узлами.

В зависимости от используемого интегрального критерия можно получить различные алгоритмы назначения групп виртуальных машин. В качестве критериев были выбраны формулы

I_{SDMA} и I_{MDMA}. Данные алгоритмы были соответственно названы Summed Distance Minimization Assignment и Maximum Distance Minimization Assignment. В силу наличия аналогии между группой виртуальных машин и параллельной программой с несколькими процессами, данные алгоритмы являются аналогами ранее предложенных автором методов назначения параллельных программ для вычислительной грид-системы [3].

Результаты исследования данных алгоритмах будут опубликованы позднее, после создания облачного вычислительного ЦОД с поддержкой OpenFlow в Оренбургском государственном университете.

4. Предложенный подход к организации HPC поверх облачных центров обработки данных с поддержкой OpenFlow

Для организации HPC поверх облачных центров был выбран подход, заключающийся в создании неавтономной грид-системы, функционирующей поверх вычислительного облака. Опишем основные преимущества и особенности реализации данного подхода.

Грид-система представляет собой совокупность вычислительных кластеров, каждый из которых развернут внутри группы экземпляров виртуальных машин в отдельном сегменте ЦОД, а также экземпляра виртуальной машины грид-диспетчера, который поддерживает единую очередь параллельных вычислительных задач.

Грид-система и облачная система не изолированы полностью друг от друга, как это характерно для обычных подходов к реализации HPC поверх облаков. Облачная система предоставляет планировщику грид-системы актуальную информацию о состоянии сети облака (оценку задержек при передаче пакетов между виртуальными вычислительными узлами), а грид-система передает информацию подсистеме маршрутизации облака о выполненных назначения процессов параллельных задач и их коммуникационные схемы.

Такой осознанный подход к снижению уровня абстракции и универсализации облачной системы позволяет, с одной стороны, планировщику грида, обладающему актуальной информацией о высокопроизводительной сети, эффективно и локализовано назначать процессы параллельных задач. А с другой, подсистема маршрутизации облачной системы может использовать проактивную схему маршрутизации. Данная схема заключается в том, что, когда известны шаблоны коммуникации (например, виртуальные топологии MPI-2 для параллельных задач) между сетевыми устройствами, то маршруты передачи данных между ними можно проложить заранее, до начала коммуникаций. Это позволяет избежать задержки обработки первого пакета, которая характерна для реактивной схемы, в которой при получении первого пакета каждого нового потока маршрут прокладывает контроллер и устанавливает соответствующие правила во все OpenFlow-коммутаторы.

При использовании обеих схем имеет место проблема ограниченности размера таблицы записей о потоках OpenFlow-коммутаторов. В случае реактивной схемы она решается путем установки каждого правила коммутации на некоторый интервал времени, после оно автоматически удаляется из таблицы. В случая проактивной схемы предложен вариант, в котором агрегируются потоки между процессами параллельных задач, при условии, что процессы запущены в одних и тех же виртуальных машинах.

Все описанное выше позволяет снизить среднее время выполнения коммуникационно-интенсивных задач, а также увеличить среднюю загрузку, как грид-системы, так и всей облачной системы.

В разрабатываемой облачной системе планируется использовать смешенную схему маршрутизации: проактивную – для передачи данных между процессами параллельных программ и реактивную – для управления инфраструктурными потоками в облаке, а также при чтении или записи данных в СХД.

В качестве алгоритмов планирования параллельных задач в грид-системе планируется использовать модифицированные варианты ранее разработанных автором алгоритмов Backfill SDM и Backfill MDM [3].

5. Архитектура облачного вычислительного центра обработки данных с поддержкой OpenFlow

Предложена архитектура облачного вычислительного ЦОД с поддержкой OpenFlow. ЦОД состоит из нескольких территориально-распределенных сегментов (автономных систем), каждый из которых представляет собой ПКС на основе OpenFlow. Сегменты связаны через глобальную сеть Интернет с помощью протокола BGP, точками подключения сегментов к глобальной сети являются граничные шлюзы, обладающие сведениями о префиксах других сегментов. Сеть ПКС сегмента содержит, как OpenFlow-коммутаторы, так и обычные коммутаторы без поддержки OpenFlow, а также вычислительные узлы на которые назначаются виртуальные машины групп экземпляров виртуальных машин.

Также каждый сегмент ЦОД содержит систему управления OpenStack, включающий: планировщик OpenStack (решает инфраструктурные задачи по запуску, остановке и контролю выполнения экземпляров виртуальных машин), разработанный модуль диспетчера распределения нагрузки (реализует предложенные алгоритмы назначения групп виртуальных машин) и сетевой модуль OpenStack (модифицированный для взаимодействия с HPC системой).

Рис. 1 более детально раскрывает структуру одного сегмента облачного вычислительного ЦОД.

С целью обеспечения надежности и масштабируемости в каждом сегменте функционирует несколько экземпляров контроллера OpenFlow, каждый из них имеет свой экземпляр базы данных кластерной СУБД MySQL, хранящей текущее состояние сегмента.

Прежде всего это ориентированный взвешенный мультиграф сегмента, вершины – сетевые устройства, дуги – сетевые связи (гарантируется существование противоположной дуги). Веса вершин – статические параметры и динамические характеристики соответствующих устройств (например, для вычислительных узлов – максимальный размер оперативной и локальной дисковой памяти, количество вычислительных ядер, текущие объемы свободной оперативной и дисковой памяти, загрузку каждого вычислительного ядра), веса дуг – их максимальная пропускная способность и текущая загрузка.

Имеется несколько систем распределения нагрузки, одна из них активная, остальные – находятся в резерве. Активный экземпляр осуществляет назначение OpenFlow-коммутаторов на контроллеры, руководствуясь заложенной политикой выравнивания нагрузки. Таким образом, каждый OpenFlow-коммутатор подключен ровно к одному контроллеру.

Согласно спецификации OpenFlow существует следующий принцип функционирования OpenFlow-коммутаторов. Каждый коммутатор имеет таблицу потоков, состоящую из записей о потоках (правил). Каждая запись включает: шаблоны заголовков пакета (match) – часть, по которой определяется, применимо ли данное правило к обрабатываемому пакету, действия (actions) – набор операций, совершаемых над пакетом (передача в заданный порт, помещение в заданную очередь, ассоциированную с конкретным портом коммутатора, изменение конкретного заголовка и др.), а также статистические счетчики.

При поступлении пакета, коммутатор ищет подходящие для него правило в таблице потоков, руководствуясь match-частями. Если правило найдено, то над пакетом выполняются соответствующие действия, после чего обновляются значения счетчиков. Если же правило отсутствует, то пакет пересылается контроллеру OpenFlow. Контроллер анализирует его заголовки, после чего устанавливает в данный и возможно в другие, подключенные в него, коммутаторы правила, для обработки подобных пакетов.

OpenFlow позволяет упростить коммутатор и вынести логику управления в отдельный контроллер. В рамках предложенного подхода модифицирован контроллер OpenFlow. Реализованы эффективные алгоритмы маршрутизации потоков данных с учетом текущего состояния сети ЦОД и информации о параллельных программах, исполняющихся поверх вычислительного ЦОД.

Система сбора статистики через регулярные небольшие интервалы времени с помощью SNMP и OpenFlow собирает сведения о текущем состоянии сетевых устройств и связей. Полученные данные сохраняются в локальную копию базы данных кластерной СУБД.

Хранение данных в облаке реализовано на базе службы сетевого хранилища Swift.

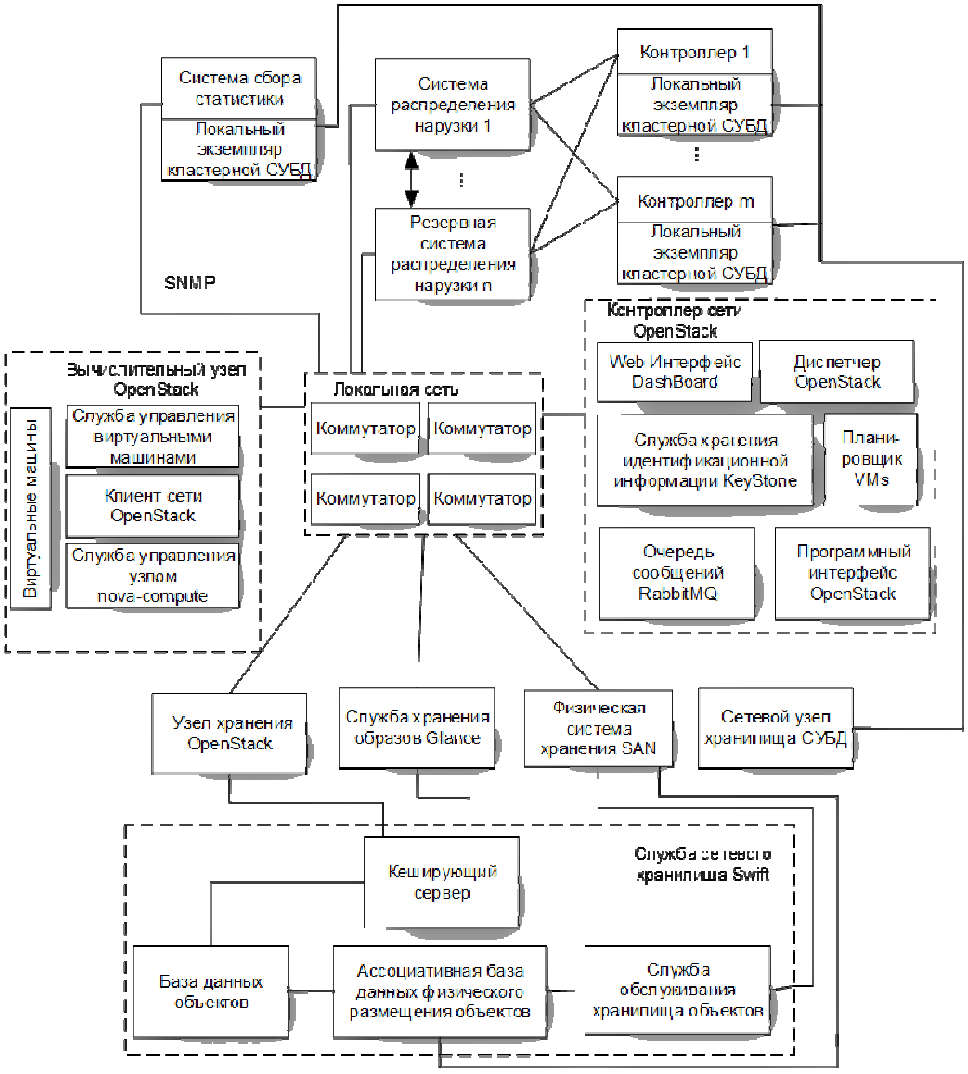


Рис. 1. Структура сегмента облачного вычислительного ЦОД с поддержкой OpenFlow

6. Заключение

Предложен подход к организации высокопроизводительных вычислений поверх вычислительного облака OpenStack с поддержкой OpenFlow. Описана архитектура облачного ЦОД с поддержкой его сегментами OpenFlow. Разработаны алгоритмы назначения групп виртуальных машин с учетом состояния физической сети облачного ЦОД.

В Оренбургском государственном университете находится в процессе создания экспериментальный образец облачного ЦОД с поддержкой OpenFlow его сегментами. В будущем планируется экспериментальное исследование всех разработанных алгоритмов и используемых технических решений.

Литература

1. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/ (дата обращения: 20.11.2012).
2. Портал OpenFlow. http://www.openflow.org/ (дата обращения: 20.11.2012).
3. Полежаев П.Н. Экспериментальное исследование алгоритмов планирования задач для грид-систем с использованием симулятора // Труды международной конференции «Высокопроизводительные вычисления HPC-UA’2012» (г. Киев), 2012. С. 278-284.

Отображение DVMH-программ на кластеры с ускорителями*

М.Н. Притула

ФГБУН Институт прикладной математики им. М.В. Келдыша РАН

В 2011 году для новых гетерогенных и гибридных суперкомпьютерных систем в Институте прикладной математики им. М.В. Келдыша РАН были разработаны языки программирования высокого уровня, представляющие собой стандартные языки Фортран и Си, расширенные директивами. В статье дается представление о логической структуре DVMH-программы и ходе ее выполнения на кластере с ускорителями; описываются методы и режимы отображения DVMH-программы на кластер с ускорителями, включая динамическое планирование выполнения DVMH-программы на многоядерных процессорах и ускорителях. Освещаются также возможности сравнительной отладки и особенности ее реализации в системе поддержки выполнения DVMH-программ.

1. Введение

В последнее время появляется много вычислительных кластеров с установленными в их узлах ускорителями. В основном, это графические процессоры компании NVIDIA. В 2012 году начинают появляться кластеры с ускорителями другой архитектуры – Xeon Phi от компании Intel. Так, в списке Top500 [2] самых высокопроизводительных суперкомпьютеров мира, объявленном в ноябре 2012 года, 62 машины имеют в своем составе ускорители, из них 50 машин имеют ускорители NVIDIA, 7 – Intel, 3 – AMD/ATI, 2 – IBM. Данная тенденция заметно усложняет процесс программирования кластеров, так как требует освоения на достаточном уровне сразу нескольких моделей и языков программирования. Традиционным подходом можно назвать использование технологии MPI для разделения работы между узлами кластера, а затем технологий CUDA (или OpenCL) и OpenMP для загрузки всех ядер центрального и графического процессоров.

С целью упрощения программирования распределенных вычислительных систем, были предложены высокоуровневые языки программирования, основанные на расширении директивами стандартных языков такие, как HPF [3], Fortran-DVM [1,9], C-DVM [1,10]. Также были предложены модели программирования и соответствующие основанные на директивах расширения языков для возможности использования ускорителей такие, как HMPP [4], PGI Accelerator Programming Model [5], OpenACC [6], hiCUDA [7].

В 2011 году в Институте прикладной математики им. М.В. Келдыша РАН была расширена модель DVM для поддержки кластеров с ускорителями [8]. Это расширение названо DVMH и позволяет с небольшими изменениями перевести DVM-программу для кластера в DVMH-программу для кластера с ускорителями.

Одним из важных аспектов функционирования такой программной модели, как DVMH является вопрос отображения исходной программы на все уровни параллелизма и разнородные вычислительные устройства. Важными задачами механизма отображения является обеспечение корректного исполнения всех поддерживаемых языком конструкций на разнородных вычислительных устройствах, балансировка нагрузки между вычислительными устройствами, а также выбор оптимального способа исполнения каждого участка кода на том или ином устройстве.

2. Схема выполнения DVMH-программы

Под DVMH-программой будем понимать программу на языке Fortran-DVMH или программу на языке C-DVMH.

* Работа выполнена при финансовой поддержке гранта Президента РФ НШ-4307.2012.9 и грантов РФФИ №11-01-00246, 12-01-33003-мол_а_вед, 12-07-31204-мол_а.

Выполнение DVMH-программы можно представить, как выполнение последовательности вычислительных регионов и участков между ними, которые будем называть внерегионным пространством. Код во внерегионном пространстве выполняется на центральном процессоре, тогда как для вычислительных регионов возможно их исполнение на разнородных вычислительных устройствах. Внутри, равно как и вне регионов могут быть как параллельные циклы, так и последовательные участки программы.

Параллелизм в DVMH-программах проявляется на нескольких уровнях:

1. Распределение данных и вычислений по MPI-процессам. Этот уровень задается специальными директивами распределения или перераспределения данных и спецификациями параллельных подзадач и циклов.
2. Распределение данных и вычислений по вычислительным устройствам при входе в вычислительный регион.
3. Параллельная обработка в рамках конкретного вычислительного устройства. Этот уровень появляется при входе в параллельный цикл, находящийся внутри региона.

Наличие этих уровней дает возможность органично отобразить программу на кластер с многоядерными процессорами и ускорителями в узлах.

Исполнение DVMH-программы начинается синхронно всеми запущенными процессами в модели SPMD. На каждый процесс выделяется одна основная последовательная нить исполнения.

При входе в вычислительный регион каждый процесс независимо выполняет дополнительное к межпроцессному распределение данных, используемых данным вычислительным регионом, по вычислительным устройствам, выбранным для выполнения региона. На этом этапе производится динамическое планирование с целью балансировки нагрузки и минимизации временных затрат на перемещения данных.

При входе в параллельный цикл внутри региона каждый процесс разделяет работу в соответствии с распределением данных по вычислительным устройствам. Затем он выбирает для каждого устройства метод обработки части цикла на конкретном устройстве, называемый обработчиком, а также оптимизационные параметры для выбранного обработчика. На этом этапе производится динамическая настройка оптимизационных параметров, включая количество ЦПУ-нитей для обработчиков на ЦПУ, а также размер и форму блока нитей для CUDA-обработчиков с целью минимизации времени исполнения на каждом отдельном устройстве.

Параллельный цикл внутри региона при исполнении распадается на несколько частей, каждая из которых обрабатывается некоторым обработчиком на некотором вычислительном устройстве. На этом этапе собирается основная информация для отладки эффективности DVMH-программы.

При выходе из вычислительного региона собирается дополнительная информация для целей динамического планирования выполнения региона, а также может быть осуществлена сравнительная отладка с целью контроля корректности результатов, полученных при счете на ускорителях.

Напомним также, что перемещения данных между регионами регулируются в директивах регионов соответствующими указаниями для используемых в них переменных, а перемещения данных во внерегионном пространстве – с помощью специальных исполнительных директив актуализации.

3. Отображение данных

В DVMH-программах все распределенные данные распределяются блочно: каждое распределенное измерение делится несколькими точками на отрезки. При этом есть возможность по каждому измерению распределенного массива задавать или равномерное блочное распределение, или распределение взвешенными блоками, т.е. с учетом заданного вектора весов. Эти указания непосредственно выполняются при распределении данных между MPI-процессами. Вложенные распределения, появляющиеся при входе в вычислительный регион, строятся с использованием этой информации, но, в силу разнородности вычислительных устройств, могут иметь отличающуюся от внешней схему распределения.

В DVM-системе разрабатываются три режима распределения данных по вычислительным устройствам в точках входа в регионы:

1. Простой статический режим.
2. Динамический режим с подбором схемы распределения.
3. Статический режим с использованием подобранной схемы распределения.

Рассмотрим подробнее эти режимы распределения.

3.1 Простой статический режим

В этом режиме в каждом регионе распределение производится одинаково. Пользователем задается вектор весов вычислительных устройств, имеющихся в каждом узле кластера (или они могут быть грубо определены автоматически), затем эти веса накладываются на параметры внешнего распределения данных, которое по каждому распределенному измерению может быть распределено как равномерно, так и с заданным вектором весов. В таком режиме сводятся к минимуму перемещения данных в связи с их перераспределением, но не учитывается различное соотношение производительности вычислительных устройств на разных участках кода.

3.2 Динамический режим с подбором схемы распределения

В этом режиме в каждом регионе распределение выбирается на основе постоянно пополняющейся истории запусков данного региона и его соседей, определяющихся динамически.

Каждый регион в данном режиме рассматривается в виде нескольких родственных инстанций, каждая из которых определяется парой (регион, соответствие данных), где под соответствием данных понимается соответствие используемых в коде региона локальных переменных реальным переменным (массивам или скалярам).

Для каждой инстанции определяются:

- Динамически предшествующие инстанции, являющиеся поставщиками актуальных входных данных.
- Зависимость времени работы от распределения данных, причем принимаются во внимание все инстанции одного и того же региона с учетом их разных соответствий данных.
- Общее время выполнения, как сумма всех времен выполнения инстанции.
- Количество вхождений

Зависимость времени работы от распределения данных строится в виде табличной функции времени от распределений распределенных массивов, которая является суммой таких же табличных функций, построенных для параллельных циклов, последовательных участков и хост-секций, содержащихся внутри данной инстанции региона.

В этом режиме периодически включается построение субоптимальной схемы распределения данных во всех инстанциях регионов на основе накопленных сведений в целом для программы, анализируя целиком последовательность инстанций регионов и их характеристики выполнения. При построении таких схем учитываются как внутренние показатели инстанций регионов в виде зависимости времени работы от распределения данных, так и последовательность исполнения инстанций с целью минимизации в том числе и временных затрат на перераспределение данных. После построения такой схемы, она применяется и происходит дальнейшее накопление характеристик и информации о регионах.

Есть возможность записи построенных схем распределения в файл для использования в последующих запусках программы, как в этом режиме, так и в третьем режиме. Также в качестве начального приближения может быть использован вектор весов вычислительных устройств в том же виде, как и для простого статического режима.

Из данного режима возможен переход в третий режим в любой точке выполнения программы, что обеспечивает упрощенную схему подбора и использования схемы распределения без необходимости сохранения параметров в файл и повторного запуска программы.

3.3 Статический режим с использованием подобранной схемы распределения

В этом режиме в каждом регионе распределение выбирается на основе предоставленной схемы распределения, построенной при работе программы во втором режиме, причем есть возможность, как перейти в этот режим непосредственно из второго, так и использовать схему распределения из файла, полученного в результате работы программы во втором режиме. При использовании схемы распределения из файла не гарантируется ее корректное использование в случае, если параметры программы были изменены, в особенности это касается тех параметров, которые влияют на путь выполнения программы (выбор другого метода расчета, отключение или включение этапов расчета).

4. Отображение вычислений

При отображении на разнородные вычислители параллельных циклов внутри вычислительных регионов необходимо решать задачи двух типов:

- Выбор метода обработки (обработчика) для конкретного вычислительного устройства, а также параметров для конкретного обработчика.
- Выполнение отображения на параллельную архитектуру.

Выбор обработчика и его параметров производится для всех вычислительных устройств в каждой инстанции вычислительного региона, содержащего данный параллельный цикл.

Так как вычислители разнородные, то разберем отдельно эти вопросы для ЦПУ и графических процессоров с архитектурой CUDA.

4.1 Отображение на ЦПУ

Обработчиков для ЦПУ может быть несколько, но, как правило, он присутствует в единственном числе и может быть параметризован количеством нитей и методом планирования их расписания.

В простом статическом режиме выбирается ЦПУ-обработчик по-умолчанию, а в других режимах – или исходя из имеющейся схемы распределения, или лучший по результатам произведенных запусков.

В простом статическом режиме количество нитей выбирается максимально доступное. В других режимах – максимальное из тех, которые дают лучшее время, чем меньшее количество нитей.

Метод планирования расписания нитей используется динамический или статический (в терминах OpenMP) и выбирается динамически, исходя из равномерности загрузки нитей при статическом планировании.

Выполнение отображения на ЦПУ делается простейшим образом с использованием технологии OpenMP.

4.2 Отображение на CUDA-устройство

Обработчиков для CUDA-устройств может быть несколько – могут быть применены разные подходы к оптимизации на уровне кода CUDA-ядра, использованы различные целевые архитектуры при их компиляции. Каждый обработчик может быть параметризован размерами CUDA-блока нитей и способом обработки редукционных операций.

В простом статическом режиме для всех CUDA-устройств выбирается CUDA-обработчик по-умолчанию, а в других режимах – или исходя из имеющейся схемы распределения, или лучший по результатам произведенных запусков.

Размеры CUDA-блока нитей для параллельного цикла могут быть заданы в тексте программы, и в этом случае будут использоваться эти размеры. Если же размеры CUDA-блока нитей не были заданы в тексте программы, то в простом статическом режиме блок выбирается фиксированным, заданным в настройках DVM-системы. В других режимах – или исходя из имеющейся схемы распределения, или лучший по результатам произведенных запусков.

Способ обработки редукционных операций может быть двух видов – с большим потреблением памяти устройства, но с более быстрым алгоритмом или с меньшим потреблением памяти устройства, но с менее быстрым алгоритмом. Если этот параметр поддерживается обработчиком, то способ выбирается динамически на основании имеющегося в данный момент объема свободной памяти устройства.

Выполнение отображения на CUDA-устройство выполняется с использованием технологии CUDA на усмотрение обработчика. Стандартным практикуемым подходом является отображение ровно одного витка параллельного цикла на одну CUDA-нить, причем пространство витков параллельного цикла разбивается на блоки методом замощения, при котором внутренние три измерения цикла ставятся в соответствие с измерениями блока нитей, а решетка блоков полагается одномерной и отвечает за покрытие всего пространства витков блоками. Для циклов с регулярными зависимостями применяются особого вида CUDA-обработчики, обрабатывающие пространство витков гиперплоскостями.

5. Сравнительная отладка вычислительных регионов

Для DVMH-программ есть возможность сравнительной отладки регионов, это специальный режим работы вычислительного региона, при котором вычисления повторяются на ЦПУ и других вычислительных устройствах с целью сравнения значений выходных переменных при завершении вычислительного региона. Такой механизм позволяет выявлять и локализовывать ошибки, проявляющиеся при работе на ускорителях.

Включение и использование этого механизма не требует от программиста менять программу или что-либо дополнительно сообщать о своей DVMH-программе. Реализация этого механизма основывается на том, что, во-первых, вычислительный регион может быть выполнен одновременно независимо на нескольких вычислительных устройствах и, во-вторых, системе поддержки выполнения DVMH-программ известен исчерпывающий набор входных и выходных данных региона, так как он ей необходим для управления перемещениями данных между устройствами.

В сравнение включаются все выходные данные вычислительного региона. При этом целочисленные данные сравниваются на совпадение, а вещественные числа сравниваются с заданной точностью по абсолютной и относительной погрешности. В случае нахождения расхождений пользователю выдается информация о них, и далее в программе используется та версия данных, которая была получена при счете на центральном процессоре.

6. Выводы

Новый подход к созданию прикладного программного обеспечения, разработанный в Институте прикладной математики им. М.В. Келдыша РАН, существенно упрощает создание прикладных программ для суперкомпьютерных систем с ускорителями. Языки модели DVMH обеспечивают высокий уровень переносимости прикладного ПО на системы с другими архитектурами ускорителей, поскольку перенос не требует изменения программы.

Кроме того, DVMH-программы обладают способностью гибко подстраиваться под конкретную вычислительную аппаратуру, на которой эти программы запущены.

Литература

1. DVM-система [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.keldysh.ru/dvm/ – 01.12.2012
2. Top500 List – November 2012 | TOP500 Supercomputer Sites [Электронный ресурс] – : [web-сайт] – Режим доступа: http://top500.org/list/2012/11/ – 01.12.2012
3. High Performance Fortran [Электронный ресурс] – : [web-сайт] – Режим доступа: http://hpff.rice.edu/ – 01.12.2012

4. Romain Dolbeau, Stéphane Bihan, and François Bodin. HMPP™: A Hybrid Multi-core Parallel Programming Environment.
URL: http://www.caps-entreprise.com/wp-content/uploads/2012/08/caps-hmpp-gpgpu-Boston-Workshop-Oct-2007.pdf (дата обращения 02.12.2012)

5. The Portland Group. PGI Accelerator Programming Model for Fortran & C.
URL: http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf (дата обращения 02.12.2012)

6. OpenACC [Электронный ресурс] – : [web-сайт] – Режим доступа: http://www.openacc-standard.org/ – 01.12.2012

7. T. D. Han and T. S. Abdelrahman. *hi*CUDA: High-Level GPGPU Programming. IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 78-90, Jan. 2011

8. В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. – Вестник Южно-Уральского государственного университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12 – Челябинск: Издательский центр ЮУрГУ, 2012, с. 82-92

9. Н.А. Коновалов, В.А. Крюков, С.Н. Михайлов, А.А. Погребцов. Fortran DVM – язык разработки мобильных параллельных программ. – Программирование, № 1, 1995, стр. 49-54

10. Н.А. Коновалов, В.А. Крюков, Ю.Л. Сазанов. C-DVM – язык разработки мобильных параллельных программ. – Программирование, № 1, 1999, стр. 54-65

Идентификация моделей радикально-цепного окисления органических соединений в присутствии ингибиторов параллельными методами условной глобальной оптимизации*

В.В. Рябов[1], М.В. Тихонова[2]

ННГУ им. Лобачевского [1], Институт нефтехимии и катализа РАН[2]

Данная работа предлагает новые методики идентификации параметров математических моделей сложных химических реакций на примере реакции радикально-цепного окисления органических соединений в присутствии ингибитора. Для учета ряда взаимосвязей между физико-химическими величинами, а также структурной зависимости между различными механизмами реакции обратная кинетическая задача решается как задача условной глобальной минимизации. В настоящей работе для решения задачи идентификации применяется параллельный индексный метод глобальной условной оптимизации, разрабатываемый в ННГУ им. Лобачевского и неоднократно опробованный ранее на моделях реакций металлокомплексного катализа.

1. Введение

Разработка математической модели химической реакции обычно сводится к определению неизвестных кинетических параметров по экспериментальным данным, то есть решается так называемая обратная кинетическая задача.

Практически любая обратная задача, как отмечается в работах академика А.Н. Тихонова, является некорректной.

Понятие корректной постановки задачи включает в себя три требования, которым должно удовлетворять решение поставленной задачи [1]:

1) существование решения;

2) единственность решения;

3) устойчивость решения к изменениям в исходных данных.

Точного решения обратной кинетической задачи обычно не существует из-за погрешности, с которой измеряются значения концентраций в химическом эксперименте. Концентрации некоторых веществ не измеряются из-за технологических ограничений, поэтому недостаточная информативность эксперимента часто приводит к неединственности решения. Одна из причин нарушения третьего условия – чувствительность констант скоростей реакций к температуре, особенно для стадий с большой энергией активации.

Поскольку в реальной практике нет возможности получить дополнительную информацию об изменении концентрации достаточного количества реагентов, в данной работе предлагается сузить круг получаемых решений обратной кинетической задачи за счет более полного использования информации о взаимосвязях между физико-химическими величинами и структурной зависимости между различными механизмами реакций. Для каждого химического эксперимента экспертные оценки физико-химических величин различаются, поэтому повышение информативности задачи для идентификации моделей требует индивидуального подхода для каждой исследуемой химической реакции.

Окислительно-восстановительные реакции представляют значительный интерес в органической химии. Получение продуктов окисления органических соединений является ценным крупнотоннажным производством важных кислородсодержащих соединений, таких как ацетон, фенол и др. Окисление органических соединений относится к классу вырожденно-разветвленных цепных реакций, механизм которых достаточно хорошо изучен [2, 3]. В то же время, далеко не все элементарные стадии этого механизма охарактеризованы кинетическими

* Работа выполнена при поддержке РФФИ (грант 12-07-00324), а также при поддержке гранта НШ-1960.2012.9.

константами скорости, что связано с методическими сложностями регистрации малых концентраций и крайне малым временем жизни промежуточных частиц (атомов и радикалов).

В связи с необходимостью подавления нежелательных эффектов в окислительно-восстановительных реакциях применяют ингибиторы, тормозящие окисление и порчу продуктов. В случае добавок ингибиторов или ингибирующих композиций описание механизма значительно усложняется, поскольку число элементарных стадий и промежуточных частиц увеличивается. Важно учитывать, что моделирование «усложненных» химических реакций должно проходить в согласовании с кинетическими параметрами исходного механизма. В таком случае задачу идентификации расширенной модели целесообразно решать в связке с исходной как задачу условной оптимизации.

В настоящей работе для решения задачи идентификации применяется параллельный индексный метод глобальной условной оптимизации, разрабатанный в ННГУ им. Лобачевского и неоднократно опробованный ранее на моделях реакций металлокомплексного катализа [4, 5]. Метод использует редукцию размерности на основе кривых Пеано и информационно-статистический подход, дополненный схемой построения множественных отображений (вращаемые развёртки) с адаптивно растущим уровнем детализации [6], позволяющих эффективно использовать сотни процессоров. Для ускорения поиска используется ряд модификаций.

2. Реакция радикально-цепного окисления н-декана

Детальный анализ кинетики и восстановления механизма реакций радикально-цепного окисления реакций в настоящей работе проведен на примере модельной реакции ингибированного окисления н-декана. Для разработки алгоритмов и создания программных модулей в качестве базового эксперимента выбраны экспериментальные результаты [3] по изучению окисления н-декана, ингибированного добавками классического ингибитора окисления - параоксидифениламина (ПОДА). При этом рассмотрена классическая схема ингибированного окисления (табл. 1), состоящая из ключевых элементарных стадий [1, 2]. В качестве инициатора И в работе [3] использовался пероксид дикумила, RH – субстрат окисления н-декана, InH – ингибитор, ROOH - гидроперекись.

Таблица 1. Классическая схема ингибированного окисления

Номер стадии	Химическое уравнение
(i1)	$И \rightarrow 2r^{\bullet}$
(i2)	$r^{\bullet} + RH \rightarrow R^{\bullet} + rH$
(1)	$R^{\bullet} + O_2 \rightarrow RO_2^{\bullet}$
(2)	$RO_2^{\bullet} + RH \rightarrow R^{\bullet} + ROOH$
(6)	$RO_2^{\bullet} + RO_2^{\bullet} \rightarrow ROOR + O_2$
(7)	$RO_2^{\bullet} + InH \rightarrow In^{\bullet} + ROOH$
(←7)	$In^{\bullet} + ROOH \rightarrow RO_2^{\bullet} + InH$
(8)	$In^{\bullet} + RO_2^{\bullet} \rightarrow ROOIn$

В окисляющихся спиртах скорость расходования ингибитора гораздо ниже, чем скорость образования радикалов. Связано это с явлением регенерации ингибиторов в реакциях обрыва цепей [2]. В углеводородах возможно вызвать регенерацию ингибитора, добавляя в окисляющийся субстрат спирт. В настоящей работе рассматривается окисление н-декана (*RH*), ингибированного композицией состава ПОДА (*InH*) и н-децилового спирта (*R'OH*). В этом случае механизм реакции ингибированного окисления (табл. 1) усложняется и дополняется несколькими элементарными стадиями (табл. 2).

Таблица 2. Дополнительные стадии регенерации ингибитора с добавлением спирта

Номер стадии	Химическое уравнение
(i2')	$r + (CH_3)_2CHOH \rightarrow (CH_3)_2C^{\bullet}(OH) + rH$
(1')	$O_2 + (CH_3)_2C^{\bullet}(OH) \rightarrow (CH_3)_2COHOO^{\bullet}$
(2')	$RO_2^{\bullet} + (CH_3)_2CHOH \rightarrow ROOH + (CH_3)_2C^{\bullet}(OH)$
(8')	$In^{\bullet} + (CH_3)_2COHOO^{\bullet} \rightarrow O_2 + InH + (CH_3)_2CO$

В лабораторных условиях (табл. 3) реакция ингибированного окисления н-декана с добавлением в окисляющийся субстрат спирта проводилась при различных концентрациях ингибитора. В ходе эксперимента наблюдалось поглощение кислорода O_2 и выдавалась его концентрация (моль/л) в отдельные моменты времени.

Таблица 3. Лабораторные условия проведения реакций ингибированного оксиления н-декана

Условие проведения реакции	**Классическая схема с ПОДА**	**Схема с ПОДА + спирт**		
	Опыт №6	**Опыт №29**	**Опыт №30**	**Опыт №31**
Объем V, мл	7	7		
t, °C	135	140		
[RH], моль/л	5.13			
[O2], моль/л	7.510-3			
[InH], моль/л	210-4			
[И], моль/л	3.8810-4			
[(CH3)2CHOH]. моль/л.	–	0.1	0.5	1.0
Длина индукционного периода, мин	18	20	33	54

В работе [7] с применением индексного метода глобальной оптимизации были удовлетворительно воспроизведены известные константы скоростей элементарных стадий, а также рассчитаны ранее неизвестные, описывающие механизм реакции с ингибитором ПОДА. Основной задачей для расширенной схемы реакции ингибированного окисления н-декана с добавлением спиртов является нахождение кинетических параметров дополнительных стадий (i2'), (1'), (2'), (8') на основе найденных констант таблицы 2.

3. Постановка задачи

В предыдущих работах [4, 5] обратная кинетическая задача решалась как задача глобальной минимизации критерия отклонения экспериментальных и расчетных данных:

$$\varphi(y) = \frac{1}{P}\sum_{k=1}^{P}\sum_{i=1}^{n}\sum_{l=1}^{H}\frac{\left|x_{kil}^{расч} - x_{kil}^{эксп}\right|}{x_{kil}^{эксп}} \to \min, \quad (1)$$

$$y = \left(\ln k_{(j)}, \ln k_{(-j)}\right), \quad (2)$$

где y – вектор минимизируемых параметров; $k_{(j)}$, $k_{(-j)}$ - приведенные константы скоростей прямой и обратной j-й элементарной стадии , *1/ч*, соответственно; *j,-j* – индекс прямой и обратной элементарной стадии; $x^{р}_{kil}$ - расчетные значения концентраций наблюдаемых веществ, получаемые в результате решения прямой кинетической задачи на векторе кинетических констант y, *мольные доли*; $x^{э}_{kil}$ - экспериментально полученные значения концентраций наблюдаемых веществ, *мольные доли*; H - количество наблюдаемых веществ; n - количество точек эксперимента; P - количество экспериментов. Вектор параметров (2) задачи (1) задается в виде логарифмов констант скоростей элементарных стадий, чтобы немного упростить структуру минимизируемого функционала: уменьшить эффект "широкого плато", который негативно сказывается на алгоритмах липшицевой оптимизации.

Исследование реакции ингибированного окисления н-декана, осложненного стадиями регенерации ингибитора (табл. 1 + табл. 2), требует учета констант скоростей элементарных стадий классического механизма (табл. 1), а именно, общие стадии обоих механизмов должны описываться одними и теми же кинетическими параметрами. В связи с явлением регенерации ингибиторов концентрация *InH* в окисляющихся спиртах должна быть больше, чем в классической схеме ингибированного окисления.

Ввиду структурной взаимосвязи между двумя описанными механизмами, задачу математического моделирования процесса ингибированного окисления н-декана целесообразно решать как задачу условной оптимизации.

Введем некоторые обозначения:

- набор кинетических параметров классического механизма (табл. 1) охарактеризуем вектором y ^{classic} :

$$y^{classic} = \left(\ln k_{(i1)}, \ln k_{(i2)}, \ln k_{(1)}, \ln k_{(2)}, \ln k_{(6)}, \ln k_{(7)}, \ln k_{(8)}, \ln k_{(\leftarrow 7)} \right).$$

- набор кинетических параметров расширенного механизма со стадиями регенерации ингибитора (табл. 1 + табл. 2) обозначим через вектор y:

$$y = \left(y^{classic}, \ln k_{(i2')}, \ln k_{(1')}, \ln k_{(2')}, \ln k_{(8')} \right)$$

- *g(y)* – функционал обратной кинетической задачи (1) для классического механизма реакции ингибированного окислений н-декана (табл. 1);
- g_{min} – минимум функционала *g*, полученный на наборе констант из таблицы 4.
- *F(y)* – функционал обратной кинетической задачи (1) для расширенного механизма со стадиями регенерации ингибитора (табл. 1 + табл. 2).

Требуется найти набор констант скоростей элементарных стадий *y*, который минимизирует функционал *F(y)*, то есть описывает механизм реакции ингибированного окисления н-декана в окисляющихся спиртах (табл. 1 + табл. 2). При этом нужна гарантия, что описание классического механизма не ухудшится, то есть отклонение $g\left(y^{classic}\right)$ от уже найдено минимума g_{min} не превысит заданного ε:

$$g\left(y^{classic}\right) \leq g_{\min} + \varepsilon, \tag{3}$$

Учет соотношения концентрации ингибитора *InH* в исследуемых механизмах задается следующим неравенством:

$$g_H\left(y\right) = \sum_{m=1}^{M} \left([InH]\left(t_m, y\right) - [InH]^{classic}\left(t_m, y^{classic}\right) \right) \geq 0, \tag{4}$$

где *M* – некоторое количество моментов времени t_m для сравнения концентраций.

Для автоматизированной выборки адекватных решений накладывается дополнительное условие на длину индукционного периода. Период индукции – одна из важных кинетических особенностей самоускоряющихся реакций. Период индукции какого-то вещества (или нескольких веществ) определяется в [1] как интервал времени, отсчитанный с момента, когда реакция начала протекать в режиме прогрессирующего самоускорения и до момента, при котором это самоускорение резко усиливается.

Сумма относительных отклонений рассчитанных в ходе решения прямых кинетических задач индукционных периодов $T^{расч}{}_k$ от экспериментальных $T^{эксп}{}_k$ по всем P экспериментам не должно превышать 10%:

$$g_T\left(y\right) = \frac{1}{P} \sum_{k=1}^{P} \frac{\left| T_k^{расч}\left(y\right) - T_k^{эксп}\left(y\right) \right|}{T_k^{эксп}\left(y\right)} \leq 0.1. \tag{5}$$

Для вычисления условия (5) обратной кинетической задачи автоматизирован лабораторный способ определения индукционного периода как пересечения двух касательных, проведенных в начальный t_0 и конечный t_N момент времени протекания реакции (рис. 1).

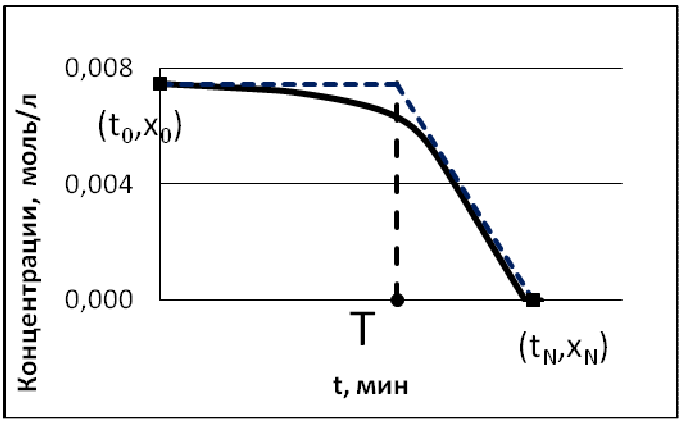


Рис. 1. Лабораторный способ определения периода индукции

Тогда период индукции T будет вычисляться из формулы:

$$T = \frac{x(0) - x(t_N) + x'(t_N) \cdot t_N}{x'(t_N) - x'(t_0)}. \tag{6}$$

Таким образом, обратная кинетическая задача идентификации моделей ингибированного радикально-цепного окисления н-декана формируется как задача условной глобальной оптимизации:

$$\begin{aligned} & F(y) \to \min, \\ & g(y) - g_{\min} - \varepsilon \le 0, \\ & -g_H(y) \le 0, \\ & g_T(y) - 0.1 \le 0. \end{aligned} \tag{7}$$

4. Параллельный индексный метод условной глобальной оптимизации

Алгоритмы, развиваемые Нижегородской научной школой многоэкстремальной оптимизации, предполагают следующую постановку задачи:

$$\begin{gathered} \varphi^* = \min\{\varphi(y):\ y \in D\}, \\ g_j(y) \le 0,\ \ 1 \le j \le M, \\ D = \{y \in R^N:\ a_i \le y_i \le b_i,\ \ 1 \le i \le N\}, \end{gathered} \tag{8}$$

где целевая функция $\varphi(y)$ удовлетворяет условию Липшица с соответствующей константой L, а именно

$$|\varphi(y_1) - \varphi(y_2)| \le L \|y_1 - y_2\|,\ \ y_1, y_2 \in D.$$

Функции $g_j(y)$ также удовлетворяют условию Липшица с константами L_j.

Используя кривые типа развертки Пеано $y(x)$, однозначно отображающие отрезок $[0,1]$ на N-мерный гиперкуб P

$$P = \{y \in R^N:\ -2^{-1} \le y_i \le 2^{-1},\ \ 1 \le i \le N\} = \{y(x):\ 0 \le x \le 1\},$$

исходная задача редуцируется к одномерной:

$$\varphi(y_D(x^*)) = \min\{\varphi(y_D(x)):\ x \in [0,1]\}.$$

Для решения редуцированной одномерной задачи используется эффективный алгоритм с индексной схемой учёта ограничений [8] и рядом модификаций [6, 9, 10].

4.1 Индексная схема учёта ограничений и ε-резервирование

Итерация в каждой точке состоит в последовательной проверке ограничений $g_j(y(x))$, которая заканчивается либо обнаружением нарушенного ограничения, либо вычислением критерия $\varphi(y(x))$. Номер последней вычисленной функции в точке x называется её *индексом* и обозначается $1 \le iz(x) \le M+1$. Результатом итерации в точке является пара (iz, z), где z – значение последней вычисленной функции.

Данная схема допускает частичную вычислимость функций задачи (8), а также не приводит к катастрофическому росту константы Липшица в недопустимой области (как это происходит в методе штрафных функций). Она подробно описана во множестве работ, например, в [8].

Индексная схема учёта ограничений обладает одним недостатком: на границе допустимой области (где одно или несколько ограничений активны) строится неоправданно плотное покрытие точками итераций. Для борьбы с этим эффектом используется так называемое ε-резервирование с адаптивной оценкой резервов [10]. Важным параметром для данной модификации является достаточно малое число q, которое обычно берут равным 0.01 или 0.1 (чем оно ближе к нулю, тем ближе алгоритм к исходной реализации).

5. Вычислительные эксперименты на кластере ННГУ

В полученной задаче условной оптимизации целевая функция и ограничения являются многоэкстремальными, как это показано на одномерных сечениях (рис. 2 и рис. 3), сделанных относительно найденной оптимальной точки (зафиксированы оптимальные значения всех параметров, кроме y_1). Рисунок 2 также иллюстрирует частичную вычислимость целевой функции, которая допускается в индексном методе.

