ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2013
Т. 2, № 3

ISSN 2305-9052

СЕРИЯ

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»

ПОСВЯЩАЕТСЯ 70-ЛЕТИЮ ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

Учредитель — Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Южно-Уральский государственный университет» (национальный исследовательский университет)

Основной целью издания является пропаганда научных исследований в следующих областях:

- Вычислительная математика и численные методы
- Математическое программирование
- Распознавание образов
- Вычислительные методы линейной алгебры
- Решение обратных и некорректно поставленных задач
- Доказательные вычисления
- Численное решение дифференциальных и интегральных уравнений
- Исследование операций
- Теория игр
- Теория аппроксимации
- Информатика
- Математическое и программное обеспечение высокопроизводительных вычислительных систем
- Системное программирование
- Распределенные вычисления, облачные и грид-технологии
- Технология программирования
- Машинная графика
- Интернет-технологии
- Системы электронного обучения
- Технологии обработки баз данных и знаний
- Интеллектуальный анализ данных

BULLETIN

OF THE SOUTH URAL STATE UNIVERSITY

2013
Vol. 2, no. 3

ISSN 2305-9052

SERIES

«COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING»

South Ural State University

The main purpose of the series is publicity of scientific researches in the following areas:

- Numerical analysis and methods
- Mathematical optimization
- Pattern recognition
- Numerical methods of linear algebra
- Reverse and ill-posed problems solution
- Computer-assisted proofs
- Numerical solutions of differential and integral equations
- Operations research
- Game theory
- Approximation theory
- Computer science
- High performance computer software
- System programming
- Distributed, cloud and grid computing
- Programming technology
- Computer graphics
- Internet technologies
- E-learning
- Database and knowledge processing
- Data mining

Содержание

Contents

УДК 004.001.85

ПРАВДА, ИСКАЖАЮЩАЯ ИСТИНУ. КАК СЛЕДУЕТ АНАЛИЗИРОВАТЬ TOP500?[1]

С.М. Абрамов

После каждого выпуска рейтинга Top500 выполняются подсчеты и публикуются суждения, вида: «Подавляющее большинство суперкомпьютеров списка Top500 используется в промышленности». Появляются и другие подобные подсчеты и суждения о долях в списке Top500 разных типов процессоров, различных типов интерконнекта, производителей суперкомпьютеров, стран и т.п. Часто на базе подобных суждений принимаются серьезные решения, в том числе и на правительственном уровне.

В данной работе показано: все, что фиксируется в подобных суждениях — правда, однако эта правда серьезно искажает истину и не отражает истинное положение дел в суперкомпьютерной отрасли. Кроме того, дается анализ причины серьезного отличия «правды» от «истины», приводятся методика корректного анализа данных Top500 и результаты такого анализа.

Ключевые слова: рейтинг Top500, использование суперкомпьютеров, высокопроизводительные вычисления.

Введение

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

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 из *Top500*, или какие доли приходятся на суперкомпьютеры, использующие те или иные микропроцессоры. Анализируют и другие процентные распределения: 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 (http://s.Top500.org/static/lists/2012/11/TOP500_201211_Poster.pdf) и рассмотрим диаграмму *Installation Type* (рис. 1). В диаграмме 40 столбцов — каждый столбец соответствует одному выпуску рейтинга, на один год приходятся два столбца (июнь и ноябрь). Столбец состоит из частей разных цветов; размер частей определяется долями различных

[1] Статья рекомендована к публикации Программным комитетом международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2013».

сегментов применения суперкомпьютеров из соответствующего рейтинга *Top500*. Различают шесть значений для сегментов применения: *Vendor, Research, Industry, Government, Classified и Academic*.

Действительно, легко взять полную *Excel*-таблицу (см. список *Top500* за ноябрь 2012 года http://s.Top500.org/static/lists/2012/11/TOP500_201211.xls) и посчитать, сколько суперкомпьютеров в колонке *Segment* имеют то или иное значение области применения. Результат представлен ниже (табл. 1). Доли, посчитанные в третьей колонке, естественно, в точности соответствуют длинам цветных частей правого столбца диаграммы *Installation Type* (рис. 1). Тем самым, будет справедливым следующее суждение:

§1 В ноябре 2012 года самая бо́льшая часть (49,40 %) суперкомпьютеров использовалась в промышленности (Segment=Industry). При этом индустриальное применение превосходило научное применение (44,2 % = 24,6 % + 19,6 %, Segment=Research и Segment=Academic).

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

§2 В ноябре 2009 года в промышленности использовалась подавляющая часть (62,4 %) суперкомпьютеров. При этом индустриальное применение значительно (почти вдвое) превосходило применение для научных задач (34 % = 18,2 % + 15,8 %).

Таблица 1

Распределение суперкомпьютеров по «сегментам» применения
(*Top500* за ноябрь 2012 г.)

Применение (колонка «*Segment*»)	**Количество систем**	**Доля**
Vendor	12	2,4 %
Research	123	24,6 %
Industry	247	49,4 %
Government	16	3,2 %
Classified	4	0,8 %
Academic	98	19,6 %
ВСЕГО	**500**	**100 %**

Подобные вычисления и суждения (§1, §2) сделать легко — для этого не нужно быть бо́льшим специалистом, достаточно начальных навыков владения программой *Excel*. Более того, график *Installation Type* входит в официальный плакат рейтинга *Top500* и очень наглядно иллюстрирует распределение суперкомпьютеров по так называемым сегментам применения и то, как с течением времени меняется это распределение.

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

§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): порядка 35 % от государства, порядка 65 % ВБС от индустрии (см. §2).

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

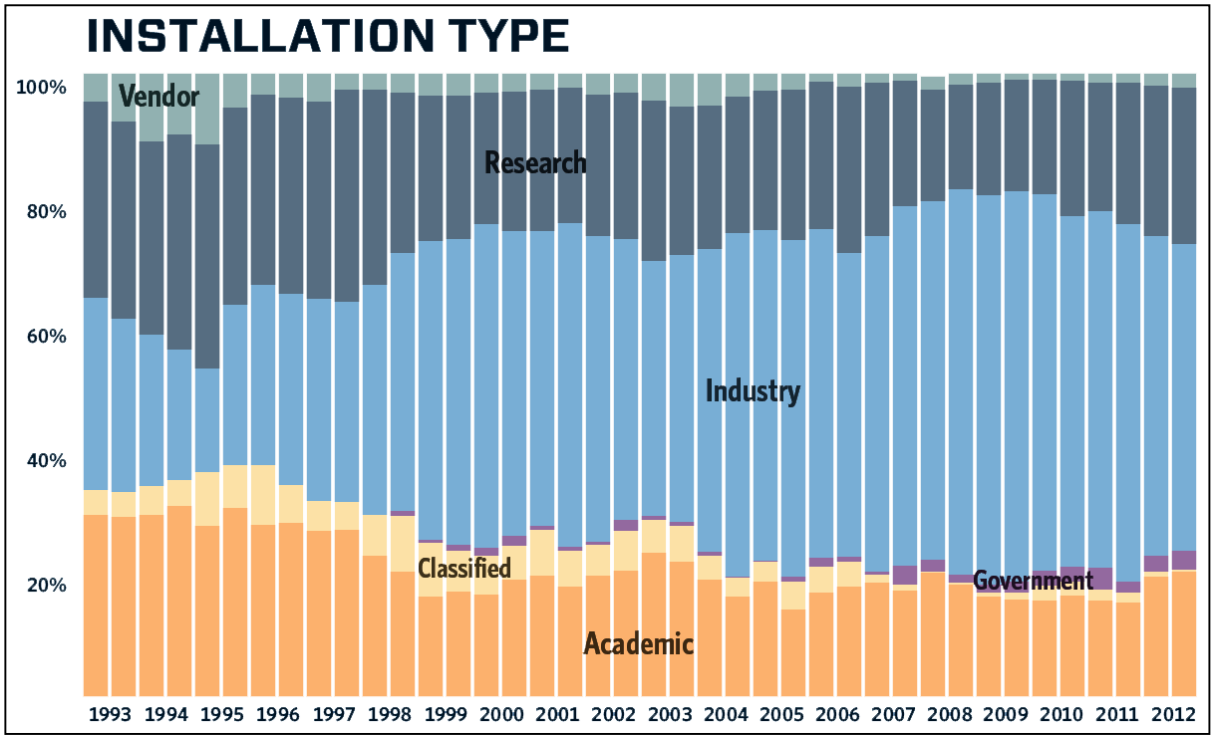


Рис. 1. Диаграмма *«Installation Type»* с плаката рейтинга *Top500* за ноябрь 2012 г.

Если же посчитать истинные доли (как их вычислять — рассмотрим в последующих разделах), приходящиеся на различные сферы применения суперкомпьютеров (табл. 2), то увидим, что различие между «правдой» (колонка A) и «истиной» (колонка B) оказывается весьма значительным — в разы. Степень искажения истины — самая правая колонка — вычисляется как $max(A,B)/min(A,B)$; она указывает, во сколько раз «правда» приуменьшает (знак «↓» перед числом) или преувеличивает (знак «↑») «истину».

Столь же разительно отличается от §1 истинное суждение:

§6 В ноябре 2012 года подавляющая доля производительности суперкомпьютеров (77,67 % = 59,23 % + 18,44 %) была использования в науке (Segment=Research и Segment= Academic), что многократно (в 4,4 раза) превосходит долю использования суперкомпьютеров в промышленности (17,56 %, Segment=Industry).

Серьезное (в разы) отличие «правды» от «истины» показывает недопустимость использования графика *Installation Type* (рис. 1) и суждений §1 и §2 для обоснования любых

управленческих решений. На их основе легко сделать ложные выводы и, как результат, — принять вредные управленческие решения (например, §3 и §4).

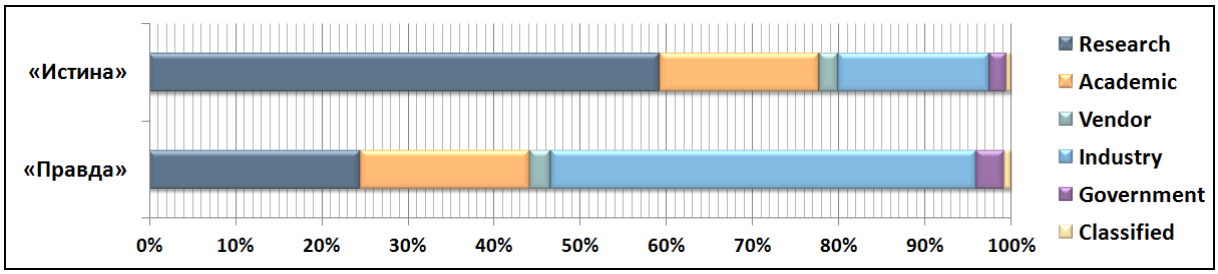
Распределение долей вычисляют не только в отношении применения суперкомпьютеров. В общем случае, если суперкомпьютеры некоторой редакции Top500 каким-то образом разбиты на категории, то процентные доли этих категорий можно посчитать двумя способами:

- *A* — по общепринятой процедуре, когда вычисляются доли числа суперкомпьютеров (среди всех 500 систем), соответствующих каждой категории;
- *B* — вычислить истинные доли категорий (методика подсчета обсуждается в разделе 3).

Таблица 2

Истинное распределение долей различных «сегментов» применения суперкомпьютеров (*Top500* за ноябрь 2012 г.)