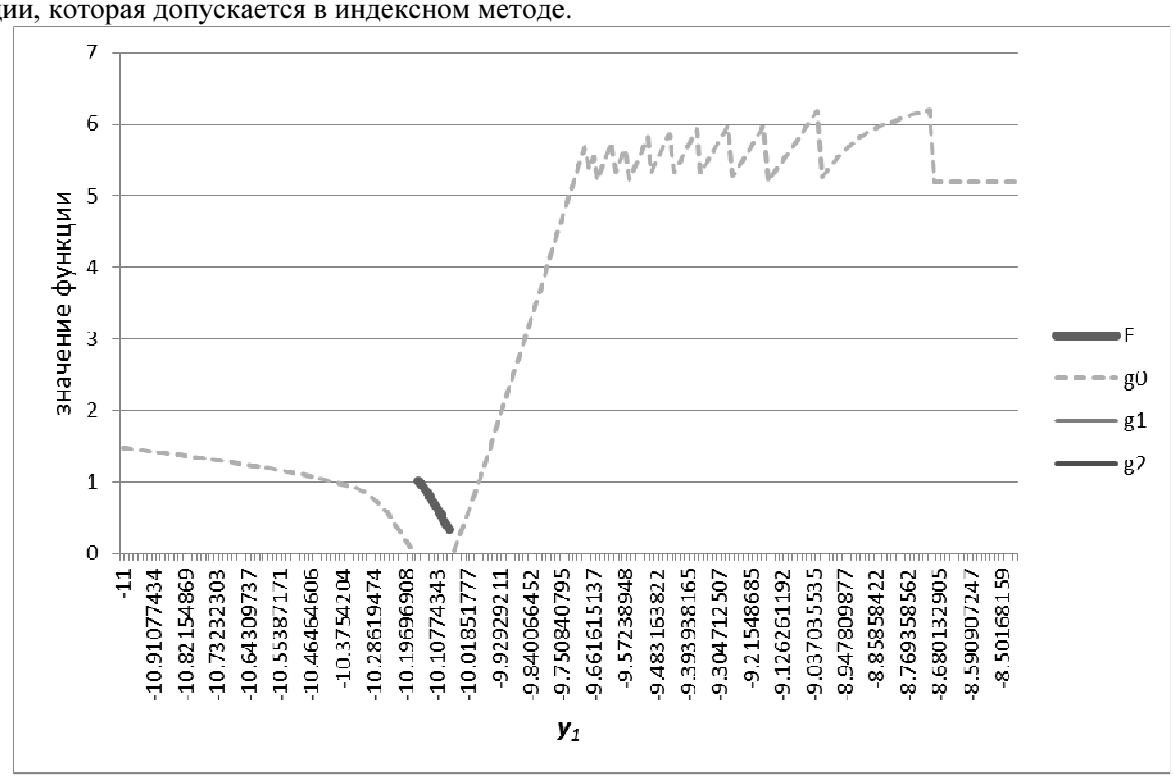


Рис. 2. Ограничение $g_0(y)$ и целевая функция $F(y)$ в допустимой области. Сечение по y_1.

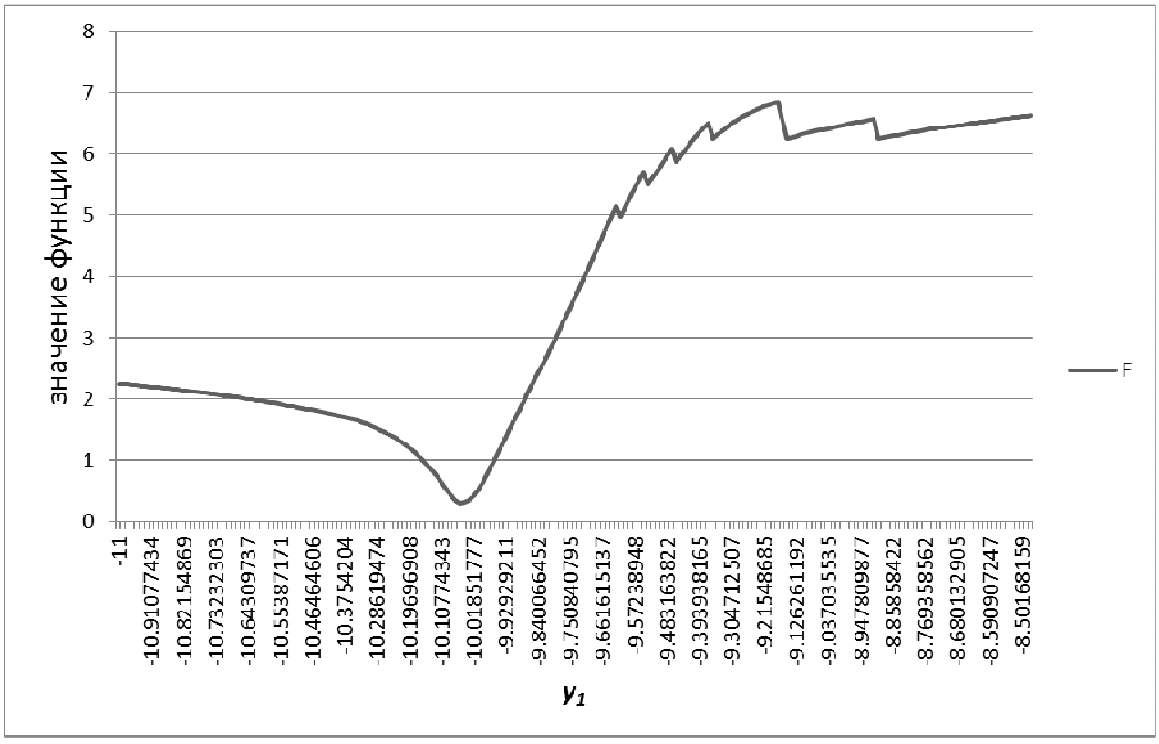


Рис. 3. Сечение целевой функции *F(y)* при отсутствии функциональных ограничений

Вычисления выполнены на новом кластере ННГУ, содержащем 16 вычислительных узлов. Характеристики каждого узла:

- два 4-ядерных процессора Intel Xeon L5630 2,13 ГГц (8 ядер на узел, всего – 128);
- 24 Гб оперативной памяти DDR3 (3 Гб на ядро);

Кластер работает под управлением операционной системы Windows Server 2008. Теоретическая пиковая производительность кластера (без учёта GPU) составляет 2176 Гигафлоп.

Параметры индексного метода глобальной оптимизации: надежность r=2.5, число вращаемых разверток равно количеству вычислительных ядер, максимальный уровень детализации разверток m=15, параметр адаптивного ε-резервирования q=0.1, условие остановки по числу итераций – 500 000 (пятьсот тысяч), условие остановки по точности ε=0.001.

При любом количестве вычислительных ядер параллельный индексный метод находит оценку оптимума задачи (6), приведенную в таблице 5, хотя во всех случаях алгоритм завершает работу по максимальному числу итераций. Эффективность использования вычислительность ресурсов иллюстрируют таблица 4 и рисунок 4.

Таблица 4. Время решения обратной задачи на кластере ННГУ

Кол-во развёрток	4	4	4	8	12	16	20
Кол-во ядер	1	2	4	8	12	16	20
Время, сек.	60274	45922	20772	13738	8734	12011	6056
Ускорение		1.31	2.9	4.39	6.9	5.02	9.95
Эффективность		65.5%	72.5%	54.9%	57.5%	31.4%	49.8%

Кол-во развёрток	30	40	50	60	70	80	90
Кол-во ядер	30	40	50	60	70	80	90
Время, сек.	5785	4731	7245	2900	3490	1986	2085
Ускорение	10.42	12.74	8.32	20.78	17.27	30.35	28.9
Эффективность	34.7%	31.9%	16.6%	34.6%	24.7%	37.9%	32.1%

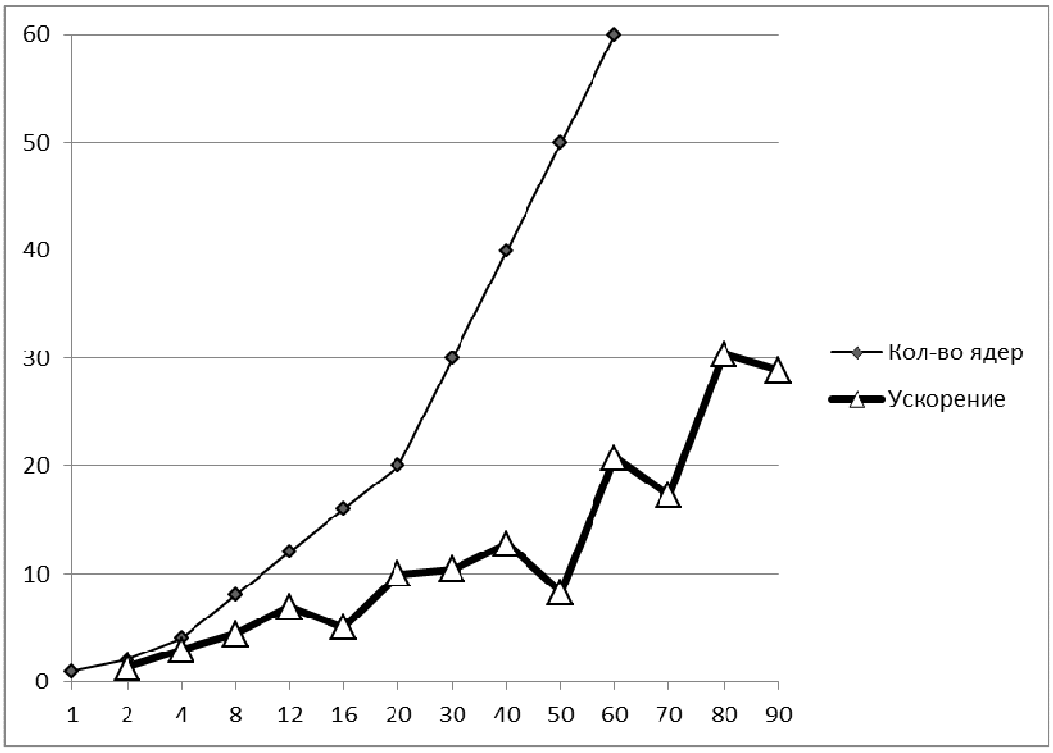


Рис. 4. Ускорение параллельного индексного метода на данной задаче

Полученные данные свидетельствуют о том, что существуют резервы для наращивания эффективности параллелизма в технической реализации индексного метода, поскольку теоретические оценки избыточности [8] для классического параллельного индексного метода (без модификаций) говорят о том, что идеально достижимая алгоритмическая эффективность распараллеливания – 83% и выше (избыточность – не более 17%).

6. Результаты моделирования реакции ингибированного окисления н-декана

Для построения математических моделей реакции ингибированного окисления н-декана проведен ряд вычислительных экспериментов по решению обратных кинетических задач. Область поиска кинетических параметров ограничивалась, исходя из экспертной информации о порядках величин констант скоростей элементарных стадий для сохранения радикально-цепного режима протекания реакции.

На каждой итерации решения обратной кинетической задачи проводилась проверка выполнения ограничений (2), (3). В качестве g_{min} бралось значение функционала (1), рассчитанного для классического механизма реакции ингибированного окисления (табл.1), которое составило 0.48. Допускалось отклонение функционала g на 30% от найденного минимума, таким образом, $\varepsilon = 0.6$.

В результате решения задачи (1)-(3) был определен оптимальный набор констант скоростей элементарных стадий (табл. 5). Важным требованием, предъявляемым к искомому решению обратной кинетической задачи, является не противоречие констант литературным данным о порядке их величин. При этом в различных литературных данных величины констант различаются в пределах порядка. Это допущение было заложено в виде границ области поиска варьируемых параметров. В таблице 5 представлено сопоставление найденных кинетических констант литературным данным [1, 2].

Значение функционала (1) в этой точке составило 0.304. Суммарное относительное отклонение рассчитанных длин периодов индукции по условию (5) составило 9%. На рисунке 5 представлено сопоставление расчетных и экспериментальных данных для двух исследуемых механизмов (рис. 5).

Таблица 5. Константы скоростей элементарных стадий реакции ингибированного окисления н-декана с добавлением спиртов

Номер	Константы скоростей стадий	
	Рассчитанные индексным методом	**Литературные данные для 135°C**
$k_{(i1)}$, 1/с	$4.14 \cdot 10^{-5}$	10^{-4}
$k_{(i2)}$, л/(моль•с)	$1.02 \cdot 10^{9}$	10^{9}
$k_{(1)}$, л/(моль•с)	$1.02 \cdot 10^{9}$	10^{9}
$k_{(2)}$, л/(моль•с)	3.04	10
$k_{(6)}$, л/(моль•с)	$1.34 \cdot 10^{6}$	10^{6}
$k_{(7)}$, л/(моль•с)	$6.52 \cdot 10^{5}$	10^{5}
$k_{(\leftarrow 7)}$, л/(моль•с)	$2.70 \cdot 10^{2}$	10^{3}
$k_{(8)}$, л/(моль•с)	$5.77 \cdot 10^{6}$	10^{6}
$k_{(i2')}$, л/(моль•с)	$2.08 \cdot 10^{9}$	10^{10}
$k_{(1')}$, л/(моль•с)	$4.85 \cdot 10^{8}$	10^{9}
$k_{(2')}$, л/(моль•с)	1.11	-
$k_{(8')}$, л/(моль•с)	$3.28 \cdot 10^{4}$	-

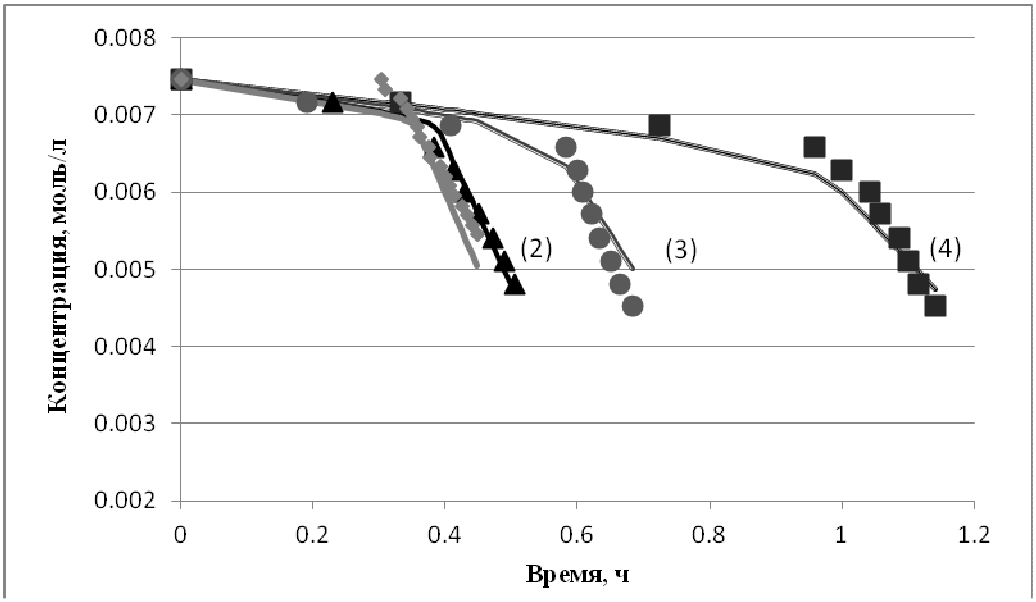


Рис. 5. Кинетические кривые расходования кислорода O_2 при окислении н-декана в присутствии ПОДА (1) и ПОДА+спирт: (2) - [(CH3)2CHOH] = 0.1 моль/л, (3) - [(CH3)2CHOH] = 0.5 моль/л, (4) - [(CH3)2CHOH] = 1.0 моль/л. Точки – экспериментальные данные, сплошные линии – расчет.

Значение функционала *g* на найденном наборе параметров (табл. 5) составило 0.577. Таким образом, достигнуто выполнение условия (3). Кинетические кривые (рис. 6) иллюстрируют выполнение условия (4). Вместе с тем наблюдается замедление расходования ингибитора *InH* с ростом начальной концентрации спирта. Условие (5) выполняется с точностью 0.012.

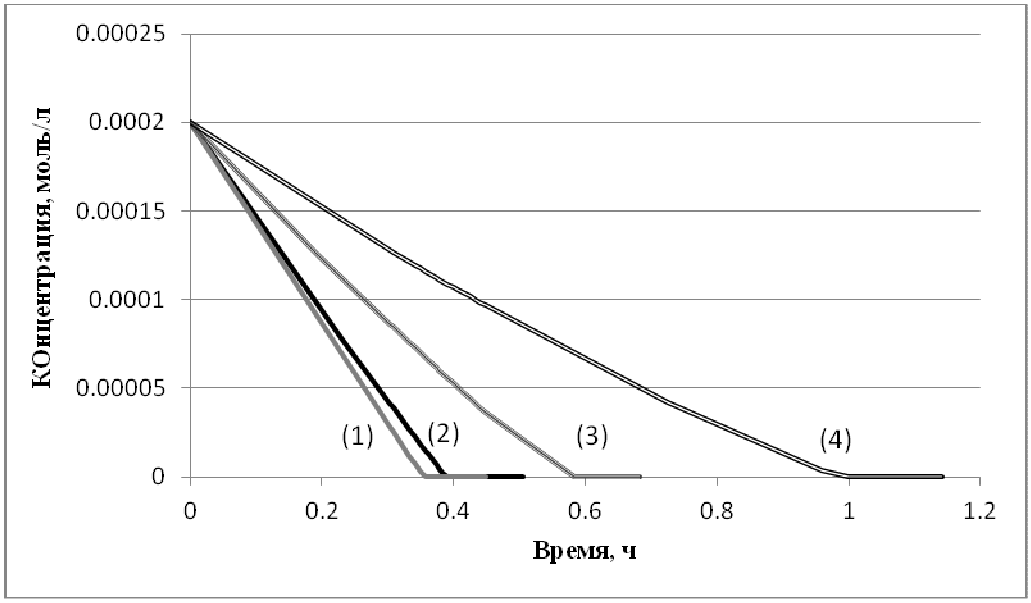


Рис. 6. Сравнение кинетических кривых расходования ингибитора *InH* при окислении н-декана в присутствии ПОДА (1) и ПОДА+спирт: (2) - [(CH3)2CHOH] = 0.1 моль/л, (3) - [(CH3)2CHOH] = 0.5 моль/л, (4) - [(CH3)2CHOH] = 1.0 моль/л.

Заключение

В настоящей работе разработана методика решения обратных кинетических задач как задач условной глобальной оптимизации, которая позволяет идентифицировать механизмы двух реакций, не формулируя задачу как двухкритериальную. Дополнительно учитывается информация об индукционном периоде. Исследована эффективность распараллеливания индексного метода с модификациями на данной задаче. Непосредственным результатом работы стало восстановление полной кинетической картины модельной реакции ингибированного окисления н-декана, а также более сложной реакции окисления органических соединений с добавлением в окисляющийся субстрат спирта.

Литература

1. Денисов Е.Т., Азатян В.В. Ингибирование цепных реакций. // Черноголовка, 1996. –268 с.
2. Денисов Е.Т., Мицкевич Н.И., Агабеков В.Е. Механизм жидкофазного окисления кислородсодержащих соединений. // Минск. Наука и техника, 1975. – 335 с.
3. Гарифуллина Г.Г., Герчиков А.Я., Осипова С.А.. Регенерация ингибитора добавками многоатомных спиртов в окисляющемся декане. // Башкирский химический журнал. 1997. Т.4. №3.С. 64-65.
4. Губайдуллин И.М., Рябов В.В., Тихонова М.В. Применение индексного метода глобальной оптимизации при решении обратных задач химической кинетики. // Вычислительные методы и программирование, 2011. Т.12. С. 137-145.
5. Рябов В.В., Тихонова М.В., Губайдуллин И.М. Методы глобальной оптимизации и исследование эффективности химических реакций карбоалюминирования олефинов // Труды XI Всероссийской конференции "Высокопроизводительные параллельные вычисления на кластерных системах", Нижний Новгород, 2011. С. 278-281.
6. Сидоров С.В., Рябов В.В. Параллельные алгоритмы глобальной оптимизации и использование развёрток растущего уровня детализации // Вестник ННГУ, серия “Математическое моделирование и оптимальное управление”, №3, 2011. С. 127-133.

7. Тихонова М.В., Гарифуллина Г.Г., Герчиков А.Я., Спивак С.И. Программный комплекс математической идентификации кинетической модели ингибированного окисления кумола. - Обратные задачи химии. // Сборник статей научно-практической конференции. - Бирск: БирГСПА, 2011. С. 311-317
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., Рябов В.В., Сидоров С.В. Параллельные вычисления в задачах многоэкстремальной оптимизации. // Вестник ННГУ, 2009. №6. С. 171-177.
10. Баркалов К.А. Ускорение сходимости в задачах условной глобальной оптимизации. Нижний Новгород: изд-во Нижегородского гос. ун-та, 2005.

DiVTB Server: среда выполнения виртуальных экспериментов*

Д.И. Савченко, Г.И. Радченко

Южно-Уральский государственный университет

В статье представлена архитектура программной системы DiVTB Server – среды управления виртуальными экспериментами, построенными на основе концепции распределенных виртуальных испытательных стендов (РаВИС). Описаны структура программной системы и алгоритмы ее работы, а также структура РаВИС.

1. Введение

На сегодняшний день процесс решения прикладных задач 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, таких как архитектура суперкомпьютеров, навыки работы в Unix-подобных операционных системах, настройка и администрирование удаленного доступа, умение работать с очередями приложений и т. д. С другой стороны, современные системы решения прикладных задач представляют собой многофункциональные программные комплексы, состоящие из множества отдельных программных подсистем со сложным пользовательским интерфейсом. Проблема сопряжения компонент существенно усложняется при использовании одновременно двух и более различных прикладных пакетов для решения одной задачи. Все эти факторы затрудняют широкое внедрение систем суперкомпьютерного моделирования в практику НИОКР.

Эта проблема может быть решена при помощи концепции *распределенных виртуальных испытательных стендов (РаВИС)* – обеспечивающей проведение проблемно-ориентированного моделирования заранее определенных классов задач на базе ресурсов удаленной распределенной вычислительной среды (РВС) [5] в соответствии с моделью облачных вычислений (*Cloud Computing*) [7, 8]. Именно на концепции облачных вычислений и базируется система *CAEBeans* [4], предоставляющая пользователю доступ к ресурсам распределенной вычислительной среды в виде простого пользовательского веб-интерфейса. Развитием программного комплекса *CAEBeans*, обеспечивающего формирование и использование иерархий проблемно-ориентированных оболочек для предоставления ресурсов РВС, стал программный комплекс *DiVTB*. *DiVTB* включает в себя такие компоненты как:

1. *DiVTB Portal* – веб-приложение, обеспечивающее выбор, запуск и получение результатов моделирования РаВИС [1];
2. *DiVTB Server* – хранилище и среда исполнения РаВИС;
3. *DiVTB Broker* – автоматизированная система регистрации, анализа и предоставления DiVTB-ресурсов [6];
4. *DiVTB Infoservice* – компонент, предоставляющий системе информацию об имеющихся ресурсах (версиях, лицензиях, ограничениях на доступные вычислительные ресурсы и сервисы и т. д.);
5. *DiVTB-ресурсы* – грид-сервисы, обеспечивающие удаленную постановку и решение прикладных задач.

Система DiVTB построена на основе грид-среды UNICORE [3], которая обеспечивает прозрачную интеграцию классических приложений в виде ресурсов грид-сети.

* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 11-07-00478-а и Министерства образования и науки РФ (государственное задание 8.3786.2011).

2. DiVTB Server

2.1. Основные понятия системы DiVTB

В рамках данной статьи введем следующие основные понятия:

— *проблемный слой РаВИС* – описание набора проблемно-ориентированных параметров РаВИС;

— *логический план РаВИС* – ориентированный граф, в вершинах которого могут находиться узлы двух типов: узлы действия и узлы управления, а ребра представляют собой отношения, описывающие потоки управления между двумя узлами;

— *физический слой РаВИС* – описание преобразования входных и выходных параметров действий в форматы входных и выходных файлов конкретных вычислительных сервисов;

— *дескриптор РаВИС* – полное описание РаВИС, включающее в себя структуры физического слоя и логического плана РаВИС, а также формат входных параметров для проведения виртуального эксперимента;

— *виртуальный эксперимент* – это запуск определенного РаВИС на конкретном наборе начальных данных.

2.2. Структура DiVTB Server

На рисунке 1 представлена структура DiVTB Server.

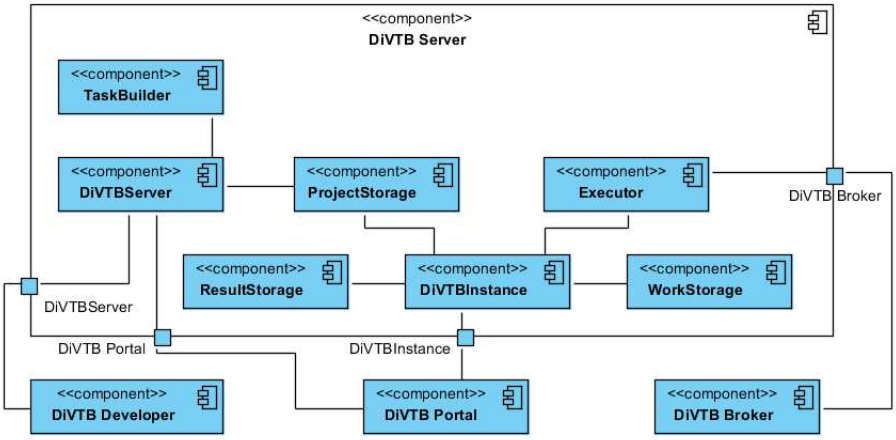


Рис. 1. Структура DiVTB Server

В состав DiVTB Server входят следующие компоненты:

1. *ProjectStorage* – ведет учет проиндексированных РаВИС и хранит их дескрипторы, по запросу выдает ссылки на файлы описания проекта РаВИС и файл проблемного слоя РаВИС;
2. *TaskBuilder* – выполняет преобразование описания РаВИС и проблемного слоя РаВИС в дескриптор РаВИС;
3. *WorkStorage* – хранит промежуточные результаты выполнения эксперимента;
4. *ResultStorage* – обеспечивает учет и хранение результатов выполнения экспериментов;
5. *DiVTBServer* – предоставляет доступ извне к методам, касающимся функциональности сервера;
6. *DiVTBInstance* – предоставляет доступ извне к методам, касающимся конкретного виртуального эксперимента;
7. *Executor* – обеспечивает исполнение эксперимента согласно дескриптора РаВИС, построенного компонентом TaskBuilder.

Для связи компонентов был использован фреймворк *Guice* [2], позволяющий каждому компоненту избавиться от ссылок на объекты и взаимодействовать с другими компонентами только при помощи ссылок на интерфейсы, т. е. DiVTB Server является слабосвязанной системой.

На рисунке 2 приведены интерфейсы компонентов DiVTBServer и DiVTBInstance.

```
public interface DiVTBServer  {
    //  Создать виртуальный эксперимент
    CreateInstanceResponse createInstance();
    //  Загрузить РаВИС
    void uploadProject(DataHandler content);
    //  Индексировать РаВИС
    void indexProject(String projectId);
    //  Индексировать все РаВИС
    void indexAllProjects();
    //  Получить идентификаторы развернутых РаВИС
    List<String> getProjectsIDs();
    //  Генерировать идентификатор нового РаВИС
    String generateID();
    //  Получить проблемный слой РаВИС
    SubmitJob getProblemCaebean(String id);
}

public interface Instance  {
    //  Запустить эксперимент
    SubmitJobResponse submitJob(SubmitJob submitJobDocument);
    //  Получить статус эксперимента
    GetStatusResponse getStatus(String instanceID);
    //  Получить время исполнения эксперимента
    long getExecTime(String instanceID);
    //  Получить результаты выполнения эксперимента
    byte[] getResultsArch(String instanceID);
    //  Получить содержимое рабочей директории эксперимента
    byte[] getWorkArch(String instanceID);
    //  Удалить директорию результатов эксперимента
    void removeResultDir(String instanceID);
    //  Загрузить дополнительные файлы
    void uploadUpdates(String projectID, DataHandler content);
}
```

Рис. 2. Интерфейсы компонентов DiVTBServer и DiVTBInstance

3. Обработка проекта РаВИС

Для разработки РаВИС используется среда *DiVTB Developer* (далее Developer), разработанная в рамках реализации технологии DiVTB. Для экспорта в РаВИС из Developer в Server создаются следующие файлы:

1. *Project.xml* содержит описание логического, физического слоя и описание дерева проблемных оболочек;
2. *<идентификатор испытательного стенда>.xml* содержит описание проблемного слоя РаВИС – возможных параметров постановки эксперимента, значения по умолчанию для всех этих параметров, а также их названия;
3. *Набор файлов-шаблонов* описывающих физический слой РаВИС и обеспечивающих автоматизированное формирование файлов постановки задач, сформированных в соответствии с требованиями конечных вычислительных DiVTB-сервисов;
4. Прочие файлы, необходимые для расчета эксперимента.

При экспорте РаВИС на сервер, происходит его индексация и создается общий дескриптор РаВИС при помощи компонента TaskBuilder.

Исполнение виртуального эксперимента можно условно разделить на три стадии: построение дескриптора задачи, исполнение эксперимента в соответствии с его логическим слоем и обращение к DiVTB-ресурсам таким образом, как это описано в физическом слое эксперимента.

3.1. Построение дескриптора задачи

Построение общего дескриптора РаВИС происходит при индексации проекта, построенный дескриптор сохраняется в базе данных РаВИС. TaskBuilder обеспечивает преобразование документов, содержащих описание РаВИС в дескриптор РаВИС, используемый затем компонентом Executor. На рисунке 3 отражена сущность преобразования описания РаВИС и проблемного слоя эксперимента в комплексное описание задачи.

Дескриптор содержит список узлов РаВИС, каждый из которых содержит ссылку на контекст (объект, содержащий текущую рабочую директорию для узла и задачи в целом, информацию из компонентного DiVTB, а также информацию о текущем проекте), список ссылок на следующие узлы (в случае начального узла – список из одного элемента, в случае конечного — пустой список), а также метод *exec*, инициирующий выполнение задачи, характерной для данного конкретного узла.

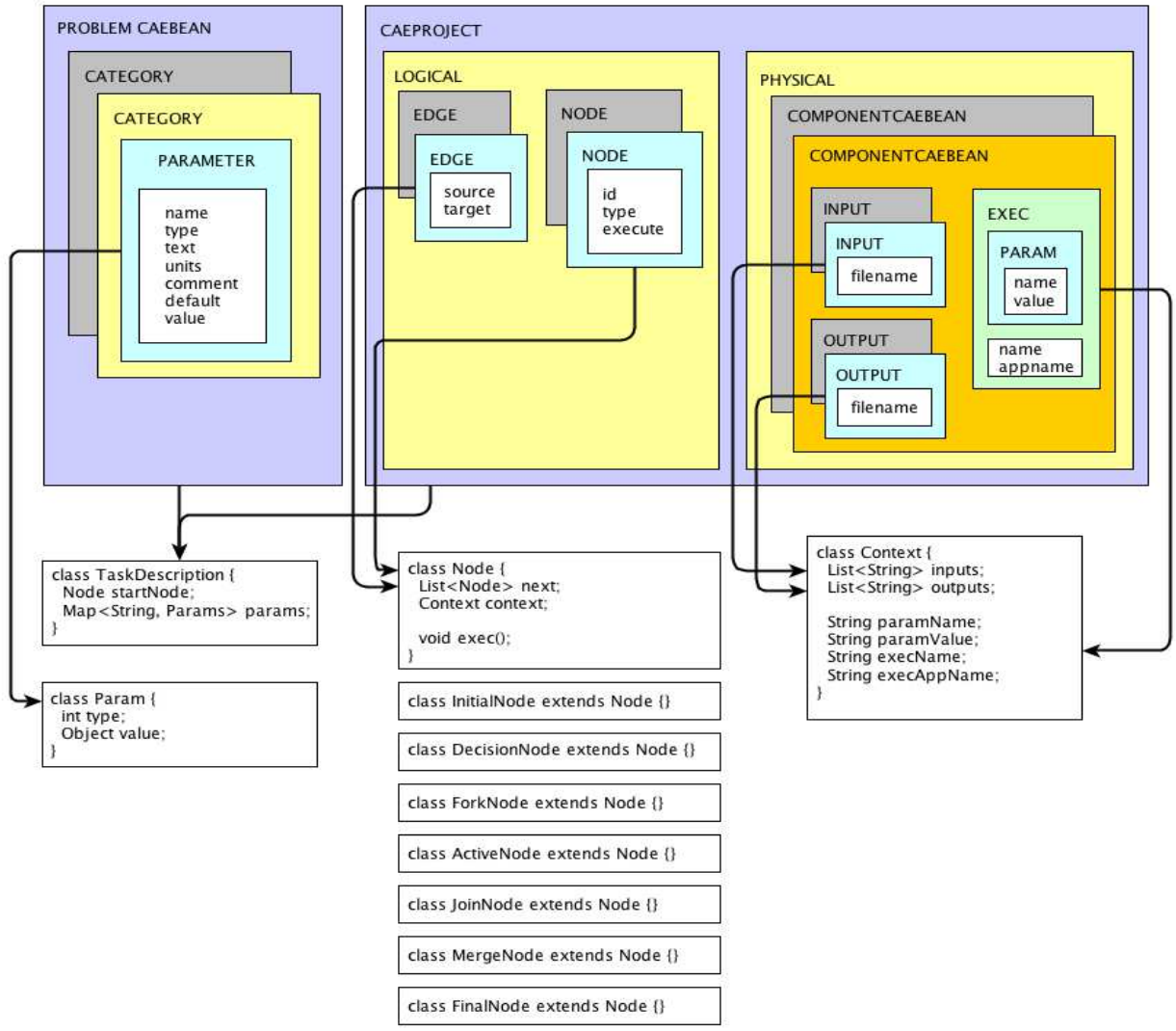


Рис. 3. Преобразование описания стенда в дескриптор

3.2. Обработка логического слоя РаВИС

Как было сказано выше, дескриптор содержит ориентированный граф, в вершинах которого могут находиться узлы, представляющие решение подзадач эксперимента посредством отдельного DiVTB-ресурса, либо специальные служебные узлы – распараллеливание, выбор,

начальный узел, конечный узел, слияние параллельных ветвей, окончание выбора. По этому графу и происходит исполнение эксперимента компонентом *Executor*.

Перед началом работы дескриптор РаВИС отправляется компоненту DiVTB Broker для анализа задач, входящих в эксперимент, и резервирования ресурсов РВС для выполнения этих задач. В ответ DiVTB Broker возвращает список идентификаторов физических узлов грид-сети (TSS – Target System Interface), на которых были зарезервированы ресурсы. Для различных подзадач могут быть зарезервированы ресурсы разных вычислительных узлов. Планирование и распределение ресурсов проводится единоразово для всего виртуального эксперимента.

Изначально исполнение виртуального эксперимента начинается с одного потока, поочередно исполняющего все узлы логического плана РаВИС, которые встречаются ему на пути. При попадании в узел ветвления (*Fork*) происходит создание новых потоков, которые начинают исполнение параллельных ветвей, основной же поток продолжает исполнение первой из веток. При попадании в узел слияния каждый поток может вести себя по-разному (см. рис. 4):

1. если поток является последним из потоков, которые должны прийти в узел, он продолжает свое исполнение обычным образом;
2. если же поток не является последним ожидаемым, его выполнение завершается.

Перед исполнением узла действия происходит вызов препроцессора, который обеспечивает формирование файлов постановки задач для очередного действия на основе соответствующих файлов-шаблонов.

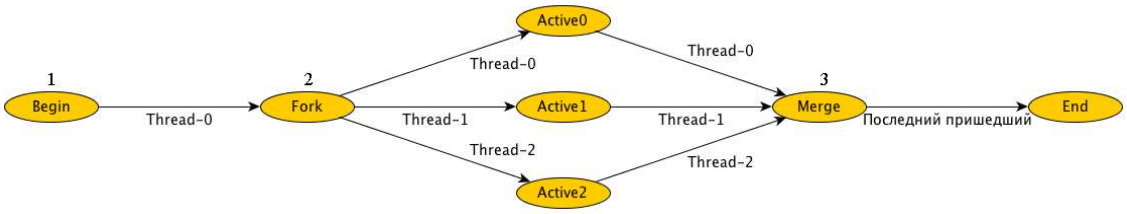


Рис. 4. Обработка логического слоя эксперимента

3.3. Обработка физического слоя РаВИС

Каждый узел действия имеет собственную физическую составляющую, описывающую входные и выходные файлы и параметры инженерного ресурса, при помощи которого будет выполнен очередной шаг эксперимента, а также атрибуты, необходимые для обращения к этому ресурсу при помощи грид-среды – имя и версию ресурса.

Перед обращением к ресурсу происходит предобработка файлов, которые поступят на вход ресурса. Файлы, проходящие предобработку, являются шаблонами на языке JMTE, в которые передается список переменных из проблемного слоя текущего эксперимента. Пример шаблона приведен на рисунке 5.

```
${foreach cart.items item}
  ${item.amount}x ${item.name} ${item.price} each = ${item.total}
${end}

${if cart.addShipping}
  Shipping = ${cart.shippingCost}
${end}
```

Рис. 5. Пример шаблона на языке JMTE

По завершению предобработки происходит загрузка файлов постановки задачи на DiVTB-ресурс , предоставленный компонентом DiVTB Broker ранее. По завершению работы инженерного ресурса, DiVTB Server выгружает результаты исполнения, описанные в физическом слое узла, обратно (см. рис. 6).

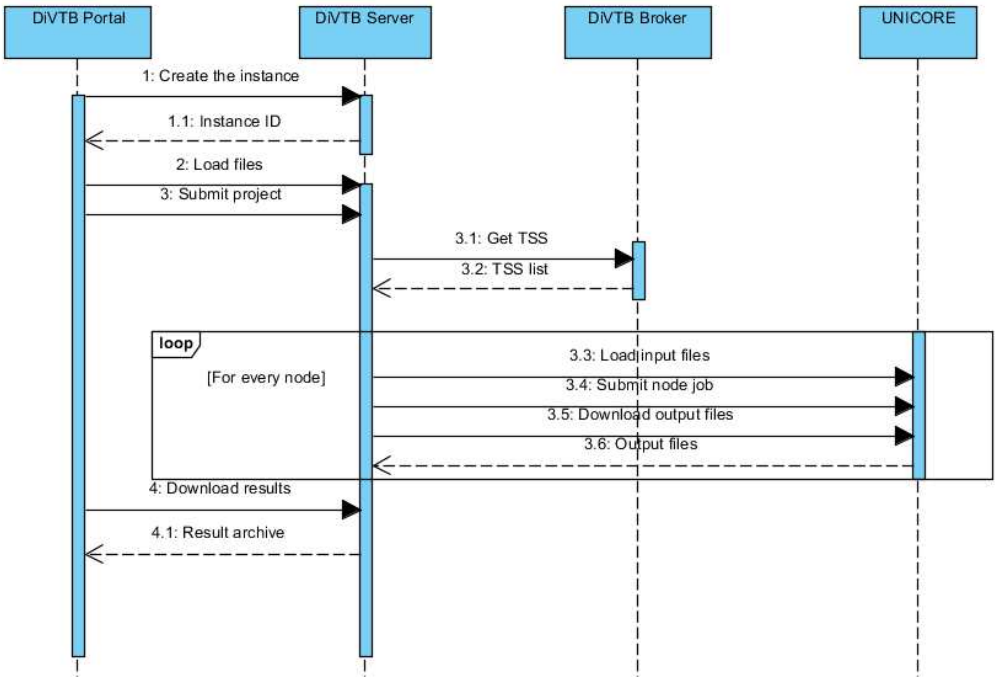


Рис. 6. Взаимодействие DiVTB Server грид-средой UNICORE

4. Тестирование DiVTB Server

4.1. Аппаратное обеспечение

Для тестирования эффективности работы механизма исполнения потоков работ DiVTB Server, было проведено сравнительное тестирование компонента Executor и UNICORE Workflow Engine. Для минимизации воздействия сторонних эффектов, вычислительные ресурсы были развернуты на одном вычислительном узле, оборудованном процессором Intel Core i5-2500k и 8 гигабайтами.

4.2. Содержание тестовых проектов

В качестве РаВИС для тестирования параллельного выполнения потоков задачи была выбрана тестовая задача обработки текстов, исполняющаяся в два потока (см. рис. 7):

1. один поток сначала утраивает содержимое файла, указанного в качестве параметра (узел *Copy*), затем добавляет в начало и в конец файла текст, который также указан в параметрах запуска (узел *Pp*);
2. второй поток капитализирует содержимое файла, указанного в качестве параметра (узел *Caps*).

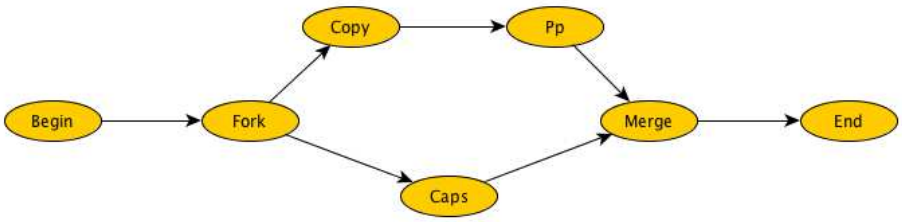


Рис. 7. Дерево исполнения РаВИС для тестирования параллельного исполнения потоков

Для сравнительного тестирования производительности DiVTB Server был выбран испытательнй стенд (см. рис. 8), состоящий из параллельно выполняющихся команд *sleep 60*, задерживающих исполнение ровно на минуту.

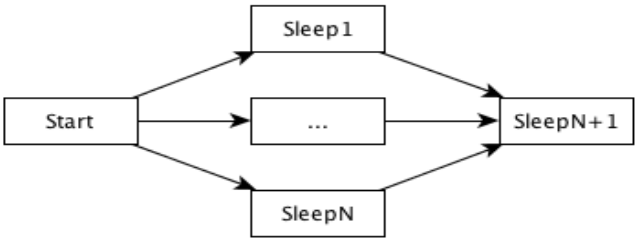


Рис. 8. Дерево исполнения РаВИС для сравнительного тестирования производительности

4.3. Результаты тестирования параллельного исполнения потоков

В качестве клиента был использован Java-клиент, использующий классы, которые были сгенерированы из WSDL-описания сервисов CAEBeans Server. Порядок обращений клиента к серверу во время тестового запуска был следующим:

1. запрос на создание экземпляра и получение его идентификатора;
2. запрос на запуск задачи;
3. проверка статуса задачи (производилась до тех пор, пока статус задачи не изменился на «выполнено» или «прервано»).

4.4. Результаты сравнительного тестирования DiVTB Server и UNICORE

На рис. 11 приведены графики сравнительной производительности UNICORE Workflow Engine и DiVTB Server на одинаковых испытательных стендах. В качестве изменяемого параметра было выбрано количество одновременно выполняющихся операций.

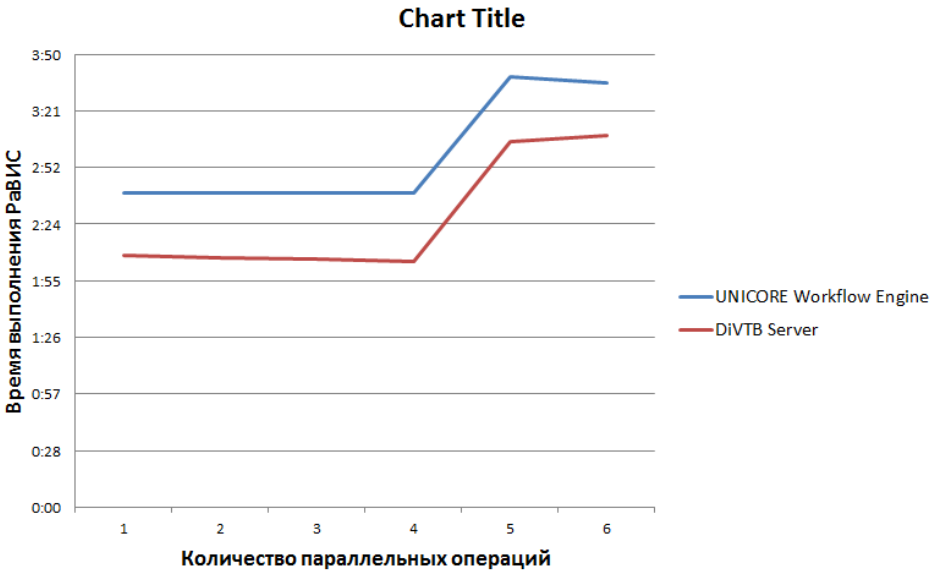


Рис. 9. Сравнительная производительность DiVTB Server и UNICORE

Полученный результат иллюстрирует тот факт, что DiVTB Server на небольших объемах данных ведет себя не хуже UNICORE Workflow Engine. Скачок, наблюдавшийся при увеличении одновременно выполняющихся узлов до 5, объясняется ограничениями среды UNICORE, которая была запущена на этом же вычислительном узле.

Результаты теста показывают, что DiVTB Server не уступает по производительности стандартному для среды UNICORE сервису Workflow Server, что делает его использование не только удобным для инженера, но и обоснованным с точки зрения производительности.