Сегмент	**(A) «Правда»: доля систем (табл. 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 %**	**100 %**	



На основе данных *Top500* за ноябрь 2012 года были построены таблицы долевого распределения для следующих категорий:

- табл. 3 — используемая технология интерконнекта: *Infiniband, Ethernet, Myrinet* или *Custom* (*Custom* — интерконнект, коммерчески недоступный на рынке, по крайней мере, как отдельный продукт. Если надо использовать такой, то аналог придется разрабатывать самостоятельно);
- табл. 4 — компания-производитель (*IBM, Hewlett-Packard, Cray Inc.* и все остальные).

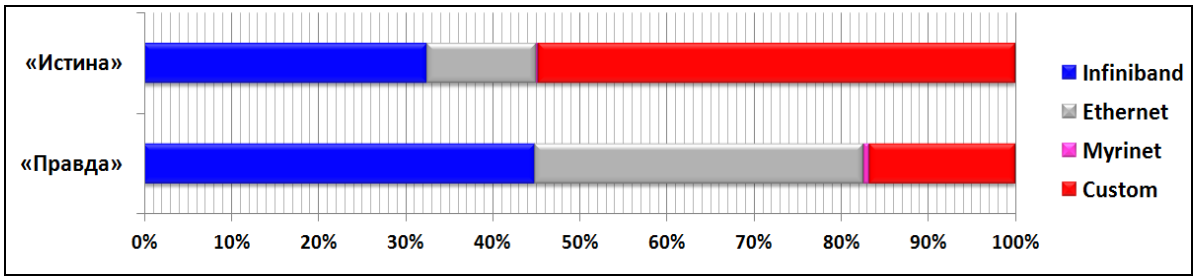
Во всех рассмотренных случаях очень часто «правда» сильно (в разы) отличается от «истины» (см. правую колонку таблиц). Вот несколько примеров:

- табл. 2 — в колонке *«A»* доля сегмента *Industry* преувеличена в 2,8 раза, а доля сегмента *Research* преуменьшена в 2,4 раза;
- табл. 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 3

Распределение долей между разными технологиями интерконнектов, используемых в суперкомпьютерах (по сведениям *Top500* за ноябрь 2012 г.)

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 %**	**100 %**	

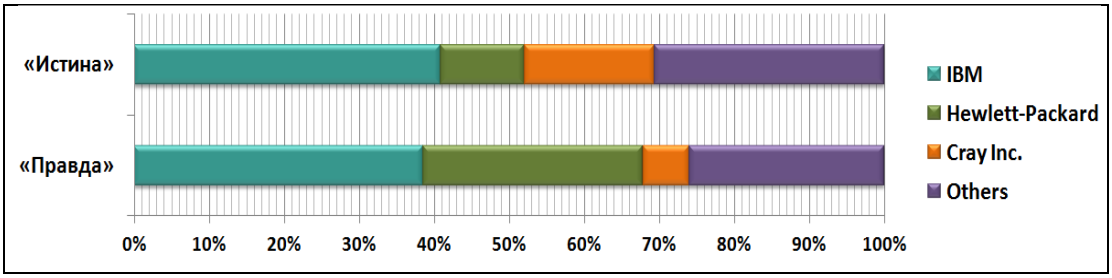


- табл. 4 — в колонке *«A»* доля компании *Hewlett-Packard* преувеличена в 2,61 раза, а доля *Cray Inc.* преуменьшена в 2,8 раза.

Таблица 4

Распределение долей между производителями суперкомпьютеров (*Top500* за ноябрь 2012 г.)

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
прочие	130	26,00 %	30,59 %	↓1,18
Всего	**500**	**100 %**	**100 %**	



Какой должна быть корректная методика вычисления истинных долей? По какой причине «правда» так сильно отличается от «истины»? Всё это будет рассмотрено 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»;

- 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 исследуются позиции России в мировой суперкомпьютерной отрасли.

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. В качестве единиц измерения служат: гигафлопс (*1 GFLOPS* = 10^9 операций в секунду); терафлопс (*1 TFLOPS* = 10^{12} операций в секунду); петафлопс (*1 PFLOPS* = 10^{15} операций в секунду); экзафлопс (*1 EFLOPS* = 10^{18} операций в секунду) и т.д.

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 может быть разной.

Для сравнения производительности различных суперкомпьютеров чаще всего берут реальную производительность на задаче *LINPACK* (решение системы линейных уравнений с большим числом неизвестных; используется в мировом рейтинге суперкомпьютеров *Top500*). В последнее время набирают популярность и другие тесты реальной производительности суперкомпьютеров — например, основанные на задачах с интенсивной обработкой данных (так, задача поиска в большом графе в ширину применяется как тест в другом мировом рейтинге суперкомпьютеров — *Graph500* [2]).

2.2. Суперкомпьютеры

Отметим интересный факт: если в известной сетевой энциклопедии *Wikipedia* [3] попытаться посмотреть термин *High-performance computing* (высокопроизводительные вычисления), то последует автоматическое перенаправление на страницу *Supercomputer*. Это верно и для англоязычной, и для русскоязычной версии Википедии. Тем самым, подчеркивается синонимичность понятий «высокопроизводительный компьютер» и «суперкомпьютер».

Дадим формальное определение: к вычислительным системам высокой производительности — **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, **500 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 (либо технические показатели позволяли её включить в рейтинг, но это не было сделано по некоторым причинам) 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*. Сегодня в открытом доступе [1] имеются данные 40 выпусков рейтинга (с июня 1993 года по ноябрь 2012 года), которые можно выгрузить в виде *Excel*-таблицы. В этом случае предоставляется самая полная информация.

Если свести все 40 таблиц вместе, то получим таблицу с $40 \times 500 = 20\,000$ строками и 40 колонками (полями). Имена 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.*

Профессиональный анализ списков *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 применения суперкомпьютера более точно, чем это указано в поле *Segment*. Чтобы точнее понять устройство интерконнекта, имеет смысл обрабатывать 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.

2.4. О частичной неполноте и частичной недостоверности Top500

Время от времени появляются публикации [5], указывающие на частичную недостоверность данных в *Top500*: установки могут попадать в рейтинг еще до того, как они реально созданы, или оставаться в рейтинге, прекратив 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 — подобно тому, что можно вполне достоверно сравнивать между собою айсберги, основываясь на неполной информации, которую дают их надводные (видимые) части.

3. Методика вычисления истинных долей

Зададимся вопросом: почему правильное суждение (абсолютная правда) «В ноябре 2012 года бóльшая часть (247 из 500) суперкомпьютеров использовалась в промышленности (*Segment=Industry*)» не может служить основой 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), по 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 количественными характеристиками, которые наиболее верно отражают **наиважнейшую характеристику суперкомпьютеров** как изделий. Точно так же, например, когда сравнивают торговые флоты разных стран, их измеряют не в штуках, а в суммарном тоннаже.

Самая важная количественная 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 данных 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 (как увидим далее, разницу можно оценить в два порядка — до 250 раз) 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**.

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, *RMax(n, i)* — *LINPACK*-производительность данной системы в n-ой редакции *Top500*.

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

Истинную долю суперкомпьютеров заданной категории в n-ой редакции *Top500* определим как долю суммарной *LINPACK*-производительности суперкомпьютеров данной категории в суммарной *LINPACK*-производительности всего списка:

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

Рассмотрим некоторый **подсписок** в n-ой редакции *Top500*, заданный множеством позиций

$J = \{... \; i \; ...\} \subseteq [1...500]$, — например, первую сотню, то есть *Top1*–100: $J = [1...100]$.

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:

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

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

Используя обозначения раздела 3.2, посчитаем «правду» — долю категории C по традиционной методике, в штуках:

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

Таким образом, при такой методике в общую копилку доли p_1 категории 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 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-й редакции рейтинга *Top500*; то есть, 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*. Это позволяет выделить:

1. суперЭВМ в крупнейших национальных центрах — единичные установки в стране, соответствующие местам 1–20 в мировом рейтинге *Top500*;
2. суперЭВМ в крупнейших региональных и отраслевых центрах — два–четыре десятка установок в стране, соответствующих местам 21–100 в мировом рейтинге *Top500*;
3. суперЭВМ в крупных региональных и корпоративных центрах — от четырех десятков до сотни установок в стране, соответствующих местам 101–250 в мировом рейтинге *Top500*;
4. 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, табл. 5).

Разница по *LINPACK*-производительности самой мощной и самой слабой системы в классе *Top1–20* (20 систем) составляет 16,7 раза (*Top1–10* — 11,6 раза, *Top11–20* — 1,3 раза); 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 уровня *Top1–20* (и особенно — системы *Top1–10*) радикально отличаются от других, а системы в классах *Top21–100*, *Top101–250* и *Top251–500* отличаются друг от друга по производительности не принципиально.

Таблица 3

Разница в *LINPACK*-производительности между суперкомпьютерами разных уровней (*Top500* за ноябрь 2012 г.)

Места в *Top500*	***LINPACK*-производительность, *max–min* (*TFLOPS*)**	**Разница *LINPACK*-производительности, *max/min* (разы)**	**Разница с *LINPACK*-производительностью *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*-производительности всего списка *Top500* обеспечивают первые n систем из списка. Видно, что глубокое расслоение суперкомпьютерной отрасли является почти точным отражением принципа Вильфредо Парето (этот принцип часто формулируют так: 20 % усилий дают 80 % результата).

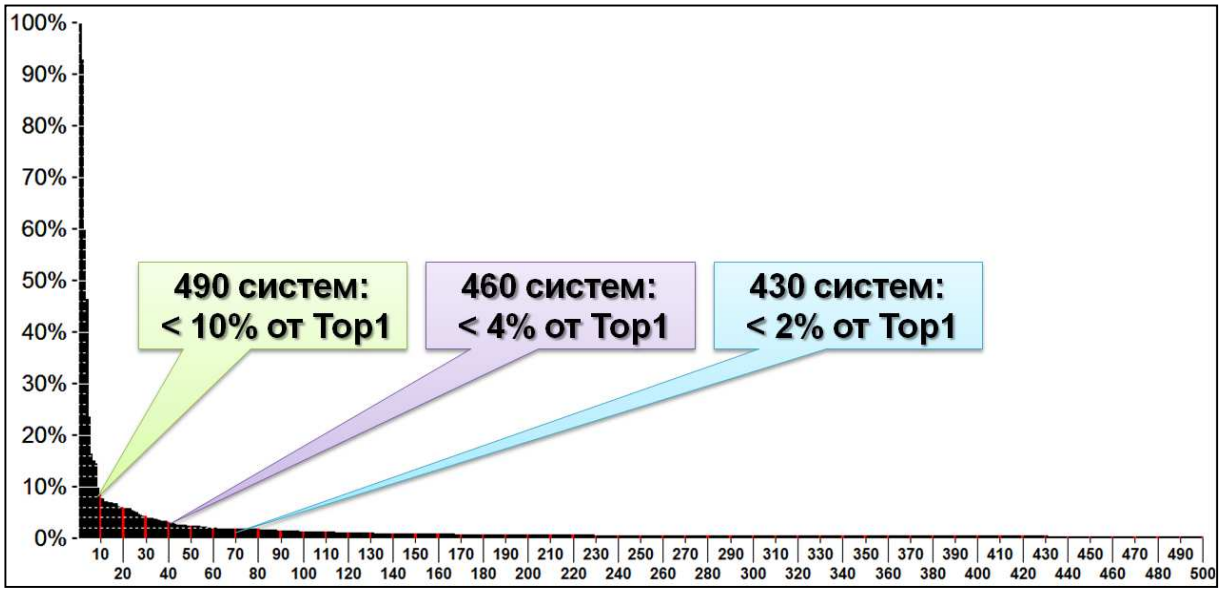


Рис. 2. Относительная *LINPACK*-производительность i-й системы в *Top500*; за 100 % принята *LINPACK*-производительность *Top1* (по данным *Top500* за ноябрь 2012 г.)