5. Заключение

В статье представлена архитектура системы DiVTB Server, выполняющей задачи управления виртуальными экспериментами, построенными на основе концепции распределенных виртуальных испытательных стендов (РаВИС), включая загрузку, запуск исполнение виртуальных экспериментов на основе загруженных РаВИС, отслеживание статуса запущенных экспериментов и получение результатов их выполнения, а также детали реализации описанной системы. Система имеет функционал для автоматической загрузки создаваемых проектов при помощи DiVTB Developer.

Дальнейшим развитием DiVTB Server станет разработка инструмента наблюдения за выполнением эксперимента и оперативного вмешательства в его ход, а также улучшение стабильности DiVTB Server путем восстановления и завершения экспериментов, которые не были корректно завершены.

Литература

1. Захаров Е.А. Web-портал для проведения виртуальных экспериментов в распределенных вычислительных средах. См. настоящий сборник.
2. Официальный сайт проекта Guice. URL: http://code.google.com/p/google-guice/ (дата обращения: 23.01.13)
3. Официальный сайт проекта Unicore. URL: http://www.unicore.eu (дата обращения: 23.01.13)
4. Радченко Г.И. Грид-система CAEBeans: интеграция ресурсов инженерных пакетов в распределенные вычислительные среды. Вестник Нижегородского университета им. Н.И. Лобачевского. 2009. № 6-1. С. 192-202.
5. Радченко Г.И. Распределенные виртуальные испытательные стенды: использование систем инженерного проектирования и анализа в распределенных вычислительных средах. Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. 2011. № 37. С. 108-121.
6. Шамакина А.В. CAEBeans Broker: брокер ресурсов системы CAEBeans // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". 2010. № 16(192). Вып. 5. С. 107-115.
7. Mell P., Grance T. The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST), USA, 2011. 7 p.
8. Vaquero L. M. et al. A break in the clouds: towards a cloud definition. ACM SIGCOMM Computer Communication Review. Vol. 39, Issue 1. 2009. P. 50-55.

Реализация параллельных алгоритмов решения модельной задачи динамики фитопланктона в мелководном водоеме с применением многопоточности в операционной системе windows

А.И. Сухинов, А.В. Никитина, И.С. Семенов

Таганрогский кампус Южного федерального университета

Работа посвящена применению функций winapi для создания параллельного алгоритма с общей памятью, реализующего задачу распространения вредоносной водоросли Sceletonema costatum в мелководном водоеме. Задача является актуальной, поскольку при ее решении производятся расчеты концентраций вредных веществ и вредоносных водорослей в Азовском море. Предложенный способ реализации параллельных вычислений позволяет сократить время работы программы более чем в 2 раза. За основу был взят механизм создания дополнительных потоков, который обеспечивает лучшее распределение ресурсов ЭВМ и повышает эффективность алгоритма.

1.Введение

В настоящее время получили широкое распространение многоядерные процессорные системы с общей памятью. Для повышения эффективности алгоритмов можно использовать библиотеку OpenMP, а также многопоточность операционной системы [1]. Под многопоточностью операционной системы будем понимать такое ее свойство, при котором вычислительный процесс, в ней порожденный, может состоять из нескольких потоков, выполняющихся параллельно, то есть без предписанного порядка во времени [2]. В данной работе разработан параллельный алгоритм с общей памятью, в котором для создания потоков используются функции winapi. Winapi – это инструмент, с помощью которого осуществлялась работа с операционной системой из прикладной программы.

2. Постановка задачи

За основу был взят последовательный алгоритм, с помощью которого было получено численное решение задачи динамики вредоносной водоросли Sceletonema costatum, имеющей наибольшее значение в питании пелагических рыб Азовского моря [3].

Рассмотрим систему из трех уравнений диффузии – конвекции – реакции в области G, представляющей собой замкнутый бассейн, ограниченный невозмущенной поверхностью водоема Σ_0, дном $\Sigma_H = \Sigma_H(x,y)$ и цилиндрической поверхностью σ, для временного интервала $0 < t \le T_0$. $\sum = \sum_0 \cup \sum_H \cup \sigma$ – кусочно-гладкая граница области G [4]:

$$\frac{\partial X}{\partial t} + div(\mathbf{u}X) = \mu_X \Delta X + \frac{\partial}{\partial z}\left(\nu_X \frac{\partial X}{\partial z}\right) + (\alpha_0 + \gamma M)\psi(S)X - \delta X, \quad (1)$$

$$\frac{\partial S}{\partial t} + div(\mathbf{u}S) = \mu_S \Delta S + \frac{\partial}{\partial z}\left(\nu_S \frac{\partial S}{\partial z}\right) - (\alpha_0 + \gamma M)\psi(S)X + B(S_p - S) + f, \quad (2)$$

$$\frac{\partial M}{\partial t} + div(\mathbf{u}M) = \mu_M \Delta M + \frac{\partial}{\partial z}\left(\nu_M \frac{\partial M}{\partial z}\right) + k_M X - \varepsilon M, \quad (3)$$

где X, S, M – концентрация фитопланктона (диатомовой водоросли Sceletonema costatum), биогенного вещества (азот, фосфор) и метаболита соответственно; $\mathbf{u} = (u, v, w)$ – вектор скорости водного потока; $\alpha = (\alpha_0 + \gamma M)$ – коэффициент роста фитопланктона; α_0 – скорость

роста фитопланктона в отсутствие метаболита; γ – параметр воздействия; $\mu_X, \mu_S, \mu_M, \nu_X, \nu_S, \nu_M$ – диффузионные коэффициенты в горизонтальном и вертикальном направлениях субстанций X, S, M соответственно; C – концентрация солености; $\delta = \delta(C)$ – коэффициент убыли фитопланктона за счет отмирания (удельная смертность), учитывающий влияние солености; B – удельная скорость поступления загрязняющего вещества; S_p – предельно возможная концентрация загрязняющего вещества; $f(x, y, z)$ – функция источника загрязнения; k_M – коэффициент экскреции; ε – коэффициент разложения метаболита; T – температура; $\psi(T, S)$ – коэффициент, учитывающий влияние температуры и концентрации биогенного вещества на рост концентрации фитопланктона.

Пусть n – вектор внешней нормали к поверхности $\sum$, U_n – нормальная по отношению к $\sum$ составляющая вектора скорости водного потока.

К системе (1) – (3) необходимо добавить начальные условия:

$$X(x, y, z, 0) = X_0(x, y, z),\ S(x, y, z, 0) = S_0(x, y, z),$$
$$M(x, y, z, 0) = M_0(x, y, z),\ (\mathrm{x,y,z}) \in \overline{G},\ t = 0 \qquad (4)$$

и граничные условия:

$$X = S = M = 0 \ \text{на}\ \sigma,\ \text{если}\ U_n < 0;$$
$$\frac{\partial X}{\partial n} = 0, \frac{\partial S}{\partial n} = 0, \frac{\partial M}{\partial n} = 0 \ \text{на}\ \sigma, \text{если}\ U_n \geq 0;$$
$$\frac{\partial X}{\partial z} = 0, \frac{\partial S}{\partial z} = 0, \frac{\partial M}{\partial z} = 0 \ \text{на}\ \Sigma_0; \qquad (5)$$
$$\frac{\partial X}{\partial z} = -\varepsilon_1 X, \frac{\partial S}{\partial z} = -\varepsilon_2 S, \frac{\partial M}{\partial z} = -\varepsilon_3 M \ \text{на}\ \Sigma_H,$$

где $\varepsilon_1, \varepsilon_2, \varepsilon_3$ – неотрицательные постоянные, ε_1 – учитывает опускание водорослей на дно и их затопление; $\varepsilon_2, \varepsilon_3$ – учитывают поглощение биогенного вещества и метаболита донными отложениями.

Для модели (1) – (5) входными параметрами являются компоненты вектора скорости водной среды, которые описываются гидродинамической моделью, описанной в работах [5 – 8].

При численной реализации модели (1) – (5) использовались функции winapi по следующим причинам:

- параллельный алгоритм не зависит от количества вычислителей на ЭВМ.
- разработка данного алгоритма является первым этапом для реализации задачи на супер-ЭВМ с общей памятью.

3. Описание последовательного алгоритма решения задачи

Для построения дискретной модели динамики вредоносной водоросли использовалась линеаризация по Ньютону [9, 10]:

$$\hat{X}_t^{p+1} + \frac{\left(\hat{U} - \left|\hat{U}\right|\right)}{2}\hat{X}_x^{p+1} + \frac{\left(\hat{U} + \left|\hat{U}\right|\right)}{2}\hat{X}_{\bar{x}}^{p+1} + \frac{\left(\hat{V} - \left|\hat{V}\right|\right)}{2}\hat{X}_y^{p+1} + \frac{\left(\hat{V} + \left|\hat{V}\right|\right)}{2}\hat{X}_{\bar{y}}^{p+1} +$$
$$+ \frac{\left(\hat{W} - \left|\hat{W}\right|\right)}{2}\hat{X}_z^{p+1} + \frac{\left(\hat{W} + \left|\hat{W}\right|\right)}{2}\hat{X}_{\bar{z}}^{p+1} = \mu_x\left(\hat{X}_{\bar{x}x}^{p+1} + \hat{X}_{\bar{y}y}^{p+1}\right) + \nu_z\left(\hat{X}_{\bar{z}}^{p+1}\right)_z +$$
$$+ \left(\alpha_0 + \gamma\hat{M}^p\right)\hat{S}^p\hat{X}^{p+1} - \delta\hat{X}^{p+1},$$

$$\hat{S}_t^{p+1} + \frac{\left(\hat{U} - \left|\hat{U}\right|\right)}{2}\hat{S}_x^{p+1} + \frac{\left(\hat{U} + \left|\hat{U}\right|\right)}{2}\hat{S}_{\bar{x}}^{p+1} + \frac{\left(\hat{V} - \left|\hat{V}\right|\right)}{2}\hat{S}_y^{p+1} + \frac{\left(\hat{V} + \left|\hat{V}\right|\right)}{2}\hat{S}_{\bar{y}}^{p+1}$$
$$+ \frac{\left(\hat{W} - \left|\hat{W}\right|\right)}{2}\hat{S}_z^{p+1} + \frac{\left(\hat{W} + \left|\hat{W}\right|\right)}{2}\hat{S}_{\bar{z}}^{p+1} = \mu_s\left(\hat{S}_{\bar{x}x}^{p+1} + \hat{S}_{\bar{y}y}^{p+1}\right) + \nu_s\left(\hat{S}_{\bar{z}}^{p+1}\right)_z -$$
$$-\left(\alpha_0 + \gamma\hat{M}^p\right)\hat{S}^{p+1}\hat{X}^p + B(S_p - \hat{S}^{p+1}) + \tilde{f}, \qquad (6)$$

$$\hat{M}_t^{p+1} + \frac{\left(\hat{U} - \left|\hat{U}\right|\right)}{2}\hat{M}_x^{p+1} + \frac{\left(\hat{U} + \left|\hat{U}\right|\right)}{2}\hat{M}_{\bar{x}}^{p+1} + \frac{\left(\hat{V} - \left|\hat{V}\right|\right)}{2}\hat{M}_y^{p+1} + \frac{\left(\hat{V} + \left|\hat{V}\right|\right)}{2}\hat{M}_{\bar{y}}^{p+1} +$$
$$+ \frac{\left(\hat{W} - \left|\hat{W}\right|\right)}{2}\hat{M}_y^{p+1} + \frac{\left(\hat{W} + \left|\hat{W}\right|\right)}{2}\hat{M}_{\bar{y}}^{p+1} = \mu_M\left(\hat{M}_{\bar{x}x}^{p+1} + \hat{M}_{\bar{y}y}^{p+1}\right) + \nu_M\left(\hat{M}_{\bar{z}}^{p+1}\right)_z +$$
$$+k_M\hat{X}^{p+1} - \varepsilon\hat{M}^{p+1},$$

где p – номер итерации по нелинейности. Предложенная разностная схема обладает большим запасом устойчивости и является консервативной в случае несжимаемой водной среды [8].

Схема последовательного алгоритма расчета по (6) представлена на Рис. 1.

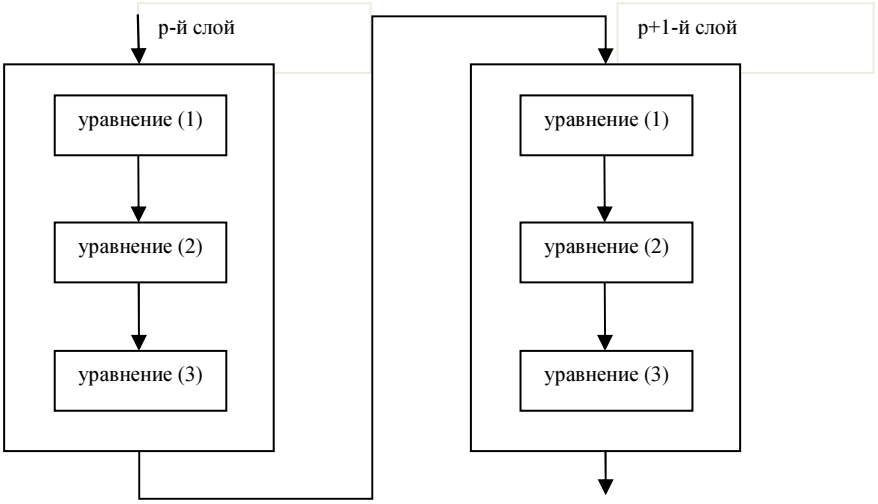


Рис. 1. Схема последовательного алгоритма решения задачи динамики токсичной водоросли

На каждой итерации по нелинейности производятся расчеты сначала для уравнения (1), затем для (2) и (3) соответственно как в работе [11]. Переход к следующему временному слою осуществляется в случае сходимости итерационного процесса по нелинейности. Критерием остановки итерационного процесса является: $\left\|\hat{X}^{p+1} - \hat{X}^p\right\|_2 < \varepsilon$, где $\|A\|_2 = \sqrt{\sum_{i,j}\left|a_{ij}\right|^2}$, ε – наперед заданная точность.

4. Описание параллельного алгоритма решения задачи

Входные данные расчетов для уравнений (1) – (3) не зависят друг от друга, поэтому параллельные расчеты производились для уравнений (1) – (3) на каждом временном слое согласно схеме, представленной на **Рис. 2**.

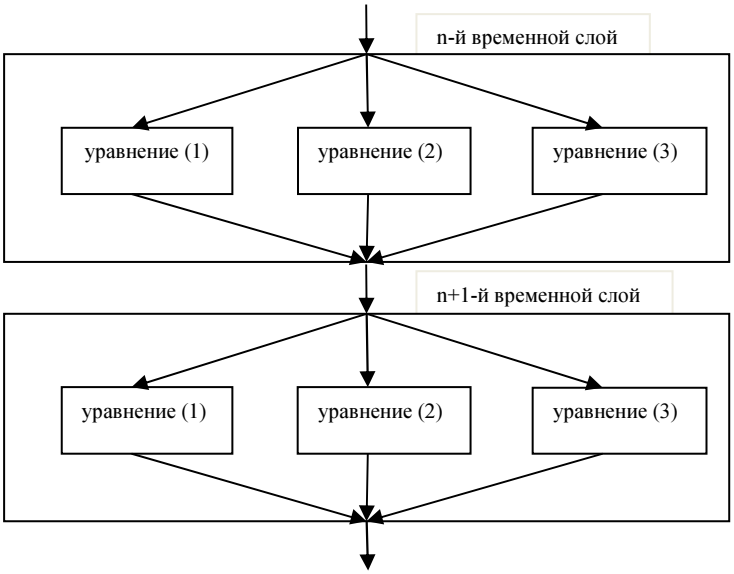


Рис. 2. Схема параллельного алгоритма решения задачи динамики токсичной водоросли

В расчете по последовательной версии программы всегда присутствует один поток. Для создания дополнительных потоков в программном комплексе использовалась функция *CreateThread.*

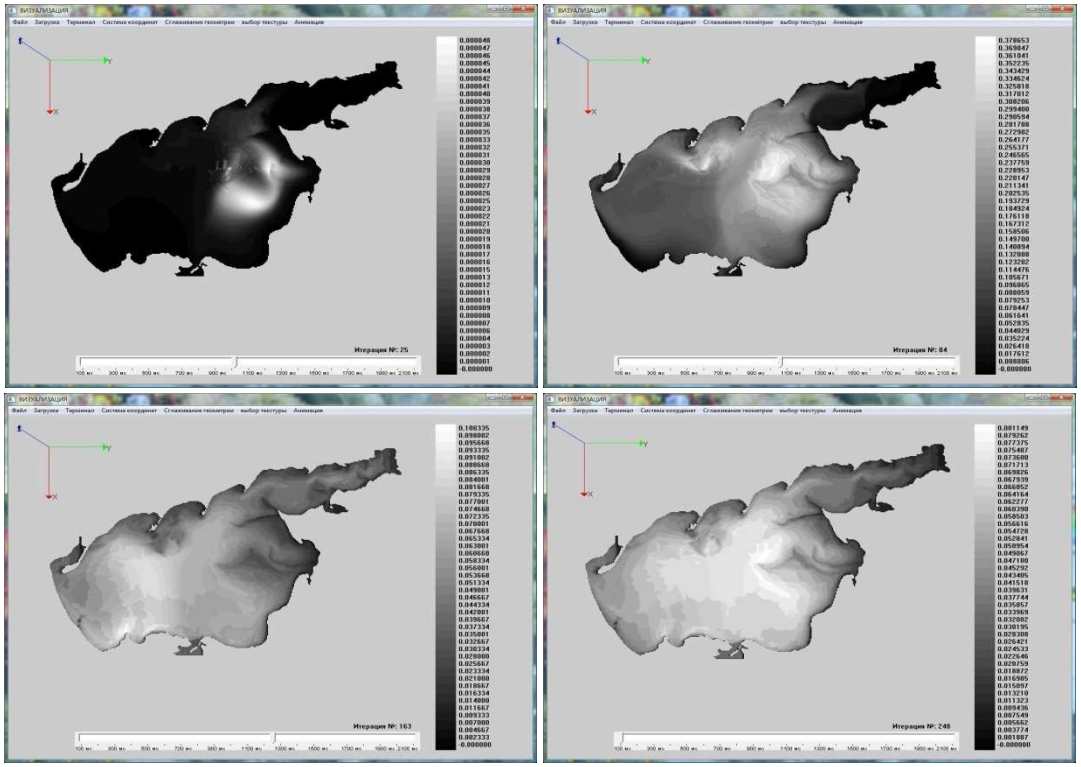


Рис. 3. Распределение концентрации вредоносной водоросли в Азовском море в разные моменты времени при северном ветре

Чтобы синхронизировать созданные потоки при расчете для описанной модельной задачи гидробиологии моря на очередном временном слое использовалась функция *WaitForMultipleObjects.*

С целью оценки эффективности использования winapi функций фиксировалось время выполнения программы расчета для 4-х временных слоев (12-иттераций по нелинейности). Время выполнения программы без winapi функций составило 101 секунду, а с winapi функциями 69 секунд. Ускорение составило 1.46, эффективность 0.73 по сравнению с обычным последовательным алгоритмом [12]. Расчет производился на 4-ми ядерном процессоре Intel Core i7-3770K 3.5GHz.

На Рис. 3 приведены результаты работы параллельной версии программы для задачи динамики фитопланктонной популяции в мелководном водоеме (Азовское море).

Приведем физические размеры расчетной области: площадь поверхности 37605 км^2, длина 343 км, ширина 231 км. Расстояние между узлами сетки по длине и ширине составляли 1 км, по глубине 1 м.

5. Заключение

На основании проведенных расчетов можно сделать вывод о том, что применение функций winapi при реализации программы для решения модельной задачи динамики вредоносной водоросли в Азовском море позволяет более рационально распределить ресурсы ЭВМ и повысить эффективность алгоритма.

Литература

1. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. Москва, Изд.-во МГУ, 2010. 534 с.
2. Воеводин В.В. Вычислительная математика и структура алгоритмов. Москва, Изд.-во МГУ, 2010. 166 с.
3. Сухинов А.И., Никитина А.В., Чистяков А.Е. Моделирование сценария биологической реабилитации Азовского моря // Математическое моделирование. 2012. Т.24, №9, С. 3–21.
4. Никитина А.В. «Численное решение задачи динамики токсичных водорослей в Таганрогском заливе». // Известия ЮФУ. Технические науки.– 2010, №6(107). С 113–116.
5. Сухинов А.И., Чистяков А.Е., Алексеенко Е.В. Численная реализация трехмерной модели гидродинамики для мелководных водоемов на супервычислительной системе// Математическое моделирование. 2011. Т. 23, № 3. С.3–21.
6. Сухинов А.И., Чистяков А.Е. Параллельная реализация трехмерной модели гидродинамики мелководных водоемов на супервычислительной системе// Вычислительные методы и программирование: Новые вычислительные технологии. 2012. Т.13. С. 290–297.
7. Сухинов А.И., Чистяков А.Е. Адаптивный модифицированный попеременно-треугольный итерационный метод для решения сеточных уравнений с несамосопряженным оператором// Математическое моделирование. 2012. Т.24, №1. С. 3–20.
8. Сухинов А.И., Чистяков А.Е., Тимофеева Е.Ф., Шишеня А.В. Математическая модель расчета прибрежных волновых процессов// Математическое моделирование. 2012. Т.24, №8. С. 32–44.
9. Самарский А.А. Теория разностных схем. М. Наука, 1989. 616 с.
10. Самарский А.А., Николаев Е.С. Методы решения сеточных уравнений. М. Наука, 1978. 592 с.
11. Никитина А.В. Модели биологической кинетики, стабилизирующие экологическую систему таганрогского залива // Известия ЮФУ. Технические науки. 2009. № 8 (97). С. 130–134.
12. Чистяков А.Е. Теоретические оценки ускорения и эффективности параллельной реализации ПТМ скорейшего спуска// Известия ЮФУ. Технические науки. 2010, №6(107). С. 237-249.

Анализ эффективности ряда параллельных итерационных методов решения СЛАУ в упругопластической задаче на кластерной системе[1]

А.В. Толмачев, А.В. Коновалов, А.С. Партин

Институт машиноведения УрО РАН

В процессе решения упругопластической задачи возникает необходимость решить систему линейных уравнений большой размерности. Данная система является ленточной и разреженной внутри ленты. Её решение прямыми методами нецелесообразно, так как требуется большой объём памяти из-за образующегося заполнения. В данной работе описаны результаты вычислительного эксперимента и анализа производительности параллельных итерационных методов: релаксации (SOR), бисопряжённых градиентов (BiCGStab) и обобщённый метод минимальных невязок (GMRES).

1. Введение

Упругопластическая задача с большими пластическими деформациями физически и геометрически существенно нелинейная [1] и требует большого количества времени для её решения методом конечных элементов на персональном компьютере. Для сокращения времени расчётов необходимо использовать параллельные вычисления, в частности, на кластерных системах.

Решение методом конечных элементов упругопластической задачи осуществляется в условиях пошагового нагружения и на каждом таком шаге состоит из трёх основных этапов [1]:

1) расчёт локальных матриц жёсткости для конечных элементов и формирование глобальной матрицы жёсткости A и вектора b правой части системы линейных алгебраических уравнений (СЛАУ)

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

относительно искомого вектора x обобщённой скорости в узлах конечно-элементной сетки;

2) решение СЛАУ (1);

3) вычисление напряжённо-деформированного состояния конечных элементов в конце шага нагружения.

На каждом шаге нагружения этап 1 выполняется один раз, а этапы 2 и 3 — от десяти до пятнадцати раз для удовлетворения итерационно с приемлемой точностью условию пластичности в конце шага нагружения. При этом матрица жёсткости остается постоянной, а изменяется только правая часть системы уравнений. Матрица A является ленточной матрицей большой размерности и разреженной внутри ленты.

Производительность различных прямых методов решения СЛАУ в упругопластической задаче рассматривалась в работах [2-4]. Использование прямых методов для решения СЛАУ приводит к образованию заполнения внутри ленты и большим затратам по как памяти, так и по времени счёта.

Целью данной работы является исследование возможностей итерационных параллельных алгоритмов метода релаксации [5], стабилизированного метода бисопряжённых градиентов (BiCGStab) [6] и обобщённого метода минимальных невязок (GMRES) [5] для решения СЛАУ в упругопластических задачах на кластерной системе.

Ранее эффективность метода релаксации исследовалась в работе [7], однако там рассматривались матрицы системы из двухмерной упругопластической задачи с малой размеренностью.

[1] Работа выполнена при поддержке программы фундаментальных исследований Президиума РАН №15, проект 12-П-1-1025

2. Использованные методы решения СЛАУ

Формы хранения матрицы системы. При формировании матрицы A использовали координатную форму хранения [5] с хранением троек из номеров строк, номеров столбцов и элементов матрицы в красно-коричневом дереве [8]. Для выполнения вычислений над матрицей, она переводится в сжатую по строкам форму [5]. Использование красно-коричневого дерева позволяет выполнить преобразование из координатной в сжатую форму без сортировок за единственный обход дерева.

Метод релаксации. Метод релаксации является одним из базовых итерационных методов линейной алгебры. Он является улучшением метода Гаусса-Зейделя [5] путём добавления коэффициента релаксации $0 < \omega < 2$. При значении коэффициента $\omega = 1$ метод релаксации сводится к методу Гаусса-Зейделя. Структура метода релаксации приведена на рисунке 1.

Повторять до схождения

$for\ i = 1,...,n$

$$\dot{x} = \frac{1}{a_{ii}}\left(b_i - \sum_{i \neq j} a_{ij} x_j\right)$$

$$\delta = \dot{x} - x_i$$

$$x_i = x_i + \omega\delta$$

Рис. 1. Алгоритм метода релаксации

Метод BiCGStab. Метод BiCGStab является вариантом метода сопряжённых градиентов для несимметричных матриц. Он относится к семейству методов, основанных на подпространствах Крылова. Рассматривается СЛАУ $Ax = b$ с начальным приближением x_0. Здесь (a,b) означает вычисление скалярного произведения векторов a и b. Алгоритм непредобусловленного метода BiCGStab представлен на рисунке 2.

$$r_0 = b - Ax_0$$

$$\rho_0 = (r_0, r_0)$$

$$p_0 = r_0$$

for $j = 0,1,...$

$$v = Ap_j$$

$$\alpha_j = \rho_j / (v, r_0)$$

$$s_j = r_j - \alpha_j v$$

$$t = As_j$$

$$\omega_j = \frac{(t, s_j)}{(t,t)}$$

$$x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j;\ \text{проверка на сходимость}$$

$$r_{j+1} = s_j - \omega_j t$$

$$\rho_{j+1} = (r_{j+1}, r_0)$$

$$\beta_j = \frac{\rho_{j+1}}{\rho_j}\frac{\alpha_j}{\omega_j}$$

$$p_{j+1} = r_{j+1} + \beta_j (p_j - \omega_j v)$$

end

Рис. 2. Алгоритм непредобусловленного метода BiCGStab

Для уменьшения количества итераций, необходимых для достижения сходимости метода до заданной точности, использовали вариант метода с предобуславливателем [5].

Обобщённый метод минимальных невязок. Обобщённый метод минимальных невязок так же относится к семейству методов, основанных на подпространствах Крылова. Его подробное описание приведено в [5].

Предобуславливатель. Для уменьшения количества итераций, необходимых для схождения итерационных методов BiCGStab и GMRES использовали предобуславливатель, основанный на процессе неполного LU-разложения без заполнения (ILU(0)) [5].

3. Характеристики матрицы системы

Для упрощения анализа эффективности рассматриваемы методов решения СЛАУ использовали регулярную сетку при разбиении тела на конечные элементы.

При решении трёхмерной задачи с регулярной сеткой имеет место ленточная матрица с пятью ненулевыми лентами. Параметры матрицы системы в зависимости от количества разбиений d по каждой координатной оси в конечноэлементной сетке приведены в таблице. Здесь n - размерность матрицы, β - полуширина ленты, z - количество элементов в матрице, удовлетворяющих условию $\left|a_{ij}\right| \geq 10^{-6}$, которое на два порядка меньше критерия остановки в методе Ньютона-Равсона, использованного при достижении условия пластичности, δ - число обусловленности матрицы. Число обусловленности для матрицы оценивалось как отношение наибольшего к наименьшему сингулярных значений матрицы. Сингулярные значения матрицы оценивались при помощи функции `svds` пакета MATLAB. Для задач с сеткой d = 30, 40 и 50 сингулярные значения оценивались с использованием библиотеки SLEPc [9] с критерием останова 10^{-3}.

Таблица 1. Параметры матрицы в трёхмерной упругопластической задаче

d	n	β	z	$\delta/10^3$
10	3993	401	234864	1,14
20	27783	1391	1910269	2,34
30	89373	2981	5856367	4,2
40	206763	5171	13630871	8,1
50	397953	7961	30080934	10,6

4. Результаты вычислительных экспериментов

Эффективность итерационных методов для решения упругопластической задачи определяли по результатам вычислительного эксперимента по сжатию параллелепипеда плоскими плитами.

Вычислительные эксперименты проводили на кластере «Уран» Института математики и механики УрО РАН. Он состоит из 208 вычислительных узлов, установленных в модулях с высокой плотностью упаковки. Каждый вычислительный узел оснащен 2-мя процессорами Intel Quad-Core Xeon, работающими на частоте 3.00 ГГц, и 16/32 гигабайтами оперативной памяти. В общей сложности пользователям доступно 1664 вычислительных ядра и 3584 Гбайт оперативной памяти. Для передачи данных между вычислительными узлами используется высокоскоростная сеть Infiniband с пропускной способностью 20 Гбит/сек.

В качестве реализаций итерационных методов решения СЛАУ была использована библиотека lis [10]. Решение СЛАУ производилось до удовлетворения условия остановки $\|r\| / \|b\| < 10^9$, где r - вектор невязки, b - вектор правой части, $\|\bullet\|$ - 2-норма вектора.

Анализ результатов, полученным методом релаксации показал, что при значениях коэффициента релаксации $\omega > 1$ на сетках с параметром $d > 15$ метод расходится. При значения $\omega < 1$

метод релаксации сходится, но решение СЛАУ производится значительно медленнее, чем при использовании методов BiCGStab и GMRES.

На рисунке 3 представлена зависимость ускорения *s* методов BiCGStab и GMRES от количества используемых MPI-процессов *p* для решения СЛАУ.

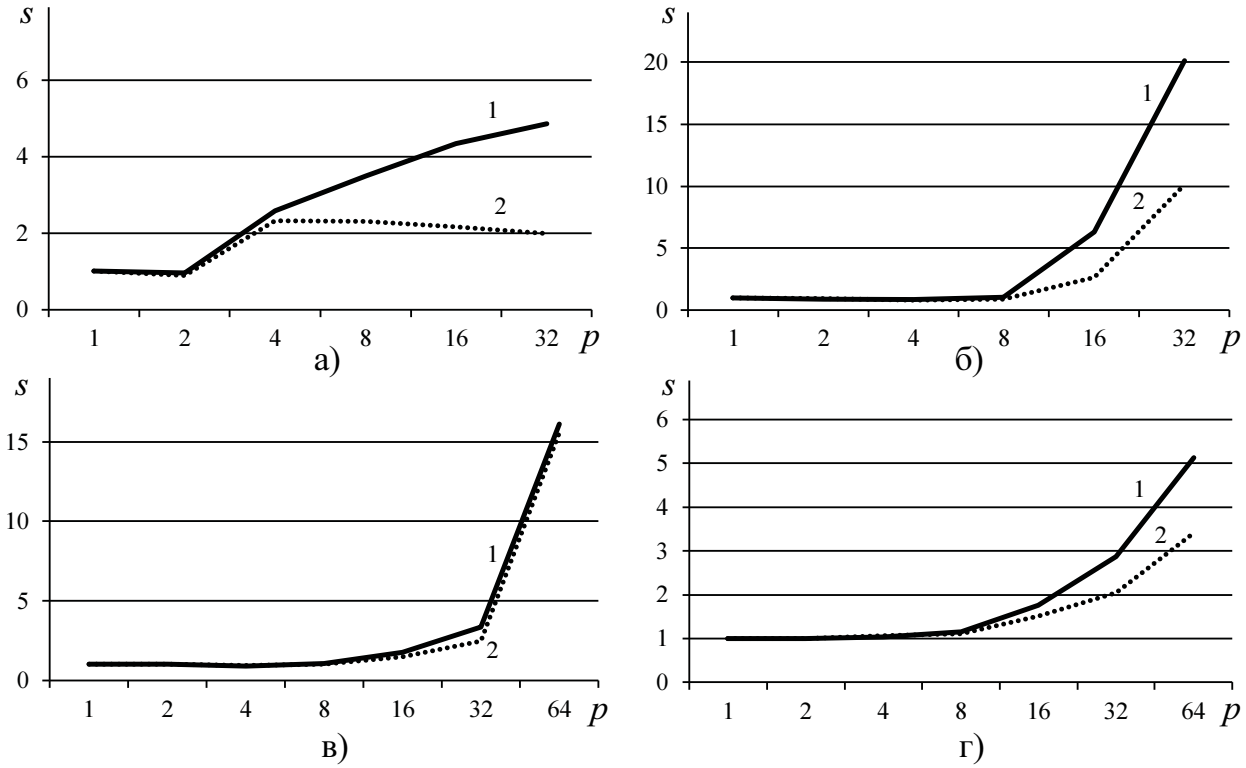


Рис. 3. Зависимость ускорения *s* решения СЛАУ от количества MPI-процессов *p* для сеток с параметром d равным 10 (а), 20 (б), 30 (в) и 50 (г): 1 - метод BiCGStab , 2 - метод GMRES

На всех рассмотренных сетках метод BiCGStab показал лучшее ускорение по сравнению с методом GMRES. Для сетки с d = 10 (см. рис. 3,а) у метода GMRES после при использовании более 4 процессов имеет место замедление вычислений за счёт времени на передачу данных. А в методе BiCGStab данный эффект отсутствует. Для сеток наблюдается скачкообразный рост ускорения в трёх случаях, а именно: для сетки с $d = 10$ при переходе с 2 на 4 процесса, для сетки с $d = 20$ при переходе с 16 на 32 процесса и для сетки с $d = 30$ при переходе с 32 на 64 процесса. Этот эффект обусловлен тем, что матрица системы уравнений начинает полностью помещаться в процессорный кэш. Характер графика для сетки $d = 40$ аналогичен графику для сетки $d = 50$. При этом ускорение метода BiCGStab достигало 5,8 на 64 процессах.

На рисунке 4,а показаны график зависимости среднего времени, затрачиваемого на одну итерацию от количества процессоров на задаче с сеткой $d = 30$. На рисунке 4,б показана зависимость наибольшего, среднего и наименьшего количества итераций при решении СЛАУ.

Уменьшение времени на итерацию одновременно с увеличением количества итераций приводит к тому, что на отрезке от 1 до 8 процессов не происходит существенного ускорения (см. рис. 3). Отсутствие ускорения на рассматриваемом отрезке объясняется тем, что использованная реализация предобуславливателя ILU(0) в библиотеке lis не производит, по нашему мнению, межпроцессорный обмен. Поэтому при увеличении количества процессов требуется большее количество итераций для достижения требуемой точности. При использовании более 8 процессов время на итерацию уменьшается быстрее, чем увеличивается количество итераций, поэтому общее время на решение СЛАУ уменьшается. Применение других предобуславливателей может существенно повлиять на данный эффект.

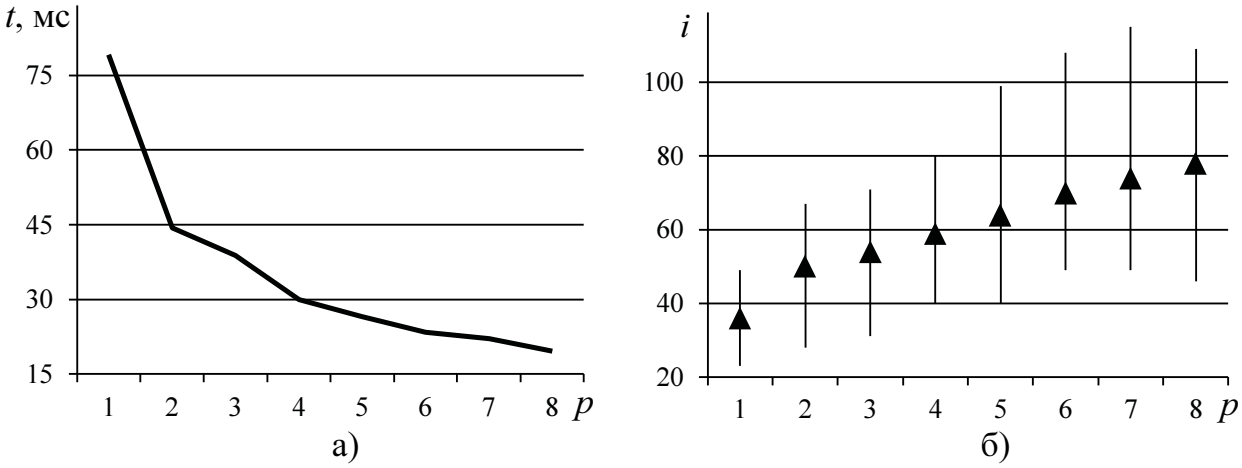


Рис. 4. Зависимость для метода BiCGStab от количества MPI-процессов p времени t на одну итерацию (а), наибольшее, среднее (маркеры) и наименьшее количество итераций при решении СЛАУ (б)

Метод GMRES показывает лучшее абсолютное время решения СЛАУ на всех рассматриваемых сетках на небольшом количестве процессов, однако при увеличении количества процессоров метод BiCGStab начинает быть быстрее, так как он требует меньше операций передачи данных. Например, для решения СЛАУ в задаче с сеткой $d = 40$ 1 процессом методу BiCGStab требуется затратить 17,5 секунд, а методу GMRES — 13,1 секунды. Однако при использовании 64 процессов методу BiCGStab требуется затратить 2,8 секунды, а методу GMRES — 3,4 секунды.

Применение прямых методов решения СЛАУ в упругопластической задаче методами из работ [3,4] не позволяло решать на кластере "Уран" ИММ УрО РАН задачи с сетками с $d > 35$ поскольку для хранения матрицы жёсткости требовалось много оперативной памяти. Применение разреженных схем хранения матрицы в совокупности с предобусловленными итерационными методами позволило сократить затраты на хранение матрицы жёсткости и дало возможность решать задачу с большим разбиением.

Выводы

1. Итерационные методы позволили решать упругопластическую задачу методом конечных элементов с большим разбиением, чем прямые методы.
2. Метод релаксаций по сравнению с методами BiCGStab и GMRES затрачивает значительно большее время решение СЛАУ.
3. Метод GMRES показывает лучшее абсолютное время при решении СЛАУ на небольшом количестве процессоров, однако метод BiCGStab показывает лучшее ускорение и при использовании большого количества процессоров показывает лучшее абсолютное время.

Литература

1. Поздеев А.А., Трусов П.В., Няшин Ю.И. Большие упруго-пластические деформации. М.: Наука, 1986. 232 с.
2. Толмачев А.В. Коновалов А.В. Партин А.С. Использование усечённого варианта алгоритма SPIKE из библиотеки Intel Adaptive SPIKE-Based Solver для решения упругопластических задач // Вычислительные методы и программирование: новые вычислительные технологии. 2011. – Т. 12. – № 1. – С. 154-159.
3. Коновалов А.В., Толмачев А.В., Партин А.С. Параллельное решение упругопластической задачи с применением трехдиагонального алгоритма LU-разложения из библиотеки SCALAPACK// Вычислительная механика сплошных сред. – 2011. – № 4. С. 34-41.

4. Толмачев А.В., Коновалов А.В., Партин А.С.. Применение распределённого по замкнутому тору алгоритма LU-разложения к решению упругопластической задачи (с. 745) // Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции (Новосибирск, 26 – 30 марта 2012 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2012. – 774 с.
5. Saad Y. Iterative methods for sparse linear systems, SIAM 2003. С. 447.
6. H. A. van der Vorst, BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 631–644. doi:10.1137/0913035
7. 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. (электронное издание).
8. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К. Алгоритмы: построение и анализ. – М.: Вильямс, 2005. – 1296 с.
9. Vicente Hernandez, Jose E. Roman, Vicente Vidal. SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems // ACM Transactions on Mathematical Software 31(3), 2005. pp. 351-362
10. Akira Nishida. Lis User Manual. Research Institute for Information Technology – Fukuoka. Электронный ресурс. URL: http://www.ssisc.org/lis/lis-manual-en.pdf (дата обращения 15.11.2012)

Методы декомпозиции для решения задач движения жидкости в пористых средах на гетерогенных вычислительных системах*

А.В. Цепаев

Федеральное государственное бюджетное учреждение науки Институт механики и машиностроения Казанского научного центра Российской академии наук

Работа направлена на решение важных задач подземной гидромеханики - задач многофазной фильтрации жидкостей. Для решения этих задач предлагаются новые алгоритмы и методы решения, основанные на методах декомпозиции. Алгоритмы реализованы на вычислительных системах нового поколения - гетерогенных суперкомпьютерах, построенных на основе современных центральных процессоров и графических ускорителей.

1. Введение

Исследование процессов разработки месторождений углеводородного сырья с использованием математических моделей течений многофазной жидкости в пористых средах со скважинами является актуальной задачей. Математические модели таких процессов представляют собой системы связанных нелинейных нестационарных уравнений с частными производными. При численном решении этих систем наиболее распространенными являются вычислительные алгоритмы, основанные на тех или иных модификациях метода IMPES (неявная схема по давлению, явная по насыщенности) [1]. Как правило, методы решения задач многофазных течений в пористых средах реализованы на сетках, сгущающихся к интервалам вскрытия скважин. Требования к точности решения приводят к необходимости получать численные решения задачи на сетках, имеющих десятки миллионов неизвестных, за приемлемое время. Повышенные требования к вычислительным ресурсам обусловлены также нестационарностью и трехмерностью рассматриваемых процессов, неоднородностью физических характеристик пласта и др. Такие задачи могут быть решены только на высокопроизводительных вычислительных системах. Поэтому необходимы алгоритмы, пригодные для распараллеливания. Наиболее популярными методами решения таких задач являются методы декомпозиции, то есть разделение задачи на части, которые могут быть параллельно решены. Одним из преимуществ методов декомпозиции является возможность их реализации на многопроцессорных вычислительных системах. Идея декомпозиции использовалась и ранее, но не в связи с параллельными вычислениями, и привела к методам подструктур, подконструкций, макроэлементов, суперэлементов, фрагментов, модуль-элементов, редуцированных элементов, а также методам Шварца, матриц емкости и т.д. Такие методы всегда использовались как методы, позволяющие сводить решение исходной задачи в области со сложной границей к последовательности задач в подобластях, граница которых достаточно простая. С развитием гетерогенных вычислительных систем основной принцип разделения области решения на подзадачи сводится к независимому решению этих подзадач на различных вычислительных комплексах (ядра, процессоры, графические ускорители). Это приводит значительному уменьшению время решения исходной задачи.

При решении задач многофазной фильтрации [2, 3] на каждом временном шаге приходится определять поля давления и насыщенности. Для решения сеточных систем уравнений по давлению и насыщенности предлагаются два различных метода декомпозиции области: один метод для решения сеточных уравнений по давлению, другой - для решения сеточных уравнений по насыщенности [4]. Метод декомпозиции области по определению поля давления основан на

Работа выполнена в рамках гранта РФФИ “Решение задач многофазного течения в пористой среде с использованием гетерогенных вычислительных систем” N«12-01-31180»*

независимом решении систем алгебраических уравнений для сгущающихся участков сетки в подобластях и новом типе согласования этих решений с решением на грубой сетке. Для решения уравнения по насыщенности разработан новый метод декомпозиции области, основанный на сочетании элементов явной и неявной схем. На основе предложенных методов декомпозиции построены алгоритмы для решения задачи трехфазной фильтрации жидкости на гетерогенных вычислительных системах. Сочетание вычислительных ядер центрального процессора и графических устройств позволило значительно сократить время решения задач. Данные методы декомпозиции применялись также при решении задач фильтрации однофазной жидкости, подчиняющейся нелинейному закону Форхгеймера [5], напорно-безнапорной фильтрации [6].

2. Постановка задачи трехфазной фильтрации

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. Тогда справедлива следующая система уравнений:

$$div(\mathbf{q_w}) + m\partial S_w / \partial t = 0, \quad (1)$$

$$div(\mathbf{q}_o) + m\partial S_o / \partial t = 0, \quad (2)$$

$$div(\rho_g \mathbf{q}_g) + m\partial(\rho_g S_g) / \partial t = 0, \quad (3)$$

$$q_\alpha = -(f_\alpha k / \mu_\alpha) grad\, p, (\alpha = o, w, g). \quad (4)$$

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, $\rho_g = \rho_g^* / B_g$ - плотность газа в пластовых условиях, ρ_g^* - плотность газа при нормальных условиях, B_g - объемный коэффициент, m - пористость. Суммируя уравнения (1)-(3) с учетом (4) получим уравнение для давления

$$(mS_g(\rho_g)'_p / \rho_g)\partial p / \partial t - div((K_o + K_w + K_g) grad\, p) = 0, \quad (5)$$

где $K_\alpha = f_\alpha k / \mu_\alpha$, $(\alpha = o, w, g)$ - фазовые подвижности.

Область решения представляется многосвязной областью, внутренние поверхности которой определены поверхностями скважин в интервалах вскрытия пласта. В начальный момент времени считаются известными распределения давления и насыщенностей в пласте. На внешней поверхности пласта задаются граничные условия 1-го или 2-го рода. Обычно это начальное пластовое давление, условие непротекания и насыщенности пласта фазами на участках внешней поверхности, через которые поступают флюиды. На скважинах задается либо забойное давление, либо суммарный расход жидкости при некотором фиксированном давлении, определяемом в процессе решения задачи.

3. Решение задачи методами декомпозиции области

Задача решается в области D, представляющей собой пласт, ограниченный кровлей, подошвой, боковыми поверхностями и поверхностями интервалов вскрытия скважин V_k, $k = 1,...,N$. Объединение $\bigcup_{k=1}^{N} V_k$ является дополнением многосвязной области D до односвязной области. Система уравнений трехфазной фильтрации без учета капиллярных и гравитационных сил записывается в виде (1)-(4) при граничных условиях

$$p = p_\Gamma \text{ на } \Gamma_1, \quad (6)$$

$$-(K_o + K_w + K_g)\partial p / \partial n = q_n \text{ на } \Gamma_2, \quad (7)$$

$$p|_{\partial V_k} = P_k, \; k = 1,..., N, \quad (8)$$

$$S_w = S_{w_\Gamma} \text{ на } \Gamma_3,\ S_w = S_{w_k},\ k = 1,...,M \tag{9}$$

и начальных условиях

$$\text{p} = \text{P}^0\ ,\ S_w = S_w^0\ ,\ S_o = S_o^0 \text{ в D}, \tag{10}$$

где $\Gamma_1 + \Gamma_2 = \Gamma$ - внешняя граничная поверхность области D, Γ_3 - часть поверхности Γ, через которую жидкость поступает в пласт, ∂V_k - поверхность интервала вскрытия пласта k-ой скважиной, P_k - заданное давление на k-ой скважине, N - число скважин, M – число нагнетающих скважин ($M < N$), S_{w_k} - заданная насыщенность в нагнетательной скважине. Для области D введем следующие соотношения: $\overline{D} = \overline{D}_0 \cup (\bigcup_{k=1}^{N} \overline{D}_k)$, $D_i \cap D_j = \varnothing$, $i \neq j$, $\overline{D}_0 \cap \overline{D}_k = \gamma_k$, $S_k \cap \gamma_k = \varnothing$, где D_k - прискважинные подобласти, S_k - суммарная поверхность интервалов вскрытия k-ой скважины. Пласт покрыт сеткой Ω, ячейки которой в прискважинных зонах D_k уменьшаются в размерах к интервалам вскрытия скважин по убывающей геометрической прогрессии. Для определения S_w и S_o на (n+1) – ом временном шаге необходимо выполнить следующие этапы [7].

Этап 1. Вычисляются давления p_i^{n+1} из системы (5)-(8),(10) со значениями K_{oi}^n, K_{wi}^n, K_{gi}^n. Для решения задачи (5)-(8),(10) предлагается следующий метод декомпозиции. Вводятся дополнительные грубые сетки ω_k, покрывающие области $D_k \cup V_k$ и имеющие размер ячеек, эквивалентный размеру ячеек Ω в D_0. Решение p в области D_0 представляется как p_1 с граничными условиями $p_1 = p_{1\gamma k}$ на границах раздела γ_k области D_0 и прискважинных подобластей D_k, а вне области D_0 как сумму двух решений p_{2k} и p_{3k}. Решения p_{2k} определяются на сетке Ω в областях D_k, а решения p_{3k} на сетках ω_k в областях $D_k \cup V_k$, при этом $p_{1\gamma k} = p_{2\gamma k} + p_{3\gamma k}$ на границах γ_k.

Постановка задачи для определения p_1 на сетке Ω:

$$\begin{gathered}(mS_g(\rho_g)'_p / \rho_g)\partial p_1 / \partial t - div(K_w + K_o + K_g)grad\ p_1 = 0 \text{ в } D_0, \\ p_1 = p_\Gamma \text{ на } \Gamma_1 \cap D_0, \\ -(K_w + K_o + K_g)\,\partial p_1 / \partial n = q_n \text{ на } \Gamma_2 \cap D_0, \\ p_1 = p_{1\gamma k} \text{ на } \gamma_k,\ k = 1...N\ .\end{gathered} \tag{11}$$

Постановка задачи для определения p_{2k} ($k = 1,...,N$) на сетке Ω:

$$\begin{gathered}(mS_g(\rho_g)'_p / \rho_g)\partial p_{2k} / \partial t - div(K_w + K_o + K_g)grad\, p_{2k} = 0 \text{ в } D_k, \\ p_{2k} = p_\Gamma \text{ на } \Gamma_1 \cap D_k, \\ -(K_w + K_o + K_g)\,\partial p_{2k} / \partial n = q_n \text{ на } \Gamma_2 \cap D_k, \\ p_{2k}\,|_{\partial V_k} = P_k, \\ p_{2k} = p_{2\gamma k} \text{ на } \gamma_k.\end{gathered} \tag{12}$$

Постановка задачи для определения p_{3k} ($k = 1,...,N$) на сетке ω_k:

$$\begin{gathered}(mS_g(\rho_g)'_p / \rho_g)\partial p_{3k} / \partial t - div(K_w + K_o + K_g)grad\, p_{3k} = 0 \text{ в } D_k \cup V_k, \\ p_{3k} = 0 \text{ на } \Gamma_1 \cap (D_k \cup V_k) \\ p_{3k} = p_{3\gamma k} \text{ на } \gamma_k.\end{gathered} \tag{13}$$

Решения p_1, p_{2k} и p_{3k} при известных граничных значениях $p_{\Gamma_{1k}}$, $p_{\Gamma_{2k}}$, $p_{\Gamma_{3k}}$ независимо определяются из систем уравнений (11), (12) и (13) соответственно. При выполнении условий

на границах раздела γ_k относительно давлений $p_{\Gamma_{1k}} = p_{\Gamma_{2k}} + p_{\Gamma_{3k}}$ и относительно нормальных составляющих скоростей фильтрации $q_{1kn} + q_{2kn} + q_{3kn} = 0$ для определения решений p_1, p_{2k}, p_{3k} достаточно задания начальных условий и граничных значений $p_{\Gamma_{3k}}$. При значениях $p_{\Gamma_{3k}} = 0$ система уравнений (11)-(13) эквивалентна исходной системе уравнений (5)-(8),(10). Таким образом, для получения решения исходной системы уравнений необходимо построить алгоритм решения системы (11)-(13), в котором $p_{3k} \to 0$.

На каждой итерации $i \geq 1$ независимо определяются p_{2k}^i из решения задач (12) с граничными значениями на γ_k

$$p_{2\gamma k}^i = p_{2\gamma k}^{i-1} + p_{3\gamma k}^{i-1}, \tag{14}$$

где $p_{2\gamma k}^0$, $p_{3\gamma k}^0$ берутся с предыдущего временного шага. Для определения p_1^i, и p_{3k}^i совместно решаются системы уравнений (11), (13) на грубой сетке с дополнительными условиями $p_{\Gamma_{1k}}^i - p_{\Gamma_{3k}}^i = p_{\Gamma_{2k}}^i$ и $q_{1kn}^i + q_{3kn}^i = -q_{2kn}^i$ на γ_k при фиксированных узловых значениях решений p_{2k}^i. В вычислительном процессе, построенном таким образом, граничные значения $p_{3\gamma k}^i$ сносятся после каждой итерации по правилу (14) на граничные значения $p_{2\gamma k}^i$. В результате p_{3k}^i с ростом i стремятся к нулю, и i-ое приближение давления определяется решением p_1^i в области D_0 и решениями p_{2k}^i в подобластях D_k.

Этап 2. Вычисляются полные расходы, выходящие из ячеек грубой сетки в единицу времени

$$Q_{j,i}^{n+1} = (p_j^{n+1} - p_i^{n+1}) / R_{i,j}^n, \tag{15}$$

где $R_{i,j}^n = L_{i,j} / D_{i,j}(K_{wi}^n + K_{oi}^n + K_{gi}^n) + L_{j,i} / D_{i,j}(K_{wj}^n + K_{oj}^n + K_{gj}^n)$, $D_{i,j}$ - площадь общей граничной поверхности i-ой и j-ой ячеек, $L_{i,j}$ - расстояние от узлового значения i-ой ячейки до общей граничной поверхности, p_i^{n+1} - давление в i-ой ячейке.

Этап 3. Для полных расходов, выходящих из ячеек грубой сетки, вычисляются фазовые расходы по явной схеме для воды и нефти соответственно

$$Q_{w\,j,i}^{n+1,я} = (K_w^n / (K_w^n + K_o^n + K_g^n))_{j,i}^{up} Q_{j,i}^{n+1},$$
$$Q_{o\,j,i}^{n+1,я} = (K_o^n / (K_w^n + K_o^n + K_g^n))_{j,i}^{up} Q_{j,i}^{n+1}$$

где $(K_w^n / (K_w^n + K_o^n + K_g^n))_{j,i}^{up} = \begin{cases} (K^n{}_{wi} / (K^n{}_{wi} + K^n{}_{oi} + K^n{}_{gi})), \; p_i \geq p_j, \\ (K^n{}_{wj} / (K^n{}_{wj} + K^n{}_{oj} + K^n{}_{gj})), \; p_i < p_j. \end{cases}$

$(K_o^n / (K_w^n + K_o^n + K_g^n))_{j,i}^{up} = \begin{cases} (K^n{}_{oi} / (K^n{}_{wi} + K^n{}_{oi} + K^n{}_{gi})), \; p_i \geq p_j, \\ (K^n{}_{oj} / (K^n{}_{wj} + K^n{}_{oj} + K^n{}_{gj})), \; p_i < p_j. \end{cases}$

Этап 4. Для каждой прискважинной зоны независимо вычисляются насыщенности по неявной схеме из системы уравнений

$$m_i V_i (S_{wi}^{n+1} - S_{wi}^n) / \Delta t = \sum_j Q_{wj,i}^{n+1}, \tag{16}$$

$$m_i V_i (S_{oi}^{n+1} - S_{oi}^n) / \Delta t = \sum_j Q_{oj,i}^{n+1}, \tag{17}$$

где сумма берется по j-ым ячейкам, окружающим i-ую ячейку, $Q_{w\,j,i}^{n+1} = Q_{w\,j,i}^{n+1,я}$, $Q_{o\,j,i}^{n+1} = Q_{o\,j,i}^{n+1,я}$ для j-ых ячеек грубой сетки, из которых расход поступает в прискважинную зону,

$$Q_{w\,j,i}^{n+1} = Q_{w\,j,i}^{n+1,ня} = (K_w^{n+1}/(K_w^{n+1} + K_o^{n+1} + K_g^{n+1}))_{j,i}^{up} Q_{j,i}^{n+1},$$
$$Q_{o\,j,i}^{n+1} = Q_{o\,j,i}^{n+1,ня} = (K_o^{n+1}/(K_w^{n+1} + K_o^{n+1} + K_g^{n+1}))_{j,i}^{up} Q_{j,i}^{n+1}$$

в остальных случаях. Фазовые расходы $Q_{w\,j,i}^{n+1,я}$, $Q_{o\,j,i}^{n+1,я}$ являются граничными условиями при решении систем (16)-(17).

Этап 5. Вычисляются насыщенности для ячеек грубой сетки

$$S_{wi}^{n+1} = S_{wi}^{n} + (\Delta t / m_i V_i) \sum_j Q_{wj,i}^{n+1},$$
$$S_{oi}^{n+1} = S_{oi}^{n} + (\Delta t / m_i V_i) \sum_j Q_{oj,i}^{n+1}$$

где $Q_{w\,j,i}^{n+1} = Q_{w\,j,i}^{n+1,ня}$, $Q_{o\,j,i}^{n+1} = Q_{o\,j,i}^{n+1,ня}$ берутся из решения системы уравнений (16)-(17) для j-ых ячеек прискважинных зон, из которых расход поступает в ячейки грубой сетки, $Q_{w\,j,i}^{n+1} = Q_{w\,j,i}^{n+1,я}$, $Q_{o\,j,i}^{n+1} = Q_{o\,j,i}^{n+1,я}$ в остальных случаях.

4. Численные эксперименты

Предложенный алгоритм тестировался при решении модельной трехмерной задачи трехфазной фильтрации жидкостей с различным числом вертикальных добывающих и нагнетающих скважин. Рассматривался десятислойный пласт (≈1км×1км×0.018км) с толщинами слоев d_1 = 1 м, d_2 = 1 м, d_3 = 3 м, d_4 = 1 м, d_5 = 1 м, d_6 = 1 м, d_7 = 2 м, d_8 = 1 м, d_9 = 2 м, d_{10} = 5 м и абсолютными проницаемостями $k_1 = 10^{-3}$ дарси, $k_2 = 10^{-2}$ дарси, $k_3 = 25\times10^{-3}$ дарси, $k_4 = 10^{-2}$ дарси, $k_5 = 10^{-3}$ дарси, $k_6 = 10^{-2}$ дарси, $k_7 = 5\times10^{-2}$ дарси, $k_8 = 10^{-2}$ дарси, $k_9 = 10^{-3}$ дарси, $k_{10} = 15\times10^{-3}$ дарси соответственно. Кровля пласта считалась непроницаемой, на боковых поверхностях и подошве пласта давление P_Γ = 125 атм, на скважинах $P_к$= 30 атм, на боковой поверхности насыщенность $S_w = 0$, на подошве $S_w = 1$. Начальная насыщенность $S_w = 0$ за исключением зоны Γ_g (содержание газа в пласте), расположенной в центре пласта у кровли, в которой $S_w = 0$, $S_o = 0$, $S_g = 1$. Динамическая вязкость воды - $\mu_w = 1 мПа \cdot с$, динамическая вязкость нефти - $\mu_o = 15 мПа \cdot с$, плотность нефти $\rho_o = 0.882 г/см^3$, плотность воды $\rho_w = 1 г/см^3$. Относительные фазовые проницаемости брались линейными функциями от насыщенностей. Каждый интервал вскрытия моделировался круговым цилиндром с радиусом основания r=0,1 м и замыкался сверху и снизу сферическими поверхностями радиуса r=0,1 м. Таким образом, для каждой точки поверхности интервалов вскрытия вектор нормали определен однозначно. Ячейки, примыкающие к скважинам, имели размеры как в горизонтальной плоскости, так и по высоте, порядка 0,1 м.

На основе предложенных методов построены алгоритмы для решения задачи на суперкомпьютерах с большим числом вычислительных ядер и с большим числом графических процессоров. Алгоритмы распараллелены с помощью MPI процессов, OpenMP и CUDA технологий. Использовался язык C++ и среда разработки приложений Visual Studio 2008. При решении задачи для каждого MPI процесса выделяется равное число сгущающихся участков сетки. Для решения задач на сгущающихся участках, соответствующих одному MPI процессу, порождаются потоки с помощью технологии OpenMP, которые распределяют эти задачи на ядра процессора и на графические устройства. При этом задачи распределяются динамически, то есть по мере их решения. Вычисления на графическом устройстве включают в себя:

1. Построение систем уравнений для определения поля давления и насыщенности.
2. Решение линейных систем уравнений для определения поля давления с помощью библиотек cublas и cusparse.
3 Решение нелинейных систем уравнений для определения поля насыщенности в прискважинной зоне по неявной схеме методом Ньютона.
4. определение поля насыщенности по явным схемам для ячеек внескважинной зоны.

Предложенные алгоритмы тестировались на кластере, состоящем из 4-ядерных вычислительных узлов с процессорами Intel Core i7 2600 и оборудованными графическими ускорителями компании NVidia GTX 560 TI, при решении модельной трехмерной задачи трехфазной фильтрации жидкости с вертикальными добывающими и нагнетающими скважинами. Каждая прискважинная зона содержала около 25000 узлов, внескважинная зона около 6000 узлов. Общее число узлов для 200 сгущающихся участков достигало 5000000. Системы линейных уравнений для определения поля давления решались методом сопряженных градиентов с полиномиальным предобуславливанием. Системы нелинейных уравнений для определения поля насыщенностей прискважинной зоны решались по неявной схеме методом Ньютона.

Задача решалась на 2-х вычислительных узлах с общим числом ядер 8(4 ядра на узел) и 2 GPU ускорителя. Задачи запускались в режиме 1 MPI процесс на один узел, каждому MPI процессу соответствовало 1 GPU устройство. Рассматривались следующие варианты запуска задач:
1. На всех доступных ядрах с использованием одного MPI процесса и без использования GPU ускорителей.
2. На двух ядрах с использованием двух MPI процессов и двух GPU устройств. В этом случае каждому MPI процессу соответствовало одно GPU устройство и одно ядро.
3. На всех доступных ядрах с использованием MPI процессов и GPU устройств. В этом случае использовались все доступные ресурсы кластера (2 GPU устройства и 8 ядер).
В **Табл. 1-3** приведены результаты решения. Сравнение проводилось с решением задачи на одном ядре без использования GPU устройств.

Таблица 1 (вариант 1)

Число ядер	1	2	3	4
Ускорение	1	1.7	2.1	2.9

Таблица 2 (вариант 2)

Число GPU устройств	1 GPU	2 GPU
Ускорение	17.3	30.1

Таблица 3 (вариант 3)

Число GPU устройств	1 GPU	2 GPU
Ускорение	20.2	36.3

В Табл. 1 показано ускорение времени решения задачи, при использовании ядер только одного вычислительного узла. Эффективность использования ядер достигает 72%. Эффективность использования GPU устройств рассчитывается из Табл 2. и достигает 80%.

Также, предложенными методами решалась задача, в которой область Γ_g (содержание газа в пласте) предполагалась пустой, т. е. фактически рассматривалась двухфазная фильтрация[8]. Задача решалась на суперкомпьютере "GraphIT!" Научно-исследовательского вычислительного центра МГУ имени М.В.Ломоносова. На момент запуска задачи на суперкомпьютере "GraphIT!" было доступно 5 вычислительных улов с общим числом ядер 60(12 ядер на узел) и 15 GPU ускорителей. Также, как и в предыдущем примере, рассматривались 3 варианта запуска задач. На **Рис. 1-3** приведены результаты решения.

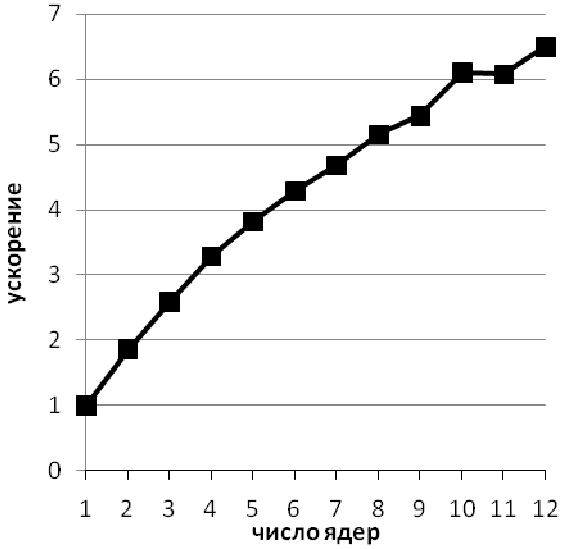


Рис. 1. Ускорение времени решения задачи на одном узле в зависимости от числа ядер (1 вариант).

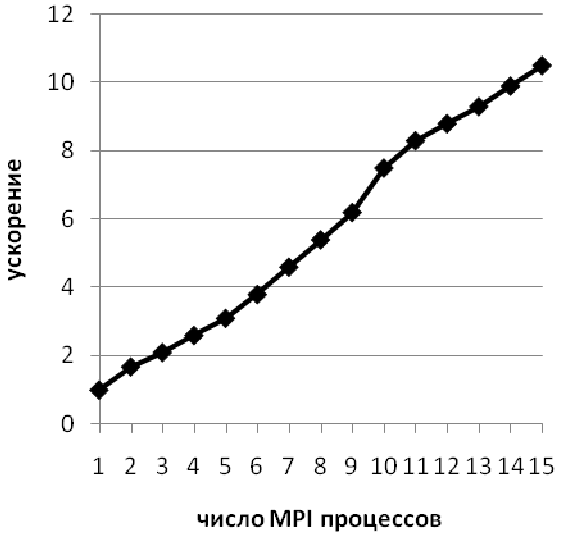


Рис. 2. Ускорение времени решения задачи в зависимости от числа MPI процессов(2 вариант).

На Рис. 1 приведено ускорение времени решения задачи в зависимости от числа ядер, в случае запуска задачи на одном узле с одним MPI процессом и без использования GPU устройств (1 вариант). Сравнение проводилось с решением на 1 ядре и без использования GPU устройств. Из рисунка видно, что получено 7-кратное ускорение на 12 ядрах (эффективность 60%). На Рис. 2 приведено ускорение времени решения задачи в зависимости от числа MPI процессов, в случае запуска задачи с использованием различного числа GPU процессоров и выделением 1 ядра для каждого MPI процесса(2 вариант). Сравнение проводилось с результатом решения на 1 ядре и 1 GPU устройстве. Показана эффективность 70% при решении на 15 GPU процессоров.

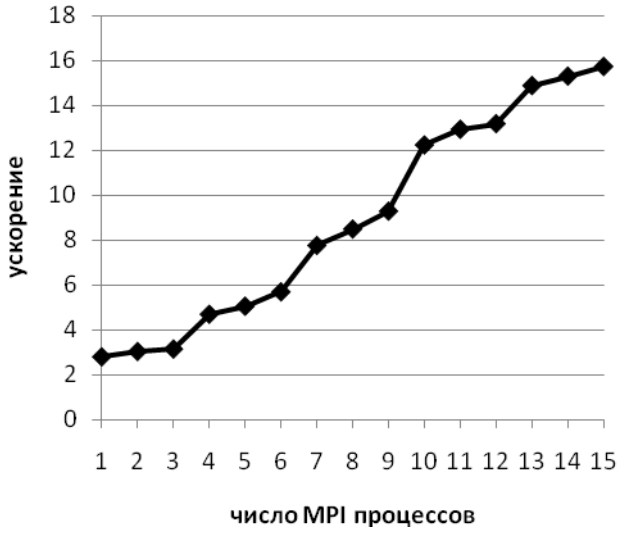


Рис. 3. Ускорение времени решения задачи в зависимости от числа MPI процессов (3 вариант).

На Рис. 3 приведено ускорение времени решения в случае запуска задач с различным числом MPI процессов и GPU процессоров (вариант 3), при этом использовались все доступные ядра узлов на которых запускались MPI процессы. Сравнение проводилось со временем решения задачи на 1 GPU процессоре и 1 ядре. Отметим, что если сравнивать решение на 1 ядре без MPI процессов и GPU процессоров с решением задачи с использованием всех доступных вычислительных устройств суперкомпьютера, то ускорение достигало 120 раз.

5. Выводы

Построены алгоритмы для решения задачи трехфазной фильтрации жидкости на сетках со сгущающимися участками, основанные на методах декомпозиции области. Декомпозиция сеточной системы уравнений по давлению основана на согласовании решений для сгущающихся участков с решением на грубой сетке за счет введения дополнительных грубых сеток. Декомпозиция сеточной системы уравнений по насыщенности основана на независимом решении уравнений на сгущающихся участках по неявным схемам и на согласовании этих решений с решением на грубой сетке с использованием элементов явной и неявной схем. На основе предложенных методов декомпозиции построены алгоритмы для решения задачи на гетерогенных вычислительных системах. Показана высокая эффективность использования многопроцессорных систем, построенных на базе графических процессоров.

Литература

1. Chen Z., Huan G., Ma Y. Computational methods for multiphase flows in porous media. SIAM, 2006.
2. Басниев К. С., Власов А.М., Кочина И.М., Максимов В.М. Подземная гидравлика. М.: Недра, 1986, 303 с.
3. Азис Х., Сеттари Э. Математическое моделирование пластовых систем. М.: Недра, 1982, 407 с.
4. Мазуров П.А., Цепаев А.В. Алгоритмы для распараллеливания решения задач двухфазной фильтрации жидкости на сетках со сгущающимися участками. // Вычислительные методы и программирование.2006, Т.7, №2, С. 115–123.
5. Мазуров П.А., Цепаев А.В. Метод решения нелинейных задач фильтрации жидкости в трехмерных пластах с гидродинамически несовершенными скважинами. // Математическое мо-

делирование. 2004, Т.16, №3, С. 33-42.
6. Губайдуллин Д.А., Мазуров П.А., Цепаев А.В. Алгоритм решения трехмерных задач напорно-безнапорной стационарной фильтрации жидкости со сгущающимися участками сетки. // Вычислительные методы и программирование. 2005, Т.6, №2, С. 217 - 225.
7. Губайдуллин Д.А., Никифоров А.И., Цепаев А.В. Алгоритмы распараллеливания на сгущающихся сетках в задачах трехфазной фильтрации жидкости. // Вычислительные методы и программирование. - 2007. - Т.8. - №2. - С. 360 - 366.
8. Цепаев А. В. Использование гетерогенных вычислительных систем для решения задач фильтрации жидкости методами декомпозиции области. // Вычислительные методы и программирование. 2012. Т. 13. № 1. 39-44.

Использование GPU для ускорения поиска в ширину на графах*

М.А. Черноскутов

Институт математики и механики УрО РАН

В работе описана реализация алгоритма поиска в ширину на графе с использованием технологии CUDA на GPU, проведено сравнение ее быстродействия с реализацией того же алгоритма на CPU на основе технологии OpenMP. Сравнение производилось с использованием графов, имеющих различное число вершин и ребер. В результате экспериментов выявлено, что GPU опережает CPU по производительности на всем диапазоне тестов.

1. Введение

В последнее время растет популярность использования GPU в качестве вычислителей общего назначения, с помощью которых потенциально можно запрограммировать решения самых разнообразных задач. Сейчас GPU используются в основном для приложений, которые принято называть вычислительно-интенсивными («computing intensive tasks»): механика сплошных сред, молекулярная динамика, трассировка лучей и т.д. Однако растет потребность и в задачах совершенно другой природы – «data intensive tasks», которые характеризуются работой с большими массивами данных и активным использованием нерегулярного доступа в память [1]. На данный момент уже появился специализированный рейтинг Graph500 (аналог рейтинга Top500) [2], целью которого является оценка производительности вычислительных систем при обработке больших массивов данных.

Одним из представителей задач класса «data intensive» является поиск в ширину на графе [3], который в данный момент используется в качестве вычислительного ядра теста Graph500 (пока реализация доступна только для CPU) [4]. Данный выбор обусловлен следующими обстоятельствами [2]:

- алгоритм обхода графа в ширину используется в различных научных и технических приложениях (анализ социальных сетей, информационная безопасность, биоинформатика, нефтедобыча);
- структура графов отображает реальные наборы данных, которые встречаются в задачах;
- результаты масштабирования задачи обхода графов должны хорошо соответствовать масштабированию реальных приложений.

В данный момент известно множество работ по обработке больших графов на CPU и GPU и сравнению их производительности: [5-9].

2. Постановка задачи

Целью работы является анализ эффективности использования GPU в задаче поиска в ширину на графе. Основным критерием эффективности является отношение времени выполнения тестовой задачи (поиск в ширину на заранее выбранном графе) на GPU по сравнению с временем выполнения этой же задачи на CPU. Распараллеливание алгоритма обхода графа осуществляется по всем доступным ядрам GPU и CPU. В качестве входных данных для алгоритма используются графы одинакового размера (имеется в виду не только объем занимаемой памяти, но и количество ребер и вершин). Результатом работы программы является время и скорость обхода заданного графа (измеряется в количестве пройденных ребер в секунду [2]). Ранжирование результатов осуществляется по скорости обхода графа.

* Работа поддержана грантами УрО РАН РЦП-12-П13, РЦП-13-П18. При проведении работ был использован суперкомпьютер “Уран” ИММ УрО РАН.

3. Реализация

В каждом эксперименте использовался ориентированный граф, с заранее заданным числом исходящих из каждой вершины ребер. При этом противоположные концы ребер выбирались случайным образом (возможность образования петель и несвязных компонент при этом не исключается). Графы с подобной структурой часто встречаются в реальных задачах, таких как анализ социальных сетей или биоинформатика. Размер графов выбирался таким образом, чтобы заполнить всю доступную DRAM память на GPU. Используемые в экспериментах графы приведены в табл.1.

Таблица 1. Графы, используемые в эксперименте

Количество исходящих из вершины ребер, шт.	**Количество вершин, шт.**	**Размер графа, ГБ**
5	66 060 288	2.21
10	48 234 496	2.52
15	35 651 584	2.52
20	28 311 552	2.53
25	23 068 672	2.49
30	19 922 944	2.52
35	16 777 216	2.44
40	14 680 064	2.41
45	13 631 488	2.49
50	12 582 912	2.53
55	11 534 336	2.54
60	10 485 760	2.50
65	9 437 184	2.43
70	8 388 608	2.31

Для поиска в ширину с использованием GPU реализовано два CUDA-ядра. Первое производит перебор вершин в текущей итерации (в порядке обхода в ширину) и заносит данные об их соседях в специальный массив, а второе, анализируя этот массив, формирует список вершин для следующей итерации алгоритма. Каждому GPU-потоку ставится в соответствие одна вершина, которую он просматривает на каждой итерации (реализация очередей, присущих стандартному алгоритму поиска в ширину, для GPU не эффективна в виду больших накладных расходов на обработку очереди). В текущей реализации в каждом блоке задействовано 1024 потока. Количество блоков варьируется в зависимости от количества исходящих ребер для каждой вершины. Исходный код обоих ядер, а также описание используемой для графа структуры данных представлено на рис.1.

Поиск в ширину на графе с помощью CPU производится аналогичным образом – используется две функции, распараллеленные на все доступные ядра CPU: первая обходит вершины на текущей итерации, а вторая формирует список вершин для следующей итерации. Для описания вершин и ребер графа использовалась та же самая структура данных, что и в версии для GPU. Исходный код обеих функций приведен на рис.2. Как видно, объем кода для обеих версий алгоритма оказался примерно одинаковым.

```
struct node {
  int x;
  int edge[DEGREE];
  int next;
  int visited;
};

__global__ void traversal(struct node *d_arr,int
*d_next_lev,int *d_count)
{
  d_count[0]++;
  int k,i = blockDim.x * blockIdx.x + threadIdx.x;
  if(d_arr[i].next==1 && d_arr[i].visited==0)
  {
    d_count[1]++;
    d_arr[i].next = 0;
    d_arr[i].visited = 1;
    for(k=0;k<DEGREE;++k)
      d_next_lev[d_arr[i].edge[k]] = 1;
  }
}

__global__ void refresh(struct node *d_arr,int *d_next_lev)
{
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if(d_next_lev[i]==1 && d_arr[i].visited==0)
    d_arr[i].next = 1;
  d_next_lev[i] = 0;
}
```

Рис. 1. Обход графа в ширину с использованием GPU

```
int traversal()
{
  int i,k,s = 0;
  int start =
omp_get_thread_num()*(SIZE/omp_get_num_threads());
  int end = start + (SIZE/omp_get_num_threads());
  for(i=start;i<end;++i)
  {
    if(arr[i].next==1 && arr[i].visited==0)
    {
      s++;
      arr[i].next = 0;
      arr[i].visited = 1;
      for(k=0;k<DEGREE;++k)
         next_lev[arr[i].edge[k]] = 1;
    }
  }
  return s;
}

void sync()
{
  int i;
  int start =
omp_get_thread_num()*(SIZE/omp_get_num_threads());
  int end = start + (SIZE/omp_get_num_threads());
  for(i=start;i<end;++i)
  {
    if(next_lev[i]==1 && arr[i].visited==0)
      arr[i].next = 1;
    next_lev[i] = 0;
  }
}
```

Рис. 2. Обход графа в ширину с использованием CPU

Для тестирования использовался GPU Nvidia Tesla M2050, имеющий 448 вычислительных ядер и 3 ГБ DRAM памяти (в действительности, доступным для использования оказалось около 2.5 ГБ из-за включенного ECC и других накладных расходов) и CPU Intel Xeon X5675, имеющий 6 вычислительных ядер и 48 ГБ оперативной памяти. Оба вычислителя располагаются в сервере HP ProLiant SL390s G7 4U.

Программа для GPU скомпилирована с использованием nvcc, входящего в комплект CUDA Toolkit 4.0. Для CPU использовался компилятор icc версии 11.1 и технология OpenMP. Для распараллеливания обхода графа в ширину на все процессоры в CPU и потоки в GPU использовались технологии OpenMP и CUDA. Для генерирования ребер в графах использовался генератор случайных числе rand(), входящий в стандартную библиотеку языка C.

4. Эксперимент

Программам на GPU и CPU подавались графы с одинаковым числом вершин и одинаковым количеством ребер, исходящих из каждой вершины. Эксперименты проводились с графами, имеющими от 5 до 70 исходящих из каждой вершины ребер. Количество вершин в графе варьировалось от 8 388 608 до 66 060 288, в зависимости от объема доступной памяти и числа исходящих из каждой вершины ребер.

Как видно из рис.3, GPU заметно опережает CPU во всех экспериментах. Наибольшее преимущество GPU имеет при обходе графов с малым числом исходящих ребер. При увеличении числа исходящих ребер, преимущество GPU постепенно уменьшается, вследствие роста количества запросов в память, производимых множеством параллельных нитей. В CPU, наоборот, параллельных потоков намного меньше и вычислительные ядра имеют большую скорость работы по сравнению с нитями GPU.

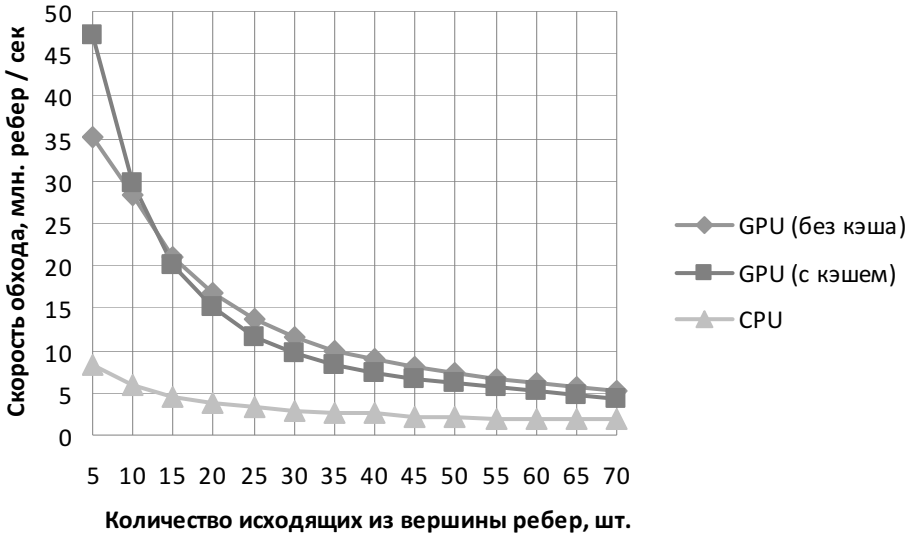


Рис. 3. Скорость обхода графов на GPU и CPU

Достигнутое на GPU ускорение показано на рис.4. Интересной особенностью является влияние кэша L1 в GPU на скорость обхода графа. При малом числе исходящих ребер кэш GPU оказывает положительное влияние на производительность, т.к. может захватить больше данных о вершинах графа, а с ростом их числа, наоборот, показывает ухудшающиеся результаты. К тому же с ростом количества исходящих ребер снижается пространственная и временная локальность [10] и степень промахов кэша из-за этого только увеличивается.

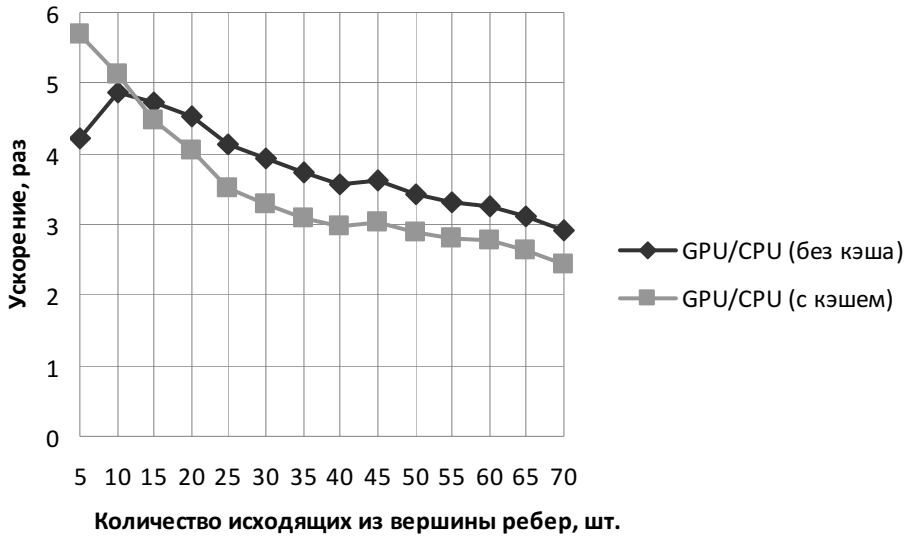


Рис. 4. Ускорение, достигнутое на GPU

5. Заключение

В результате эксперимента выявлено, что поиск на графах в ширину при использовании GPU производится с большей скоростью (от 2.4 до 5.7 раз быстрее), по сравнению с аналогичной задачей на CPU, при условии, что размер графа не превышает объем памяти в GPU. Установлено, что кэш L1 в GPU в большинстве случаев не дает выигрыша в производительности.

В качестве дальнейших направлений исследований интерес представляет:

- обход графов, созданных с помощью генераторов (таких как Kronecker Graph Generator [11]), а также графов полученных в результате решения реальных задач (к примеру графы из Sparse Matrix Collection [12] или DIMACS Implementation Challenge [13]);
- анализ эффективности обхода больших графов в multi-GPU системах, а также в кластерных системах, использующих GPU;
- решение реальных прикладных и фундаментальных задач класса «data intensive tasks» с использованием GPU.

Литература

1. Furht B., Escalante A. «Handbook of Data Intensive Computing», Springer, 2011.
2. Murphy R., Wheeler K., Barrett B., Ang J. «Introducing the Graph 500» // Cray User's Group (CUG), May 5, 2010.
3. Кормен Т., Лейзерсон Ч., Риверс Р., Штайн К. «Алгоритмы. Построение и анализ» – М.: «Вильямс», 2011.
4. Graph 500 Benchmark 1 (“Search”) // URL:http://www.graph500.org/specifications (дата посещения: 01.06.2011).
5. Harish, P. and Narayanan, P.J. «Accelerating large graph algorithms on the GPU using CUDA» // Proceedings of the 14th international conference on High performance computing (Berlin, Heidelberg, 2007), 197–208.
6. Hong, S. et al. «Accelerating CUDA graph algorithms at maximum warp» // Proceedings of the 16th ACM symposium on Principles and practice of parallel programming (New York, NY, USA, 2011), 267–276.

7. Merrill D., Garland M., Grimshaw A. «Scalable GPU graph traversal» // Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP'12), 117–128.
8. Hong S., Oguntebi T., Olukotun K. «Efficient Parallel Graph Exploration on Multi-Core CPU and GPU» // Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques (Galveston, TX, USA, October 10-14, 2011), 78–88.
9. Корж А.А. «Масштабирование Data Intensive приложений с помощью библиотеки Dislib на суперкомпьютерах BlueGene/P и Ломоносов» // Материалы Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Новороссийск, 19-24 сентября 2011 г., Изд-во Московского Университета, с.126–131.
10. Murphy R., Kogge P. «On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications» // IEEE Transactions on Computers 56(7), July 2007, 937–945.
11. Seshadhri C., Pinar A., Kolda T. «An In-Depth Study of Stochastic Kronecker Graphs» // Proceedings of the 2011 IEEE 11th International Conference on Data Mining (ICDM), 587–596.
12. Davis T., Hu Y. «The University of Florida Sparse Matrix Collection» // ACM Transactions on Mathematical Software, Vol. V, No. N, M 20YY, 1–28.
13. 10th DIMACS Implementation Challenge // URL:http://www.cc.gatech.edu/dimacs10/index.shtml (дата посещения: 01.06.2011).

Программная архитектура системы передачи интенсивного потока данных в распределенных системах*

В.А. Щапов[1,2]

Институт механики сплошных сред УрО РАН[1],
Пермский национальный исследовательский политехнический университет[2]

В статье рассмотрена программная архитектура системы передачи интенсивного потока данных в распределенных системах. Предложенная архитектура реализует разработанную в ИМСС УрО РАН модель потоковой обработки данных от экспериментальной установки PIV (Particle Image Velocimetry) на удаленных высокопроизводительных вычислительных системах с передачей данных по скоростной оптической сети большой протяженности. Описаны класс прикладных задач, для которых можно использовать систему, и схема взаимодействия распределенных компонент программного обеспечения (ПО). Приводятся результаты измерений, иллюстрирующие эффективность разработанных алгоритмов диспетчеризации параллельных потоков данных и протокола взаимодействия оконечных систем.

1. Введение

Современные экспериментальные установки генерируют большие объемы данных, нуждающихся в обработке в реальном времени. Одной из таких задач является обработка экспериментальных данных, получаемых по методу PIV (Particle Image Velocimetry) – оптическому методу измерения полей скорости жидкости или газа в выбранном сечении потока. Метод основан на обработке пар фотографий трассеров — мелких частиц, взвешенных в потоке, в моменты, когда они подсвечиваются импульсным лазером, создающим тонкий световой нож. Скорость потока определяется расчетом перемещения трассеров за время между вспышками лазера. Интенсивность порождаемого потока данных зависит от числа, разрешения и частоты работы камер и может достигать нескольких гигабит в секунду [1]. Обработка таких потоков данных с использованием только локального компьютера экспериментальной установки (ЭУ) чрезвычайно затруднена. В то же время развитие математического аппарата и появление новых высокоточных алгоритмов расчетов увеличивают требования к необходимой вычислительной мощности. Поэтому перенос вычислений на удаленные суперкомпьютеры достаточной производительности позволит применять новые высокоточные алгоритмы, обрабатывать данные в реальном времени, уменьшить объемы сохраняемых на ЭУ данных. В ИМСС УрО РАН данное направление развивается в рамках проекта «Распределенный PIV» [2] с использованием высокоскоростной оптической магистрали, создаваемой в рамках проекта «Инициатива GIGA UrB RAS» [3].

Разработанная программная архитектура использует лямбда грид парадигму распределенных вычислений [4], в которой используется параллелизм потоков данных в скоростных оптических сетях со спектральным уплотнением каналов. Достижение эффективной диспетчеризация параллельных потоков между сопрягаемыми системами является основной целью представленной работы. Измерения проводились по выделенному L2 каналу связи, объединяющий в единую подсеть ЭУ (Пермь, ИМСС УрО РАН), суперкомпьютеры «Уран» и «UM64» и систему хранения данных (СХД) EMC Celerra NS-480 (Екатеринбург, ИММ УрО РАН). В настоящий момент доступная пропускная способность выделенного канала связи составляет 1 Гбит/с.

2. Задачи, допускающие параллельную обработку потока данных

Анализ исходной задачи потоковой обработки исходных данных эксперимента PIV показал, что каждое измерение можно обрабатывать независимо от других [5]. Исходя из этого, бы-

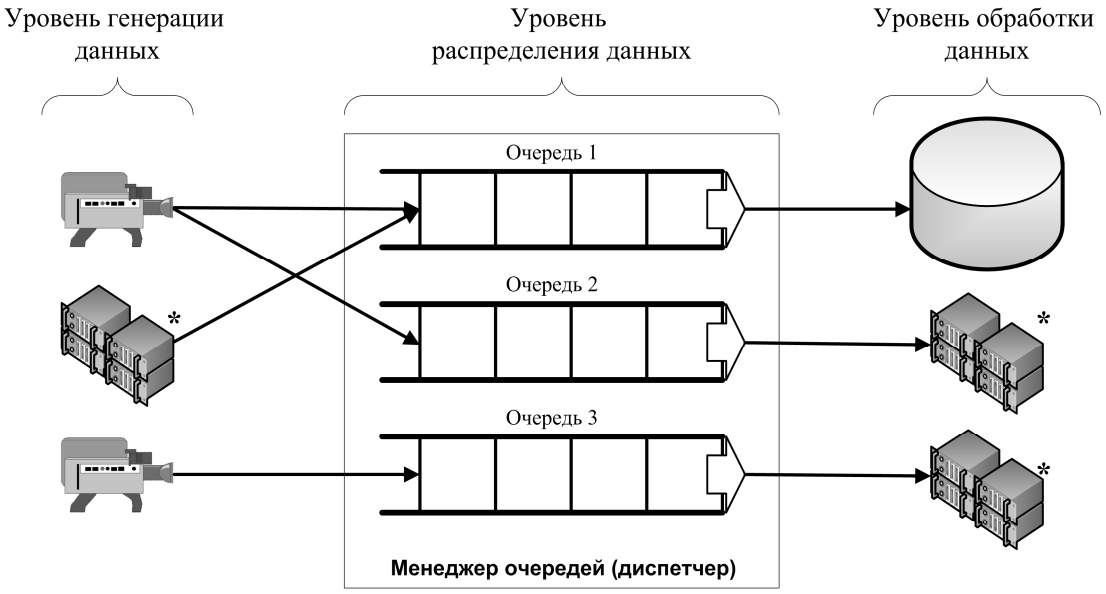
* Исследование проводится при поддержке РФФИ (грант № 11-07-96001-р_урал_а).

ло предложено отказаться от однозначного отображения измерений на вычислительные узлы и применить концепцию очередей для обработки данных [6].

Обобщение концепции очередей позволило сформулировать границы применимости подхода для расчетных алгоритмов. Параллельная потоковая обработка исходных данных возможна только тогда, когда расчетный алгоритм позволяет выделить во множестве исходных данных независимые подмножества, допускающие независимую обработку.