Подчеркнем, что анализируя приведенные данные (табл. 5, рис. 2, 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.

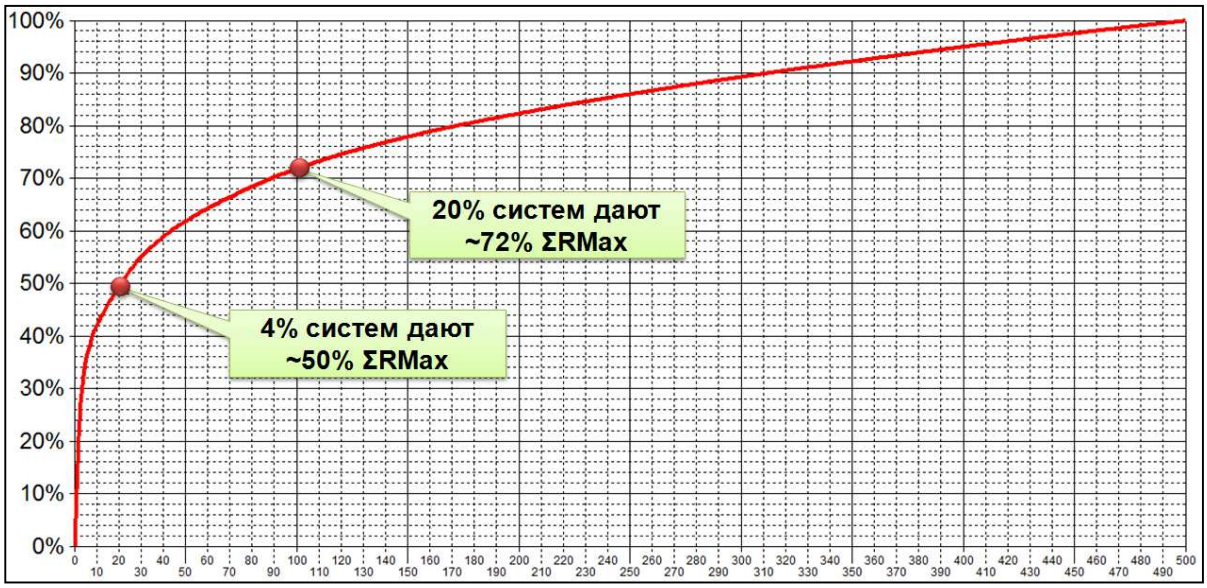


Рис. 3. Какую долю суммарной *LINPACK*-производительности всего списка *Top500* обеспечивают первые n систем из списка (по данным *Top500* за ноябрь 2012 г.)

Например, суммарная стоимость первых 20-ти суперкомпьютеров в *Top500* (рис. 3), скорее всего, примерно равна суммарной стоимости остальных 480-ти суперкомпьютеров.

5. Восстановление истины: применение 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 мы, в этом и последующих разделах, проведем исследование долей по различным категориям. Исследования будут выполняться при помощи программы *Top500 Analyzer*.

5.1. Анализ «сегментов применения суперкомпьютеров»

Начнем с анализа так называемых сегментов применения суперкомпьютеров. Все суперкомпьютеры разбиваются по категориям, в зависимости от указанных значений в поле *Segment* (в этом поле составители всегда указывают одно из шести значений — *Research, Academic, Vendor, Industry, Government, Classified*). Соответственно получаем шесть категорий суперкомпьютеров. Диаграмма (рис. 1) долей этих категорий при расчете в штуках входит в официальный плакат рейтинга *Top500*, опубликованного 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.

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.

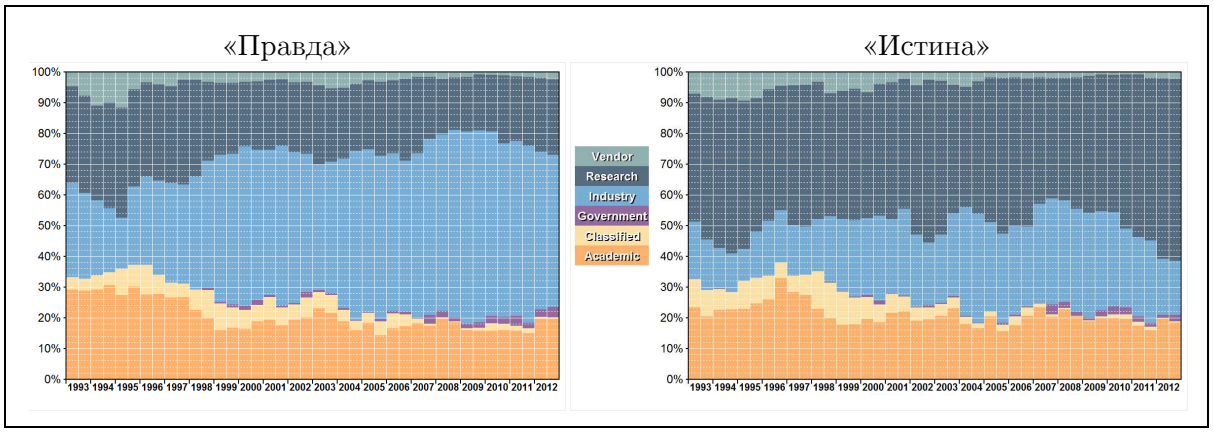


Рис. 4. Изменение долей сегментов применения суперкомпьютеров в период с июня 1993 до ноября 2012 г. по данным всех 40 списков *Top500*.
Слева — доли «в штуках» (от общего числа суперкомпьютеров), справа — истинные доли (в *LINPACK*-производительности)

5.2. Анализ областей использования суперкомпьютеров

Понятие «сегмент применения суперкомпьютеров» определяется напрямую значением поля *Segment* в рейтинге *Top500*. Кроме этого, в программе *Top500 Analyzer* поддерживается понятие «область применения суперкомпьютера», которое определяется путем анализа двух полей: *Segment* и *Application Area*. В результате программа относит все суперкомпьютеры к одной из четырех категорий:

1. *RnD* (от английского *Research* and *Development*) — использование для фундаментальных исследований и НИОКР;
2. *Industry* — использование в промышленности и в других областях реальной экономики (например, в индустрии развлечений и т.п.);
3. *Gov.Mil* — использование для государственных и военных нужд;
4. *Unknown* — недостаточно информации для отнесения к одной из предыдущих категорий.

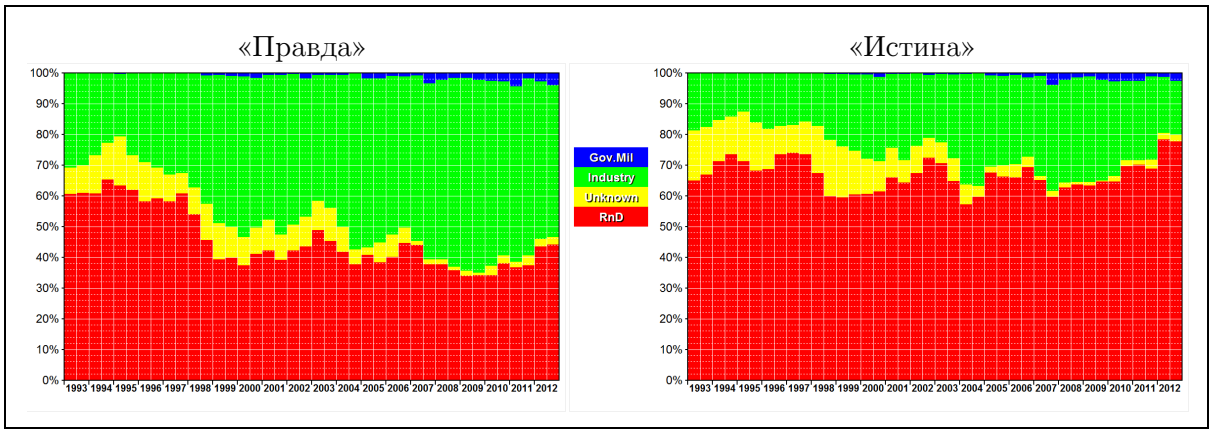


Рис. 5. Изменение долей четырех областей использования суперкомпьютеров в период с июня 1993 до ноября 2012 г. по данным всех 40 списков *Top500*.
Слева — доли «в штуках» (от общего числа суперкомпьютеров),
справа — истинные доли (в *LINPACK*-производительности)

Для данных категорий при помощи программы *Top500 Analyzer* построены диаграммы (рис. 5). Левая часть рисунка — доли областей использования суперкомпьютеров «в штуках», правая — истинные доли. Сравнивая обе части, 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 *Top500 Analyzer* построим для редакции Top500 за ноябрь 2012 года распределение областей использования суперкомпьютеров по пяти уровням суперкомпьютеров: *Top1–10, Top11–20, Top21–100, Top101–250* и *Top251–500*. Поясним структуру этой диаграммы (рис. 6). Левые пять столбцов иллюстрируют истинные доли (доли суммарной *LINPACK*-производительности — *RMax*) областей использования отдельно для пяти уровней суперкомпьютеров: от *Top1–10 до Top251–500*. Площади этих пяти столбцов (а значит и их ширины) пропорциональны суммарной *LINPACK*-производительности соответствующих уровней суперкомпьютеров: *Top1–10, Top11–20, Top21–100, Top101–250* и *Top251–500*. Таким образом, в данных пяти столбцах площадь любой области — например, области некоторого цвета,— пропорциональна суммарной *LINPACK*-производительности соответствующего множества суперкомпьютеров.

Предпоследний столбец на рис. 6 имеет некоторую фиксированную ширину и длинами цветовых сегментов отражает истинные доли областей использования для всего списка *Top500* — что, по сути, совпадает с правым столбцом в правой части рис. 5. Последний столбец на рис. 6 (он имеет ту же самую фиксированную ширину, что и предпоследний) иллюстрируют доли «в штуках» для всего списка *Top500* за ноябрь 2012 года — что, по сути, совпадает с правым столбцом в левой части рис. 5.

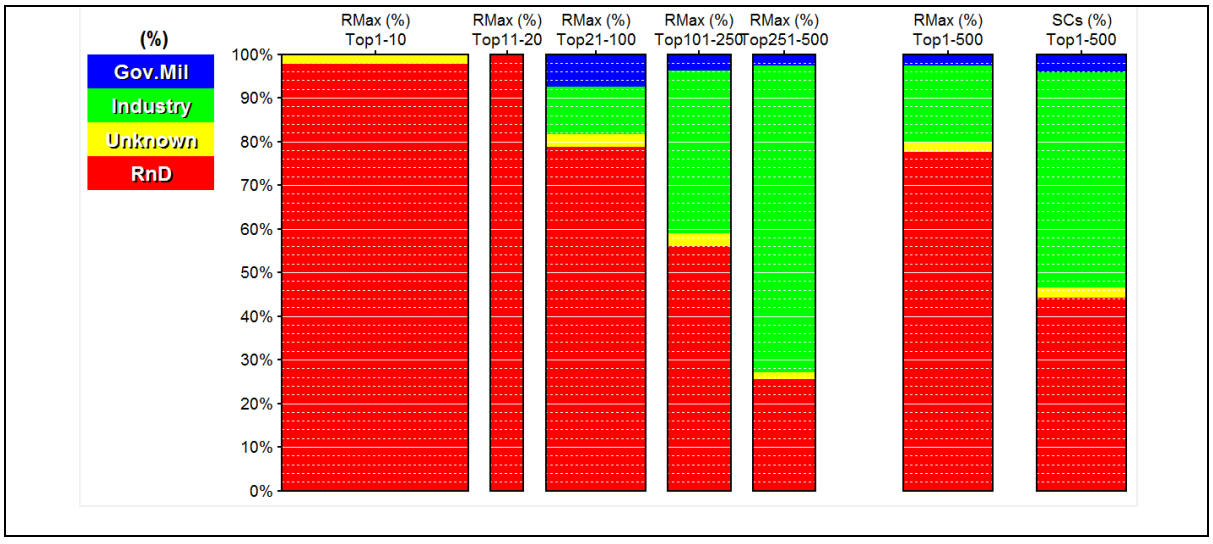


Рис. 6. Распределение областей использования суперкомпьютеров по уровням *Top1–10, Top11–20, Top21–100, Top101–250, Top251–500* (на основании списка *Top500* за ноябрь 2012 г.)