3. Трехуровневая программная архитектура

Возможность независимой обработки подмножеств исходных данных позволило применить концепцию очередей для решения задачи передачи и диспетчеризации потока данных (рис. 1).



*) Одно устройство может находиться и на уровне генерации данных, и на уровне обработки данных

Рис. 1. Трехуровневая программная архитектура

Концепция очередей приводит к разделению системы на три уровня:

1. уровень генерации данных;
2. уровень распределения данных;
3. уровень обработки данных.

Распределение очереди проводится по принципу First In First Out (FIFO). При этом передача данных между уровнями реализуется с использованием протокола, получившего название «Протокол PIV», построенного на идее модели RPC (рис. 2). В этом случае сервером является менеджер очередей, реализующий уровень распределения данных. Клиентами является ПО на уровнях генерации и обработки данных.

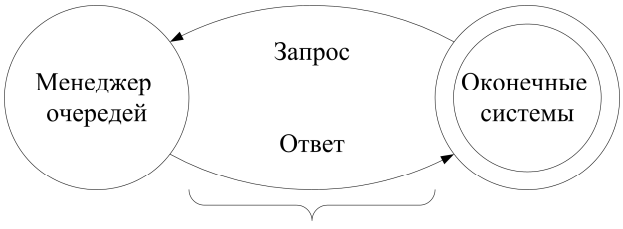


Рис. 2. Схема взаимодействия компонент

Выделение задачи диспетчеризации в отдельный слой, который может располагаться, как на ЭУ, так и на отдельном сервере, позволило отказаться от межузлового обмена данными на стороне суперкомпьютера, что дает следующие преимущества:

- автоматическая балансировка нагрузки по вычислительным узлам — более быстрые узлы будут чаще посылать запросы новых данных и, таким образом, будут получать больше данных для обработки;
- возможность изменять число используемых вычислительных узлов во время проведения эксперимента;
- возможность использовать вычислительную мощность нескольких суперкомпьютеров;
- минимизация потери данных в случае выхода из строя одного или нескольких вычислительных узлов (в худшем случае потеряется только текущее обрабатываемое измерение сбойного узла). В случае если потеря данных недопустима, то на время расчета измерения необходимо сохранять в памяти сервера и, в случае выявления сбоя, повторно добавлять эти блоки в очередь готовых к обработке измерений.

Необходимо отметить, что предлагаемый подход не ограничивает нас в выборе транспорта для передачи данных. Текущая реализация использует для передачи данных протокол PIV. Однако, возможно и использование общего СХД, подключенного и к ЭУ, и к супервычислителю. В этом случае задачей диспетчера будет распределение по вычислительным узлам не самих данных, а путей к файлам с данными на СХД, при этом вычислительные узлы будут считывать данные непосредственно из файлов на СХД. Это позволит анализировать и сравнивать поведение, как классической схемы с общим дисковым пространством, так и предлагаемой схемы без промежуточного хранения данных на дисковых массивах.

Так как технология позволяет вынести менеджер очередей на отдельный сервер, это может позволить снизить нагрузку на ЭУ благодаря переносу задачи по обслуживанию большого числа параллельных TCP-соединений на отдельный сервер, при этом ЭУ будет передавать данные менеджеру очередей через небольшое число TCP-соединений. Так как менеджер очередей будет располагаться недалеко от ЭУ, то даже небольшое число TCP-соединений будут полностью утилизировать канал связи между ЭУ и менеджером очередей.

На рис. 3 показано применение предложенной архитектуры в проекте «Распределенный PIV».

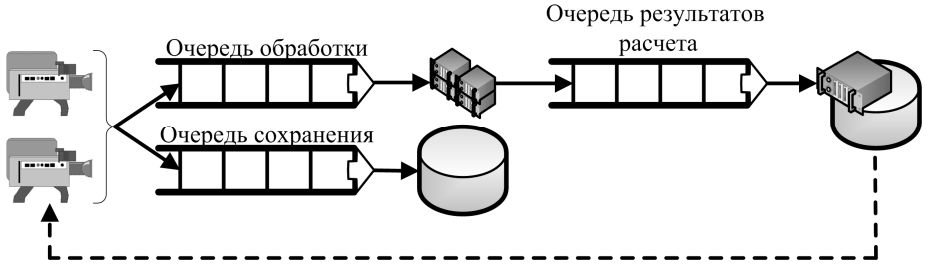


Рис. 3. Применение трехуровневой программной архитектуры в проекте «Распределенный PIV»

Данные с камер ЭУ загружаются в одну или две очереди: очередь обработки и очередь сохранения. Данные из очереди сохранения забираются ПО, которое запускается на хранилище данных, что позволяет сохранить исходные данные эксперимента в случае невозможности их сохранения на ЭУ по техническим причинам. Данные из очереди обработки передаются на вычислительные узлы суперкомпьютеров, результаты обработки помещаются в очередь результатов расчетов, откуда их получает ПО, работающее на ЭУ. Полученные ЭУ результаты расчета могут использоваться для управления экспериментом. При необходимости сохранения результатов расчетов на внешнем хранилище они также могут помещаться в две или более очереди, из одной из которых данные будут забираться ПО хранилища данных.

3.1 Протокол PIV

Протокол PIV является протоколом прикладного уровня, реализующим идею модели RPC. Протокол работает по схеме запрос-ответ и может использовать в качестве протокола транс-

портного уровня любой надежный потоковый протокол передачи данных. Текущая реализация протокола PIV поддерживает транспортные протоколы TCP и UDT [7]. Положение протокола PIV в стеке сетевых протоколов показано на рис. 4 [8].

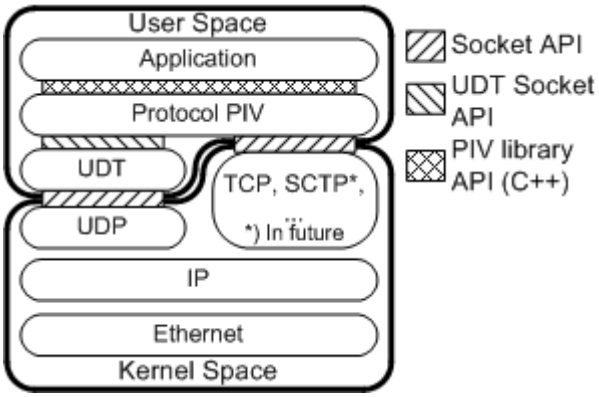


Рис. 4. Положение протокола PIV в стеке сетевых протоколов

Протокол UDT – это, основанный на UDP, протокол передачи данных для высокоскоростных сетей. Он был разработан в Университете штата Иллинойс в Чикаго. Функциональные возможности протокола UDT аналогичны протоколу TCP. UDT является дуплексным протоколом передачи потока данных с предварительной установкой соединения. Особенностью протокола UDT является оригинальная архитектура и реализация, а также оригинальный алгоритм управления перегрузкой. При этом протокол UDT позволяет программисту реализовать и использовать свой алгоритм управления перегрузкой.

Протокол PIV рассчитан на передачу нескольких именованных блоков бинарных данных. В одном пакете протокола PIV можно передать от нуля до 65535 блоков, каждый из которых может иметь размер до 4 Гбайт, при этом сохранение порядка следования блоков не гарантируется. Длина имени блока ограничена 255 байтами. На рис. 5 приведен формат пакета протокола PIV. Поля заголовка пакета протокола кодируются в сетевом порядке байт.

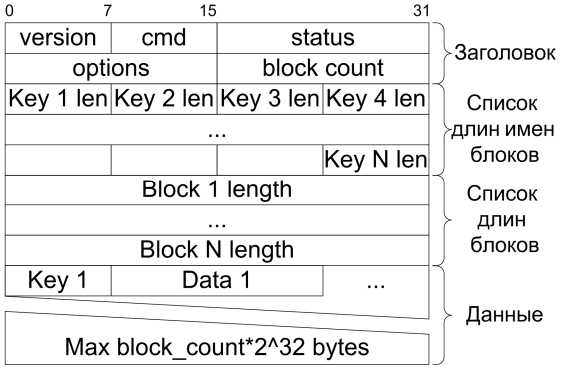


Рис. 5. Формат пакета протокола PIV

Протокол поддерживает версионирование с использованием поля `version`, что позволяет реализовать в рамках одной версии ПО поддержку нескольких версий протоколов.

Поле `cmd` содержит один из следующих кодов команд:

- `SciMQ_PROTOCOL_CMD_RESPONSE` – пакет ответа сервера на запрос клиента;
- `SciMQ_PROTOCOL_CMD_PING` – запрос, отправляемый клиентом для проверки активности канала связи;
- `SciMQ_PROTOCOL_CMD_GET` – запрос данных у сервера;
- `SciMQ_PROTOCOL_CMD_POST` – отправка данных на сервер;
- `SciMQ_PROTOCOL_CMD_CONFIRM` – команда подтверждения приема данных при использовании надежных очередей;
- `SciMQ_PROTOCOL_CMD_CREATE_QUEUE` – команда создания очереди;
- `SciMQ_PROTOCOL_CMD_DELETE_QUEUE` – команда удаления очереди;

- `SciMQ_PROTOCOL_CMD_GETINFO_QUEUE` – команда получения информации об очереди;
- `SciMQ_PROTOCOL_CMD_MODIFY_QUEUE` – команда изменения очереди.

Поле `status` предназначено для передачи клиенту статуса ответа от сервера. В зависимости от значения статуса можно судить о выполнении или невыполнении запроса клиента.

Поле `options` предназначено для кодирования дополнительных опций.

Поле `block_count` содержит число передаваемых блоков данных.

Далее в пакете располагаются список длин имен блоков и список длин блоков, после чего передаются пары, состоящие из имени блока и данных блока.

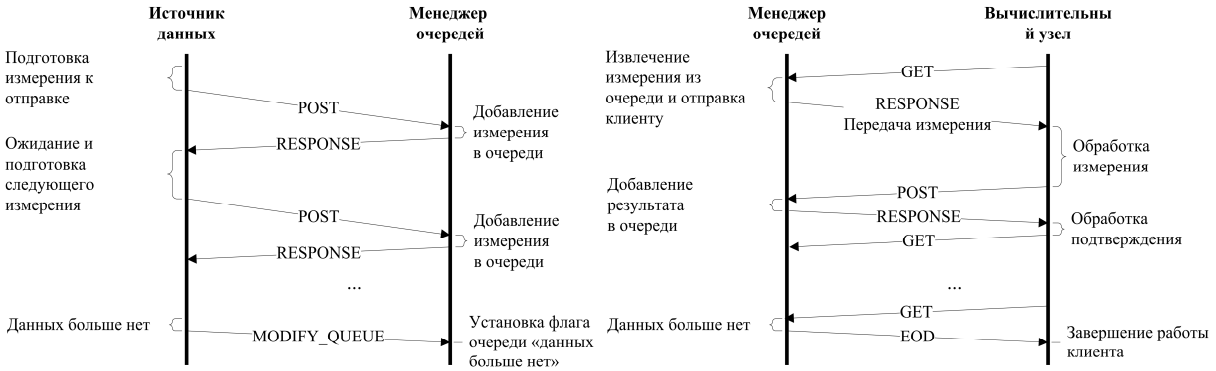


Рис. 6. Временная диаграмма работы протокола PIV

На рис. 6 приведена временная диаграмма работы протокола PIV в трехуровневой программной архитектуре. Левая часть диаграммы показывает взаимодействие уровня генерации данных, а правая часть – уровня обработки данных с менеджером очередей. Для наглядности эти процессы разделены, однако существенным является то обстоятельство, что передача по протяженной линии связи занимает значительное время и процессы передачи необходимо выполнять одновременно при наличии данных.

3.2 Уровень генерации данных

Уровень генерации данных реализует загрузку исходных данных из источника и отправку их на уровень распределения данных, используя для взаимодействия с сервером очередей специальную клиентскую библиотеку.

Существуют различные стратегии загрузки исходных данных. В случае использования, например, ПО ActualFlow данные могут загружаться из файлов в файловой системе ЭУ. В более сложных случаях, когда требуется обрабатывать данные от большого числа источников, возможна передача данных непосредственно с датчиков, например с камер ЭУ, обладающих Ethernet-интерфейсом и позволяющих реализовать логику протокола PIV.

3.3 Уровень распределения данных

Уровень распределения данных решает задачу получения блоков данных от уровня сбора данных и их передачу на уровень обработки данных. В связи с тем, что уровень обработки данных состоит из множества вычислительных процессов, запущенных на узлах суперкомпьютера, на диспетчер ложится задача по распределению очередей данных на множество узлов суперкомпьютера.

В настоящий момент реализован подход, когда, при наличии данных в очереди, они будут отдаваться сервером на все поступающие запросы, при наличии доступных ресурсов на сервере. Однако, возможно, что с точки зрения загрузки сети и отзывчивости системы, данный подход не будет являться оптимальным. Исследование поведения системы и выбор оптимальных стратегий распределения очереди в условиях большого числа вычислительных узлов (более 512) будут предметом дальнейших исследований. Предполагается определить такую конфигу-

рацию параметров, при которой будет соблюдаться баланс между числом активных параллельных соединений, уровнем загрузки сети и временем получения результата расчета после начала его отправки (отзывчивость системы).

3.4 Уровень обработки данных

На этом уровне происходит обработка данных. Это может быть расчет исходных данных с применением каких-либо алгоритмов, сохранение данных в высокопроизводительных хранилищах или на большом количестве локальных дисков, передача данных из очередей в сторонние системы и т.д.

Как и уровень генерации, уровень обработки данных взаимодействует с сервером очередей уровня распределения с помощью специальной клиентской библиотеки.

4. Программная реализация

Предложенная программная архитектура была реализована в виде комплекса ПО для всех уровней архитектуры.

Были разработаны три компоненты общего назначения:

- сервер очередей,
- клиентская библиотека,
- управляющее ПО.

И два специализированных компонента для PIV-расчетов:

- клиент загрузки исходных данных и сохранения результатов расчетов,
- клиент, упрощающий написание расчетных алгоритмов для суперкомпьютера.

Все приложения и библиотеки разработаны на языке программирования C++ с использованием библиотек Boost[1]. Использование Boost позволило ускорить разработку с выполнением требований по кроссплатформенности ПО, а выбор языка программирования C++ и использование оптимизирующих компиляторов GCC и Intel C++ позволил разрабатывать высокопроизводительный код.

Разработанный комплекс ПО поддерживает работу под управлением операционных систем Red Hat Enterprise Linux 5 и новее, SUSE 11 и новее, и Windows 7 и новее. Поддерживаются компиляторы GCC 4.1.2 и новее, Intel C++ 11 и новее, и Microsoft Visual C++ 2010 (10.0).

4.1 Сервер очередей

Сервер очередей реализует функциональность менеджера очередей (диспетчера). Архитектурно он состоит из четырех компонент:

- управляющий модуль,
- модуль очередей,
- модуль фоновых процессов,
- модуль сетевого взаимодействия.

Все модули, за исключением менеджера очередей, работают по событийно-ориентированному принципу с использованием технологий Boost.ASIO.

Управляющий модуль отвечает за начальную инициализацию сервера, загрузку параметров конфигурации, запуск и остановку остальных модулей, обработку сигналов операционной системы, таких как сигналы немедленного и плавного завершения и сигнал для переоткрытия файлов журналов. Цикл обработки сообщений реализуется объектом `boost::asio::io_service`, метод `run()` которого выполняется непосредственно в главном потоке приложения.

Модуль очередей управляет хранением очередей в памяти сервера. Каждая очередь имеет свое уникальное имя и может быть с гарантией обработки или без гарантии обработки. Если очередь без гарантии обработки, то элемент удаляется из очереди сразу же после отправки кли-

[1] Boost C++ Libraries: http://www.boost.org/

енту. Если очередь с гарантией обработки, то, после передачи клиенту, элемент помещается во временный список, из которого удаляется после получения подтверждения обработки или перемещается в основную очередь, если в течение задаваемого пользователем таймаута подтверждение получено не было. Однако использование таких очередей повышает требования к доступной оперативной памяти, так как данные приходится хранить дольше.

Для распределения памяти модуль очередей использует два менеджера памяти, которые позволяют отдельно настроить ограничение на максимальное потребление памяти для структур самой очереди и для данных, помещаемых в очередь.

Модуль фоновых процессов предназначен для выполнения низкоприоритетных отложенных задач и состоит из одного объекта-диспетчера событий `boost::asio::io_service`, методы `run()` которого выполняются в одном или более независимых потоков. Это позволяет запускать в рамках одного диспетчера событий длительные задачи, так как, пока выполняется одна задача, остальные задачи могут обрабатываться оставшимися потоками. Модуль фоновых процессов используется для отслеживания таймаутов в очередях с гарантией обработки.

Модуль сетевого взаимодействия отвечает за реализацию взаимодействия с клиентами по сети, то есть непосредственно реализует функциональность протокола PIV. Он представляет собой пул потоков, в каждом из которых есть свой объект-диспетчер событий `boost::asio::io_service`, метод `run()` которого выполняется в этом же потоке.

Для возможности обработки нескольких клиентских соединений одним потоком применяются неблокирующие сокеты, асинхронные операции и зависимые от операционной системы технологии мультиплексирования, которые реализуются внутри библиотеки Boost.ASIO. Каждому соединению с клиентом в момент его создания назначается один из объектов `io_service`, который будет обслуживать все асинхронные операции в этом соединении. Выбор такой архитектуры вызван тем, что большинство операционных систем не поддерживают параллельное ожидание событий на одном наборе сокетов из нескольких потоков. Из-за этого клиентские сокеты принудительно распределяются по нескольким потокам, что позволяет снизить накладные расходы на блокировки, которые в условиях небольшого времени работы обработчиков событий становятся значительными.

4.2 Клиентская библиотека

Клиентская библиотека предназначена для упрощения написания прикладных приложений, так как снимает с программиста необходимость написания собственной реализации протокола PIV. Клиентская библиотека `libscimqclient` предоставляет прикладному программисту синхронный API для взаимодействия с сервером очередей и скрывает все нюансы реализации протокола PIV и работы с сетью. API клиентской библиотеки показан на рис. 7.

```
namespace SciMQ { namespace Client {
class LIBSCIMQCLIENT_API ProtocolException: public std::exception
{
public:
  ProtocolException(int status) throw();
  virtual ~ProtocolException() throw();
  scimq_protocol_status_t status() const throw();
  virtual const char* what() const throw();
};

class LIBSCIMQCLIENT_API Connection: private boost::noncopyable
{
public:
  Connection();
  Connection(
    const SciMQ::Network::EndpointType& endpoint,
    const SciMQ::Network::EndpointOptionType& endpoint_options =
SciMQ::Network::EndpointOptionType()
  );
  ~Connection();
```

```
  void connect(
    const SciMQ::Network::EndpointType& endpoint,
    const SciMQ::Network::EndpointOptionType& endpoint_options =
SciMQ::Network::EndpointOptionType()
  );
  void close();
  // Запрос данных для обработки.
  // Если возвращенный указатель равен NULL, то данных больше нет.
  SciMQ::DataQueue::MessageSharedPtr get_data(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool& is_data_ended
  );
  // Отправка результатов на сервер.
  void post_data(
    const SciMQ::DataQueue::MessageKeyListType& queue_list,
    SciMQ::DataQueue::MessageSharedPtr& message,
    bool is_push = false,
    const SciMQ::DataQueue::MessageKeyType& confirm_queue =
SciMQ::DataQueue::MessageKeyType(),
    const SciMQ::DataQueue::MessageUuidType& confirm_uuid =
SciMQ::DataQueue::make_message_nil_uuid()
  );
  // Отправка подтверждения на сервер.
  void post_confirm(
      const SciMQ::DataQueue::MessageKeyType& confirm_queue,
      const SciMQ::DataQueue::MessageUuidType& confirm_uuid,
      bool is_push = false
  );
  // Управление очередями
  void create_queue(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool is_reliable   = false,
    bool is_persistent = false,
    const boost::chrono::seconds& reliable_timeout =
SciMQ::DataQueue::MESSAGE_RELIABLE_TIMEOUT
  );
  void delete_queue(
    const SciMQ::DataQueue::MessageKeyType& queue
  );
  bool get_queue_info(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool *is_reliable = NULL,
    bool *is_persistent = NULL,
    bool *is_eod = NULL,
    boost::chrono::seconds *reliable_timeout = NULL
  );
  bool exist_queue(const SciMQ::DataQueue::MessageKeyType& queue);
  void set_queue_eod(const SciMQ::DataQueue::MessageKeyType& queue, bool
eod);
  bool get_queue_eod(const SciMQ::DataQueue::MessageKeyType& queue);
};
} }
```

Рис. 7. API клиентской библиотеки libscimqclient

4.3 Управляющее программное обеспечение

Управляющее ПО предназначено для управления очередями. Дополнительно поддерживается возможность генерации тестового потока данных, загрузка данных с опциональной под-

держкой сохранения данных в файлах, а также загрузка данных из одной очереди и помещение их в другую очередь.

4.4 Программное обеспечение для PIV-расчетов

Данный компонент разработанного ПО состоит из двух частей:

- клиент загрузки исходных данных и сохранения результатов расчетов,
- клиент, упрощающий написание расчетных алгоритмов для суперкомпьютера.

Приложение ввода-вывода предназначено для загрузки исходных данных из формата хранения ActualFlow и для сохранения получаемых результатов в файлах на дисках ЭУ. Поддерживаются два режима загрузки исходных файлов. В первом случае, будут рекурсивно прочитаны все заданные каталоги, и обнаруженные файлы данных будут переданы на сервер, после чего приложение завершается. Во втором случае, помимо обнаружения всех существующих данных в заданном каталоге, будет включен мониторинг появления новых файлов данных и каталогов, которые будут отправляться на сервер по мере их появления. В этом случае, для завершения работы требуется подать специальную команду в консольном окне приложения. Для завершения наблюдения с продолжением передачи уже обнаруженных данных на сервер нужно нажать комбинацию клавиш `Ctrl+C` один раз. Для завершения работы без передачи уже обнаруженных данных на сервер необходимо нажать комбинацию клавиш `Ctrl+C` два раза.

Расчетный клиент предназначен для упрощения написания расчетных приложений на стороне суперкомпьютера. Он берет на себя функции загрузки параметров из конфигурационного файла, все взаимодействие с сервером по сети, позволяет выделять для обработки одного блока данных несколько вычислительных узлов, объединенных в MPI-группу.

Реализация прикладного алгоритма сводится к написанию класса с заданным API, который будет обрабатывать данные, полученные от приложения и передавать приложению результат расчетов.

В случае использования режима, когда один блок данных обрабатывается несколькими MPI-процессами, один из них назначается ведущим и отвечает за взаимодействие с сервером, при этом, когда данные получены, то обработкой могут заниматься все процессы группы, включая ведущий.

Разработанная программная архитектура направлена на предоставление простого API для прикладных программистов, минимизацию нагрузки на процессор ЭУ и уменьшение требований к сетевому стеку операционной системы ЭУ путем выделения компоненты по диспетчеризации и передаче на большие расстояния потока данных в отдельный слой с возможностью его переноса на отдельный сервер.

Следующим этапом планируется проанализировать эффективность загрузки интерконнекта и сети ввода-вывода суперкомпьютера и проработать двухуровневый вариант передачи данных, когда в вычислительном поле выделяются некоторые узлы, которые занимаются только обменом данными с сервером и передают данные другим вычислительным узлам с использованием сети интерконнекта.

5. Оценка эффективности разработанного решения

Оценка эффективности проводилась путем измерения эффективной пропускной способности системы при передаче данных от экспериментальной установки через сервер очередей на площадке ИМСС УрО РАН (Пермь) на вычислительные узлы удаленного суперкомпьютера URAN ИММ УрО РАН (Екатеринбург). Коннективность связки между городами осуществлялась по выделенному vlan по DWDM магистрали «GIGA UrB RAS» с доступной пропускной способностью 1 Гбит/с и временем RTT=5,5 мс.

Исследовалось поведение системы при передаче измерений, состоящих из трех блоков с длинами 128, 1048576, 1048576 байт, соответственно. Измерения поступали от эмулятора ЭУ с промежутками между отправками данных, равными 20мс, 50мс и 100мс. Это приближенно соответствует нескольким режимам работы реальной ЭУ. При этом при минимальном интервале

скорость генерации составляет порядка 750Мбит/с, что меньше предельной пропускной способности канала связи и поэтому позволяет работать в реальном времени без бесконечного роста размера очереди исходных данных. Процесс на вычислительных узлах рассчитывал контрольные суммы блоков данных по алгоритму CRC32 и передавал их обратно на эмулятор ЭУ.

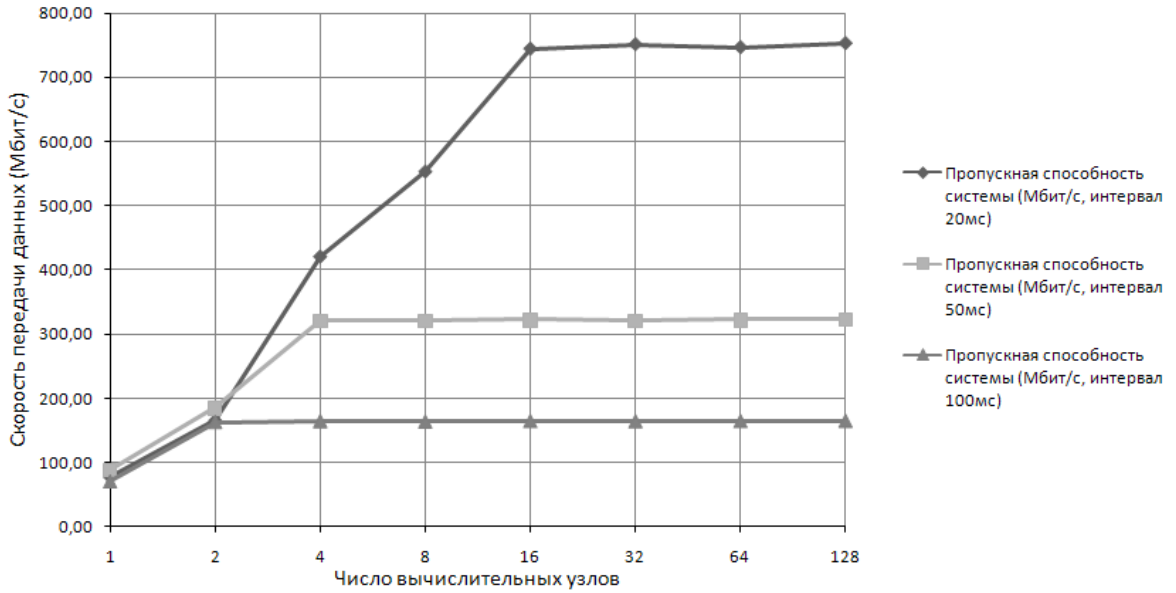


Рис. 8. Зависимости пропускной способности системы от числа задействованных вычислительных узлов

На рис. 8 показаны графики зависимости пропускной способности системы (Мбит/с) от числа задействованных вычислительных узлов. Для наглядности горизонтальная ось «число вычислительных узлов» приведена в логарифмическом масштабе.

Наклонные участки показывают рост пропускной способности системы с добавлением новых узлов, когда меньшее количество узлов не справляются с обработкой и небольшое число параллельных соединений не полностью утилизируют канал связи. При работе в этой области графика очередь растет, так как вычислительные узлы не успевают обрабатывать все данные в реальном времени.

Горизонтальные участки в правой части графика соответствуют установившемуся режиму, в котором скорость обработки данных совпадает со скоростью генерации данных. В этом случае увеличение числа задействованных вычислительных узлов не влияет на пропускную способность системы, но увеличивает ее надежность. Рост надежности обусловлен тем, что в случае отключения или потери сетевой связности части вычислительных узлов, а также в случае, если время обработки одного измерения по каким-либо причинам возрастет, то система все равно может остаться в рамках горизонтального участка графика и не допустить снижение пропускной способности ниже скорости генерации данных. Необходимо отметить, что разработанная архитектура позволяет, при необходимости, увеличивать число задействованных вычислительных узлов непосредственно в процессе работы системы, что предоставляет возможность оперативно реагировать на изменения условий проведения расчета.

6. Заключение

Предложена программная архитектура системы передачи интенсивного потока данных в распределенных системах. Сформулированы ограничения на класс прикладных задач, для которых возможно применение разработанного подхода. Спроектирован протокол, алгоритм диспетчеризации данных и разработано программное обеспечения для передачи данных с экспериментальной установки на узлы вычислительного кластера, тестирование которых подтверждает работоспособность предложенной программной архитектуры системы передачи интенсивного потока данных в распределенных системах.

Оценка эффективности показала увеличение пропускной способности системы при использовании разработанной технологии по сравнению с классическими подходами [2] и, как след-

ствие, уменьшение времени обработки массивов исходных данных. Показан возможный путь снижения нагрузки на процессор ЭУ путем переноса задачи диспетчеризации на близкорасположенный выделенный сервер. Помимо этого, использование выделенного сервера позволяет проводить сборку набора данных из нескольких несвязанных между собой источников, например, при наличии нескольких независимых групп датчиков, допускающих раздельную обработку данных с них.

Разработанная технология предоставляет принципиально новый инструмент проведения экспериментальных исследований, позволяя обрабатывать быстропротекающие процессы в течение длительного времени, например, при лабораторном изучении начальной стадии формирования тропических циклонов.

Дальнейшие исследования будут направлены на детальную разработку методики оценки необходимого вычислительного ресурса, исходя из параметров эксперимента, расчетного алгоритма и пропускной способности сети, а также на совершенствование программного обеспечения для оптимального использования ресурсов внутренних сетей суперкомпьютера, таких, как сеть ввода вывода и сеть передачи сообщений MPI.

Автор выражает благодарность научному руководителю исследования Григорию Федоровичу Масичу (зав. лабораторией телекоммуникационных и информационных систем ИМСС УрО РАН) за помощь в проведении исследований.

Литература

1. Степанов Р.А., Масич А.Г., Сухановский А.Н., Щапов В.А., Игумнов А.С., Масич Г.Ф. Обработка на супервычислителе потока экспериментальных данных // Вестник УГАТУ, Уфа, 2012. Т. 16, № 3 (48). С. 126-133.
2. Р.А. Степанов, А.Г. Масич, Г.Ф. Масич «Инициативный проект «Распределенный PIV»» // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: труды Всероссийской суперкомпьютерной конференции – М.: Изд-во МГУ, 2009. – С. 360–363. (ISBN 978-5-211-05697-8).
3. А.Г. Масич, Г.Ф. Масич «Инициатива GIGA UrB RAS» // Совместный вып. Журнала «Вычислительные технологии» и журн. «Вестник КазНУ им. Аль-Фараби». Сер. «Математика, механика, информатика», №3 (58). По материалам Междунар. конф. «Вычислительные и информационные технологии в науке, технике и образовании», - Казахстан, Алматы.-2008.- Т.13.- Ч. II. -С. 413-418 (ISSN 1560-7534).
4. А.Г. Масич, Г.Ф. Масич. GIGA UrB RAS подход к LambdaGrid парадигмам вычислений // 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. 4-11.
5. А.Г. Масич, Г.Ф. Масич, Р.А. Степанов, В.А. Щапов Скоростной I/O-канал супервычислителя и протокол обмена интенсивным потоком экспериментальных данных // Материалы X международной конференции «Высокопроизводительные параллельные вычисления на кластерных системах HPC-2010» – Пермь: Изд-во ПГТУ, 2010. - Т. 2. С. 119–128. (ISBN 978-5-398-00506-6).
6. Щапов В.А., Масич А.Г., Масич Г.Ф. Модель потоковой обработки экспериментальных данных в распределенных системах // Вычислительные методы и программирование. 2012. Раздел 2. 139-145 (http://num-meth.srcc.msu.ru/).
7. Yunhong Gu and Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.
8. Vladislav Shchapov, Alexey Masich. Protocol of High Speed Data Transfer from Particle Image Velocimetry System to Supercomputer // Proc. of The 7th International Forum on Strategic Technology (IFOST 2012) September 18-21, 2012, Vol.1. National Research Tomsk Polytechnic University, Tomsk, P. 653-657

Parallel computing of temperature fields on layered finite-element mesh

Olga Ogorodnikova[1], Konrad Weiss[2]

Ural Federal University[1], RWP[2]

Computer aided simulation of foundry technologies is an important part of product life-cycle engineering and digital production. WinCast is the software for simulation of cast metal solidification and has been developed by RWP GmbH (RWTH Aachen University), Germany. It's possible to use it in the next generation of automated technological equipment for process control in the real-time operation mode. The results of instantaneous simulation could be used to adjust the technological parameters in accordance with current sensor readings and predicted metal defects. In this connection, the acceleration of temperature computing and metal quality simulation is an actual problem to be solved. Improving the parallel algorithms and using the parallel computing systems can effectively accelerate the simulation.

TFB module of WinCast program is solving transient heat conductivity equation by finite-element (FE) method to calculate temperature distribution in the casting and in the mould during solidification. For this purpose the WinCast preprocessor creates the layered finite-element mesh following the strategy of breaking down (slicing) 3-dimensional STL geometry into several 2D/3D layers by horizontal cutting planes. In WinCast finite element is a pentahedral prism with changeable angles which could be variated to approximate any relief surfaces by element faces. Acceptable variations of the prism angles could be achieved by changing X- and Y-coordinates of nodes in cutting plane and Z-coordinates within the layer. The number of nodes is a constant for the plane and could be changed simultaneously for all the planes of the mesh. The number of prisms has the same value in each layer. Dimensions of prisms in Z-direction usually coinciding with the direction of gravity could differ.

Iterative solution of equation system proceeds using Gauss-Seidel method or PCG. Gauss-Seidel procedure is not proper method for parallel system since it uses the intermediate results of two consecutive steps. On the contrary, solver PCG (+ MPI) can reduce the computation time by 50% using four cores. Increased productivity is also achieved with the automatic generation of layered mesh from the contours of the cross section in each layer. Layered mesh is a precise presentation of cast object for simulation of melt behavior and also, it's suitable for parallel processing.

Finally, the total time required to obtain the calculated temperature field and solidification defects can be reduced to some minutes. The achieved acceleration of temperature computation permits to embed WinCast module into automatic control system equipment. The objective of simulation module is to get operational forecasts of the expected quality for control unit making a decision to change the process parameters.

Численное моделирование реакции циклоалюминирования олефинов и ацетиленов на основе кинетического эксперимента

Л.Р. Абзалилова, И.М. Губайдулин, Ю.С. Лаврентьева

Институт нефтехимии и катализа РАН

Численное моделирование реакции циклоалюминирования олефинов и ацетиленов [1] заключается в нахождении кинетических констант, вычислении концентраций веществ, а также определении скоростей стадий. Корректным описанием кинетики реакции является система дифференциальных уравнений, учитывающая изменение числа молей, проходящее в ходе реакции: $\frac{dN}{d\tau} = \sum \nu_{ij} w_j = F_N$; $\frac{dX_i}{d\tau} = \frac{F_i - X_i F_N}{N}$

где X_i – концентрации (мольные доли) веществ, участвующих в реакции; N - относительное изменение числа молей реакционной среды; ν_{ij}–элементы стехиометрической матрицы; w_j – скорость j-ой стадии, 1/с; k^+_j и k^-_j – приведенные константы скорости прямой и обратной реакции, соответственно, 1/с. Константы k^+_j и k^-_j находятся в соответствии с законом действующих масс.

При решении прямой и обратной задач определения кинетических параметров реакции циклоалюминирования олефинов и ацетиленов распараллеливание вычислительного процесса быть осуществлено на четырех уровнях [2]:

1) распараллеливание численного метода решения прямой задачи;
2) использование внутреннего параллелизма задачи;
3) распараллеливание по экспериментальной базе;
4) 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 начальным концентрациям при одинаковых температурах.

Распараллеливание вычислительного процесса по экспериментальной базе осуществляется по принципу процессорной фермы (или стратегии «главный – подчиненные»).

Рассматриваемый процесс также обладает внутренним параллелизмом, который заключается в том, что общая схема реакции включает 9 протекающих параллельно стадий. 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). При этом область изменения кинетических параметров задается априорно.

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 технологией многопоточного программирования OpenMP.

Литература

1. Рамазанов И.Р., Кадикова Р.Н., Джемилев У.М. Реакция циклоалюминирования функционально-замещенных олефинов и ацетиленов // International symposium on advance science in organic chemistry. – Miskhor, Crimea. – 2010. – С.323

2. Линд Ю.Б., Губайдуллин И.М., Мулюков Р.А. Методология параллельных вычислений для решения задач химической кинетики и буровой технологии // «Системы управления и информационные технологии». – № 2(36), 2009. – С. 44-50.

Модель памяти в технологии графосимволического программирования

П.В. Аболмасов, В.В. Жидченко

Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет)

На кафедре программных систем Самарского государственного аэрокосмического университета разработана и развивается технология графосимволического программирования (технология ГСП), предназначенная для визуального проектирования и разработки параллельных программ. Технология использует графическую форму представления программ, семантически близкую к блок-схемам, дополненную механизмами описания параллелизма и синхронизации. Пример представления программы в технологии ГСП приведен на рисунке 1.

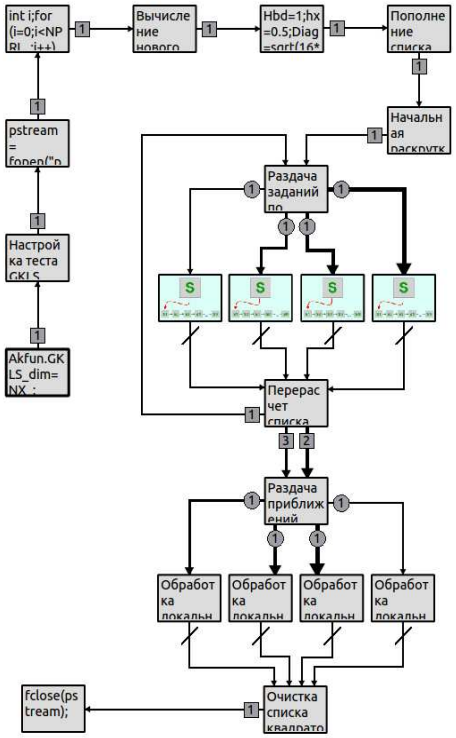


Рис 1. Граф-модель параллельного алгоритма нахождения глобального экстремума

Вершинами графа служат подпрограммы на языке С или другие графы. Направленные дуги графа определяют передачу управления между вершинами. По графическому описанию программы специализированный компилятор технологии ГСП строит исходный текст на языке С. Исполняемый файл формируется из этого исходного текста обычным компилятором С с использованием библиотеки MPI.

Изначально технология создавалась для разработки последовательных программ, поэтому она ориентирована на модель общей памяти. Для исполнения разрабатываемых программ в распределенных системах необходимо расширение модели памяти, используемой в технологии ГСП.

Все переменные программы разделяются на два подмножества: глобальные и локальные. Глобальные хранятся и изменяются единственным процессом – диспетчером данных. Для локальных переменных в каждом использующем их процессе создается отдельная копия переменной. Обмен значениями локальных переменных между процессами осуществляется с помощью сообщений. Сообщения изображаются графически в виде направленной дуги от вершины – источника сообщения, в вершину – получатель сообщения. Дуга помечается именами передаваемых в сообщении переменных.

Компилятор технологии ГСП генерирует исходный текст программы, в котором все переменные инкапсулируются объектами класса TPOData. Доступ к переменным предоставляется через поля-свойства. Свойство - это способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Обращение к свойству выглядит так же, как и обращение к полю объекта, но в действительности оно реализовано через вызов функции. При попытке задать значение свойства вызывается один метод (setter), а при попытке получить - другой (getter). Объекты класса TPOData создаются в каждом процессе параллельной программы. Поля-свойства для локальных переменных, а также для глобальных переменных в диспетчере данных обращаются к переменным по указателям. При доступе к глобальным переменным из остальных процессов поля-свойства работают с переменными, формируя диалог с диспетчером данных с помощью сообщений MPI.

Класс TPOData напрямую недоступен для пользователя технологии ГСП. Пользователь работает непосредственно с переменными программы и не обязан знать об их реализации.

Текущие возможности и перспективы развития кластерного планировщика CSched

Р.А. Ахметшин, Е.А. Ермолаев, А.Л. Штангеев, А.В. Юлдашев

Уфимский государственный авиационный технический университет

Эффективная эксплуатация суперкомпьютерной техники невозможна без использования развитого системного программного обеспечения, предназначенного для предоставления интерфейса взаимодействия с суперкомпьютером, управления, планирования, мониторинга выполнения вычислительных задач и т.д. В Уфимском государственном авиационном техническом университете (УГАТУ) для этого используется как стороннее системное программное обеспечение, так и собственные разработки. К последним, в частности, относятся «Программный комплекс автоматизированных расчетов на кластерных системах CCS», а также «Кластерный планировщик задач CSched».

CSched предназначен для планирования выполнения задач на кластерных вычислительных системах в связке с системой очередей TORQUE. Планировщик поддерживает учет различных ресурсов вычислительных узлов (вычислительных ядер, оперативной памяти и т.д.), а также плавающих ресурсов, например, сетевых лицензий на программное обеспечение, что позволяет использовать его при проведении расчетов средствами коммерческих пакетов компьютерного моделирования, в том числе ANSYS, SIMULIA Abaqus, Schlumberger Eclipse, Roxar Tempest More и т.п. Поддерживается настройка политик планирования для различных классов задач, в качестве которых могут выступать упомянутые выше пакеты.

Одной из основных целей разработки собственного планировщика являлось повышение производительности вычислений за счет использования алгоритмов планирования, позволяющих минимизировать задержки при конкурентном доступе к общим ресурсам кластерной системы: оперативной памяти и коммуникационной среде [1]. Необходимо отметить, что в известных планировщиках подобные алгоритмы не реализованы. Единственной системой, в которой произведена попытка учесть влияние конкуренции при доступе к оперативной памяти, является Platform LSF, где имеется поддержка требований задач к пропускной способности памяти в виде ресурсного запроса.

Наш подход предполагает выполнение автоматической априорной оценки характеристик поступающих задач, мониторинг загрузки общих ресурсов многоядерных узлов в динамике работы кластерной системы и накопление статистики использования общих ресурсов, а также возможность оценки влияния конкуренции за общие ресурсы на время выполнения задач для более эффективного планирования, что в альтернативных планировщиках не поддерживается. В то же время проведенные экспериментальные исследования на суперкомпьютере УГАТУ подтверждают, что планирование с учетом конкуренции при доступе к оперативной памяти многоядерных узлов позволяет получить ускорение при решении реальных задач около 1,5 раз.

В настоящее время ведутся работы по созданию специализированного модуля для планировщика CSched, который позволит проводить планирование с учетом конкуренции за общие ресурсы многоядерных узлов. Кроме этого ведется реализация алгоритмов справедливого распределения ресурсов между пользователями кластерной системы (fairshare [2]), что позволит обеспечить высокий уровень доступности кластерной системы для всех ее пользователей.

Литература

1. Юлдашев А.В. Минимизация времени выполнения MPI-программ с учетом конкуренции за каналы передачи данных коммуникационной среды кластерной системы // Вестник УГАТУ, 2011. – Т.15. – №2 (42). – С. 99-105.
2. Jackson D., Snell Q., Clement M. Core Algorithms of the Maui Scheduler. URL: http://www.eecs.harvard.edu/~chaki/bib/papers/jackson01maui.pdf (дата обращения: 02.12.2012).

Технология параллельного программирования RiDE

М.О. Бахтерев, П.А. Васёв

Институт Математики и Механики УрО РАН

RiDE это технология программирования в параллельных распределенных средах на основе модели потока данных (dataflow, [1]). RiDE основана на анализе различных, в том числе и собственных, моделей потока данных [2]. Цель – упростить процесс создания параллельных программ, и сделать это не в ущерб эффективности исполнения вычислительных кодов.

Технология RiDE базируется на понятиях хранилища, задач и правил. Хранилище содержит в себе именованные данные, по отношению к которым доступны три операции – запись (создание), чтение и удаление (возможно в автоматическом режиме с распределенной сборкой мусора). Хранимые данные есть единицы информации с уникальными именами. Задачи выполняют программы, считывающие данные с определенными именами из хранилища, обрабатывают их и формирует новые данные, которые записываются в хранилище. Правила описывают взаимосвязи между задачами и содержимым хранилища, определяя тем самым поток данных параллельного вычисления. Более подробно: http://www.ridehq.net.

Описание вычислительных приложений в предложенных терминах представляется авторам более простой задачей, чем разработка и реализация параллельных схем работы в более традиционных терминах моделей MPI и OpenMP. Действительно, программисту необходимо, по сути, описать вычислительные процедуры, и зависимости между ними. И такое описание достаточно для автоматического формирования эффективного процесса параллельного исполнения программы в режиме совмещения счета и обменов (что является преимуществом dataflow).

Предлагаемая технология в перспективе позволит относительно просто реализовать проведение вычислительного эксперимента на гибридных архитектурах с динамическим изменением количества вычислительных узлов во время самого счета (что актуально при больших вероятностях сбоев на машинах экзафлопного класса), работу в существенно неоднородных коммуникационных средах, автоматическое создание контрольных точек, приостановку и продолжение вычисления прозрачным для программиста образом, использование распределенные хранилищ данных, а также обеспечивает ряд других преимуществ.

Авторы выражают надежду, что результатом развития системы RiDE станет повышение эффективности труда программистов, разрабатывающих приложения для современных неоднородных высокопроизводительных систем. На данный момент для технологии RiDE разработана методика [3], и силами компании LineAct ведется ее реализация [4].

Литература

1. Dennis J. Data Flow Supercomputers // Computer, 1980, Vol.13, No.11, P.48-56.
2. Бахтерев М.О., Описание параллельных вычислений при помощи замыканий // Тезисы 10-го Международного семинара "Супервычисления и Математическое моделирование", РФЯЦ-ВНИИЭФ, Саров, с. 31-32, 2008.
3. М.О. Бахтерев, П.А. Васёв, А.Ю. Казанцев, И.А. Альбрехт, Методика распределенных вычислений RiDE // Параллельные вычислительные технологии (ПаВТ’2011): труды международной научной конференции (Москва, 28 марта – 1 апреля 2011 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2011, с. 418–426.
4. M. Bakhterev, A. Kazantzev, P. Vasev, I. Albrekht, Dataflow-Based Distributed Computing System // Proceedings of the Euromicro PDP 2011 Work in Progress Session (Eds. E. Grosspietsch, K. Kloeckner) p.6-7, SEAA-Publications No. SEA-SR-29 Johannes Kepler University Linz (Austria), ISBN 978-3-902457-29-5.

Численное моделирование турбулентных течений с помощью RANS/ILES-методов высокого разрешения в авиационных приложениях*

Л.А. Бендерский, Д.А. Любимов, А.Ю. Макаров, И.В. Потехина, А.Э. Федоренко

ЦИАМ им. П.И. Баранова, г. Москва.

С помощью комбинированного RANS/ILES-методов высокого разрешения [1] были проведены мультивариантные расчеты сложных турбулентных течений, которые важны для авиационных приложений. Были выполнены совместные расчеты течения в соплах различной конфигурации и их до- и сверхзвуковых струях, расчет натекания сверхзвуковой струи на стенку, влияние элементов планера на течение в струе из двухконтурного сопла ТРД. Исследовано активное управление отрывным течением в диффузоре с помощью синтетических струй, обтекание открытой полости. Высокое разрешение представленного метода позволяет на сетках 0.6-3.3×10^6 ячеек рассчитывать перечисленные прикладные задачи. Это позволяет проводить расчет задачи на одном узле кластера в сравнительно небольшое время. Появляется возможность проводить параметрические расчеты на кластере, что важно для инженерных приложений. Высокое разрешение метода достигается использованием монотонной схемы 9го порядка аппроксимации для расчета конвективных потоков уравнений Навье-Стокса на гранях ячеек.

При расчетах для дозвуковой струи получены акустические характеристики. Для угла наблюдения 90° в 1/3 октавном спектре шума струи, расчет хорошо совпадает с экспериментом для Sh=0.05-3. Для нерасчетных холодных и горячих сверхзвуковых струй наблюдается удовлетворительное соответствие по распределению осредненного статического давления на оси струи с экспериментом. Рассчитано натекание сверхзвуковой нерасчетной холодной струи на стенку, при расстоянии сопла от стенки h/d=2.08, 2.8 и 4.16. Определены пульсации давления вдоль стенки. Перечисленные расчеты выполнялись на сетках с $1.3\text{-}1.6\times10^6$ ячеек. Влияние компоновки (пилон и крыло с закрылками) на течение в струе из сопла двухконтурного ТРД было исследовано на сетке 3.3×10^6 ячеек. Обнаружено, что взаимодействие концевых вихрей от закрылков со струей приводит к сильному изменению течения в ней: изменяется форма струи, возрастает уровень энергии турбулентности. Рассмотрено несколько вариантов геометрии закрылков.

На сетке 0.5×10^6 ячеек выполнены расчеты по исследованию активного управления отрывными течениями в диффузорах с помощью синтетических струй, выдуваемых со стенки диффузора, на характеристики турбулентного отрывного течения в нем. Показано, что с помощью соответствующего выбора геометрических и газодинамических параметров синтетических струй, уровень пульсаций статического давления и скорости на выходе из диффузора может быть снижен в 1.3–1.5 раза.

Исследована структура и характеристики течения в открытой полости (каверны) в присутствии внешнего потока над ней с числом Маха M=0.9 на сетке с 0.7×10^6 ячеек. Получены картины течения в полости, а также мгновенные и осредненные параметры и турбулентности в каверне и в области над ней.

Для задачи истечения сверхзвуковой струи из биконического сопла был выполнен анализ производительности расчета данным методом на узлах с разной архитектурой. Получено, что разница в ускорении расчетов в зависимости от архитектуры может достирать 2.5 раз.

Литература

1. Любимов Д.А. Разработка и применение метода высокого разрешения для расчета струйных течений методом моделирования крупных вихрей // Теплофизика высоких температур. 2012. Т. 50, № 3. С. 450.

* Работа выполнена при поддержке гранта РФФИ №12-08-00951-а.

Применение многоядерных сопроцессоров в параллельных системах баз данных*

К.Ю. Беседин, П.С. Костенецкий

Южно-Уральский государственный университет

По мере совершенствования графических ускорителей все больший интерес вызывает использование их вычислительных возможностей в областях, отличных от компьютерной графики. Базы данных являются одной из областей, где могут быть успешно применены GPU [1]. На данный момент опубликован ряд работ, описывающих эффективную реализацию специфичных для СУБД алгоритмов с использованием GPU [3]. Некоторое число работ посвящено поиску архитектурных решений, позволяющих эффективно использовать аппаратные особенности GPU, например [4,5]. Еще одной перспективной многоядерной архитектурой является Intel MIC, недавно представленная широкой публике. Уже существует несколько работ, посвященных использованию MIC в базах данных. Так, в работе [2], помимо CPU и GPU подробно рассматривается реализация предложенного алгоритма поиска в дереве индекса на Intel MIC.

В данной работе описывается незаконченное исследование, посвященное использованию многоядерных сопроцессоров и графических ускорителей для обработки запросов в параллельных системах управления базами данных. На данный момент разработан эмулятор параллельной СУБД, использующий вычислительный кластер для выполнения запросов SELECT и JOIN. Разработаны версии запросов как для CPU, так и для GPU. Алгоритмы для CPU реализованы с использованием технологий MPI и OpenMP. Для GPU алгоритмы реализованы с использованием MPI и Nvidia CUDA. Для достижения высокой степени параллелизма при обработке запроса используется фрагментация отношений по вычислительным узлам. Выполняемые запросы делятся на подзапросы, поровну распределяемые между рабочими процессами (параллельными агентами). Дополнительно разработана реализация алгоритма, в которой для обработки своей порции кортежей входного отношения каждый параллельный агент может использовать как центральный процессор, так и графический ускоритель. Для коммуникации процессов между собой используется интерфейс MPI. Ведутся работы по исследованию эффективности алгоритмов для GPU.

На следующем этапе работы будет выполнена адаптация алгоритмов для сопроцессора Intel Xeon Phi и проведено сравнение их эффективности с версиями для центрального процессора и для графического ускорителя.

Литература

1. Blas A. D., Kaldewey, T. Data Monster // IEEE spectrum 2009. –Vol. 46, No. 9.
2. Kim C., Chhugani J., Satish, N. et al., Designing fast architecture-sensitive tree search on modern multicore/many-core processors, // ACM Trans. Database Syst. 2011 –Vol 36, –No 4., –P 22:1–22:34
3. Kim C., Chhugani J., Satish N., et al. FAST: fast architecture sensitive tree search on modern CPUs and GPUs // ACM SIGMOD International Conference on Managment of data, Indianapolis, USA, June 6–10, 2010, Proceedings. ACM, 2010 –P. 339–350
4. He B., Yu J. X. High-throughput transaction executions on graphics processors // VLDB Endowment, Seattle, Washington, USA, August 29 – September 3, 2011, Proc. VLDB Endowment 2011 –Vol 4, –No 5. –P 314–325
5. He B., Lu M., Yang K., et al. Relation query coprocessing on graphics processors // ACM Trans. Database Syst. 2009 –Vol 34, –No 4. –P 21:1–21:39

*Работа выполнена при поддержке гранта РФФИ № 12-07-31082 (2012–2013 гг.) и гранта Президента РФ № МК-3711.2013.9 (2013–2014 гг.)

Библиотека статистического анализа данных в гетерогенной распределенной среде *

А.К. Богушов, А.А. Морозов

ФГБОУ ВПО "Южно-уральский государственный университет"(НИУ)

Открытые библиотеки научных инструментов для языка программирования Python (Numpy, Scipy) содержат большое количество статистических функций, в то же время, они имеют ряд недостатков: невысокая скорость обработки больших объемов данных, отсутствие поддержки длинной арифметики. Использование Numpy на многопроцессорных, многоядерных или распределенных системах приводит к усложнению программы [1], а работа в гетерогенной среде невозможна.

Применение графических ускорителей в библиотеках статистической обработки "больших данных"позволяет получить значительные преимущества. Например, библиотека gpustats [2] реализует эти преимущества с помощью технологии CUDA и техники метапрограммирования.

В данной работе предложен другой подход к реализации библиотеки статистических функций. В основе библиотеки лежит технология OpenCL, которая позволяет работать приложению в гетерогенной среде с большими объемами данных. Для хранения данных предлагается новая абстракция Resilient Distributed Arrays (RDA), основанная на Resilient Distributed Datasets (RDD) [3], и позволяющая хранить в оперативной памяти распределенные данные, осуществлять контроль разбиения и размещения данных по вычислительным узлам. Создать или изменить RDA можно с помощью крупномодульных операций. Примером таких операций служат функции высших порядков (map, reduce, filter, join, ...), которые применяют одни и те же команды ко всем элементам массива. Многие статистические функции могут быть выражены через эти функции. Данное ограничение дает возможность обеспечить отказоустойчивость через сохранение всей истории применения операций для каждого RDA, которая может быть потом использована для восстановления актуального состояния данных из исходных данных, вместо постоянного сохранения промежуточных состояний системы на жесткий диск в контрольных точках после каждой операции. Это позволяет хранить массивы данных непосредственно в оперативной памяти и обрабатывать данные в интерактивном режиме. RDA в отличие от RDD имеют ряд преимуществ, связанных с акцентом на хранение массивов данных: многомерность, доступ к срезам данных, только числовые типы данные, совместимые с типами Numpy, поддержка длинной арифметики.

Литература

1. Parallel Programming with numpy and scipy:
URL: http://www.scipy.org/ParallelProgramming (дата обращения: 01.12.2012).

2. Andrew Cron, Wes McKinney gpustats: GPU Library for Statistical Computing in Python
URL: http://ftp.stat.duke.edu/WorkingPapers/11-17.pdf (дата обращения: 01.12.2012).

3. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing, NSDI 2012, April 2012.

*Работа выполнена при поддержке гранта РФФИ №12-07-31013

Реализация технологии «Персональный виртуальный компьютер» на базе открытых технологий

К.В. Бородулин, А.П. Свиридов

Южно-Уральский государственный университет

В рамках технологии «Персональный виртуальный компьютер» [1] для каждого студента первого курса создается персональный виртуальный компьютер, доступ к которому осуществляется с домашнего компьютера, ноутбука, нетбука и др. В результате, в качестве компьютерного класса может быть использована любая учебная аудитория, оборудованная Wi-Fi сетью и электрическими розетками. Рассматривается система SpicePVC, которая реализует технологию «Персональный виртуальный компьютер» на базе открытых технологий, состоящая из следующих компонентов. Разрабатываемый сервер Desktop delivery – это менеджер, управляющий выдачей рабочих столов виртуальных машин клиентам. Для отображения (рендеринга) удаленного дисплея используется технология Spice [2], которая позволяет просматривать виртуальный «рабочий стол» вычислительной среды через компьютерную сеть, причем для просмотра можно использовать широкий спектр операционных систем и устройств.

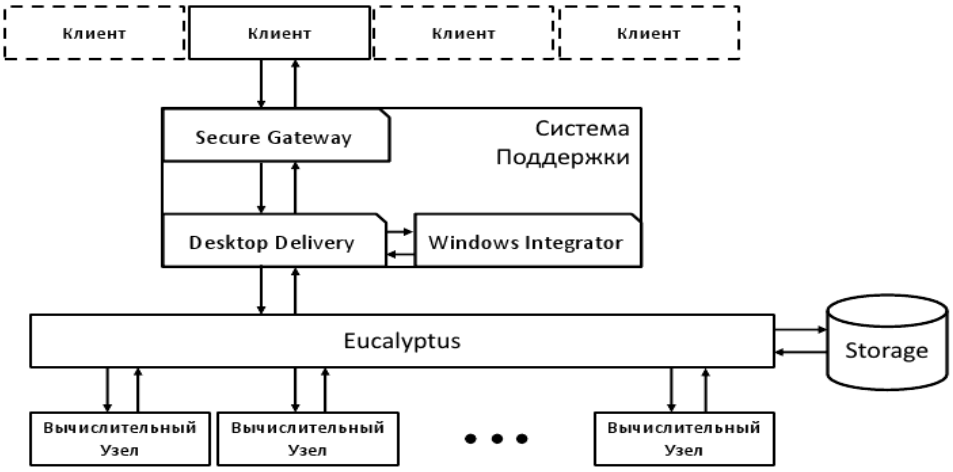


Рис. 1. Компоненты системы SpicePVC

Разрабатываемый сервер Secure gateway – это сервер, который управляет защищенными подключениями к системе SpicePVC, и осуществляет передачу данных по протоколу Spice через сеть Internet. Клиент – программа, служащая для работы с персональным виртуальным компьютером на устройстве пользователя системы.

Можно выделить следующие преимущества предлагаемого подхода:

1. Представленная система является системой с открытым исходным кодом, доступной под лицензией GPLv3.

2. Возможность масштабирования данной системы для обеспечения одновременной работы 2500 пользователей.

3. Возможность работы с персональным виртуальным компьютером через любой современный обозреватель без установки дополнительного программного обеспечения.

Литература

1. Костенецкий П.С., Семёнов А.И., Соколинский Л.Б. Создание образовательной платформы "Персональный виртуальный компьютер" на базе облачных вычислений // Научный сервис в сети Интернет: экзафлопсное будущее: Труды международной научной конференции (19-24 сентября 2011 г., г. Новороссийск). М.: Изд-во МГУ, 2011. С. 374-377.
2. Spice User Manual. URL: http://spice-space.org/docs/spice_user_manual.pdf (дата обращения: 09.01.2013).

Веб-система визуализации, анализа и мониторинга работы программ

П.А. Васёв[1], М.С. Согомонян[2]

[1]Институт Математики и Механики УрО РАН, г. Екатеринбург
[2]Уральский Федеральный Университет, г. Екатеринбург

Визуализация процесса и параметров работы программ представляет известный интерес для разработчиков этих программ. В научном плане эти вопросы изучает область «визуализация программного обеспечения» [1], которая особенно активно развивается на западе [2]. Настоящая работа описывает наш начальный опыт разработки в данной области.

Система предназначена для визуализации, анализа и мониторинга работы программных комплексов, включая и параллельные программы. В основе системы лежит возможность:

- программным путем наполнять себя информацией;
- графически отображать накопленную информацию.

Накопление информации происходит путем передачи HTTP-запроса к серверной части системы. Каждый такой запрос формирует «событие». Событие имеет имя, числовое значение, комментарий, дату и время, а также набор произвольных атрибутов. Накопление событий в сериях и их последующее изучение как раз и является сутью работы с системой.

Программист, используя протокол HTTP или специальные поставляемые с системой библиотеки, внедряет в исходный код своих программ функции для формирования событий. Например, это может быть событие «пользователь запустил программу», «пользователь использовал функционал X», «пользователь загрузил файл размером Y», «программа завершена, время работы составило Z минут», и так далее. Таким образом, программист сам указывает, какие элементы логики работы программы его интересуют.

Просмотр событий осуществляется с помощью веб-интерфейса. Система позволяет:

- Посмотреть перечень различных имен событий (серий), сформированных в результате накопления информации.
- Посмотреть график, сформированный событиями какой-либо серии. При этом допускается возможность агрегации данных, например по часам, дням, с выбором максимального значения события, среднего, и так далее.
- Экспортировать данные в графическом или текстовом виде.

В ходе эксплуатации системы было обнаружено, что интерес представляет не только накопление и визуализация, но и автоматическая оценка происходящих событий.

Для этого система научена оценивать вновь приходящие события с заданным именем, проверяя их значения на соответствие заданным условиям. Своевременная сигнализация и визуализация различных ситуаций не раз помогли авторам, позволяя оперативно оценивать ситуацию или выявлять сбои в работе ряда программ.

Система доступна с открытыми исходными кодами. Информация о разработке публикуется в сети Интернет по адресу: www.lineact.com/evented.

Литература

1. Авербух В.Л., Байдалин А.Ю., Разработка средств визуализации программного обеспечения параллельных вычислений. Визуальное программирование и визуальная отладка параллельных программ. // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов, 2003, вып. 4., с. 68-80.

2. Martin Beck, Jonas Trumper, Jurgen Dollner. A Visual Analysis and Design Tool for Planning Software Reengineerings // Proceedings of the 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2011, Williamsburg, VA, USA, September 29-30, 2011, pp.~54-61.

Функциональные возможности среды-конструктора систем научной визуализации SharpEye

П.А. Васёв, С.С. Кумков, Е.Ю. Шмаков

Институт Математики и Механики УрО РАН

Существующие системы научной визуализации можно разделить на три группы: универсальные системы (VIZIT, ParaView), системы, специализированные для некоторого класса задач (IVS3D, Venus, VolVis); и системы, специализированные для конкретной задачи. Недостатки первых двух групп – сложность в освоении, неизменность встроенных алгоритмов представления или высокая сложность их модификации.

Идеальными системами визуализации являются системы из третьей группы – разработанные под конкретную исследовательскую задачу или проект. В этом случае они учитывают всю имеющуюся специфику. Однако создание таких систем весьма затратно, и поэтому зачастую применяют универсальные системы, с соответствующей потерей качества работы.

В течение последних двух лет ведётся разработка [1-2] среды визуализации, решающей обозначенную проблему с помощью открытой модульной структуры. Сама среда реализует лишь интерфейсную часть, а также механизм программного доступа и управления сценой. Процедуры загрузки данных и восстановления геометрических образов подключаются в виде внешних модулей – dll-библиотек, ruby-скриптов, а также программ (через потоки stdin/stdout).

В настоящее время предлагаемая программа умеет производить следующие действия (как программно через API, так и вручную через пользовательский интерфейс):

- загружать произвольные файлы данных с помощью внешних модулей;
- группировать объекты статически при добавлении в сцену; возможно несколько иерархий объектов;
- задавать дополнительные свойства объекта и описывать элементы интерфейса, с ними связанные;
- изменять стандартные и дополнительные атрибуты объекта или группы объектов; при работе с группой объектов выводятся для изменения свойства, общие для всех объектов группы;
- вращать, перемещать, масштабировать сцену;
- управлять камерами и источниками света;
- экспортировать текущий вид сцены в виде графического файла, трёхмерного файла в формате OBJ, а также автоматически размещать экспортируемые материалы в сети Интернет.
- вести историю работы, откат и повторное исполнение действий, сохранение списка действий и его загрузка с воспроизведением;

Таким образом, создание новой системы визуализации трансформируется из сложного проекта в написание модуля-загрузчика необходимого формата данных. Система и документация к ней размещаются в сети Интернет по адресу: www.sharpeye.lact.ru.

Литература

1. Васёв П.А, Кумков С.С., Шмаков Е.Ю., О создании среды разработки систем научной визуализации // Труды XIII Международного семинара «Супервычисления и математическое моделирование» (3–7 октября 2011 г.) под редакцией Р.М. Шагалиева. — ИПК ФГУП «РФЯЦ-ВНИИЭФ», г. Саров. С. 131-140.
2. Васёв П.А., Кумков С.С., Шмаков Е.Ю., Конструктор специализированных систем визуализации // Научная визуализация. 2012. Т.4, №2. С. 64--77.

Распараллеливание клеточно автоматной модели ионной имплантации в 4-х ядерной системе

Р.Р. Вильданов, С.В. Коробов, И.В. Матюшкин

ОАО «НИИ молекулярной электроники»

В своей работе мы рассматриваем возможность расспараллелить ранее нами описанную ионно-пролетную модель [1] с целью сокращения времени моделирования. Наша модель строится с помощью синхронных клеточных автоматов. Клеточный автомат (КА) является фундаментальной абстракцией параллельных вычислений, поэтому неудивительно, что математические модели, сформулированные в терминах КА, распараллеливаются наиболее эффективно.

Ионно-пролетная модель была ранее реализована в авторской разработке – программе SoftCAM. Все тестирование производились на ПК следующей конфигурации: процессор Intel Core i7 3770, модуль памяти Corsair XMS CMX16GX3M2A1600C11 DDR3 2x 8Гб, материнская плата GIGABYTE GA-Z77-DS3H LGA 1155.

Мы создали три варианта архитектуры программы, реализующие ионно-пролетную модель:

1. Для каждой ячейки выделяется свой поток;
2. Для каждого столбца КА, выделяется свой поток;
3. Все поле КА разбивается на несколько равномощных подмножеств (по количеству потоков), а каждому потоку ставится во взаимооднозначное соответствие одно из них.

Выше описанные архитектуры тестировались с помощью библеотек: Boost и Qt4Thread. В ходе исследования выяснилось, что применение 1-ой архитектуры привело к снижению производительности в 6 раз в случае использования Qt4Thread и в 2.5 при использовании Boost. Наиболее перспективной оказалась 3-я архитектура,которая дала максимальное ускорение в 3.4-3.6 раза, при этом результат не менялся от использования различных библиотек. Таким образом, можно сделать вывод о том, что при распараллеливании модели имплантации двумя независимыми способами (с использованием библиотеки Boost или Qt4Thread) для 4-х ядерной системы достигается выигрыш в 3.5 раза по времени выполнения программы.

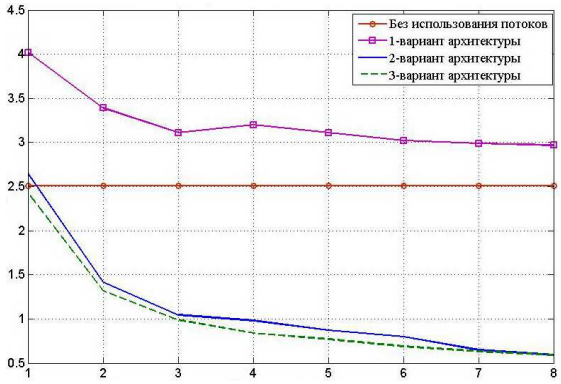


Рис. 1. Зависимость времени выполнения первого полухода ионно-пролетной модели (Boost) в зависимости от используемого количества потоков.

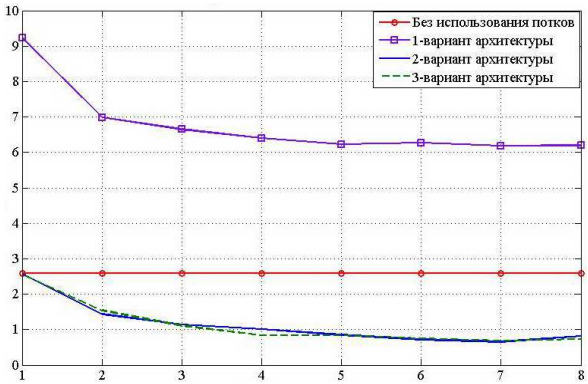


Рис. 2. Зависимость времени выполнения первого полухода ионно-пролетной модели (Qt4Thread) в зависимости от используемого количества потоков.

Литература

1. Матюшкин И.В., Коробов С.В., Зайцев Н.А., Хомяков И.А., Орлов С.Н., Михайлов А.Н., Гусейнов Д.В. Клеточно-автоматный подход к моделированию дефектообразования при ионной имплантации // Материалы IV Всероссийской конференции «Физические и физико–химические основы ионной имплантации». / Новосибирск: 23-26 октября 2012. С.49.

Решение разреженных СЛАУ с использованием графических процессоров в задаче моделирования фильтрационных течений углеводородов в пористой среде

И.И. Газизов, А.В. Юлдашев

Уфимский государственный авиационный технический университет

Решение систем линейных алгебраических уравнений с разреженной матрицей (разреженных СЛАУ) является вычислительным ядром множества задач математического моделирования. Например, в основе моделирования многофазных фильтрационных потоков в пористых средах лежит система уравнений в частных производных, описывающая распределение давлений и насыщенностей фаз в пласте, которая в результате пространственно-временной дискретизации сводится к СЛАУ с сильно разреженной матрицей. Эффективность алгоритмов решения разреженных СЛАУ играет ключевую роль при полномасштабном гидродинамическом моделировании процессов нефтегазодобычи [1].

В последнее время для ускорения вычислений начали активно использоваться гибридные (гетерогенные) вычислительные системы, в которых наряду с процессорами традиционной архитектуры используются массивно-параллельные сопроцессоры, к примеру, графические процессоры (GPU) производства компании NVIDIA с поддержкой программно-аппаратной архитектуры CUDA, обладающие относительно высокой производительностью и энергоэффективностью. Одной из библиотек, содержащей базовые блоки для реализации алгоритмов решения разреженных СЛАУ, является библиотека cuSPARSE, которая входит в стандартный комплект средств разработки CUDA. В последней версии CUDA 5.0 функционал библиотеки расширился за счет добавления нового формата хранения разреженных матриц BCSR, предобуславливателя ILU0 (на основе неполного LU-разложения) и др., что позволяет использовать данную библиотеку для построения программ, реализующих решение разреженных СЛАУ, характерных для задач гидродинамического моделирования процессов нефтегазодобычи.

В нашей работе исследуется возможность ускорения на гибридных вычислительных системах решения разреженных СЛАУ, возникающих в процессе моделирования фильтрационных течений углеводородов в пористой среде. Для этого с использованием библиотек cuSPARSE и cuBLAS на основе работы [2] нами был реализован метод бисопряженных градиентов со стабилизацией (BiCGStab) с предобуславливателем ILU0. Эффективность полученной реализации исследована на вычислительных системах с GPU NVIDIA Tesla M2050 и K20. Проведено сравнение производительности разработанной программы и многопоточной версии решателя СЛАУ, лежащего в основе корпоративного гидродинамического симулятора из состава пакета NGT BOS, используемого в компании «НК «Роснефть». Его производительность получена на двухпроцессорной системе с Intel Xeon X5670 в многопоточном режиме (12 потоков). Сравнение проведено на тестовом наборе из 13 разреженных СЛАУ содержащих от 873 до 6 610 тысяч неизвестных. В докладе будут представлены результаты проведенного сравнения.

В дальнейшем планируется оптимизация разработанной программы под архитектуру графических процессоров NVIDIA Kepler нового поколения.

Литература

1. Борщук О.С. О модификации двухступенчатого метода предобуславливания при численном решении задачи многофазной фильтрации вязкой сжимаемой жидкости в пористой среде // Вестник УГАТУ. 2009. Т. 12. № 1. С. 146-150.
2. Naumov M. Incomplete-LU and Cholesky Preconditioned Iterative Methods Using CUSPARSE and CUBLAS.
URL: https://developer.nvidia.com/content/incomplete-lu-and-cholesky-preconditioned-iterative-methods-using-cusparse-and-cublas (дата обращения: 02.12.2012).

Программная система Complex-Macro для структурной и параметрической идентификации кинетических моделей химических реакций*

И.М. Губайдуллин[2], А.П. Карпенко[1], Е.Ю. Селиверстов[1], М.В. Тихонова[2]

МГТУ им. Баумана[1], Институт нефтехимии и катализа РАН[2]

В Институте нефтехимии и катализа РАН ведутся исследования в области каталитических систем. Возникает задача разработки математических методов, алгоритмов и программного обеспечения для структурной и параметрической идентификации кинетических моделей химических реакций в присутствии катализаторов. Разрабатываемый программный комплекс *Complex-Macro* содержит совокупность систем решения задач моделирования таких процессов.

Подсистема Direct-Complex-Macro предназначена для решения прямых кинетических задач в виде системы ОДУ в нормальной форме Коши. В подсистему включен расширяемый набор программных модулей решения СОДУ. Предложена библиотека решения сверхжестких СОДУ с заданной точностью.

Подсистема Back-Complex-Macro ориентирована на решение обратной параметрической задачи химической кинетики (параметрической идентификации). Полагается заданным набор экспериментальных значений концентраций веществ в различные моменты времени и набор функционалов, определяющих меры близости решения экспериментальным данным. Задача рассматривается как многокритериальная многопараметрическая задача оптимизации. В основе решателей лежит набор модулей построения скалярных сверток (скалярная аддитивная свертка, свертка Гермейера, свертка Джоффриона), модулей решения задач глобальной условной оптимизации (метод роя частиц, генетические алгоритмы, индексный метод), модулей решения задачи Парето-аппроксимации (NGSA-II, SPEA2).

Подсистема включает два вида решателей – *PFS* (*Pareto Front Solver*), предназначенный для построения аппроксимации фронта Парето и решения данной задачи, и *MCS* (*MultiCriteria Solver*), построенный на основе модулей сверток и модулей решения задачи оптимизации. Данная подсистема считается наиболее вычислительно сложной и требует применения параллельных вычислений, планируется применения различных классов вычислительных систем. В рамках работы разработан модуль подсистемы *MCS* решения задачи глобальной оптимизации поведенческим методом роя частиц с использованием параллельных графических процессорных систем (технология Nvidia CUDA).

Подсистема *Complex-Macro* осуществляет структурную идентификацию или решение обратной структурной задачи химической кинетики. Ставится задача отыскания вектор-функции, доставляющей минимум векторному критерию оптимальности. В подсистему включены два решателя: *NWOS* (*Network Operator Solver*), реализующий метод сетевого оператора Дивеева А.И. на основе выбранного пользователем множества унарных и бинарных операторов, и *AS* (*Approximation Solver*), обеспечивающий решение задачи аппроксимации вектор-функции функцией из выбранного класса. Планируется применение параллельных вычислительных систем для реализации метода сетевого оператора Дивеева А.И.

С использованием программного комплекса решены две вычислительно сложные задачи: разработка кинетических моделей реакций гидро-, карбо- и циклоалюминирования олефинов; задача выбора сокатализаторов для прогнозирования выхода целевых продуктов.

Литература

1. Царева З.М., Орлова Е.И.. Теоретические основы химической технологии. – К: Вища шк. Головное издательство, 1986. С. 35.

* Работа выполнена при поддержке РФФИ, грант 12-07-00324.

Численное моделирование обледенения газоходов переменного сечения на кластере ПНИПУ

Д.В. Зимин, В.Я. Модорский

ФГБОУ ВПО «Пермский национальный исследовательский политехнический университет»

Одной из задач требующих использования высокопроизводительных задач является задача обеспечения работоспособности крупногабаритного газохода переменного сечения в зимнее время в связи с возможностью обледенения.

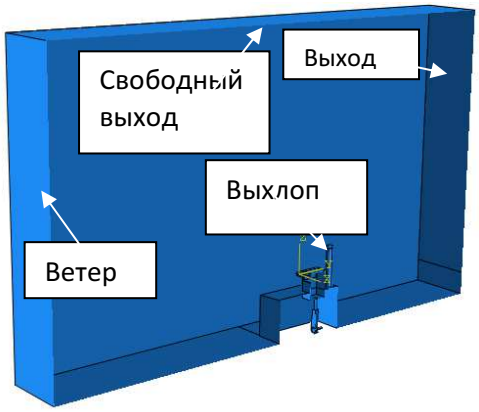


Рис. 1. Твердотельная модель газохода

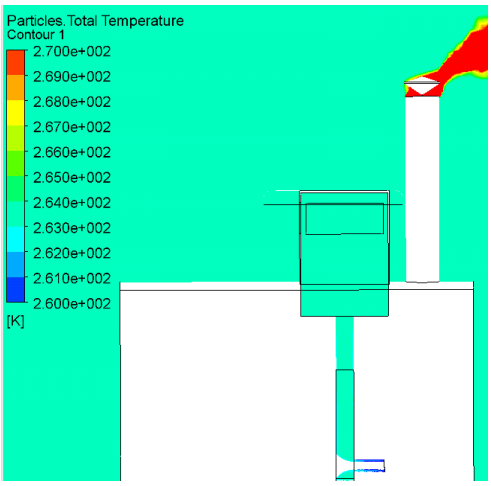


Рис. 2. Температура частиц воды, К

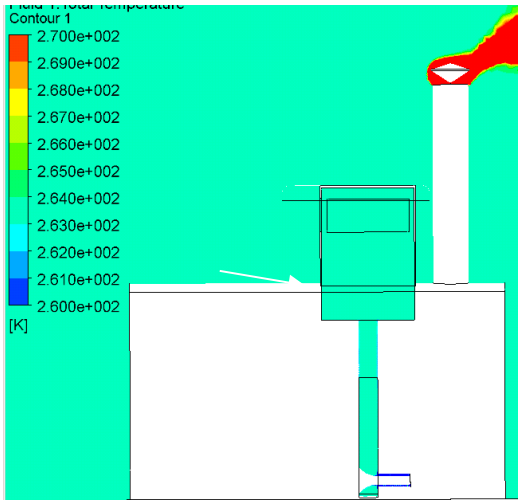


Рис. 3. Температура частиц воздуха, К

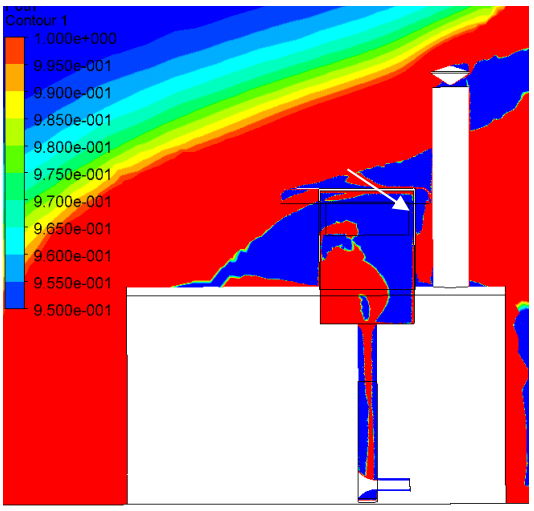


Рис. 4. Относительная влажность, %

- Сформулирована физическая модель для описания процесса обледенения конструкции габаритного газохода. Сформулирована математическая модель процесса.
- Получены распределения переменных, для различных начальных условий, по которым можно проводить анализ возможности обледенения.

Система мониторинга с прогнозированием ошибок

К.В. Иванов

ФГУП ВНИИА имени Н. Л. Духова

В работе представлен прототип разрабатываемой мониторинговой системы с прогнозированием ошибок для управления ресурсами вычислительных кластеров и предсказания системных ошибок. В статье описана одна из основных составляющих системы – подсистема сбора и хранения данных с узлов кластера.

1. Введение

С ростом сложности вычислительных систем проблема обслуживания и своевременного решения задач как прикладного, так и аппаратного уровня, становится все более трудно решаемой без автоматизированных систем управления и прогнозирования [1]. Современные суперкомпьютеры требуют от администраторов знаний о многих сторонних инструментах, необходимых для расчета той или иной задачи, при этом обеспечение объективной оценки эффективности выполнения этих задач не представляется возможным [2]. Для того чтобы оценить производительность и эффективность программы требуются специальные средства – профилировщики, трассировщики, средства анализа. При этом нельзя упускать из виду аппаратную составляющую оборудования, некорректная работа которого может сильно повлиять на результат анализа данных мониторинга [3].

Система мониторинга с прогнозированием ошибок - это попытка предложить комплексное решение для автоматизированного обслуживания кластера или системы кластеров. Данная система позволит администраторам уделять меньше внимания однотипным задачам и поддерживать работоспособность аппаратной части без потери времени, отслеживая все значимые параметры и решая возникшие задачи за счет ресурсов самого кластера. За счет алгоритмов прогнозирования и анализа данных вычислительного кластера, система позволяет получить интерактивную карту специфичных свойств решаемых задач и с помощью профилировщика решить задачу оптимизации ресурсов вычислительного поля [4].

В отличие от существующих подобных систем (SCMS, ClusterX, Ovis…), данная система основывается на самообучающейся нейронной сети, что позволит увидеть неявную зависимость работоспособности вычислительного кластера от выполняемых на нем задач.

Интеллектуальная составляющая системы состоит из трех основных жестко связанных подсистем:

1. Подсистема сбора и хранения данных
2. Подсистема анализа данных
3. Подсистема прогнозирования

Система поддерживает многие типы устройств, включая Infiniband адаптеры, контроллеры BMC, процессоры, оперативную память и т.п.

При высокой детализации кластера (когда количество параметров мониторинга очень высоко) возникает проблема быстроты выборки данных и размещения базы данных. Данная проблема решается несколькими методами – аккумулированием данных, резервным копированием или использованием интервального метода сбора информации.

Данная схема работы позволяет очень гибко настраивать конфигурацию кластерных систем, так как она не требует агентов на каждом из узлов системы, и может не иметь прямого доступа к расчётным узлам, а подключаться через систему туннелей.

Статистический анализ загрузки кластера «Уран»*

А.С. Игумнов

ИММ УрО РАН

В 2012 году на кластере «Уран» была запущена система тотального сбора показателей загрузки вычислительных узлов. С периодичностью в одну минуту с каждого из узлов собирается информация о загрузке CPU и GPU, выделении памяти, использовании сетей Ethernet и Infiniband. Главной целью внедрения этой системы является создание предпосылок для объективной оценки соответствия конфигурации кластера потоку вычислительных задач.

В основу архитектуры системы сбора статистики были положены два базовых принципа:

1. непрерывный сбор информации о нагрузке со всех узлов;
2. независимость от системы управления заданиями.

Сбор статистики на узлах выполняется с помощью модифицированной утилиты collectl [2]. Из собираемых данных удалены параметры, слабо связанные с отладкой кластера, и добавлены специфические параметры, например, загрузка сети Infiniband. При анализе статистики данные о нагрузке узлов объединяются с информацией о выполнявшихся на этих узлах прикладных задачах, извлекаемой из лог-файлов системы управления потоком задач. В настоящий момент написаны модули, обрабатывающие лог-файлы систем запуска СУППЗ и Slurm.

В процессе работы над системой сбора статистики были опробованы различные хранилища данных, в том числе, базы данных SQL. Был сделан вывод о том, что на текущем этапе развития проекта для хранения статистики вполне подходят плоские текстовые файлы с разбиением по месяцам и вычислительным узлам. Разбиение на отдельные файлы позволяет производить быструю выборку данных, относящихся к одной задаче, которая характеризуется группой выделенных ей вычислительных узлов и временем выполнения. При статистическом анализе загрузки кластера в целом во многих случаях достаточно иметь возможность прочитать все накопленные данные без необходимости делать какие-либо выборки или сортировки.

Примерная оценка объемов данных такова: 20 собираемых параметров, представленных в среднем 4-х разрядными десятичными числами, при сборе с периодичностью в одну минуту дают файл размером примерно 20x5x60x24 = 140 КБ в сутки на процессор, или 140 МБ в сутки для системы с 1000 вычислительных ядер. Такие объёмы без труда помещаются в оперативную память даже в тех случаях, когда необходимо проанализировать поведение задачи, использовавшей тысячу ядер в течение нескольких суток.

Отдельно рассматривались вопросы о возможности изменения в будущем набора измеряемых параметров и об обеспечении совместимости накопленных файлов статистики различных версий. Для обеспечения необходимой гибкости было принято решение использовать формат файла с разделителями и фиксированной семантикой отдельных полей.

В 2013 году на основе накопленных данных планируется провести поиск наиболее значимых для анализа поведения кластера наборов параметров и развитие соответствующей системы визуализации.

* Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-1-1034).

Проблемы организации параллельных многоразрядных вычислительных процессов

С.А. Инютин

МАТИ –РГТУ им. К.Э. Циолковского

При математическом моделировании тонких технологических процессов в ряде отраслей производства, в прикладной теории чисел появляются вычислительные задачи, для которых требуется специальная организация вычислительных процессов. В них значения операндов целочисленных операций на сто, тысячу, … порядков превышают максимум типового компьютерного диапазона серийной вычислительной техники. Типовой компьютерный диапазон связан с длиной машинного слова, в котором отображаются данные во внутренних форматах и обрабатываются в операционных регистрах процессора с аппаратной и микропрограммной поддержкой. Машинная арифметика для их обработки имеет наибольшее быстродействие.

Для анализа вычислительных процессов с превышением типового компьютерного диапазона введем специальную терминологию. Назовем диапазонными числовые величины, для отображения которых типовой диапазон достаточен. В зависимости от длины машинного слова вычислительной системы типовой диапазон 16, 32 или 64 разрядный (бинарные разряды). Для диапазонных величин, отображаемых в беззнаковом целочисленном формате для $n = 16 \mid 32 \mid 64$ выполняются неравенства $0 \le A \le 2^n - 1$, а для числовых величин, отображаемых в целочисленном формате со знаком выполняются неравенства $-2^{n-1} \le A \le 2^{n-1} - 1$. Компьютерные форматы данных с двойной точностью, условно относятся к диапазонным, в регистрах процессора обработка этих форматов имеет микропрограммную и аппаратную поддержку. Назовем сверхдиапазонными (многоразрядными) числовые величины, для отображения которых типовой и удвоенный диапазоны недостаточны. Для них введем производные беззнаковый и знаковый целочисленные форматы. Для сверхдиапазонных числовых величин, отображаемых в беззнаковом формате, для $n = 16 \mid 32 \mid 64$ -разрядного процессора выполняются неравенства $2^{2n} - 1 < C \le D$, а для величин, отображаемых в формате со знаком, выполняются неравенства $-D \le C < -2^{2n-1}$ *и* $2^{2n-1} - 1 < C \le D$, где D – константа, зависящая от предельных возможностей программно эмулируемой многоразрядной арифметики, операционных ресурсов и объема памяти вычислительной системы. Множество многоразрядных числовых величин назовем большим диапазоном *W*, для которого $D = \max W$.

Задачи тестирования и факторизации больших числовых величин называют вычислительными проблемами из-за большой временной сложности. В частности, это проблема тестирования на простоту при больших показателях степени чисел Мерсенна $M_q = 2^q - 1$, где $q \in N$ – множество натуральных чисел, $q>2$. Современные методы ориентированы на тестирование чисел Мерсенна при $q>300000$, а числовые величины для таких процессов содержат более миллиона десятичных разрядов позиционного представления. Программная реализация эффективных методов вычислений в больших диапазонах позволяет решать задачи, требующих вычислений в сверхбольших диапазонах, появляющихся при вычислении функций от многоразрядных величин. Для ряда задач вычислительной теории чисел максимум сверхбольшого компьютерного диапазона должен достигать значения константы Виноградова - Гольбаха $3^{3^{15}}$. Такие задачи возникают, при вычислении выражений вида:

$$C = \sum_{i=1}^{k} A_i^{f(B)} - [\sum_{i=1}^{k} A_i^{f(B)} / B] \cdot B,$$

где A_i , B, а также область значений функции $f(B)$ принадлежат большому диапазону W.

Для эффективного решения таких задач предлагается использование распараллеливаемой модулярной программно-эмулируемой арифметики и производных форматов данных.

Решение задачи прогнозирования осложнений в бурении с использованием искусственных нейронных сетей

А.Р. Кабирова[1], Ю.Б. Линд[2], А.Р. Мурзагалин[1]

ФГБОУ ВПО «Башкирский государственный университет»[1], ООО «БашНИПИнефть»[2]

Целью данной работы является прогнозирование осложнений при бурении новых скважин на основе минимума информации по ранее пробуренным скважинам данного месторождения. Задача решается в применении к наиболее распространенному виду осложнений на месторождениях Республики Башкортостан (РБ) – поглощениям бурового раствора. В качестве исходных данных выступает следующая информация по пробуренным скважинам: координаты устья скважины и класс интенсивности поглощения (без поглощений, малой интенсивности (<5 м^3/час), средней (15-40 м^3/час), высокой (>40 м^3/час)). Необходимо кластеризовать месторождение согласно введенным классам, что позволит прогнозировать класс интенсивности поглощения для любой новой скважины месторождения по проектным координатам ее устья.

Для модельного примера (на плоскости дано 30 точек, цвета которых соответствуют классам интенсивности поглощения: синий – нет поглощений, зеленый – поглощения малой интенсивности, желтый – средней, красный – высокой) было рассмотрено несколько вариантов искусственных нейронных сетей, решающих поставленную задачу (рис. 1).

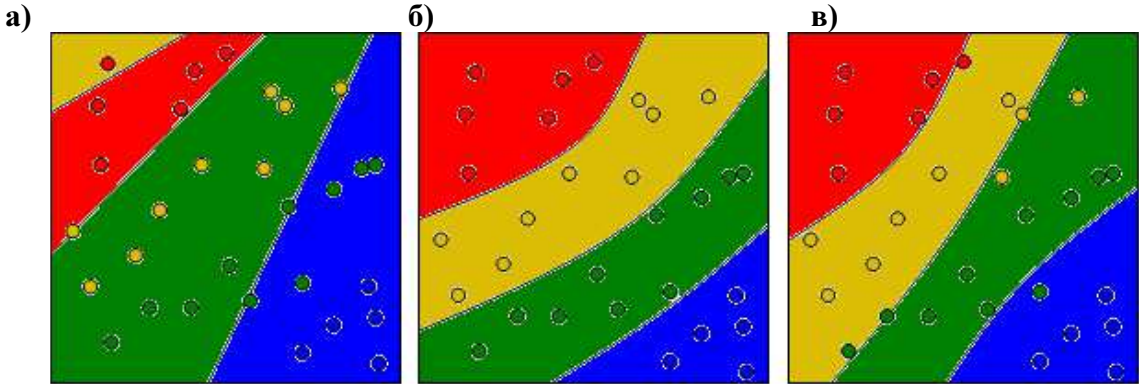


Рис. 1. Результаты вычислительного эксперимента (точки – факт):
а) однослойный персептрон с 4 выходами; б) двухслойный персептрон с 1 выходом; в) карта Кохонена

Наиболее эффективным (показавшим наибольшую точность при разумном времени реализации) оказался двухслойный персептрон с одним выходным нейроном (в скрытом слое 6 нейронов). В настоящее время проводится сбор промысловых данных для уточнения результатов.