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 промышленности совсем не применяются суперкомпьютеры

первого и второго уровней, а использование систем третьего и четвертого уровней незначительно. То есть, для задач категории *Industry* в основном задействованы только самые слабые и самые многочисленные системы (пятый уровень, 250 систем с производительностью в 150–230 раз меньше, чем у *Top1*).

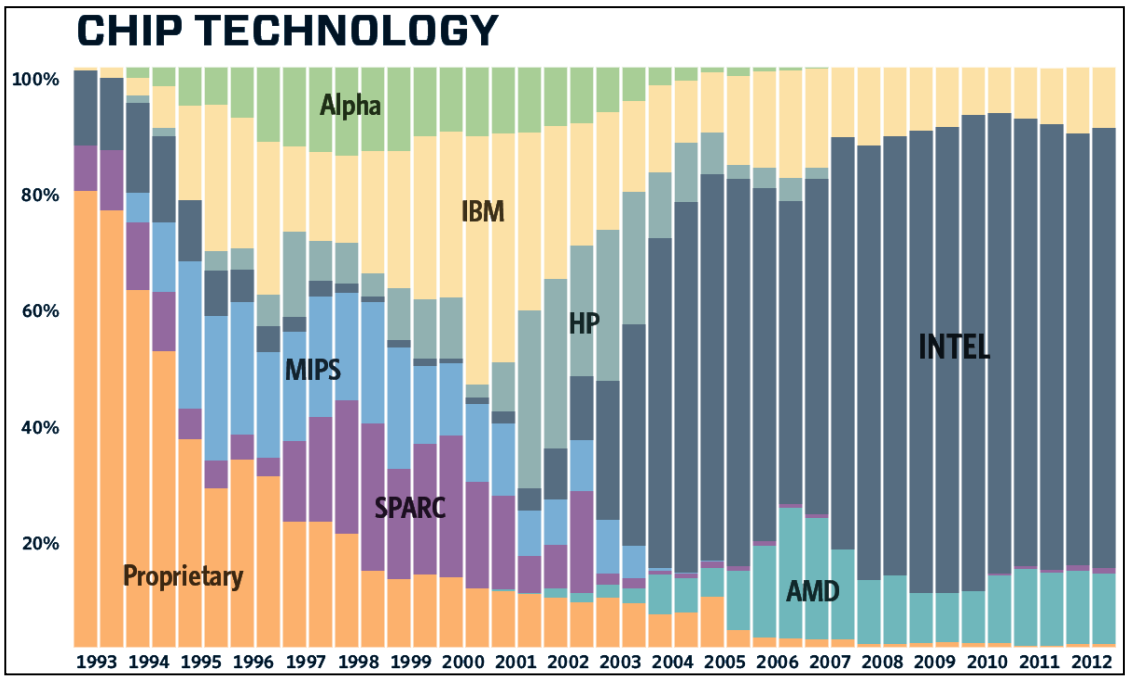


Рис. 7. Диаграмма *«Chip Technology»* с плаката *Top500* за ноябрь 2012 г.

6. Восстановление истины: типы микропроцессоров, используемых в суперкомпьютерах

На плакатах рейтинга *Top500* кроме диаграммы *Installation Type* традиционно размещают диаграмму *Chip Technology,* отображающую доли используемых в суперкомпьютерах процессоров, изготавливаемых по разным технологиям. На плакате за ноябрь 2012 года (рис. 7) диаграмма состоит из 40 столбцов — каждый столбец соответствует одному выпуску рейтинга, на один год приходятся два столбца (июнь и ноябрь). 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, Proprietary.*

Рассматривая эту диаграмму, легко сделать весьма ошибочные суждения, например: *§7 По данным на ноябрь 2012 года подавляющая часть (76 %) суперкомпьютеров Top500 построена на процессорах Intel. 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 построим, с помощью программы *Top500 Analyzer,* диаграммы долей различных технологий процессоров, используемых в суперкомпьютерах (рис. 8). Чтобы узнать технологию процессора, для каждой записи в *Top500* анализируются четыре поля: *Processor, Processor Family, Processor Generation, Processor*

Technology. Левая часть рисунка — доли «в штуках» — в точности совпадает с диаграммой *Installation Type* на официальном плакате. Правая диаграмма показывает истинные доли различных технологий процессоров, используемых в суперкомпьютерах.

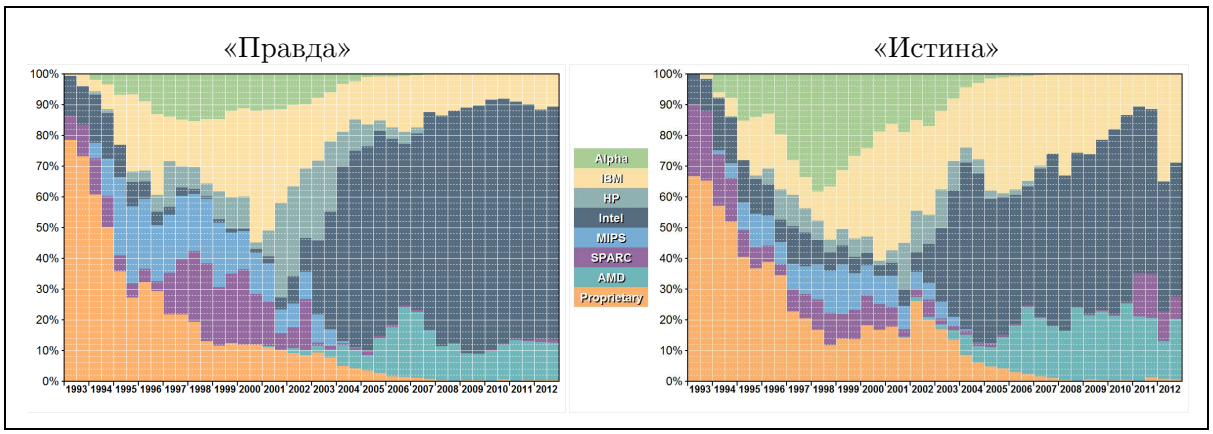


Рис. 8. Изменение долей различных технологий процессоров, используемых в суперкомпьютерах, в период с июня 1993 до ноября 2012 г. по данным всех 40 списков *Top500*. Слева — доли «в штуках» (от общего числа суперкомпьютеров), справа — истинные доли (в *LINPACK*-производительности)

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*.

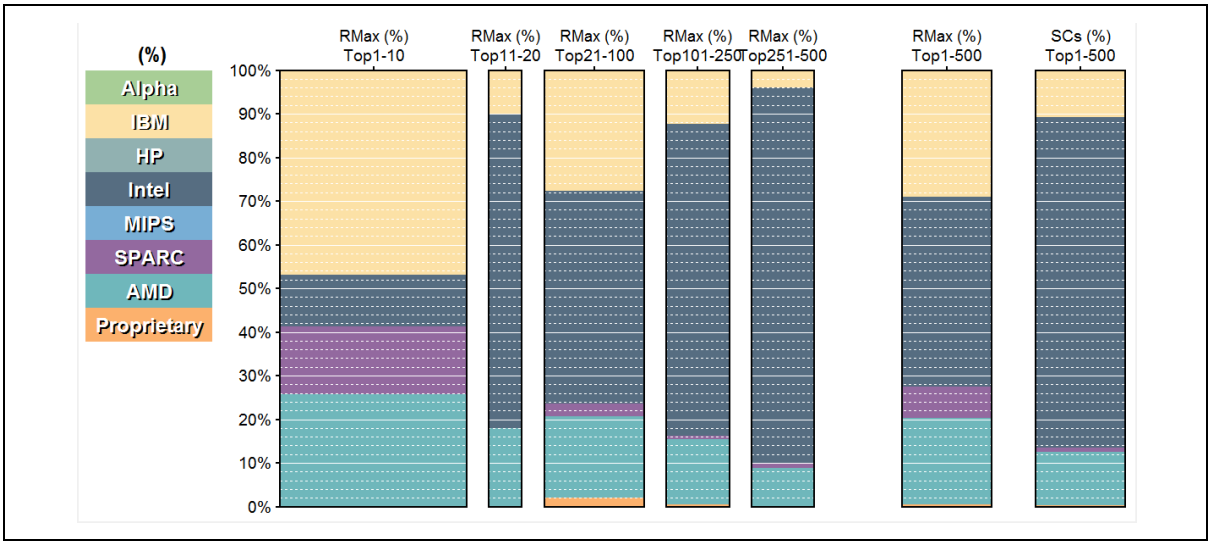


Рис. 9. Распределение технологий процессоров по уровням *Top1–10, Top11–20, Top21-100, Top101–250, Top251–500* (на основании списка *Top500* за ноябрь 2012 г.)

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, но производительность сравнительно слабая). 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* оказались сильно преуменьшены, а доля категории *Intel* — серьезно преувеличена. Истинное суждение (исправляющее 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: IBM (доля — 29 %) отстает в 1,5 раза, AMD (20 %) — в 2,2 раза. Заметная доля (7 %) приходится на процессоры SPARC.

7. Восстановление истины: компании-производители суперкомпьютеров

Теперь проанализируем показатель «компания-производитель»; соответствующее поле в записях *Top500* называется *Manufacturer*. Безусловные лидерские позиции здесь принадлежат трем компаниям, поэтому в программе *Top500 Analyzer* введем 4 категории (заинтересованный читатель-программист легко может изменить эти установки): *Cray* — суперкомпьютер изготовлен компанией *Cray Inc.; IBM* — компанией *IBM; HP* — *Hewlett-Packard; Other* — любой другой 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 Top500 за ноябрь 2008 года, компания Hewlett-Packard построила больше суперкомпьютеров, входящих в Top500, чем любая другая — 42 % от общего количества. Ближайшие конкуренты: IBM (37 %, отставание в 1,13 раза) и Cray (5 %, отставание в 8,4 раза); все остальные производители, даже 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 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 производителей вместе взятых — 22 %.

Сравнивая эти два утверждения, отметим, что в утверждении §9 истинная доля категории *HP* была серьезно (в 1.76 раз) преувеличена, истинная доля *Cray* — серьезно (в 3 раза) преуменьшена; истинная доля категории *Others* — преуменьшена в 1,4 раза. Кроме того, совершенно 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.

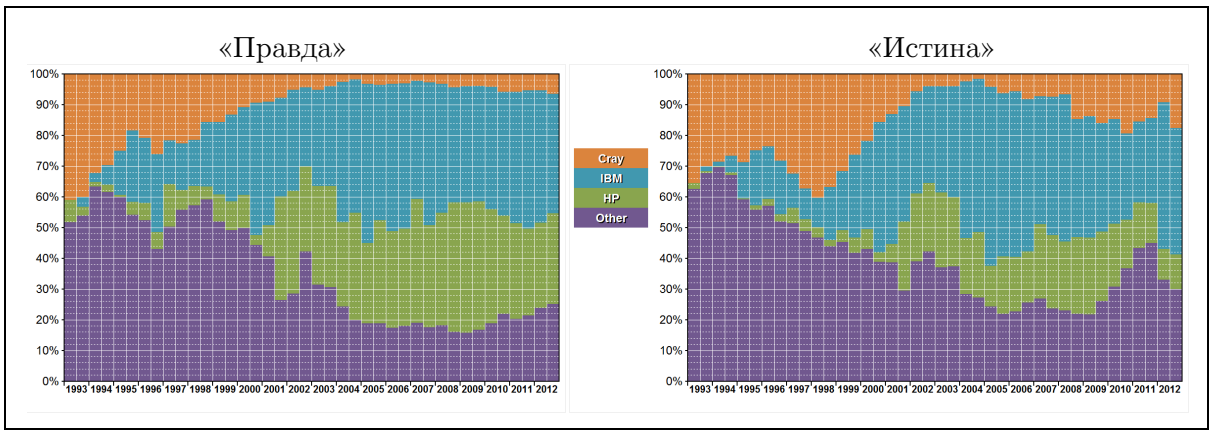


Рис. 10. Изменение долей производителей суперкомпьютеров в период с июня 1993 г. до ноября 2012 г. по данным всех 40 списков *Top500*. Слева — доли «в штуках» (от общего числа суперкомпьютеров), справа — истинные доли (в *LINPACK*-производительности)

Подобные (§9) мнимые признаки абсолютного лидерства в принципе дают компании аргументы для настойчивого продвижения своих решений — даже в тех сегментах, где её позиции на самом деле 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.

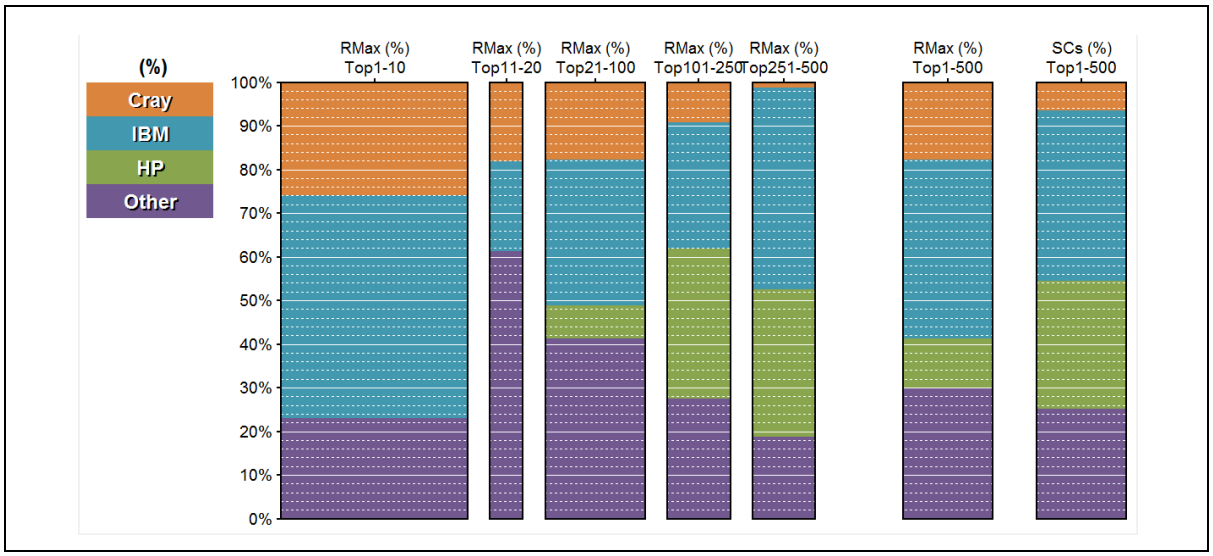


Рис. 11. Распределение долей производителей суперкомпьютеров по уровням *Top1–10, Top11–20, Top21–100, Top101–250, Top251–500* (на основании списка *Top500* за ноябрь 2012 г.)

На примере редакции *Top500* за ноябрь 2012 года разберем распределение систем компаний-производителей по пяти уровням суперкомпьютеров: *Top1–10, Top11–20,*

Top21–100, Top101–250 и *Top251–500.* На диаграмме (рис. 11) 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 3-м уровне *(Top21–100),* но заметно присутствуют на 4-м и 5-м 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, где суперкомпьютеров по количеству мало, но производительность их велика. Суперкомпьютеры категории «*IBM*» занимают сравнимые доли на всех пяти уровнях.

8. Восстановление истины: технологии интерконнекта

Пришел черед проанализировать показатель «используемая технология интерконнекта». Для этого следует принимать во внимание два поля: *Interconnect* и *Interconnect Family.*

В программе *Top500 Analyzer* введем шесть категорий для обозначения технологии интерконнекта. Пять из них явно указывают используемую сетевую технологию: *Infiniband, Ethernet, Myrinet, SCI* и *Quadrics.* Все эти технологии являются коммерчески доступными: любой разработчик суперкомпьютеров может отдельно приобрести соответствующие сетевые изделия в период их производства (сетевые адаптеры, коммутаторы, кабели или даже микросхемы для адаптеров и коммутаторов) и на этой базе разрабатывать свои собственные суперкомпьютеры.

Шестая категория — *Custom* — объединяет технологии, которые нельзя приобрести как отдельные сетевые решения (поясним: можно купить целиком суперкомпьютер *IBM Blue Gene,* но невозможно купить отдельно интерконнект, который используется в *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 создания российского суперкомпьютера с подобным интерконнектом, то этот интерконнект (аналог) придется разрабатывать самостоятельно. Естественно, принимая решение о такой разработке, следует ответить на вопросы:

§11 Надо ли тратить ресурсы на разработку российской технологии интерконнекта, подобной представленным в категории Custom? Может быть, 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», а правая диаграмма показывает истинные доли.

И снова мы наблюдаем серьезную разницу между «правдой» и «истиной». Среди прочего видно, что в последние годы 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 за ноябрь 2012 года, технологии Infiniband и Ethernet использовались в большинстве суперкомпьютеров, вошедших в эту редакцию рейтинга Top500 (45 %+38 %=83 %). Доли категорий Custom (16,7 %) и Myrinet (0,3 %) незначительны.
§13 Немногим ранее ситуация выглядела еще радикальнее. По данным за июнь 2010 года, технологии Infiniband и Ethernet применялись в подавляющем большинстве суперкомпьютеров, вошедших в эту редакцию рейтинга Top500 (41 %+49 %=90 %). Доли категорий Custom (9 %), Myrinet (0,5 %) и «Quadrics Myrinet» (0,5 %) — незначительны.

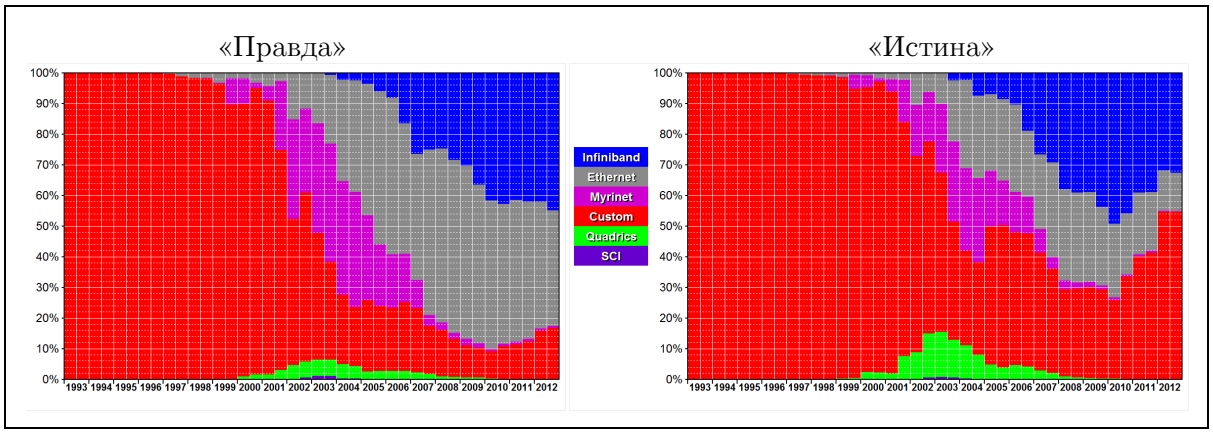


Рис. 12. Изменение долей различных технологий интерконнекта в суперкомпьютерах в период с июня 1993 до ноября 2012 г. по данным всех 40 списков *Top500*. Слева — доли «в штуках» (от общего числа суперкомпьютеров), справа — истинные доли (в *LINPACK*-производительности)

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!) отличалось от утверждений §12 и §13; на это уже указывалось выше (табл. 3). При этом не только многократно искажены доли технологий интерконнекта, но и неверно указана лидирующая категория: истинным и абсолютным лидером по данным *Top500* за ноябрь 2012 года является категория *Custom* (55 %). Еще раз напомним: 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 основанием на пятом. Суперкомпьютеры с интерконнектом на базе Ethernet вообще отсутствуют на высших двух уровнях *(Top1–10, Top11–20),* слабо представлены на 3-м уровне *(Top21–100),* но заметно присутствуют на 4-м и 5-м уровнях *(Top101–500)* — где суперкомпьютеров по количеству много, а по производительности они слабые.

Ареал 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 смещен к старшим уровням, где суперкомпьютеров по количеству мало, но они мощные. Именно на таких технологиях интерконнекта строятся рекордные установки, обладание которыми стратегически важно для России.

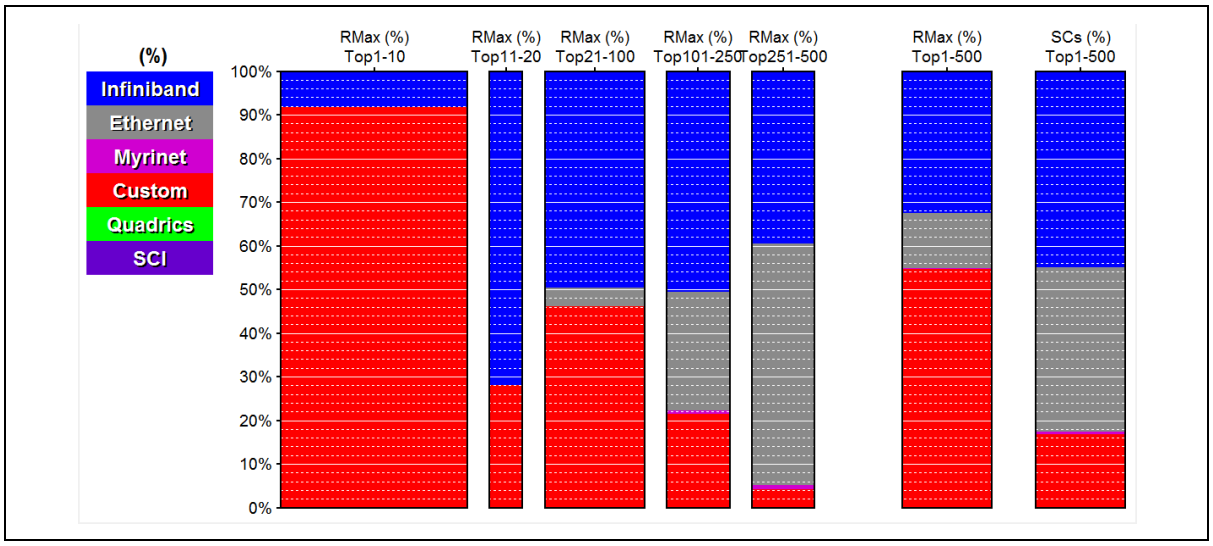


Рис. 13. Распределение долей технологий интерконнекта в суперкомпьютерах по уровням *Top1–10, Top11–20, Top21–100, Top101–250, Top251–500* (на основании списка *Top500* за ноябрь 2012 г.)