Большие объемы входных данных и итеративная природа алгоритма обучения нейросети порождают необходимость использования высокопроизводительных вычислительных систем для получения прогноза требуемой точности за разумное время. Разработана трехуровневая модель распараллеливания вычислительного процесса, которая включает распараллеливание по экспериментальной базе, выявление и использование внутреннего параллелизма задачи и распараллеливание метода ее решения [1]. На данный момент реализовано сочетание первых двух уровней: параллельная кластеризация карт по разным месторождениям (на территории РБ их более 200) и использование внутреннего параллелизма, которое заключается в возможности одновременного рассмотрения разных объектов поглощения, составляющих геологический разрез (всего около 30 поглощающих стратиграфических объектов). В настоящее время производится отладка последнего уровня – распараллеливания алгоритма обучения нейронной сети.

Литература

1. Линд Ю.Б., Кабирова А.Р. Применение современных ИТ для прогнозирования осложнений в бурении нефтегазовых скважин // Искусственный интеллект, №3, 2012. С. 451-457.

Математическое моделирование течений многокомпонентных сред в кольцевых соплах с применением высокопроизводительных вычислений

А.Л. Карташев, М.А. Карташева

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно−Уральский государственный университет»
(национальный исследовательский университет)

Рассмотрена задача определения зависимости тяговых характеристик кольцевых сопел от параметров конденсированной фазы. При этом полагалось, что все частицы конденсированной фазы состоят из одного вещества, а химические реакции в смеси и фазовые переходы между газом и конденсированной фазой отсутствуют [1].

Течение в кольцевом сопле является «смешанным» течением. Наличие в поле течения до–, транс–, сверхзвуковых областей характерно для большинства геометрических конфигураций кольцевых сопел, причем дозвуковые области могут возникать даже вблизи выходного сечения сопла. Таким образом, в различных подобластях решения задачи уравнения газовой фазы могут принадлежать к эллиптическому или гиперболическому типу, что приводит к необходимости применять различные методы для решения задачи в каждой из этих областей, при этом границы указанных подобластей заранее неизвестны. Поэтому для расчета параметров «смешанного» течения целесообразно также применить метод установления, как и при моделировании течений чистого газа.

Вследствие громоздкости уравнений двухфазного течения с учетом процессов коагуляции и дробления частиц и сложной геометрической конфигурации области решения численное интегрирование этих уравнений представляет собой довольно сложную алгоритмическую задачу. Поэтому, в рамках предлагаемого подхода, для математического моделирования использовались уравнения непрерывной модели, записанные для дискретной функции распределения [2].

В ряде возможных случаев течения для решения рассматриваемой задачи целесообразно применять метод псевдоустановления, заключающийся в разделении системы уравнений на две подсистемы: для газа и для частиц, первая из которых решается методом установления, а для решения второй подсистемы – расчета параметров частиц – используются стационарные уравнения, интегрирование которых возможно различными способами, например, с помощью неявных разностных схем.

С помощью описанных выше методов и алгоритмов проведено математическое моделирование параметров многофазной полидисперсной смеси для типичной конфигурации кольцевого сопла. В ходе проведения численных исследований получены значения импульса осаждения частиц конденсированной фазы J_{pr} на центральное тело и внешнюю обечайку сопла. Исследованы траектории движения частиц полидисперсной конденсированной фазы в условиях сложной геометрической конфигурации и значительных градиентов газодинамических параметров.

Результаты математического моделирования получены с применением высокопроизводительных вычислений на суперкомпьютере «СКИФ-Аврора ЮУрГУ» ФГБОУ ВПО «ЮУрГУ» (НИУ) с производительностью до 117 TFlops.

Литература

1. Карташев, А.Л. Математическое моделирование течений в кольцевых соплах: монография/А.Л. Карташев, М.А. Карташева. – Челябинск: Издательский центр ЮУрГУ, 2011. – 158 с.

2. Газовая динамика двухфазных течений в соплах/ И.М. Васенин, В.А. Архипов, В.Г. Бутов и др. – Томск: Изд–во Томского университета, 1986. – 262 с.

Численные исследования динамики совершенного газа в кольцевых соплах с применением высокопроизводительных вычислений

А.Л. Карташев, М.А. Карташева

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно−Уральский государственный университет»
(национальный исследовательский университет)

Кольцевое сопло, как и любое другое сопло, представляет собой газодинамическое устройство, предназначенное для создания осевой тяги и управляющих усилий либо для создания на выходе из сопла газового потока с заданными свойствами. Тяговые характеристики сопла при заданных параметрах рабочего тела на входе и параметрах внешней среды полностью определяются его геометрической конфигурацией. Поэтому при заданной геометрической конфигурации кольцевого сопла задача определения тяговых характеристик сводится в основном к определению газодинамических характеристик потока в рассматриваемом сопле, то есть к решению «прямой» задачи теории сопла (расчету поля течения в сопле с последующим определением его тяговых характеристик).

Решение поставленной выше задачи находится с помощью математического моделирования течения газа [1], для проведения которого целесообразно использовать численные методы, не связанные с выделением особенностей в поле течения и обеспечивающие проведение «сквозного» расчета.

Алгоритм расчета смешанных течений идеального газа в кольцевых соплах построен на основе общего подхода к численному решению задач газовой динамики, предложенного в работе С.К. Годунова, А.В. Забродина, М.Я. Иванова, А.Н. Крайко, Г.П. Прокопова «Численное решение многомерных задач газовой динамики» [2], с модификацией В.П. Колгана.

Для расчета параметров сверхзвукового течения в кольцевых соплах в случаях, когда заранее известен характер течения, используется обобщенная разностная схема М.Я. Иванова – А.Н. Крайко – Н.В. Михайлова.

На основе указанных расчетных методов разработаны вычислительные алгоритмы, позволяющие проводить расчеты течений в кольцевых соплах без предварительного выделения особенностей и определить газодинамические параметры потока и тяговые характеристики кольцевого сопла. Основное отличие представляемых вычислительных алгоритмов от применяемых ранее – их модификация в части построения разностной сетки и постановки граничных условий. Такие различия обусловлены особой геометрией расчетной области кольцевого сопла: наличием двух обтекаемых поверхностей и ее сложной искривленной.

В соответствии с целью настоящего исследования проведено математическое моделирование кольцевых сопел внешнего расширения. По результатам численного моделирования определены картины течения в кольцевых соплах различных конфигураций [1]. Полученные результаты математического моделирования могут быть использованы при определении газодинамической структуры потока в кольцевых соплах летательных аппаратов различного назначения и их тяговой эффективности.

Результаты математического моделирования получены с применением высокопроизводительных вычислений на суперкомпьютере «СКИФ-Аврора ЮУрГУ» ФГБОУ ВПО «ЮУрГУ» (НИУ) с производительностью до 117 TFlops.

Литература

1. Карташев, А.Л. Математическое моделирование течений в кольцевых соплах: монография / А.Л. Карташев, М.А. Карташева. – Челябинск: Издательский центр ЮУрГУ, 2011. – 158 с.

2. Численное решение многомерных задач газовой динамики / С.К. Годунов, А.В. Забродин, М.Я. Иванов и др. – М.: Наука, 1976. – 400 с.

Профилирование оптимального кольцевого сопла с многокомпонентным рабочим телом

М.А. Карташева

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Южно−Уральский государственный университет»
(национальный исследовательский университет)

Рассматривается задача построения оптимальной (обеспечивающей максимальное значение тяги) конфигурации кольцевого сопла внешнего расширения с многокомпонентным рабочим телом [1].

Поиск оптимальной конфигурации кольцевого сопла проводится путем решения вариационной задачи, при постановке которой в число оптимизируемых параметров включаются геометрические характеристики сопла (при этом параметры конденсированной фазы и показатель изоэнтропы *k*, а также располагаемый перепад давлений в сопле p_o/p_a считаются постоянными), и которая сводится к задаче нелинейного программирования [2].

Основными элементами данного подхода являются прямые расчеты поля течения и метод поиска экстремума функций многих переменных, что делает его применимым ко всем газодинамическим задачам, для которых известны методы расчета поля течения, в том числе и для кольцевых сопел с многокомпонентными потоками.

Процедура оптимизации геометрической конфигурации кольцевого сопла представляет собой сочетание аналитических методов построения оптимизируемого функционала и задания геометрического профиля кольцевого сопла, с методами поиска экстремума целевой функции, являющейся функцией нескольких переменных, и прямыми расчетами поля течения с помощью численных методов. Построение оптимального по тяге кольцевого сопла проводится в условиях заданных ограничений на его геометрические характеристики.

Прямые расчеты поля течения проведены для многокомпонентной среды с моно– и полидисперсной конденсированной фазой.

Для поиска экстремума целевой функции использованы различные методы с дискретным шагом и с минимизацией по направлению.

Построены оптимальные конфигурации кольцевых сопел с многофазным полидисперсным рабочим телом при заданных геометрических параметрах кольцевого сопла и параметрах многофазной смеси. Проведено сравнение оптимальных по тяге кольцевых сопел с многофазным рабочим телом с оптимальными кольцевыми соплами для чистого газа. Построены оптимальные кольцевые сопла для смеси газа и полидисперсного конденсата при различных условиях, ограничивающих выпадение частиц конденсированной фазы на стенки сопла.

Результаты математического моделирования получены с применением высокопроизводительных вычислений на суперкомпьютере «СКИФ-Аврора ЮУрГУ» ФГБОУ ВПО «ЮУрГУ» (НИУ) с производительностью до 117 TFlops.

Литература

1. Карташев, А.Л. Математическое моделирование течений в кольцевых соплах: монография/А.Л. Карташев, М.А. Карташева. – Челябинск: Издательский центр ЮУрГУ, 2011. – 158 с.

2. Бутов, В.Г. Применение методов нелинейного программирования для решения вариационных задач газовой динамики/В.Г. Бутов, И.М. Васенин, А.И. Шелуха// ПММ. – 1977. – Т. 41. – Вып.1. – С.59-64.

Распараллеливание алгоритмов численного решения функционально-дифференциальных уравнений при решении задачи стабилизации сгорания топлива в жидкостном ракетном двигателе*

А.В. Ким, В.М. Кормышев, М.А. Сафронов

Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Классическая математическая модель процесса сгорания топлива в жидкостном ракетном двигателе представляет собой систему функционально-дифференциальных уравнений (ФДУ), которую можно привести к виду [1]:

$$\dot{x}(t) = Ax(t) + A_\tau x(t-1) + Bu(t). \quad (1)$$

В работе [2] разрабатываются теоретические аспекты аналитического конструирования регуляторов в системах с запаздыванием, основанные на методе явных решений Обобщенных Уравнений Риккати (ОУР), а также рассматриваются способы нахождения параметров оптимального стабилизирующего управления для систем ФДУ. В [3] описываются разработанные методы численного решения дифференциальных уравнений с последействием.

Задача состоит в исследовании возможности распараллеливания численных алгоритмов решения системы (1), а также построения оптимального стабилизирующего управления на современных высокопроизводительных системах, используя управление с обратной связью, основывающееся на втором варианте явных решений системы обобщенных уравнений Риккати. При этом требуется решение специального экспоненциального матричного уравнения (ЭМУ), которое приближенно находится методом установившихся решений с использованием разработанного алгоритма.

Возможные подходы к распараллеливанию методов Рунге-Кутта для решения дифференциальных уравнений описаны, например, в [4]. Для численного расчета ЭМУ и ФДУ разработана специальная модификация данного алгоритма.

Литература

1. *Сафронов М.А.* Математическое моделирование процесса сгорания топлива в жидкостном ракетном двигателе // Международный научн.-техн. журнал "Информационные технологии моделирования и управления". 2011. №7(72). С. 806-811.

2. *Квон В.Х., Ким А.В., Кормышев В.М., Пименов В.Г., Солодушкин С.И.* Аналитическое конструирование и синтез регуляторов для систем с последействием. — Екатеринбург: УрФУ, 2010. — 170 с.

3. *Ким А.В., Пименов В.Г.* i-Гладкий анализ и численные методы решения ФДУ. — М.-Ижевск: Институт компьютерных исследований, 2004. — 256 с.

4. *Van Der Houwen P.J., Sommeijer B.P.* Parallel iteration of high-order Runge-Kutta methods with stepsize control // Journal of Computational and Applied Mathematics. 1990. Vol. 29. Pp. 111-127.

*Работа выполнена при поддержке программы президиума РАН «Фундаментальные науки — медицине», РФФИ (проекты 08-0100141, 10-01-00377) и Урало-сибирского междисциплинарного проекта 12-С-1-1017.

Разработка и анализ высокопроизводительного параллельного алгоритма решения кооперативных игр сведением к биматричным играм

А.С. Кириллов

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Оренбургский государственный университет»

В работах [1,2,3] рассмотрен подход к решению задачи нахождения значений характеристической функции кооперативной игры, заданной множеством биматричных игр, с помощью решения матричных игр отдельно коалиции и антикоалиции. Но можно рассматривать полученные матрицы как биматричную игру и её решение позволяет получить значения характеристической функции $v(S)$ и $v(P)$ для коалиции S и антикоалиции P соответственно.

Рассмотрим подход к решению полученной биматричной игры. Так как перебор векторов x и y аналогичны, то рассмотрим его на примере вектора x.

Компоненты вектора удовлетворяют уравнению гиперплоскости $\sum_{i=1}^{m} x_i = 1$ и при этом должны оставаться положительными, то есть мы имеем область гиперплоскости ограниченную отрезками соединяющие точки пересечения этой плоскости с осями координат. Назовем эти точки V_i. Радиус-вектора представленные этими точками будут иметь единичную длину, и лежать на осях координат m-мерного пространства.

Для получения всех точек лежащих на гиперплоскости, достаточно интерполировать $m-1$ не коллинеарных векторов принадлежащих ей: $V = V_1 + \sum_{i=1}^{m-1} (V_{i+1} - V_1) t_i$, где t_i – параметры интерполяции. Причем если $t_i \in [0,1]$, то точки будут лежать внутри области сформированной векторами $V_{i+1}-V_1$. При этом возникают отрицательные значения первой компоненты. Поэтому при реализации перебора, происходит проверка первой компоненты на отрицательность, и такие вектора отсеиваются. Таким образом, перебору подлежат $m-1$ переменная, каждая из которых лежит в диапазоне [0, 1]. В программе перебор происходит в несколько этапов. Задается достаточно большой начальный шаг перебора, вычисляются вектора x и y, для каждой такой пары вычисляется значение функции ошибки. При этом отсеиваются точки, в которых эти значения максимально близки к 0. Далее поиск продолжается при значениях параметров t_i лежащих вблизи найденной точки. Так продолжается до тех пор, пока функция ошибки не станет меньше какого-то наперед заданного значения. Такой перебор может выполняться параллельно по аналогии с алгоритмами, описанными в [1,2,3].

Литература

1. Кириллов А.С. Анализ масштабируемости параллельных алгоритмов решения кооперативной игры – Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XII Всероссийской конференции (Н. Новгород, 26–28 ноября 2012 г.) / Под ред. проф. В.П. Гергеля. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2012. – сс. 180-184.
2. Нестеренко М.Ю., Кириллов А.С. Разработка высокопроизводительного параллельного алгоритма решения кооперативных игр сведением к биматричным играм - Вестник ОГУ апрель 2011, сс. 215-218.
3. Нестеренко М.Ю., Кириллов А.С. Разработка и анализ высокопроизводительных параллельных алгоритмов решения кооперативных игр – Вестник ЮУрГУ, серия Математическое моделирование и программирование, выпуск 8, № 17 (234) 2011, сс. 92-101.

Подход к представлению вычислительных задач в терминах dataflow на распределенных системах*

С.О. Кисляков, А.Н. Сальников

Московский государственный университет имени М.В.Ломоносова

Многие университеты и исследовательские группы активно изучают концепцию управления выполнением программы с помощью потока данных (dataflow). Данный подход лишен недостатков управления с помощью единственного счетчика команд, препятствующего распараллеливанию на уровне инструкций. В dataflow единственным критерием для выполнения следующей операции является готовность данных. Нами были рассмотрены существующие dataflow системы: Dryad [1], LuNA [2], PARUS [3]; выявлены преимущества и недостатки подходов представления задач в синтаксисе языка и способов генерации исполняемого кода для распределенных вычислительных систем.

Нами разрабатывается система Frigate — следующий виток развития системы PARUS. Frigate позволяет программисту описать алгоритм в специальном формате. По данному представлению генерируется набор функций на языке C++, для обмена данными используется библиотека MPI. Затем код компилируется и связывается с библиотекой, которая реализует подсистему времени запуска и размещает функции по узлам вычислительной системы. Среди MPI-процессов выделяются процессы координаторы, управляющие выполнением программы. Полученная программа запускается на вычислительном кластере.

Пользователю необходимо представить алгоритм решения как граф, состоящий из набора ориентированных ациклических подграфов. Вершинам в графе соответствуют вычисления, обрабатывающие входные данные и получающие в результате данные на выход. Рёбра в графе бывают трёх типов: внутренние, внешние и управляющие. Внутренние рёбра представляют собой зависимости по данным между вершинами одного подграфа, Внешние рёбра соединяют вершины разных подграфов. Данные, передаваемые по внешнему ребру, буферизуются до окончания выполнения текущего подграфа. Управляющее ребра направлены от вершин подграфа к сущности координатора, их назначение — изменять глобальные данные на тех MPI-процессах, которые являются координаторами.

Система Dryad предоставляет широкие возможности для описания графа программы, но, в отличии от Frigate, не дает возможности перестраивать граф в процессе выполнения. За счет введения конструкций, явно описывающих граф, процесс фрагментирования программы более интуитивен для программиста, чем в системе LuNA, хоть и остается полностью за ним.

Литература

1. M. Isard, M. Budiu, Y. Yu, A. Birrell, D. Fetterly Dryad: distributed data-parallel programs from sequential building blocks //European Conference on Computer Systems (EuroSys), March 21–23, 2007, Lisbon, Portugal. P. 59–72.

2. Малышкин В.Э. Технология фрагментированного программирования //Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции Новосибирск: 26–30 марта 2012 г. С. 592–599.

3. Alexey N. Salnikov PARUS: A Parallel Programming Framework for Heterogeneous Multiprocessor Systems //Lecture Notes in Computer Science (LNCS 4192) Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2006. P. 408–409.

*Работа проводится при поддержке грантов РФФИ 11-07-00756-а, 11-07-00614-а.

Многопараметрический параллельный вычислительный эксперимент при структурной идентификации кинетических моделей обобщенного механизма реакций*

К.Ф. Коледина[1], И.М. Губайдуллин[2]

Башкирский Государственный Университет[1],
Институт нефтехимии и катализа РАН[2]

На основании разработанной кинетической модели частных реакций гидроалюминирования олефинов [1] необходимо решить обратную многоинтервальную параметрическую задачу для общей реакции, с использованием параллельных алгоритмов и внутреннего параллелизма самой задачи. При решении обратной задачи необходимо использовать параллельные технологии, ввиду большого объема вычислений. Причем распараллеливание можно вести не только за счет метода решения обратной задачи – генетического алгоритма, но и за счет внутреннего параллелизма самой задачи. Для реакции гидроалюминирования олефинов существует несколько механизмов протекания как общих, так и частных выделенных реакций, кинетические параметры для общих стадий в которых должны описываться одной областью значений. Для каждой реакции обрабатывается несколько экспериментов при различных условиях. Для каждого эксперимента необходимо найти набор кинетических параметров с помощью генетического алгоритма. Таким образом, распараллеливание, основанное на внутреннем параллелизме, состоит из трех уровней:

I. Определение общей области значений кинетических параметров частных и общих реакций гидроалюминирования олефинов, описывающей все экспериментальные данные.

II. Параллельная обработка каждого эксперимента для определения значений кинетических параметров.

III. Распараллеливание генетического алгоритма – обратной задачи для каждого эксперимента.

Для определения интервалов констант скоростей реакций можно использовать алгоритм, основанный на методе ветвей и границ. Апробация алгоритма показала перспективность определения области значений, в которой гарантировано содержится глобальный минимум целевой функции.

Программно реализованы и распараллелены генетический алгоритм решения обратной задачи химической кинетики и алгоритм определения трехмерных областей допустимых значений кинетических параметров химических реакций с заданной точностью. Выбор кинетических параметров происходит согласно значимости их влияния на функционал минимизации расчетных и экспериментальных данных, в соответствие с методом Парето.

Необходимо исследовать полученную модель на выявление ключевых процессов. Для этого необходимо:

- разработать метод моделирования индукционного периода сложных реакций для определения параметрической чувствительности к условиям проведения реакции;
- провести вычислительный эксперимент для определения параметров и предпосылок индукционного периода, увеличения выхода целевого продукта, определения промежуточных соединений, регулирования времени проведения реакции.

Литература

1. Губайдуллин И.М., Линд Ю.Б., Коледина К.Ф. Методология распараллеливания при решении многопараметрических обратных задач химической кинетики \\ Вычислительные методы и программирование. – 2012. Т 13, №1 – С.236–244.

* Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации (Госконтракт №02.740.11.0631) и РФФИ (гранты №12-07-00324 и №12-07-31029).

О параллельной реализации алгоритма поиска наибольшего значения в классе функций, определяемом кусочно-линейной мажорантой

А.Г. Коротченко, В.М. Сморякова

Нижегородский государственный университет им. Н.И. Лобачевского
г. Нижний Новгород

Вопросы, связанные с исследованием эффективности численных методов оптимизации, представляют значительный интерес, как с теоретической, так и с практической точек зрения. С теоретической точки зрения, такие исследования интересны выяснением потенциальных возможностей алгоритмов оптимизации на рассматриваемых классах задач. С точки зрения практики, использование эффективных методов в качестве вспомогательных процедур является важным при решении сложных задач, требующих большого объёма вычислений или проведения дорогостоящих экспериментов.

Эффективность алгоритма во многом определяется классом рассматриваемых задач. Например, в работах [2] и [3] построены подобного рода алгоритмы для различных классов функций, отражающих существенные свойства задач, встречающихся в приложениях (в том числе многокритериальных) и допускающих построение достаточно эффективных и простых в реализации алгоритмов.

В работе [1] построены оптимальный одношаговый и приближенно оптимальный алгоритмы для классов функций, определяемых кусочно-линейной мажорантой.

В данной работе рассматриваются подклассы класса функций определяемых кусочно-линейной мажорантой, позволяющие построить простой и более эффективный алгоритм в сравнении с алгоритмами, построенными в [1].

В ходе проведённых исследований была реализована параллельная версия алгоритма, которая позволила в 2-3 раза сократить время работы алгоритма на тестовых примерах по сравнению с последовательной версией при одной и той же точности вычислений.

Дальнейшие исследования связаны с применением алгоритма в качестве вспомогательной процедуры для поиска экстремумов функций многих переменных в диагональных методах.

Литература

1. Коротченко А.Г. Об одном алгоритме поиска наибольшего значения одномерных функций // Журнал вычислительной математики и математической физики. 1978. Т.18. N 3. С.563-573.
2. Коротченко А.Г. Приближенно-оптимальный алгоритм поиска экстремума одного класса функций // Журнал вычислительной математики и математической физики. 1996. Т.36. N 5. С.30-39.

Использование асинхронных вычислений в функциональном языке потоково-параллельного программирования

И.В. Матковский

Сибирский Федеральный Университет

Одной из актуальных проблем организации параллельных вычислений является обработка асинхронных вычислений. Различная скорость работы отдельных модулей может привести к тому, что необходимые для работы системы в целом данные будут поступать с задержками. Асинхронная модель вычисления должна эффективно обрабатывать данные вне зависимости от того, с какими задержками и в каком порядке они поступают.

Предлагаемая модель асинхронных списков [1] предназначена для функционального языка потоково-параллельного программирования “Пифагор” [2] и построена на использовании асинхронных списков. Асинхронный список хранит поступающие в него фрагменты данных в порядке поступления. Список может находиться в одном из двух состояний – либо он пуст, либо содержит как минимум один элемент.

```
Async_Sum<< funcdef A {
    x1<< A:1;
    tail_1<< A:-1;
    [(tail_1:[IsEmptyDataList, IsNotEmptyDataList]:?]^
    ( x1,
      {  block {
         x2<< tail_1:1; s<< (x1,x2):+;  tail_2<< tail_1:-1;
         [(tail_2:[IsEmptyDataList,  IsNotEmptyDataList]:?]^
         (   s,
            { asynch( tail_2:[], s):A_VecSum }
         ):. >>break;
       }}):. >>return;}
```

Рис. 1. Алгоритм суммирования асинхронного списка

Выделяется первый (головной) элемент асинхронного списка x1 (получаемый командой выборки A:1); после этого проверяется хвост списка tail_1 (получаемый командой выборки-исключения A:-1). Если хвост списка tail_1 пуст, то список A состоит из одного элемента x1, который и следует вернуть в качестве результата функции. Если хвост списка tail_1 не пуст, то выделяем его головной элемент x2 и суммируем его с x1; после этого проверяем “хвост хвоста” tail_2. Если tail_2 пуст, то A содержал только элементы x1 и x2. Возвращаем их сумму в качестве суммы списка. Если tail_2 не пуст, создаем новый асихронный список, в который помещаем сумму x1 и x2 и tail_2; вызываем для него рекурсивно функцию суммирования.

Литература

1. Легалов А.И., Редькин А.В. Расширение асинхронного управления по готовности данных / Труды III Международной конференции «Параллельные вычисления и задачи управления» PACO’2006. — ISBN 5-201-14990-1 / М.: Институт проблем управления им. В.А. Трапезникова РАН, 2006. С 1272-1281. (Электронное издание)

2. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1 (10). С.71-89.

Коэффициенты связи физических и модельных параметров в клеточно-автоматных моделях диффузионного процесса с целочисленным алфавитом*

Ю.Г. Медведев

ИВМиМГ СО РАН

Клеточно-автоматные модели вообще и модели диффузии в частности представляют огромный интерес для исследований в настоящее время [1]. Они являются естественно-параллельными с мелкозернистым параллелизмом, поэтому их программные реализации на суперкомпьютерах хорошо масштабируются и, как правило, показывают эффективность 80-90% на тысячах вычислительных ядер [2].

В работе исследуются две предложенные ранее клеточно-автоматные модели диффузионного процесса — обобщения асинхронного клеточного автомата элементарной диффузии Т. Тоффоли и блочно-синхронного клеточного автомата диффузии Н. Марголуса до клеточных автоматов с целочисленным алфавитом [3]. При описании диффузионного процесса этими моделями для представления концентрации веществ, времени и других физических параметров используются дискретные безразмерные модельные величины. В результате получается довольно точная качественная картина моделируемого процесса, но отсутствует возможность найти его абсолютные количественные характеристики.

В качестве исходных данных задаются размеры моделируемой области X [м] и Y [м], точность Δl [м], начальное распределение концентрации вещества $c(x,y)$, коэффициент диффузии D [м^2/с], и время моделирования t [с]. Перед началом моделирования выбираются максимальная модельная концентрация $C_{\max}$ [частиц в клетке] с учетом разрядности компьютера и модельный коэффициент диффузии k из интервала (0,1). Затем вычисляется масштаб линейных размеров $\mu_l = \Delta l / 10$ [м] так, чтобы в пределах точности Δl размещалось как минимум 10 клеток. Далее вычисляются размеры клеточного массива $I = \lfloor X / \mu_l \rfloor$ [клеток] и $J = \lfloor Y / \mu_l \rfloor$ [клеток], масштаб концентрации $\mu_c = c_{\max} / C_{\max}$ [м$^{-2}$], концентрация в каждой клетке $C(i,j) = \lfloor c(x(i),y(j)) / \mu_c \rfloor$ [частиц в клетке], масштаб времени $\mu_t = k\,\mu_l^2 / 2D$ [с], количество итераций $T = \lfloor t / \mu_t \rfloor$ [итераций]. Физические и модельные координаты связаны соотношениями: $i(x) = \lfloor x / \mu_l \rfloor$, $j(y) = \lfloor y / \mu_l \rfloor$, $x(i) = \mu_l\, i$ и $y(j) = \mu_l\, j$. После выполнения T итераций в клеточном массиве будут находиться значения модельной концентрации, соответствующие физической по истечении времени t, из которых могут быть вычислены значения физической концентрации вещества в каждой точке $c(x,y) = \mu_c\, C(i(x),j(y))$ [м$^{-2}$], являющиеся результатом моделирования.

С введением описанных в работе коэффициентов, связывающих физические величины с внутренними дискретными, исследуемые модели становятся полнофункциональными. Использование таких масштабных коэффициентов делает пригодными исследуемые модели для практического применения.

Литература

1. Бандман О.Л. Клеточно-автоматные модели пространственной динамики // Системная информатика. 2006. №10. С. 59–113.

2. Медведев Ю.Г. Программный комплекс клеточно-автоматного моделирования газопорошковых потоков // Параллельные вычислительные технологии (ПаВТ’2012): труды международной научной конференции. Челябинск: Изд. центр ЮУрГУ, 2012, с. 732.

3. Medvedev Yu. Multi-particle Cellular-Automata Models for Diffusion Simulation // LNCS. Vol. 6083. Springer — 2010. P. 204–211.

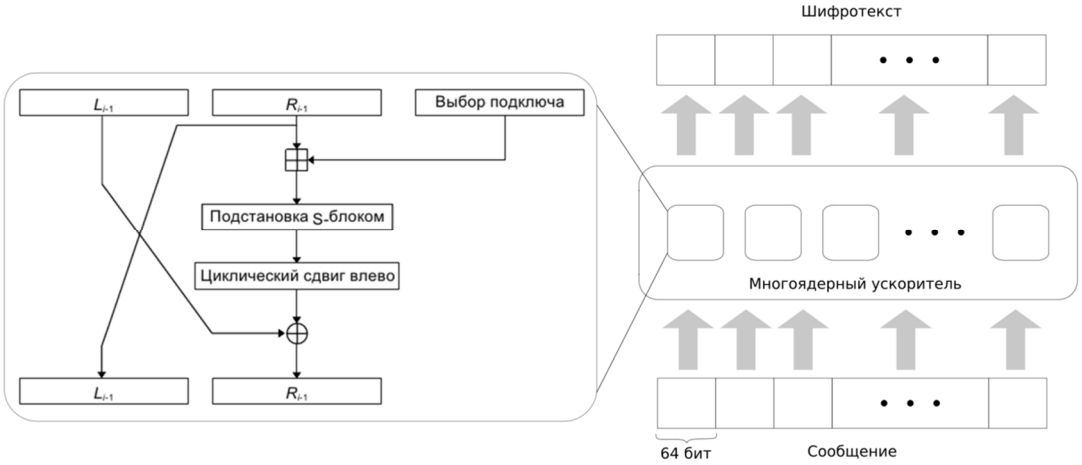
* 1) Исследование выполнено по Программе фундаментальных исследований Президиума РАН, проект № 15.9.
2) Исследование выполнено при финансовой поддержке РФФИ в рамках проекта № 11-01-00567а.

Разработка параллельного алгоритма шифрования ГОСТ 28147-89 на платформе IntelXeonPhi

М.С. Миниахметова, М.Л. Цымблер

Южно-Уральский государственный университет

В соответствии с российским законодательством технические средства защиты информации должны осуществлять шифрование данных, используя криптографический стандарт ГОСТ 28147-89. Блочная реализация алгоритма ГОСТ 28147-89 предполагает разбиение сообщения на блоки равной длины, шифрование которых осуществляется независимо (см. рис. 1а). Таким образом, при использовании блочного метода шифрования возможна параллельная обработка всех блоков сообщения на многопроцессорной и/или многоядерной системе (см. рис. 1б).



а) последовательный алгоритм б) схема распараллеливания

Рис. 1. Шифрование по ГОСТ 28147-89

На сегодняшний день известны параллельные реализации блочного алгоритма шифрования ГОСТ 28147-89 с использованием технологий OpenGL, DirectX и CUDA [2, 3].

В данной работе в качестве аппаратной платформы параллельного алгоритма шифрования ГОСТ 28147-89 предлагается использовать сопроцессоры Intel Xeon Phi с архитектурой Many Integrated Core (MIC). Представленная корпорацией Intel в 2012 г. аппаратная платформа гибридных многопроцессорных систем с многоядерными ускорителями Intel Xeon Phi совмещает в себе преимущества графических ускорителей с традиционной архитектурой x86. По результатам экспериментов компании-разработчика, гибридная многопроцессорная платформа с использованием MIC превосходит по производительности и эффективности многопроцессорные системы с графическими ускорителями [1].

Целью данной работы является создание параллельного алгоритма шифрования ГОСТ 28147-89, адаптированного для многопроцессорных систем с многоядерными ускорителями Intel XeonPhi.

Литература

1. Hughes C.J, Changkyu K., Yen-Kuang C. Performance and Energy Implications of Many-Core Caches for Throughput Computing // IEEE Micro. 2010. Vol. 3, No. 6. P. 25-35.
2. Коробицын В.В., Ильин С.С. Реализация симметричного шифрования по алгоритму ГОСТ 28147 на графическом процессоре // Информационные технологии. 2008. № 10. С. 46-51.
3. Коробицын В.В., Ильин С.С. Реализация симметричного шифрования по алгоритму ГОСТ 28147 на графическом процессоре с использованием технологии CUDA // Информационные технологии. 2011. № 4. С. 41-46.

Моделирование грид-системы CAEBeans*

П.А. Михайлов, А.В. Шамакина

Южно-Уральский государственный университет

В процессе разработки и эксплуатации распределенных вычислительных систем и систем грид возникают вопросы, связанные с изменением структуры данных систем, алгоритмов их работы и архитектуры. Особый интерес представляет исследование влияния этих изменений на ключевые параметры системы: производительность, отказоустойчивость, безопасность, масштабируемость и др.

Применение средств моделирования распределенных вычислительных систем позволяет решить основные задачи: проверка правильности работы алгоритмов системы; выявление «слабых мест» системы; тестирование различных сценариев использования грид; поиск оптимальной конфигурации ресурсов [1].

При этом оценка эффективности управления грид-средой может проводиться по следующим наиболее популярным критериям: минимизация среднего времени ожидания задачи в очереди; минимизация максимального времени выполнения группы задач; максимизация пропускной способности – числа завершенных задач в единицу времени; минимизация простоя процессора; оптимизация заполнения дискового пространства грид-ресурсов и др. [2].

На сегодняшний день существует большое число средств моделирования распределенных вычислительных систем, которые могут быть применены для решения поставленной задачи. Примерами таких средств могут служить OptorSim, GridSim и SimGrid.

Проектирование прототипа модели грид-системы CAEBeans [3, 4], предоставляющей программные ресурсы CAE-пакетов, производилось с использованием средства моделирования распределенных вычислительных систем SimGrid [5]. Базовая модель в пакете SimGrid строится на основе набора классов и двух конфигурационных файлов. Каждый из классов описывает конкретный элемент системы или некоторую структуру данных, пересылаемых между компонентами. Класс содержит логику работы реального компонента распределенной вычислительной системы. Конфигурационные файлы описывают моделируемую платформу, ее параметры и связи между элементами системы.

В работе приводится анализ существующих средств и пакетов для моделирования распределенных вычислительных систем. Разработан прототип модели грид-системы CAEBeans на основе пакета моделирования распределенных вычислительных систем SimGrid.

Литература

1. Кореньков В.В., Нечаевский А.В. Пакеты моделирования DataGrid // Системный анализ в науке и образовании. 2009. № 1. С. 21-35.
2. Грушин Д.А., Поспелов А.И. Система моделирования Grid: реализация и возможности применения // Труды Института системного программирования РАН. 2010. Т. 18. С. 243-260.
3. Радченко Г.И. Распределенные виртуальные испытательные стенды: использование систем инженерного проектирования и анализа в распределенных вычислительных средах. Вестник Южно-Уральского государственного университета. Серия «Математическое моделирование и программирование». 2011. № 37(254). Вып. 10. С. 108-121.
4. Шамакина А.В. CAEBeans Broker: брокер ресурсов системы CAEBeans // Вестник ЮУрГУ. Серия «Математическое моделирование и программирование». 2010. № 16(192). Вып. 5. С. 107-115.
5. Fujiwara K., Casanova H. Speed and accuracy of network simulation in the SimGrid framework // Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools. ICST. 2007. P. 12:1-12:10.

* Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 11-07-00478-а и Министерства образования и науки РФ (государственное задание 8.3786.2011).

Параллельный рендеринг воксельной графики

И.О. Михайлов

Институт математики и механики УрО РАН, УрФУ

В работе описывается проводимая в ИММ УрО РАН разработка программы параллельного рендеринга воксельной графики на многоядерном персональном компьютере и кластере, с разбиением данных на несколько частей.

В общем случае воксельная графика предусматривает хранение объекта в виде набора (облака) точек. Обычно данных представляются в виде набора узлов 3-х мерной сетки. Одним из основных преимуществом воксельной графики по сравнению с полигональной является возможность продемонстрировать внутреннюю структуру объекта. А основным недостатком - высокие требования по памяти и малое распространение специализированных аппаратных средств для расчёта графики.

Базовым алгоритмом рендеринга использованным в работе является трассировка лучей (точнее Ray casting). При этом из каждой точке изображения строится луч. Затем вычисляется ближайшая точка пересечения этого луча с областью данных (куб). Затем алгоритм движется по трёхмерному массиву данных с шагом в одну ячейку, до попадания в непустую ячейку. Основным преимуществом данного метода является большие возможности распараллеливания (вплоть до каждой точки изображения), а основным недостатком большая вычислительная сложность.

Для повышения производительности часто используются плотные октодеревевья в виде набора 3х-мерных массивов. Первый из них в 8 раз меньше массива исходных данных. Каждый последующий в 8 раз меньше предыдущего. В каждом элементе массива хранится количество непустых ячеек в соответствующей области основного массива (с данными). Это даёт возможность быстро пропускать пустые участки при трассировке лучей. Этот подход даёт те же преимущества что и разряженные октодеревья, кроме экономии памяти, однако значительно упрощает доступ к отдельным вокселям и их модификацию.

Для корректного отображения данных с полупрозрачными ячейками, применяется буфер, суммирующий цвета точек с учётом их прозрачности.

Область данных представляет собой 3-х мерный массив из 4-байтовых целых чисел, в котором ненулевые ячейки задают объект, а нулевые считаются пустыми. Ненулевые ячейки содержат информацию о цвете и прозрачности точек объекта (RGBA). Возможно расширения до 8-байтовых целых чисел, для хранения вспомогательной информации. В текущей реализации используются кубические массивы со стороной равной степени двойки (от 128 до 512). Для ускорения работы строится набор вспомогательных массивов, хранящих информацию о группах пустых ячеек для быстрого их пропуска.

Первые результаты были получены на многоядерном персональном компьютере. При этом использовалась технология OpenMP, для параллельного рендеринга точек изображения. Область данных была общая для всех потоков. При этом было получено ускорение близкое к линейному.

Затем эксперименты были перенесены на суперкомпьютер Уран ИММ УрО РАН, где для реализации использовалась библиотека MPI. При этом узлы были разделены на несколько групп. Области данных внутри одной группы дублировались. Сперва производился независимый рендеринг изображений в каждой группе, а затем полученные изображения совмещались в одно с учётом буфера глубины. В данной серии экспериментов основной задачей была проверка принципиальной возможности рендеринга больших объёмов данных (куб со стороной 4096).

Благодаря тому, что окончательное изображение получает сложением нескольких вспомогательных изображений с учётом буфера глубины, есть возможность использовать алгоритмы отличные от трассировки лучей. Например, некоторые данные возможно эффективней хранить в виде облака точек, с последующим проецированием на плоскость экрана. Описанный выше алгоритм можно использовать в средах виртуальной реальности.

Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-1-1034).

Расчетно-экспериментальное моделирование аэроупругих колебаний на кластере ПНИПУ

В.Я. Модорский, Л.Н. Бутымова

ФГБОУ ВПО «Пермский национальный исследовательский политехнический университет»

Значительных вычислительных ресурсов требуют для своего решения междисциплинарные задачи. Вместе с тем, совершенствование рабочих процессов в энергетических установках (ЭУ) сопряжено с решением ряда подобных проблем. В частности, необходима разработка расчетных моделей и методик, позволяющих прогнозировать опасные аэроупругие колебания в конструкции. В связи со сложностью задачи, для верификации результатов необходимо совместное использование ресурсов современных суперкомпьютеров и подходов физического эксперимента. На первом этапе изучение этой проблемы предлагается проводить по двум направлениям, независимо: численное моделирование волновых процессов в газе и численное исследование динамического напряженно-деформированного состояния конструкции (рис.1). Кроме того, на первом этапе разработана установка для проведения физических экспериментов (рис.1) и реализована подготовка методик проведения физических экспериментов. Получены экспериментальные амплитудно-частотные характеристики модельной камеры (рис.2). На втором этапе необходима реализация связанного численного решения и его верификация на базе результатов физических экспериментов.

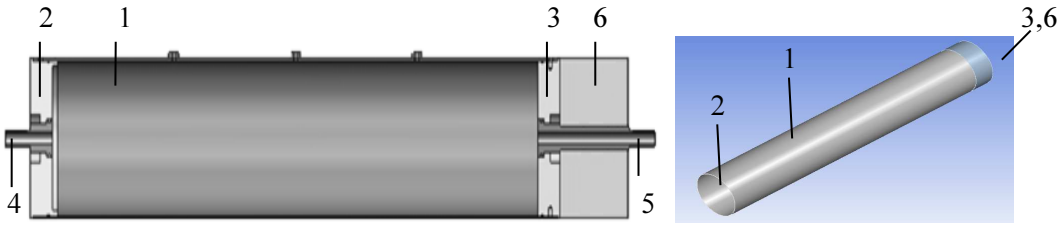


Рис. 1. Конструктивная и расчетная схемы экспериментальной установки:
1– корпус, 2, 3 – крышки, 4,5 – штуцеры, 6 – модельные элементы

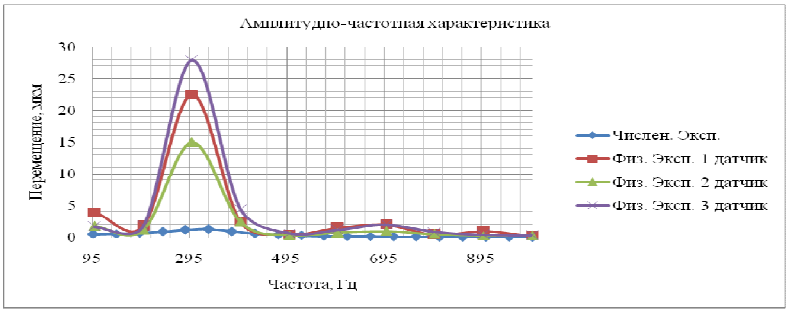


Рис. 2. Амплитудно-частотная характеристика модельной камеры

Выводы:

1. Отработана методика анализа колебательных режимов ЭУ с учетом условий закрепления в программном комплексе ANSYS WB на ВВК ПНИПУ.
2. Проведены экспериментальные и численные исследования по оценке влияния геометрических и массовых характеристик конструкции ЭУ, параметров газодинамической нагрузки на колебательные процессы в динамической системе «поток-конструкция».

Планируется провести численные эксперименты по оценке влияния физико-механических характеристик на характеристики колебаний в динамической системе «поток-конструкция», с помощью кластера ПНИПУ, верификация численных исследований будет проведена с использованием результатов физических экспериментов.

Аналитическая модель оценки эффективности выполнения параллельного кода на GPU

М.Д. Нгуен

Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

В данной работе описывается аналитическая модель A3, предназначенная для оценки эффективности выполнения параллельного кода на GPU. Теоретическая часть модели была разработана в рамках исследования по распараллеливанию метода SCF, входящий в состав пакета Siesta [1]. Поскольку модель A3 будет внедрена в систему автоматизации распараллеливания САПФОР [2] в качестве одного из инструментов для предсказания эффективности при распараллеливании последовательных Fortran программ, алгоритмы реализации модели были изменены с учетом требований САПФОР. Модель может быть использована для оценки возможности распараллеливания с помощью высокоуровневого языка Fortran DVMH и также для выбора оптимальных схем распараллеливания. Предлагаемая модель является востребованным инструментом для распараллеливания больших программ, время выполнения которых занимает от нескольких часов и более.

A3 статически анализирует регионы кода исходной программы, которые могут выполняться параллельно на GPU, и оценивает эффективность их выполнения без программирования и запуска на реальном GPU. По сравнению с существующей моделью оценки, входящей в состав фреймворка Grophecy [**Error! Reference source not found.**], A3 более универсальна, т.к. она не требует участия программиста в процессе работы. Такое преимущество возможно благодаря мощному анализатору системы САПФОР. Более того, A3 анализирует не только CWP (Compute Warp Parallelism) и MWP (Memory Warp Parallelism), но и ILP (Instruction Level Parallelism) и TLP (Thread Level Parallelism) – совокупность всех этих четырех факторов позволяет значительно быстрее сократить пространство поиска возможных схем распараллеливания. Другим преимуществом является статическое моделирование объема используемых регистров, что позволяет оценить степень занятости GPU до получения параллельного кода и его компиляции.