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

§15 Технологии категории Custom обеспечивают подавляющую долю LINPACK-производительности (55 % — по данным редакции Top500 от ноября 2012 года). А если говорить про самые мощные суперкомпьютеры (уровни Top1–10 и Top10–20), которые вряд ли будут проданы России и которые предстоит построить самостоятельно, то эти системы практически всегда строятся на сетевых 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. Восстановление истины: положение России в мировой суперкомпьютерной отрасли

Оценивая положение России в мировой суперкомпьютерной отрасли, как правило, совершают ту же методологическую ошибку: рассматривают число суперкомпьютеров из *Top500,* установленных в стране. Как пример, процитируем фрагмент из сообщения [9]: «Позиции России в рейтинге несколько улучшились с точки зрения количества представленных систем: в него вошли восемь суперкомпьютеров против пяти в прошлой редакции». К чести автора сразу отметим, что он понимает слабость такой оценки и делает правильное замечание: «с точки зрения количества представленных систем». Однако, к сожалению, многие читатели не заметят этой мелкой детали. И если их спросить, сильно ли Россия улучшила за полгода (июнь–ноябрь 2012 года) свое положение в мировой суперкомпьютерной отрасли, то не стóит удивляться ответу: «Было пять, стало восемь,

налицо рост на целых 60 % за полгода — замечательный результат!» И если суперкомпьютеры считать в штуках, то такой вывод будет правдой. И опять эта правда серьезно (в 3,5 раза) искажает истину: если и можно говорить о росте, то он составляет 17,2 %.

Основываясь на анализе *LINPACK*-производительности российских суперкомпьютеров, а не их числа, в данной работе рассмотрим три методики оценки истинного положения России в мировой суперкомпьютерной отрасли:

1) анализ отставания от ведущих стран;
2) анализ отставания от суперкомпьютерных технологий «переднего края»;
3) анализ доли России в мировой суммарной LINPACK-производительности.

Первые два подхода дают ответ на классический вопрос «на сколько лет мы отстаем».

9.1. Анализ отставания России от ведущих стран суперкомпьютерной отрасли

Для ответа на вопрос «на сколько лет Россия отстает от ведущих стран в суперкомпьютерной отрасли?» в каждый момент времени сравним суммарную *LINPACK*-производительность суперкомпьютеров, установленных в России, и в других регионах, которые являются лидерами в суперкомпьютерной отрасли: США, объединенная Европа, Китай и Япония. Соответствующий график представлен на рис. 14.

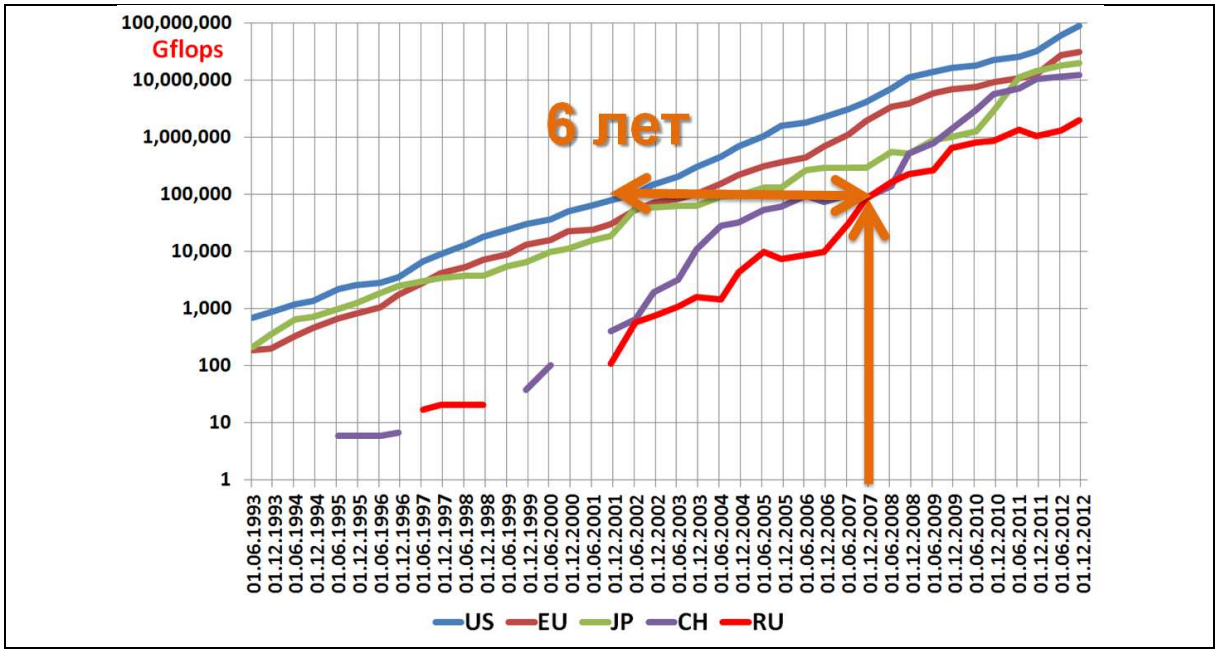


Рис. 14. Изменение суммарной *LINPACK*-производительности суперкомпьютеров *Top500*, установленных в США *(US)*, объединенной Европе *(EU)*, Японии *(JP)*, Китае *(CH)* и России *(RU)*. В качестве примера показано отставание России от США по этому показателю по состоянию на ноябрь 2007 года — шесть лет

Далее можно в каждый момент времени рассмотреть суммарную *LINPACK*-производительность суперкомпьютеров, установленных в России, и определить, сколько лет назад суммарная *LINPACK*-производительность суперкомпьютеров, установленных в США, была такой же или меньшей. Эта величина и будет показывать отставание России от США в рассмотренный момент времени по показателю «суммарная *LINPACK*-производительность суперкомпьютеров».

В качестве примера на рис. 14 это построение выполнено для ситуации на ноябрь 2007 года.

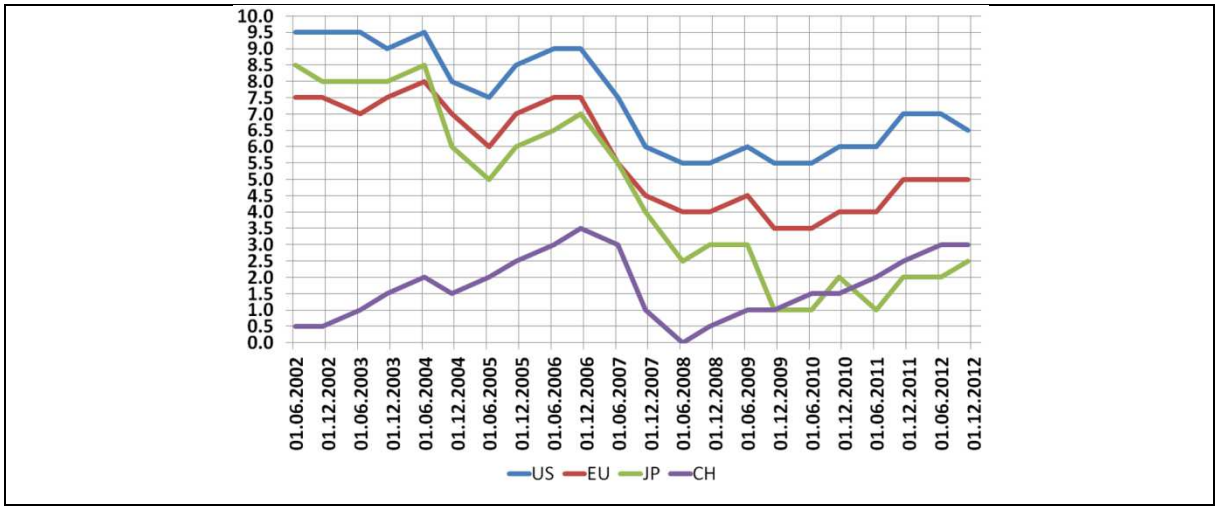


Рис. 15. Графики изменения отставания (в годах) России по показателю «суммарная *LINPACK*-производительность суперкомпьютеров» от США *(US)*, Евросоюза *(EU)*, Японии *(JP)* и Китая *(CH)*

На рис. 15 показан график отставания России по показателю «суммарная *LINPACK*-производительность суперкомпьютеров» от США, Евросоюза, Японии и Китая, начиная с июня 2002 года. Хорошо видны периоды, когда Россия сокращала свое отставание от стран-лидеров, и периоды, когда отрыв только увеличивался. К сожалению, последние два с половиной года можно охарактеризовать как период упрочнения отставания России от стран-лидеров суперкомпьютерной отрасли.

9.2. Анализ отставания от суперкомпьютерных технологий переднего края

Для ответа на вопрос «На сколько лет суперкомпьютерные технологии России отстают от технологий переднего края?» достаточно в каждый момент времени t сравнить *LINPACK*-производительность *ru1(t)* самого мощного суперкомпьютера, установленного в России, и *LINPACK*-производительность систем, находящихся на переднем крае развития суперкомпьютерных технологий в мире. Мы рассмотрим суперкомпьютеры, занимающие первое, пятое и десятое место в *Top500* — *top1(t)*, *top5(t)* и *top10(t)*. Далее, точно таким же способом, как это описано в предыдущем разделе, в каждый момент времени t вычисляется отставание от мирового уровня технологий переднего края: уровня *Top1 (d1(t))*, уровня *Top5 (d5(t))* и уровня *Top10 (d10(t))*:

$$d1(t) = \min\{\,\Delta t \mid top1(t - \Delta t) \le ru1(t)\}$$
$$d5(t) = \min\{\,\Delta t \mid top5(t - \Delta t) \le ru1(t)\}$$
$$d10(t) = \min\{\,\Delta t \mid top10(t - \Delta t) \le ru1(t)\}$$

На соответствующих графиках (рис. 16) хорошо видны периоды, когда Россия приближалась к суперкомпьютерным технологиям переднего края, и периоды, когда отставание только увеличивалось. К сожалению, последние два с половиной года можно охарактеризовать как период увеличения отставания от развития суперкомпьютерных технологий переднего края.

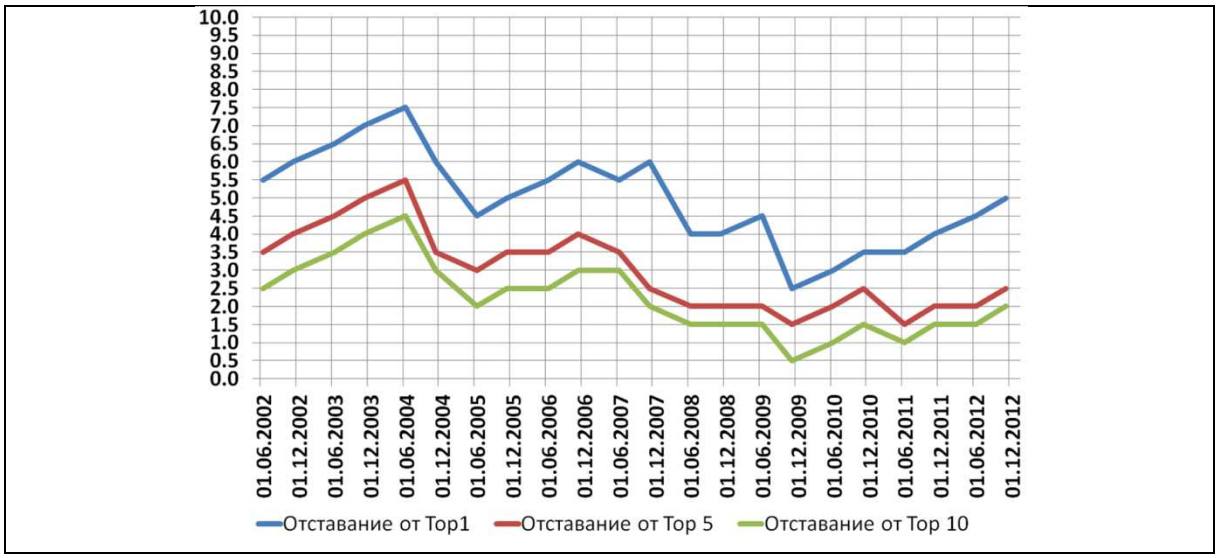


Рис. 16. Графики изменения отставания (в годах) суперкомпьютерных технологий в Российских от уровня переднего края — *Top1, Top5* и *Top10*

9.3. Анализ доли России в мировой суммарной *LINPACK*-производительности

Наконец, адекватной заменой для ошибочного показателя «количество суперкомпьютеров, установленных в России и входящих в рейтинг *Top500*» является доля России в суммарной *LINPACK*-производительности по всему рейтингу *Top500*. Для каждой редакции рейтинга эта доля рассчитывается как отношение суммы *LINPACK*-производительности систем, установленных в России, к сумме *LINPACK*-производительности по всему рейтингу *Top500*.

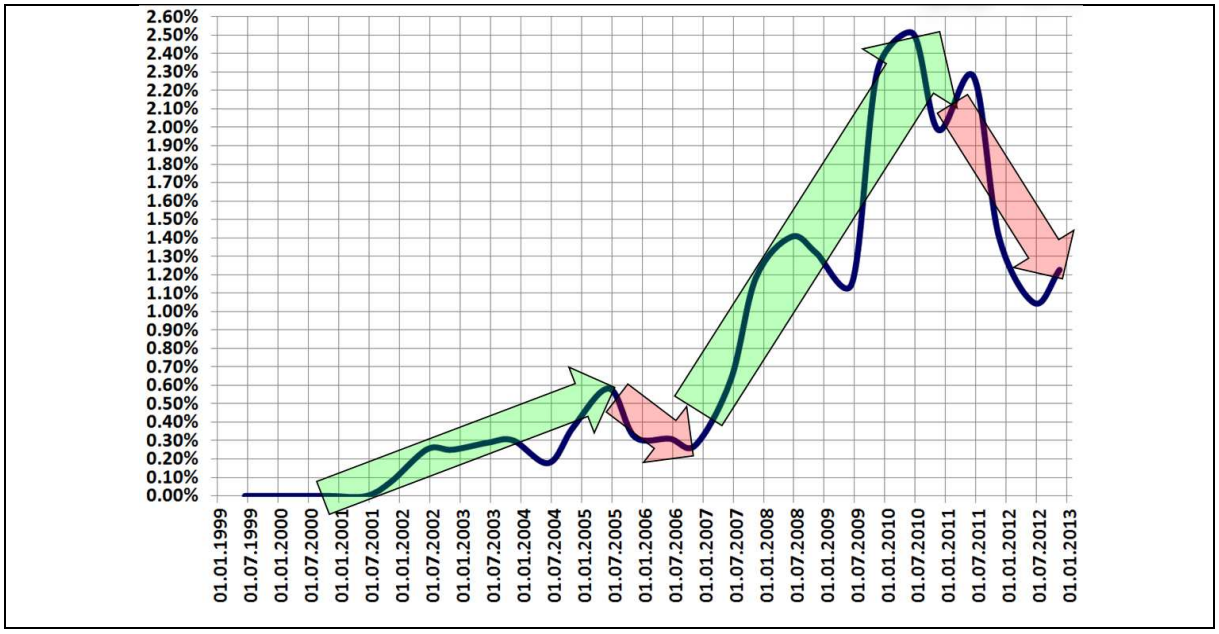


Рис. 17. График изменения доли России в суммарной *LINPACK*-производительности рейтинга *Top500* в различные моменты времени. Стрелками зеленого цвета обозначены периоды исполнения суперкомпьютерных программ «СКИФ» (2000–2004 гг.) и «СКИФ-ГРИД» (2007–2010 гг.) Союзного государства

Соответствующий график представлен на рис. 17. Хорошо видны периоды укрепления и ослабления позиций России в мировой суперкомпьютерной отрасли. Так, позиции России серьезно укреплялись в периоды действия суперкомпьютерных программ «СКИФ» (2000–2004 гг.) и «СКИФ-ГРИД» (2007–2010 гг.) Союзного государства. К сожалению, последние годы следует охарактеризовать как период ослабления позиций России в мировой суперкомпьютерной отрасли.

Заключение

Целью написания данной статьи было продемонстрировать читателю:

- насколько традиционный и широко распространенный способ анализа *Top500* — подсчет долей от общего количества суперкомпьютеров — искажает истинное положение дел в суперкомпьютерной отрасли (разделы 1, 5-9);
- как важно разработать и грамотно применять правильные методики анализа данных *Top500* (раздел 3);
- как легко на базе неверного поверхностного анализа принимаются ошибочные управленческие решения с серьезными последствиями;
- насколько значительно сегодня расслоение (по реальной производительности) в суперкомпьютерном мире (раздел 4) — кажется, даже профессионалы (чисто психологически) пока еще не всегда в полной мере осознают всю его глубину;
- насколько важно для профессионального анализа *Top500* обладать правильно построенным инструментарием.

Хочется надеться, что эти цели были достигнуты, хотя бы частично. Конечно, формат статьи не позволяет продемонстрировать все возможности программы *Top500 Analyzer*. Так, распределение долей категорий (например, рис. 13) по уровням *(Top1–10, Top11–20, Top21–100, Top101–250, Top251–500)* можно просмотреть для каждой редакции *Top500,* причем в режиме анимации: один год (две редакции *Top500)* — за секунду. При таком просмотре можно разглядеть эпохи появления, расцвета и угасания той или иной категории, можно увидеть, 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 сделать ряд 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* и к национальному рейтингу 50-ти 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 по записям и по полям;
- предположительно малое число формализованных полей в записях рейтинга.

Завершая, хочу 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. URL: http://www.Top500.org
2. Graph500 — мировой рейтинг самых производительных (на задаче поиска в большом графе в ширину) вычислительных машин мира. URL: http://www.graph500.org
3. Википедия — свободная энциклопедия, которую может редактировать каждый. URL: http://wikipedia.org.
4. Абрамов, С.М. Top500 Analyzer — программа для анализа данных рейтинга Top500 / С.М. Абрамов. URL: http://skif.pereslavl.ru/psi-info/rcms-skif/top500analyzer/.
5. Воейков, Д. Рейтинг Top500. Соревнование с гандикапом / Д. Воейков // PC Week/RE. – 2008. – № 27–28. URL: http://www.pcweek.ru/themes/detail.php?ID=112308.
6. Абрамов, С.М. Суперкомпьютерные технологии России: объективные потребности и реальные возможности / С.М. Абрамов // CAD/CAM/CAE Observer. – 2010. – № 2. – С. 74–84.
7. Абрамов, С.М. Состояние и перспективы развития вычислительных систем сверхвысокой производительности / С.М. Абрамов, Е.П. Лилитко // VI Международная конференция «Параллельные вычисления и задачи управления» (24–26 октября 2012 г, Москва). – М.: ИПУ РАН, 2012. – Т. 1. – С. 10–32.
8. Top50 суперкомпьютеров — рейтинг 50 вычислительных систем, установленных на территории СНГ и показавших наибольшую производительность на тесте LINPACK. URL: http:// top50.supercomputers.ru.
9. Лаврентьева, Н. Российский суперкомпьютер-«призрак» вошел в мировой рейтинг Топ-500 / Лаврентьева Н. // Cnews, 12 ноября 2012 г., URL: http://www.cnews.ru/news/top/index.shtml?2012/11/12/509454.

Абрамов Сергей Михайлович, д.ф.-м.н., член-корреспондент РАН, директор Института программных систем имени А.К. Айламазяна Российской академии наук (Переславль-Залесский, Российская Федерация), abramov@botik.ru.

TRUE JUDGMENTS THAT DISTORT THE REAL TRUTH. HOW TO ANALYZE THE TOP500?

S.M. Abramov, Ailamazyan Program Systems Institute of the RAS (Pereslavl-Zalessky, Russian Federation)

Each new edition of the *Top500* list brings various calculations and judgments, such as *«Supercomputers listed in the Top500 are the most used in industry (247 of 500, 49,4 %)»*. It is easy to find similar calculations and judgments about *Top500* in percentages: (i) percentage of different types of processors used in supercomputers; (ii) percentage of different types of interconnect; (iii) percentage of manufactures; (iv) percentage of countries, etc. Important decisions — even government decisions — are often made with reference to such calculations and judgments. This work shows that these calculations and judgments are true but seriously distort the real truth — and mispresent the real situation in the HPC industry. In the paper, the author analyses the reasons

of deep differences between «true judgments» and «the real truth». Furthermore, the paper offers an approach to a correct analysis of the *Top500* and the results of this analysis.

Keywords: Top500, the Use of Supercomputers, High Performance Computing.

References

1. Top500 — mirovoj rejting pyatisot samykh proizwoditelnykh (na teste LINPACK) wychislitelnykh mashin mira [The Top500 list of the world's most powerful computers (according to the Linpack benchmark)]. URL: http://www.Top500.org.
2. Graph500 — mirowoj reiting samykh proizwoditelnykh (na zadache poiska w bolshom grafe w shirinu) wychislitelnykh mashin mira [The Graph500 rating of supercomputer rystems focused on data intensive loads (based on a breadth-first Search in a large undirected Graph)]. URL: http://www.graph500.org.
3. Wikipediya — svobodnaya entsyiklopediya, kotoruyu mozhet redaktirowat` kazhdyj [Wikipedia, the free Encyclopedia that Anyone can edit]. URL: http://wikipedia.org.
4. Abramov S.M. Top500 Analyzer — programma dlya analiza dannykh reitinga Top500. [The Top500 Analyzer — The Software for the Top500 Analysis]. URL: http://skif.pereslavl.ru/psi-info/rcms-skif/top500analyzer/.
5. Wojejkow D. Rejting Top500. Sorewnowanie s gandikapom [The Top500 List. Handicap Competition] // PC Week/RE. 2008. No. 27–28. URL: http://www.pcweek.ru/themes/detail.php?ID=112308.
6. Abramov S.M. Superkompjuternyje tekhnologii Rossii: ob`ektiwnye potrebnosti i real`nyje wozmozhnosti [Supercomputing Technologies in Russia: Objective Needs and Real Opportunities] // CAD/CAM/CAE Observer. 2010. No. 2. P. 74–84.
7. Abramov S.M., Lilitko E.P. Sostojanie i perspektivy razwitija wychislitel`nykh system swerkhwysokoj proizwoditelnosti [The State and Perspectives of Development of Ultra High Performance Computing] // VI Mezhdunarodnaja konferentsija "Parallel`nye wychislenija i zadachi uprawlenija" (24–26 oktjabrja 2012 g., Moskwa) [Parallel Computations and Control Problems: Proceedings of the VI International Conference (24–26 October, 2012, Moscow)]. Moscow: Institute of Control Sciences of the RAS, 2012. Vol 1. P. 10–32.
8. Top50 supercomp`uterow — reiting 50 wychislitel`nykh system, ustanowlennykh na territorii SNG i pokazawshikh naibol`shuju proizwoditel`nost` na teste LINPACK . [The Top50 Ranking of the 50 most powerful supercomputers in CIS countries (according to the Linpack benchmark)]. URL: http://top50.supercomputers.ru.
9. Lawrentjewa N. Rossijskij superkompjuter-prizrak woshel w mirowoj reiting Top500 [Russian phantom supercomputer entered the world's Top500 ranking] // Cnews. 2012. November, 12. URL: http://www.cnews.ru/news/top/index.shtml?2012/11/12/509454.