В модели A3 каждый параллельный регион рассматривается как отдельная функция-ядро (далее ядро). Для каждого ядра собираются характеристики, влияющие на возможность его распараллеливания и на эффективность выполнения. Эффективность выполнения ядра – это время его выполнения на GPU. Оптимальная схема распараллеливания ядра – это схема с минимальным временем выполнения. Оптимальная схема распараллеливания всей программы – это множество схем распараллеливания ядер, при которых время выполнения программы минимально.

Применение модели A3 позволяло распараллелить метод SCF за 11 дней. Ускорение на тестах, связанных с молекулой h2o (32h2o, h2o, h2o_2, h2o_4, h2o_basis, и др.) составило от 6x и выше. В данный момент ведется работа над внедрением модели A3 в систему САПФОР.

Литература

1. Siesta's Official Website. URL: http://icmab.cat/leem/siesta/Documentation/Publications/index.html (дата обращения: 02.12.2012).
2. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Труды Международной научной конференции "Научный сервис в сети Интернет: экзафлопсное будущее", Новороссийск, сентябрь 2011 – М.: Изд-во МГУ, 2011, С. 67-70.
3. J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, T. D. Uram. GROPHECY: GPU performance projection from CPU code skeletons // Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, November 2011.

Интервальный поиск параметров кинетической модели реакции аминометилирования тиолов с помощью тетраметилметандиамина*

А.В. Новичкова[1], К.Ф. Коледина[2], Л.Ф. Нурисламова[1], Ю.О. Бобренёва[2], И.М. Губайдуллин[1]

Институт нефтехимии и катализа РАН[1],
Башкирский Государственный Университет[2]

Для качественного построения кинетических моделей химических реакций необходимо постоянное сочетание натурного и вычислительного экспериментов. На основе кинетической модели можно варьировать начальные данные и режимы проведения реакции, прогнозировать эффективные новые натурные эксперименты с целью получения максимального выхода целевых продуктов за наименьшее время с минимальными затратами.

Азот- и серасодержащие органические соединения находят применение в качестве эффективных средств защиты растений, антиокислительных, противокоррозионных, противозадирных, противоизносных присадок к топливам и маслам. До настоящего времени одним из наиболее известных методов синтеза аминосульфидов остается классическая реакция аминометилирования тиолов по Манниху [1] с помощью вторичных аминов и альдегидов. В настоящей работе рассматривается аминометилирование тиолов с помощью тетраметилметандиамина (**Рис. 1**).

$$1) N_2(CH_3)_4 + [Sm] \rightarrow N_2(CH_3)_4 \cdot [Sm]$$
$$2) N_2(CH_3)_4 \cdot [Sm] + HSC_5H_{11} \rightarrow (CH_3)_2NSC_5H_{11} + (CH_3)_2NH + [Sm]$$

Рис. 1. Реакция аминометилирования тиолов с помощью тетраметилметандиамина

Нами был определен один набор кинетических параметров (константы скорости, энергии активации). В связи с тем, что экспериментаторам неизвестны данные о промежуточных веществах, полученный нами набор параметров не является единственным. Для полного физико-химического анализа о механизме реакции необходимо рассмотреть все возможные варианты. Таким образом, необходим поиск интервалов кинетических параметров.

Была разработана последовательная программа для определения двумерных и трехмерных областей неопределенности кинетических параметров. Планируется распараллелить данную программу [2] с использованием программной модели CUDA и протестировать ее на видеокарте NVIDIA GeForce GTS 250.

Литература

1. Новичкова А.В., Акманов Б.Ф., Миникеева Л.Р., Хайруллина Р.Р., Губайдуллин И.М. Кинетическая модель реакции аминометилирования тиолов//Дифференциальные уравнения и их приложения: Труды Всероссийской научной конференции с международным участием/Уфа:Гилем, 2011. С.320-323
2. Губайдуллин И.М., Аристархов А.В., Спивак С.И. Использование параллельных распределенных вычислений для определения областей пространства кинетических параметров // Труды международной научной конференции «Параллельные вычислительные технологии (ПаВТ'2009)». Челябинск: Изд-во ЮУрГУ, 2009. С. 439-443.

* Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации (Госконтракт №02.740.11.0631) и РФФИ (гранты №12-07-00324 и №12-07-31029).

Использование новых технологий памяти в системах хранения данных высокопроизводительных вычислительных систем

П.Г. Овчинников

Снежинский физико-технический институт – филиал федерального государственного автономного образовательного учреждения высшего профессионального образования «Национальный исследовательский ядерный университет «МИФИ»

Рост количества обрабатываемой информации в сфере высокопроизводительных вычислений ведет к тому, что системы хранения данных, основанные на магнитных жестких дисках, перестают удовлетворять требованиям к объему и производительности. Экстенсивное наращивание объема и использование массивов для повышения производительности системы хранения приводит к повышению тепловыделения, энергопотребления и занимаемой площади на единицу объема. Применение такого подхода затруднительно для будущих компьютеров экзафлопного класса.

Новая технология памяти storage-class memory (SCM) – разрабатываемая альтернатива устройств хранения – является эволюцией существующих твердотельных накопителей (solid state drives – SSD), однако с применением новых способов сохранения информации [1]. .Цель разработки – создание энергонезависимой твердотельной памяти с большим числом циклов перезаписи, высокой скоростью чтения/записи, низкой ценой за единицу объема и широким рабочим диапазоном температур. Память с такими характеристиками найдет широкое применение во многих типах компьютерных систем.

Использование storage-class memory в высокопроизводительных системах даст ощутимый прирост производительности системы хранения данных, и, как следствие, в системе ввода/вывода. Сравнение производительности RAID-0 массива из четырех жестких дисков с интерфейсом SATA-2 и моделей двух наиболее перспективных технологий SCM (phase-change memory(PCM)[2] и Spin-Torque Magnetoresistive RAM(ST-MRAM)) на шинах PCI-Express и DDR3[3] показало огромное преимущество новых технологий над магнитными накопителями.

Задачей исследования является изучения новой технологии памяти в контексте ее применения для высокопроизводительных вычислений. Целью исследования является создание концепции и, возможно, модели архитектуры высокопроизводительного кластера с применением storage-class memory.

Задача включает изучение физических и технологических особенностей SCM (PCM и ST-MRAM), изучение архитектуры современных суперкомпьютерных кластеров, оценку узких мест в работе системы хранения и анализ возможных вариантов применения SCM для решения существующих проблем, повышения производительности и надежности, снижения расходов на энергопотребление и обслуживание.

Литература

1. Storage-Class Memory: the next storage system technology. R.F. Freitas, W.W. Wilcke; IBM J. Res. & dev. Vol. 52 no. 4/5 july/september 2008
2. Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. *SIGARCH Comput. Archit. News* 37, 3 (June 2009), 2-13.
3. Adrian M. Caulfield, Joel Coburn, Todor Mollov, Arup De, Ameen Akel, Jiahua He, Arun Jagatheesan, Rajesh K. Gupta, Allan Snavely, and Steven Swanson. 2010. Understanding the Impact of Emerging Non-Volatile Memories on High- Performance, IO-Intensive Computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '10). IEEE Computer Society, Washington, DC, USA, 1-11. DOI=10.1109/SC.2010.56

Реализация вычислительного эксперимента в грид-системе с удаленным доступом

А.И. Огородников[1], С.И. Рябухин[2]

Уральский федеральный университет[1], Московский физико-технический институт[2]

На базе Техноцентра компьютерного инжиниринга Механико-машиностроительного института УрФУ сформирована инфраструктура грид-системы, позволяющая эффективно использовать машинный парк и удаленно управлять вычислительным экспериментом через глобальные или мобильные средства связи. Созданная система обеспечивает интенсивную эксплуатацию и окупаемость дорогостоящего лицензионного программного обеспечения, способного решать как образовательные, так и исследовательские задачи приборо- и машиностроения.

Грид-офис является эффективным методом производительных вычислений для исследовательской лаборатории или небольшой инжиниринговой компании. Метод позволяет связать компьютеры в вычислительную систему и рационально использовать аппаратные ресурсы в ночное или свободное время, когда рабочие места сотрудников не задействованы по прямому назначению. Экономически выгодно проводить вычислительные эксперименты на базе грид-офиса в университетах, где большое количество однотипных компьютеров вовлечено в учебный процесс по жесткому расписанию, но вынужденно простаивает, когда студенты уходят из аудиторий на каникулы, на практики или на сессии. Существенным доводом в пользу интенсивной эксплуатации машинного парка с привлечением грид-технологий является окупаемость дорогостоящего лицензионного программного обеспечения, способного решать как образовательные, так и исследовательские задачи.

Проблемной областью приложения высокопроизводительных вычислений является исследование, проектирование и оптимизация технологических процессов машино- и приборостроения. Преобладающим методом дискретизации математических моделей в этой области является метод конечных элементов. Математические модели технологий механической обработки носят комплексный характер, они должны отражать движение инструмента и контролируемое разрушение материала. Соответствующие вычислительные модели нелинейны по ряду признаков: динамические процессы, высокоскоростные процессы, жесткое нагружение, значительные геометрические изменения, изъятие материала, зависимость свойств материалов от температуры и величины нагрузки, взаимное перемещение объектов, изменение контактных условий, пластическое течение материала, переключение моделей поведения материалов в зависимости от расчетных результатов на текущем шаге.

В качестве вычислительного инструмента для моделирования технологических процессов механообработки выбран программный комплекс ANSYS, поскольку он имеет международную сертификацию в области инженерных расчетов и используется для расчетного обоснования проектов в машино- и приборостроении. Построена трехмерная нелинейная вычислительная модель без учета тепловых, инерционных и демпфирующих эффектов. На внутреннем языке программирования ANSYS APDL написан макрос, который позволяет выявить зависимость напряженно-деформированного состояния основного материала и покрытия от геометрии режущего инструмента, от толщины и свойств материала покрытия, от силовых параметров режима резания.

Для проведения вычислительных экспериментов построена параллельная система типа «грид-офис» на базе 10 ведомых компьютеров под управлением мастера с большей оперативной и дисковой памятью. Грид-офис представляет собой локальную вычислительную систему, но разрешает взаимодействие с пользователем через локальные, глобальные и мобильные средства связи. Сформированный грид-офис в дневное время обеспечивает очное обучение студентов или решает задачи активного электронного обучения по техническим дисциплинам, а в ночное время выполняет вычислительный эксперимент.

Численное моделирование на кластере ПНИПУ напряженно-деформированного состояния оправки для непрерывной намотки конструкций из композитов

О.В. Пищулина, В.Я. Модорский

ФГБОУ ВПО «Пермский национальный исследовательский политехнический университет»

Для эффективного использования труб из стеклопластика необходимо снизить их себестоимость до уровня стальных. Для этого необходимо уменьшить энергоемкость приводов, увеличить скорость наработки трубы.

Анализ существующих конструкций оправок (рис. 1а) показал, что возможна их существенная модернизация путем использования принципа уравновешивания действующих внешних сил (см. рис. 1б). Эта идея защищена патентом РФ (см. рис. 2).

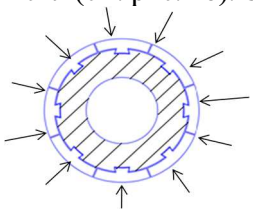


Рис. 1. Конструктивная схема
а – существующая конструкция б – новая конструкция

Рис. 2. ПАТЕНТ № 2390415

Твердотельная модель и граничные условия представлены на рис. 3. На рис.4 представлено напряженно-деформированное состояние оправки с учетом собственного веса и давления на наружную поверхность оправки при намотке. На рис. 5 показано выявленное в ходе вычислительных экспериментов влияние геометрических характеристик элементов оправки на следующие параметры: напряжения σ, перемещения вдоль оси OY Uy и массу m.

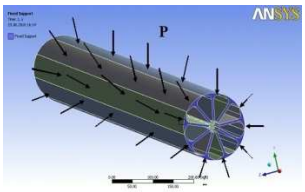


Рис. 3. Твердотельная модель и граничные условия

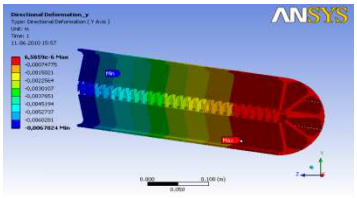


Рис. 4. Распределение перемещений (ось OY)

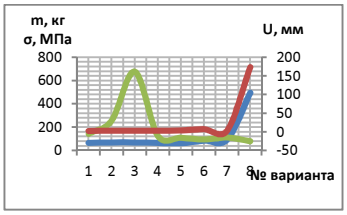


Рис. 5. Влияние параметров оправки на σ, U_y, m

Анализ полученных результатов позволил найти варианты конструкции с минимальной массой, с минимальными напряжениями и перемещениями, что явилось основой для выработки рекомендаций по проектированию таких конструкций.

Разработанная модель для оценки НДС оправки в дальнейшем будет использована для решения следующих задач:

- анализа масштабируемости кластера ПНИПУ при решении такого класса задач в ANSYS;
- анализа сходимости решения для такого класса задач;
- проведения многопараметрической оптимизации конструкции;
- разработки мобильных технологических комплексов для непрерывного производства труб различных типоразмеров.

Параллельные эвристические методы в обратных задачах химической кинетики на примере кинетики процесса ароматизации углеводородов C_5

Л.В. Сайфуллина[1], И.М. Губайдуллин[1], О.Ю. Забейворота[2]

Институт нефтехимии и катализа РАН[1], Башкирский государственный университет[2]

Каталитическое превращение легких предельных и непредельных углеводородов C_2-C_5 в более ценные продукты, такие как бензол, толуол и ксилолы (БТК), привлекает достаточно серьезное внимание различных исследователей. В данной статье рассматривается процесс ароматизации углеводородов C_5. Процесс описывается жесткими системами нелинейных дифференциальных уравнений, что приводит к необходимости использования параллельных вычислений при оптимизации параметров системы.

Каталитическое превращение легких предельных и непредельных углеводородов C_2-C_5 в более ценные продукты, такие как бензол, толуол и ксилолы (БТК), привлекает достаточно серьезное внимание различных исследователей [1]. В данной статье рассматривается процесс ароматизации углеводородов C_5.

Схема химических превращений и соответствующие им кинетические уравнения для реакции ароматизации углеводородов C_5H_{12} имеют вид [1]. Изменение концентраций веществ во времени рассматриваемых реакций описывается следующей системой обыкновенных дифференциальных уравнений.

Исследование процесса ароматизации углеводородов включает в себя решение как прямой задачи (решение системы при заданных начальных параметрах), так и обратной (восстановление параметров модели по имеющемуся экспериментальному материалу).

Решение обратной кинетической задачи сводится к прогону серии прямых задач и минимизации критерия отклонения расчета от эксперимента:

$$F = \sum_{i=1}^{N} \sum_{j=1}^{n} \left| x_{ij}^{calc} / x_{ij}^{exp} - 1 \right|, \; x_{ij}^{exp} > 0,$$

где x_{ij}^{calc} – расчетные значения; x_{ij}^{exp} – экспериментальные данные; N – количество точек эксперимента; n – количество веществ, участвующих в реакции.

Решение обратной задачи проводится параллельными вариантами генетического алгоритма и метода имитации отжига. Применение генетического алгоритма в обратных задачах химической кинетики получило широкое распространение ввиду несложности его распараллеливания на многопроцессорных системах, тогда как метод имитации отжига в данных задачах требует дальнейшего развития, что делает исследование актуальным. В этом подходе переменные, характеризующие решение, представлены в виде генов в хромосоме [2].

Имитация отжига (simulated annealing) представляет собой эвристическую процедуру поиска, которая допускает случайные переходы, что ведет к более дорогостоящим (и соответственно, худшим) решениям. Это выглядит как шаг назад, но позволяет удержать поиск от зацикливания на оптимальном локальном решении. Идея имитации отжига является аналогией физического процесса остывания расплавленных материалов и сопутствующего перехода в твердое состояние [2].

Планируется провести детальное сравнение данных методов на примере задачи оптимизации процесса ароматизации углеводородов C_5.

Литература

1. Кутепов Б.И., Белоусова О.Ю. Ароматизация углеводородов на пентасилсодержащих катализаторах. − М.: Химия, 2000. − 95с.

2. Скиена С. Алгоритмы. Руководство по разработке. – СПб: БХВ-Петербург, 2011. – 720 с.

Объектно-атрибутная модель программирования для параллельных вычислительных систем с управлением потоком данных

С.М. Салибекян[1], П.Б. Панфилов[2], Г.Н. Гончарук[1]

Национальный исследовательский университет «Высшая школа экономики»[1],
Минэкономразвития Российской Федерации[2]

Объектно-атрибутная (ОА) архитектура вычислительной системы (ВС) является разработкой Московского института электроники и математики национального исследовательского университета «Высшая школа экономики» (МИЭМ НИУ ВШЭ). ОА-архитектура обеспечивает эффективную работу параллельной ВС согласно модели организации вычислений с управлением потоком данных (так называемая dataflow-парадигма организации вычислений) как на программном, так и на аппаратном уровнях. ОА-ВС представляет особой совокупность параллельно работающих функциональных устройств (ФУ), производящих обработку данных и обменивающихся между собой «токенами» (операндами, снабженными служебной информацией), состоящими из двух полей: нагрузки (данные или указатель на данные) и атрибута (идентификатор нагрузки). ФУ в ОА-архитектуре ВС могут быть реализованы аппаратно или программно.

Программная реализация ФУ в ОА-модели программирования в какой-то степени напоминает «актор» в акторной модели программирования. Однако, в отличие от актора, ФУ осуществляет обмен информацией с «внешним миром» через универсальный интерфейс (т.е. все процедуры реализации алгоритма работы ФУ имеют одинаковый заголовок), через который операнды передаются по одному. И если в акторной модели связи между акторами задаются «статически» с помощью графического редактора или XML-подобного языка, то ОА-модель позволяет создавать и уничтожать ФУ, инициализировать ФУ, задавать и перестраивать связи между ФУ «динамически» непосредственно во время вычислительного процесса. К тому же, ОА-модель дает возможность динамически перераспределять вычислительные ресурсы ВС.

В настоящий момент осуществлена программная реализация ВС ОА-архитектуры: создана инструментальная среда ОА-программирования и моделирования, в состав которой входят ОА-платформа (совокупность программ виртуальных ФУ), включающая в себя более 70 типов ФУ; ОА-язык параллельного программирования; а также инструментальные средства, облегчающие процесс программирования и отладки параллельной программы. Стиль написания программ на ОА-языке идеально подходит для описания параллельного вычислительного процесса, что значительно упрощает работу программиста: программа естественным образом делится на параллельные блоки, и обеспечивается самосинхронизация работы этих блоков по данным. Среда также предоставляет возможность имитационного моделирования прикладной ОА-системы: настройка задержек вычислений на ФУ и при передаче данных по линии связи (шине данных-атрибутов), отслеживание модельного времени событий в ВС во время вычислительного процесса и т.п.

С помощью ОА-среды были решены некоторые прикладные задачи и осуществлено моделирование суперкомпьютерной системы ОА-архитектуры на тестовых задачах-бенчмарках, включая задачи сеточных вычислений из широкого класса задач физического моделирования, задачи на графах теста GRAPH500, задачи поиска подстрок в тексте теста GREP и др. В результате моделирования была доказана эффективность ОА-архитектуры для организации параллельного вычислительного процесса (на некоторых тестовых задачах обеспечивался линейный рост производительности ВС в зависимости от объема аппаратных ресурсов в ее составе).

В планы дальнейших исследований в области ОА-архитектуры параллельной ВС и ОА-модели программирования входит развитие инструментальной среды ОА-программирования и моделирования (усовершенствование языка ОА-программирования, доработка инструментальных средств программирования и отладки, расширение ОА-платформы путем добавления новых типов ФУ) и реализация прототипа ОА-ВС в «железе», например, на базе ПЛИС/FPGA.

Определение максимально эффективной реализации алгоритма на основе концепции Q-детерминанта

Д.И. Свирихин, В.Н. Алеева

Южно-Уральский государственный университет

В настоящее время параллельные вычисления используются практически во всех сферах научной деятельности. При этом одной из главных задач является выбор алгоритма для решаемой проблемы. Для решения данной задачи с приемлемыми затратами времени может быть использован автоматизированный анализ ресурса параллелизма различных алгоритмов. С этой целью было введено понятие Q-детерминанта алгоритма, позволяющее находить максимально быструю реализацию алгоритма и связанные с ней характеристики: количество процессоров и количество тактов работы, необходимых для ее выполнения [1]. На основе максимально быстрой реализации алгоритма может быть определена последовательность выполнения операций, требующая наименьших затрат времени при заданном количестве процессоров. Такая последовательность называется максимально эффективной реализацией алгоритма [2].

Данная работа состоит в формализации модели определения максимально эффективной реализации алгоритма для заданного количества процессоров, а также связанных с ней характеристик:

– последовательность вычислений – некоторая структура, описывающая порядок выполнения операций алгоритма (оказывает непосредственное влияние на время выполнения);

– время выполнения – количество тактов работы, необходимое для выполнения всех действий алгоритма;

– коэффициент замедления – величина изменения времени выполнения относительно максимально быстрой реализации алгоритма.

Основная задача построения максимально эффективной реализации алгоритма заключается в распределении операций между доступными процессорами таким образом, чтобы обеспечить минимальное время выполнения. В качестве исходных данных принимается максимально быстрая реализация алгоритма. В этом случае решение задачи сводится к оптимизации порядка выполнения операций, определенных максимально быстрой реализацией. С увеличением сложности алгоритма и повышением числа зависимостей между отдельными операциями, количество вариантов распределения будет также увеличиваться. В таких случаях задача построения максимально эффективной реализации алгоритма может оказаться достаточно ресурсоемкой, поэтому необходимо исследовать её возможные эвристические решения.

Использование описанной модели для различных алгоритмов дает возможность выбрать из них наиболее эффективный и масштабируемый. В дальнейшем планируется добавить к задаче построения максимально эффективной реализации алгоритма учет накладных расходов, что позволит автоматизировать определение числа процессоров, на котором алгоритм будет выполнен максимально эффективно.

Литература

1. Алеева В.Н. Анализ параллельных численных алгоритмов : Препринт №590. Новосибирск, 1985. 23с. В надзаг.: ВЦ СО АН СССР.

2. Свирихин Д.И. Определение ресурса параллелизма алгоритма и его эффективного использования для конечного числа процессоров // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (17-22 сентября 2012 г., г. Новороссийск). М.: Изд-во МГУ, 2012. С. 257-260.

Математическое и программное обеспечение решения обратных задач химической кинетики

С.И. Спивак[1,2], А.С. Исмагилова[1], А.А. Ахмеров[1]

Башкирский государственный университет[1], Институт нефтехимии и катализа РАН

Огромное значение для современной химической кинетики имеет интенсивное развитие вычислительной техники, появление все более быстродействующих компьютеров. Это позволяет вести статистическую обработку больших массивов экспериментальных данных по кинетике химических превращений, использовать для нахождения кинетических параметров, характеризующих отдельные стадии превращений, сложные, требующие большого объема вычислительной работы процедуры минимизации функции отклонения, рассчитывать протекание процессов, описываемых системами большого числа дифференциальных и алгебраических уравнений.

При исследовании химических реакций для практики нужно знать не только возможность осуществления данной реакции, но и скорость её протекания. Ответ на этот вопрос дает химическая кинетика. Для получения кинетических закономерностей должны быть известны не только начальное и конечное состояние системы, но и путь, по которому протекает реакция, а он обычно заранее не известен. Зная эти закономерности, т.е. математическую модель изучаемой реакции и ее кинетические параметры, можно рассчитать ее скорость и оптимальные условия проведения в промышленном реакторе. Целью настоящей работы является разработка программы, позволяющей находить линейно независимые маршруты сложных химических реакций.

Входными данными для разработанной программы является количество реакций, общее количество участвующих веществ и количество промежуточных веществ, а также коэффициенты стехиометрической матрицы. Программа позволяет найти линейно независимые векторы-маршруты и соответствующие итоговые уравнения, нарисовать граф механизма реакции и подграфы, соответствующие найденным маршрутам, вывести матрицу инцидентности, с помощью которой осуществляется поиск маршрутов. Также программа позволяет сохранить исследуемый механизм реакции и открыть раннее исследованные механизмы, сохранить в отчет полученные результаты.

С помощью разработанной программы были исследованы механизмы некоторых известных реакций. Результаты полностью удовлетворяли поставленным целям, т.е. программа находила все линейно независимые маршруты для каждой реакции, выводила соответствующие этим маршрутам итоговые уравнения и подграфы механизма реакции.

Исходя из проделанной работы и полученных результатов, мы можем сделать вывод, что построенная на описанных в исследовании алгоритмах математическая модель анализа механизмов сложных химических реакций адекватна. Также мы убедились, что разработанная программа, основой которой является наша математическая модель задачи, выполняется корректно и может быть применена при поиске маршрутов реакций.

В дальнейшем планируется продолжение тестирования программы путем исследования новых механизмов химических реакций, а также применение технологий параллельного программирования для исследования сложных механизмов.

Параллельные вычисления и задача расчета областей неопределенности в математической теории анализа измерений*

С.И. Спивак

Башкирский государственный университет, Институт нефтехимии и катализа РАН

Задачей настоящей работы является создание методов расчета и анализа областей неопределенности кинетических констант. Постановка задач определения интервалов при условии удовлетворения системы ограничений принадлежит Л.В.Канторовичу [1].

Постановка задачи обработки эксперимента, как она сделана Канторовичем, не требует знания информации о статистических свойствах распределения погрешности измерений.

Метод Канторовича в наших работах был развит применительно к решению обратных задач химической кинетики [2-3], анализе страховых тарифов в актуарных исследованиях по медицинскому страхованию [4], при определении параметров уравнений системной динамики [5]. Такое разнообразие областей применения говорит об общности используемых походов.

Способ нахождения области неопределенности методом перебора по точкам множества является весьма трудоемкой задачей и требует большого объема вычислений. Эффективным средством решения таких ресурсоёмких задач является использование технологии параллельных вычислений на распределенных системах кластерного типа.

Нами создано компьютерное обеспечение расчета интервалов неопределенности и областей неопределенности в двухмерном случае.

Основная проблема – расчет многомерных областей неопределенности. Возникающие задачи носят как математический, так и физико-химический характер. В частности, становится главной задачей физико-химическая интерпретация областей неопределенности. Эти проблемы являются предметом наших дальнейших исследований в этом направлении.

Литература

1. Канторович Л.В. О новых подходах в теории обработки наблюдений. // Сибирский математический журнал. 1962. Т.3. С.701-708.
2. Spivak S.I., Slinko M.G., Timoshenko V.I., Mashkin V.Yu. Interval estimation in the determination of parameters of a kinetic model. // Reaction Kinetics and Catalysis Letters. 1974. V.3, N1. P.105-113.
3. Спивак С.И. Информативность кинетических измерений. // Химическая промышленность сегодня. 2009. №9. С.52-56.
4. Спивак С.И., Райманова Г.К., Абдюшева С.Р. Обратные задачи для Марковских моделей медицинского страхования. // Страховое дело. 2008. №9(188). С.36-42.
5. Спивак С.И., Кантор О.Г. Качество моделей математической обработки наблюдений социально-экономических систем // Системы управления и информационные технологии. 2012. № 2(48). С.44-49.

* Работа выполнена в рамках исследований по Российскому фонду фундаментальных исследований, проект № 11-01-97020.

Распараллеливание механизмов сложных химических реакций при решении обратных задач химической кинетики*

С.И. Спивак[1,2], А.С. Исмагилова[1]

Башкирский государственный университет[1], Институт нефтехимии и катализа РАН[2]

Основным моментом в обратной задаче является представление кинетической модели в виде, содержащем независимые комбинации констант. Если число таких комбинаций меньше общего числа констант, то в технологических целях удобно иметь дело с моделями, содержащими меньшее число параметров. Понятно, что это крайне трудоемкий процесс [1].

Для устранения данной проблемы построен алгоритм анализа информативности кинетических измерений с учетом их погрешности, который позволяет распараллелить исходную задачу анализа выделения базиса нелинейных параметрических функций (НПФ) кинетических констант на более простые подзадачи с использованием маршрутов.

Важной особенностью маршрута является тот факт, что часть компонент в маршруте обычно равна нулю. Это значит, маршрут выделяет из всей совокупности стадий некоторую подсистему, в которую входит только часть стадий исходного механизма. Основная идея предлагаемого метода состоит в том, что вместо анализа информативности для всей сложной схемы реакций рассматриваются те схемы, которые отвечают за протекание реакции по каждому из независимых маршрутов.

Справедлива теорема [2]:

Теорема 1. Маршрут реакции есть циклический подграф исходного графа. Объединение таких подграфов образует полный граф, т.е. граф исходной системы реакций. Число независимых маршрутов равно числу независимых циклов графа Вольперта.

Основным результатом является следующая теорема:

Теорема 2. Совокупность стадий химической реакции можно разбить на подсистемы, в которые входят части стадий исходного механизма. Число таких подсистем равно числу независимых маршрутов. Объединение базисов НПФ кинетических параметров, допускающих однозначное оценивание, совпадает с базисом НПФ исходной сложной системы реакций.

Таким образом, алгоритм анализа информативности можно разложить на следующие этапы:

1. Выделение числа и вида независимых маршрутов реакции.
2. Декомпозиция исходной системы на подсистемы меньшей размерности в числе, равном числу независимых маршрутов, соответствующим базисным итоговым уравнениям.
3. Анализ информативности каждой из подсистем.
4. Объединение полученных базисов нелинейных параметрических функций в единый базис, соответствующий сложному механизму протекания реакции.

Литература

1. Спивак С.И., Исмагилова А.С. Информативность кинетических измерений при определении параметров математических моделей химической кинетики // Журнал СВМО. 2009. Т.11, №2. С.131-136.
2. Спивак С.И., Исмагилова А.С., Хамитова И.А. Теоретико-графовый метод определения маршрутов сложных химических реакций // ДАН. 2010. Т. 434, № 4. С.499-501.

* Работа выполнена в рамках исследований по Российскому фонду фундаментальных исследований, проект № 11-01-97020.

Влияние подсистемы межпроцессного взаимодействия на эффективность работы параллельных программ*

Г.В. Тарасов, Д.И. Харитонов

Институт автоматики и процессов управления ДВО РАН

Общепринятой единицей измерения производительности вычислительных систем является количество операций с плавающей точкой в секунду (FLOPS). На основе этой единицы измерения строятся мировой рейтинг суперкомпьютеров Top500 и рейтинг Российских суперкомпьютеров Top50. Однако, с точки зрения пользователя эту величину не всегда следует рассматривать как объективный показатель. Она лишь говорит о том, что на вычисление решения системы линейных уравнений заданной размерности и заданным методом будет затрачено некоторое известное (вычислимое по формуле) количество операций, и процесс вычисления займет некоторое время. Если алгоритм задачи пользователя коррелирует с алгоритмом, используемом в HPL Benchmark, то тогда пользователю важен результат теста HPL и, зная его для заданной вычислительной машины, пользователь в состоянии оценить временные затраты, требуемые на решение его задачи. В противном случае возникает необходимость проводить дополнительное и всестороннее тестирование искомой задачи.

В данной работе авторы исследуют вопросы поиска новых частных характеристик производительности вычислительной системы, которые бы отражали ее архитектурные особенности и позволяли бы оценить временны́е и ресурсные затраты на исполнение параллельных программ. В качестве начального направления исследований на различных архитектурах рассматривается эффективность работы стандартной библиотеки pthreads (POSIX Threads), повсеместно используемой на системах с общей памятью как при создании собственных приложений, так и в качестве базиса для построения более высокоуровневых средств реализации параллелизма в прикладных программах. Для данной библиотеки разработан синтетический тест, который замеряет время выполнения некоторой pthread-функции при определенном общем уровне загрузки вычислительной системы. В частности, рассматривается скорость выполнения функции pthread_create, отвечающей за порождение нового потока управления в параллельной программе. Нагрузка на вычислительную систему формируется созданием группы потоков.

В работе вводится частная характеристика *отношения времени создания одного параллельного потока в рамках группы к общему количеству потоков в этой группе*. Используя данный синтетический коэффициент, для различных вычислительных системах с процессорами Intel и AMD были получены следующие основные результаты. Наблюдается примерно 7-10% превосходства процессоров Intel (Xeon L5609 vs. Opteron 6164HE). Протестированные системы на базе Intel процессоров, имеющихся в распоряжении авторов, расположились в следующем порядке убывания коэффициента: Core-i5-3550, Xeon L5609, QuadCore-Q8600. Разница в скорости выполнения функции pthread_create в приведенной группе процессоров составляет 3-5 мкс и коррелирует с частотой процессора. Также следует отметить несколько фактов, выявленных в результате тестирования и влияющих на скорость выполнения параллельных программ. Включение флага affinity при запуске программ может давать до 30% сокращения времени накладных расходов за счет уменьшения затрат на переключение контекстов потоков между физическими ядрам процессоров. Запуск параллельных программ на процессорах Intel Xeon L5609 с 4 потоками оказывается более эффективным, чем с 8 потоками. В среднем время создания одного параллельного потока увеличивается до 30 раз при полной нагрузке на все 8 ядер.

*Работа выполнена при финансовой поддержке грантов ДВО №12-I-П18-03, №12-III-A-01И-015.

Об одном высокомасштабируемом методе численного решения уравнений математической физики

А.Б. Терентьев, С.А. Савихин, С.А. Золотов, В.В. Мохин, А.А. Панкратов, Р.М. Дмитриенко

ООО «Научно-исследовательский центр специальных вычислительных технологий»

1. Предлагаемый подход

В основе предлагаемого решения лежит использование локальных правил для описания глобальных процессов. При этом взамен больших сложно связанных систем дифференциальных уравнений для описания глобального поведения системы используется множество описаний поведения и взаимодействия её малых частей (локальные правила). В силу локальности правил распараллеливание происходит на уровне математического метода (алгоритма), из чего следует возможность практически бесконфликтного распределения задач на большое число вычислителей. Этот подход также позволяет представить неоднородность среды, граничные условия, взаимозависимости в виде набора параметров для каждой из подобластей пространства, что дополнительно повышает регулярность решения.

2. Область применения

1. Задачи, непосредственно связанные с потоками и течениями: токи жидкостей, газов и сыпучих тел, обтекание тел сложной формы, распределение температур и давления в токе жидкости или газа, атмосферные процессы.

2. Волновое моделирование: движение волн и их влияние на берег, тектонические процессы и землетрясения, электрические, магнитные, электромагнитные поля и свет, сети связи и другие информационные потоки.

3. Задачи, связанные с моделированием деформации твердого тела и распространения изломов.

4. Моделирование биологических и социологических процессов: биологические и клеточные популяции, миграции в обществе и в животном мире, потоки людей, транспорта, грузов.

5. Совместное решение разнородных (многофазных, многокомпонентных, мультифизичных) задач, в том числе с обратной связью.

3. Наработки по теме

На сегодняшний день решатель реализован на гетерогенном кластере (CPU+GPU). Тестирование, проведённое на традиционных задачах гидродинамики (обтекание твердого тела потоком жидкости или газа) на небольшой вычислительной системе (до 80 GPU), показало практически линейную (94%) масштабируемость. В настоящее время продолжается тестирование на кластерах большего масштаба (свыше 200 GPU) и на смешанных задачах гидро-газо-динамики.

Моделирование гомогенной нуклеации на гибридных суперкомпьютерах*

А.О. Типеев, К.С. Бобров, В.Г. Байдаков

ФГБУН Институт теплофизики УрО РАН

На суперкомпьютерах гибридного типа с использованием методов молекулярной динамики и Монте-Карло смоделированы процессы кристаллизации, плавления и кавитации леннард-джонсовсого вещества. Расчеты велись в свободно распространяемых распараллеленных программных комплексах LAMMPS [1] и HOOMD [2], модифицированных под конкретную архитектуру суперкомпьютера.

Исследовалась начальная стадия фазовых переходов – процесс зародышеобразования (нуклеации) – в системах, содержащих до 10 миллионов леннард-джонсовских частиц. Использование суперкомпьютеров значительно ускоряет процесс интегрирования уравнений их движения. Нами использовалось до 512 CPU и 8 GPU.

Применялись методы transition interface sampling [3], среднего времени жизни [4], среднего времени первого перехода [5] в компьютерном эксперименте. Совокупность данных методов позволила изучить процесс нуклеации в интервале частот зародышеобразования от 10^{20} до 10^{35} $м^{-3}с^{-1}$, достигнув значений, фиксируемых в натурных экспериментах. Расчеты проведены по десяти изотермам в областях положительного и отрицательного давлений. Смоделировано более 10000 событий нуклеации. Показано, что их временное распределение при фиксированных термодинамических параметрах системы хорошо описывается законом Пуассона.

Результаты моделирования кинетики фазовых переходов сопоставлялись с расчетами по классической теории гомогенной нуклеации. Оценена поверхностная свободная энергия на искривленной границе критический зародыш–материнская фаза и ее температурная зависимость при кристаллизации и кавитации метастабильной жидкости. Определены характерные параметры критических зародышей.

Обнаружены термодинамические состояния растянутой жидкости, в которых события кристаллизации и кавитации реализуются с равной вероятностью.

Процессы нуклеации визуализированы. Для этого разработаны модули идентификации принадлежности частиц к той или иной фазе.

Литература

1. Plimpton S. Fast Parallel Algorithms for Short-Range Molecular Dynamics // J. Comput. Phys. March 1995. Vol. 117, No 1. P. 1-19. (URL: http://lammps.sandia.gov)
2. URL: http://codeblue.umich.edu/hoomd-blue
3. van Erp T. Moroni D., Bolhuis P. A novel path sampling method for the calculation of rate constants // J. Chem. Phys. January 2003. Vol. 118, No. 17. P. 7762(13).
4. Скрипов В.П. Метастабильная жидкость. — М.: Наука, 1972. — 312 с.
5. Wedekind J., Strey R., Reguera D. New method to analyze simulations of activated processes // J. Chem. Phys. April 2007. Vol. 126, No. 13. P. 134103(7).

* Работа выполнена в рамках программы Президиума РАН № 18 "Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности" при поддержке УрО РАН (проект 12-П-2-1049).

Параллельная реализация модифицированного алгоритма реконструкции трехмерной сцены по стереоизображениям*

В.А. Фурсов, С.А. Бибиков, Е.В. Гошин, Д.А. Жердев

СГАУ им. академика С.П.Королёва[1], ИСОИ РАН[2]

В работе предлагается технология построения карты глубин, ориентированная на ее последующую параллельную реализацию. Технология позволяет в значительной степени исключить влияние ряда источников погрешностей. В работе используется модель камеры-обскуры [1]. Рассматривается случай, когда параметры камер, а также их координаты и ориентация известны. Для их характеристики вводятся матрицы параметров камер [2] $\mathbf{K}_1$ и $\mathbf{K}_2$.

Пусть $\mathbf{M}$ - координаты некоторой точки в глобальной системе координат. Координаты этой точки в системах координат первой и второй камер определяются как

$$\begin{cases} \mathbf{m}_1 = \mathbf{K}_1 \left[\mathbf{R}_1 \vdots \mathbf{t}_1 \right] \mathbf{M}, \\ \mathbf{m}_2 = \mathbf{K}_2 \left[\mathbf{R}_2 \vdots \mathbf{t}_2 \right] \mathbf{M}. \end{cases}$$

Здесь $\mathbf{R}_1$, $\mathbf{R}_2$ - матрицы размерности 3×3, описывающие поворот систем координат первой и второй камер относительно глобальной, а $\mathbf{t}_1$, $\mathbf{t}_2$ – векторы координат начала глобальной системы координат в системах координат первой и второй камер, соответственно.

Пусть некоторая точка $\hat{\mathbf{m}}_1$ – некоторая точка на первом виде, принадлежащая эпиполярной плоскости $\mathbf{\Pi}$, а $\mathbf{l}_1$, $\mathbf{l}_2$ – соответствующие этой точке эпиполярные прямые. При точном задании матриц проекций соответствующая точка $\mathbf{m}_2$ на втором виде обязана лежать на прямой $\mathbf{l}_2$. В данном случае для $\mathbf{m}_2$ ищется ее оценка – $\hat{\mathbf{m}}_2$, а оценка $\hat{\mathbf{M}}$ пространственных координат точки $\mathbf{M}$ в плоскости $\mathbf{\Pi}$ определяется как точка пересечения лучей ($\mathbf{c}_1$, $\hat{\mathbf{m}}_1$) и ($\mathbf{c}_2$, $\hat{\mathbf{m}}_2$), где $\mathbf{c}_1$, $\mathbf{c}_2$ - центры соответствующих камер.

Основанная на указанном подходе технология определения трехмерной модели сцены строится путем «сканирования» трехмерного пространства пучками эпиполярных линий на пучке эпиполярных плоскостей. Определение соответствующих точек на эпиполярных линиях осуществляется путем сравнения малых фрагментов, ориентированных (по углу) в соответствии с направлениями этих линий.

Алгоритм — параллельный по обрабатываемым эпиполярным линиям. Поскольку длины эпиполярных линий не одинаковы, решалась задача повышения эффективности параллельной обработки за счет формирования заданий на обработку процессам по мере их освобождения. Для сравнения вычислительной сложности и оценки эффективности реализации алгоритмов [3] проведены эксперименты как с использованием тестовых стереоизображений, полученных путем моделирования, так и на реальных изображениях.

Литература

1. Форсайт Д., Понс Ж. Компьютерное зрение. Современный подход. М.: Издательский дом "Вильямс", 2004. 928 с.
2. Цифровая обработка изображений в информационных системах: Учебное пособие / И.С. Грузман [и др.] Новосибирск: Изд-во НГТУ, 2002. 352 с.
3. Гергель В.П., Фурсов В.А. Лекции по параллельным вычислениям: Учебное пособие. Самара: Издательство СГАУ, 2009. 164 с.

* Работа выполнена при поддержке Программы № 14 фундаментальных исследований Президиума РАН и РФФИ (проект № 12-07-00581-а).

Этапы развития методики моделирования взаимодействия излучения с веществом. Доплеровские контуры

М.А. Чащин, Л.И. Рубина, О.Н. Ульянов

Институт математики и механики им. Н.Н. Красовского УрО РАН.

Авторы на протяжении ряда лет разрабатывают методику численного решения задачи о взаимодействии ионизирующего излучения с веществом [1,2]. Населенности уровней вычисляются из системы уравнений кинетики с коэффициентами, зависящими от интенсивности излучения, которая, в свою очередь, определяется из уравнения (или системы уравнений) переноса с коэффициентами, зависящими от населенностей уровней. В методике реализуются два метода решения задачи, предложенных и разрабатываемых авторами [1,2].

По мере развития вычислительной техники и параллельных технологий, совершенствования используемых авторами математических моделей, разрабатываемых ими численных алгоритмов и параллельных программ, методика совершенствуется, постановки задач все больше приближаются к описанию реальных процессов.

В докладе излагаются результаты, полученные для веществ с доплеровскими профилями излучения и поглощения при использовании новой (не применявшейся авторами ранее в этом случае), более полной математической модели процесса.

В этой модели:

- интенсивность излучения определяется из единого спектрального уравнения переноса излучения без разбиения на "непрерывную" и "дискретную" части, в отличие от ранее использовавшейся модели, в которой спектральное уравнение переноса разбивалось на систему стационарных интегро-дифференциальных уравнений переноса излучения в "дискретном спектре" (в каждой спектральной линии) и уравнение переноса излучения в "непрерывном спектре";
- расчет вклада каждой линии производится с учетом их возможного пересечения (ранее пересечением спектральных линий пренебрегалось);
- система уравнений дополнена уравнением энергобаланса.

Разработаны параллельные алгоритмы и программы, предполагающие (на данном этапе), что слой неподвижного однокомпонентного вещества, на который падает излучение, однороден, и учитываются две спектральные линии. Численные расчеты проводились на суперкомпьютере УРАН. Результаты, полученные при реализации двух различных методов, достаточно близки. Выяснены условия, при которых более полная математическая модель процесса взаимодействия излучения с веществом, дает результаты, совпадающие с результатами, полученными при использовании модели с прежним приближением, а при каких — значительно отличающиеся.

Дальнейшее развитие методики моделирования взаимодействия излучения с веществом для доплеровских контуров излучения предполагает: увеличение количества спектральных линий, количества слоев вещества, рассмотрение смесей веществ, а также областей, состоящих из возможно движущихся слоев неоднородных по плотности, скорости, составу смеси.

Литература

1. Е.Ф. Леликова, Л.И. Рубина, О.Н. Ульянов, М.А. Чащин Параллельные вычислительные технологии в задаче о переносе радиационного излучения // Вопросы атомной науки и техники. Сер. Мат. моделирование физ. процессов.- 2002.- Вып.3.- С.3-13.
2. М.А. Чащин, Л.И. Рубина, О.Н. Ульянов // Тр. конференции. ПаВТ`2012.- Новосибирск, 2012.- С.313-324.

Индекс по фамилиям

Содержание

Полные статьи

Короткие статьи

Плакаты

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ
(ПаВТ'2013)

Труды международной научной конференции
(г. Челябинск, 1–5 апреля 2013 г.)

Издательский центр Южно-Уральского государственного университета

Подписано в печать 15.03.2013. Формат 60 × 84 1/8.
Усл. печ. л. 73,47. Заказ 39.