Поступила в редакцию 23 марта 2013 г.

УДК 004.75

ЭФФЕКТИВНЫЙ ЗАПУСК ГИБРИДНЫХ ПАРАЛЛЕЛЬНЫХ ЗАДАЧ В ГРИДЕ[1]

А.П. Крюков, М.М. Степанова, Н.В. Приходько, Л.В. Шамардин, А.П. Демичев

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 веб-сервиса запуска с локальным менеджером ресурсов организовано через специальные обработчики разных типов заданий (single, openmp, mpi или hybrid), что обеспечивает передачу локальному менеджеру СК правильных параметров для резервирования ресурсов и запуска задачи. Представленное решение было опробовано на грид-полигоне, развернутом на базе промежуточного ПО ГридННС.

Ключевые слова: параллельные вычисления, распределенные вычисления, грид, гибридные задачи

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 ресурсным менеджером (ЛРМ) на конкретном ресурсе. Чтобы ЛРМ мог вы-

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2013».

полнить 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 [3] 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 [4] – пользователь должен передать скрипт с явно установленными параметрами и вызовом MPI-Start. Интерфейс 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 [5] 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 ресур-

сов 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 задач типа MPI+OpenMP.

Структура работы следующая. Во первом разделе описаны особенности гибридных задач и используемые методы их запуска. Второй раздел посвящен описанию методов запуска разных типов параллельных задач в среде ГридННС. В двух следующих разделах представлены некоторые аспекты реализации предложенных методов, а именно, в третьем разделе описана адаптация грид-сервиса «Пилот» для запуска гибридных задач, а в четвертом – взаимодействие веб-сервиса запуска с локальным менеджером ресурсов. В пятом разделе представлены результаты тестов. В заключении перечислены основные результаты работы и возможные направления их развития.

1. Особенности гибридных задач и методы их запуска

Классические параллельные приложения реализуются на основе технологий MPI [6] или OpenMP [7]. 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 [8-12], 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 mpiexec):

```
mpirun [ options ] <program> [ <args> ]
```

Предварительно должно быть определено необходимое окружение (пути к библиотекам, переменные окружения и др.), 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 M×K процессоров/ядер.

В табл. 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 табл. 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 табл. 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 табл. 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

№ п/п	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

2. Запуск параллельных задач в ГридННС

2.1. 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, 13].

Рассмотрим ее с точки зрения функционирования компонент, определяющих запуск задания. Общая схема прохождения задания представлена на рис. 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 функции брокера. Проверка корректности синтаксиса описания выполняется на уровне pilot-cli.
3. «Пилот» [14] 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;

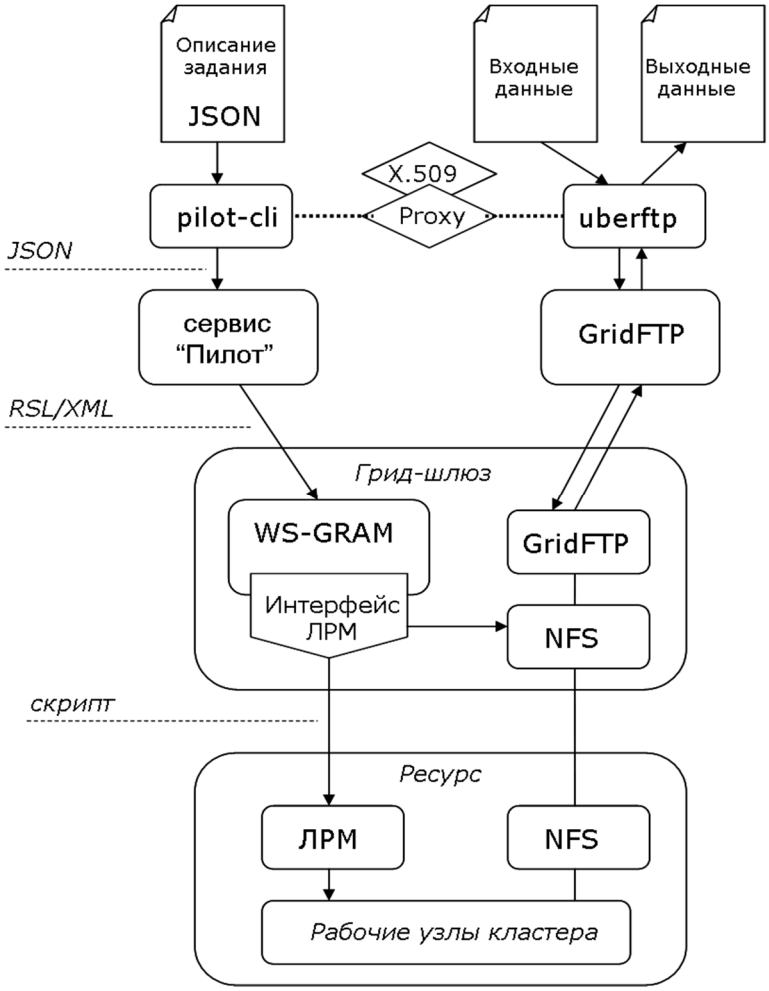


Рис. 1. Схема прохождения задания в ГридННС

- 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.

2.2. 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.

2.3. Типы задач

Расширение спецификации описания задач представлено на рис. 2.

```
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 не менее «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 рекомен-

дуется 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 табл. 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>}

3. Адаптация грид-сервиса «Пилот» 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 гибрид-

ных 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.

4. 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. раздел 2.3) 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.

1) jobtype=single

Запуск задания предполагает запуск одного однопоточного процесса на одном ядре рабочего узла кластера. Схема запуска повторяет стандартную схему запуска Globus Toolkit.

2) 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.

3) 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.

4) 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.

5. 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 рис. 3. 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.

```
{ "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

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.

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.

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.

Работа поддержана грантом РФФИ (№ 11-07-00434-а) и грантом Президента РФ (НШ-3920.2012.2)

Литература

1. Stepanova, M.M. Running Parallel Jobs on the Grid / M.M. Stepanova, O.L. Stesik // Distributed Computing and Grid Technologies in Science and Education: Proceedings of the 5th International Conference (Dubna, 16-21 July, 2012). – Dubna: JINR, 2012. – P. 383–387.
2. Ильин, В.А. ГридННС: состояние и перспективы / В.А. Ильин, В.В. Кореньков, А.П. Крюков // Труды 5-й международной конференции «Распределенные вычис-

ления и Грид-технологии в науке и образовании» (Дубна, 16-21 июля, 2012 г.). – Дубна: ОИЯИ, 2012. – С. 332–336.
3. Burke, S. gLite 3.2 User Guide. – Manual Series. – CERN-LCG-GDEIS-722398. – 2012./S. Burke, S. Campana, E. Lanciotti, et al. URL: http://edms.cern.ch/document/722398 (дата обращения: 03.04.2013).
4. MPI-Start / URL: http://grid.ifca.es/wiki/Middleware/MpiStart (дата обращения: 22.02.2013).
5. Extended Resource Specification Language. – Reference Manual for ARC versions 0.8 and above. – NORDUGRID_MANUAL-4. – 2013 / URL: http:www.nordugrid./documents/arc-ui.pdf (дата обращения: 03.04.2013).
6. MPI: A Message-Passing Interface Standard. Version 2.2. – 2009 / URL: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf (дата обращения: 22.02.2013).
7. OpenMP Application Program Interface. Version 3.1. – 2011 / URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf (дата обращения: 22.02.2013).
8. Makris, I. Mixed Mode Programming on Clustered SMP Systems / I. Makris – M.Sc. Thesis in High Performance Computing. The University of Edinburgh, 2005.– 108 p.
9. Rabenseifner, R. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes / R. Rabenseifner, G. Hager, G. Jost // Proceedings of 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.– Nov. 2009. – P. 427–236.
10. Rane, A. Experiences in tuning performance of hybrid MPI/OpenMP applications on quad-core systems / A. Rane, D. Stanzione // Proceedings of 10th LCI International Conference on High-Performance Clustered Computing – Jan. 2009. – P. 1–10.
11. Глазкова, Е.А. Анализ эффективности гибридного параллельного программирования на примере системы BLUE GENE/P / Е.А. Глазкова, Н.Н. Попова // Материалы Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ» (Новороссийск, 21–26 сентября 2009 г.). – Москва: Издательство Московского университета, 2009. – С. 36–39.
12. Balaji, P. MPI Forum Hybrid Programming Working Group / P. Balaji. URL: http://meetings.mpi-forum.org/mpi3.0 hybrid.php (дата обращения: 22.02.2013).
13. Основные подходы к построению грид-инфраструктуры национальной нанотехнологической сети / А.П. Крюков, А.П. Демичев, В.А. Ильин, Л.В. Шамардин // Вычислительные технологии в естественных науках. Перспективные компьютерные системы: устройства, методы и концепции: Труды семинара (Таруса, 24 марта 2011). – Москва: Издательство ИКИ РАН, 2011. – С. 51–68.
14. Реализация программного интерфейса грид-сервиса Pilot на основе архитектурного стиля REST / А.П. Демичев, В.А. Ильин, А.П. Крюков, Л.В. Шамардин // Вычислительные методы и программирование. Новые вычислительные технологии. – 2010. – Т. 11. – С. 62–65.

Крюков Александр Павлович, ведущий научный сотрудник, кандидат физ.-мат. наук, НИИ ядерной физики имени Д.В. Скобельцына МГУ имени М.В. Ломоносова (Москва, Российская Федерация), kryukov@theory.sinp.msu.ru

Степанова Маргарита Михайловна, доцент, кандидат физ.-мат. наук, Санкт-Петербургский государственный университет (Санкт-Петербург, Российская Федерация), mstep@mms.nw.ru

Приходько Николай Валерьевич, программист II категории, Новгородский государственный университет имени Ярослава Мудрого (Великий Новгород, Российская Федерация), niko2004x@mail.ru

Шамардин Лев Витальевич, старший научный сотрудник, кандидат физ.-мат. наук, НИИ ядерной физики имени Д.В. Скобельцына МГУ имени М.В. Ломоносова (Москва, Российская Федерация), shamardin@theory.sinp.msu.ru

Демичев Андрей Павлович, старший научный сотрудник, кандидат физ.-мат. наук, НИИ ядерной физики имени Д.В. Скобельцына МГУ имени М.В. Ломоносова (Москва, Российская Федерация), demichev@theory.sinp.msu.ru

EFFICIENT SUBMISSION OF HYBRID PARALLEL TASKS IN GRID

A.P. Kryukov, Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University (Moscow, Russian Federation),
M.M. Stepanova, Saint Petersburg State University (Saint Petersburg, Russian Federation),
N.V. Prikhodko, Yaroslav-the-Wise Novgorod State University (Novgorod, Russian Federation),
L.V. Shamardin, Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University (Moscow, Russian Federation),
A.P. Demichev, Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University (Moscow, Russian Federation)

The method of efficient submission of hybrid tasks using both MPI and OpenMP technologies is considered. For flexible control of parameters of the parallel tasks submission to supercomputers (SC), the specification of the task description language was expanded. Support of the new attributes was implemented for all key components of the infrastructure. Interaction of the submission web service with a local resource manager is organized through special handlers for tasks of different types (single, openmp, mpi or hybrid). This provides transmission of the correct parameters for reservation of resources and task submission to the local SC-manager. The provided solution was tested on a grid testbed deployed on the basis of GridNNN middleware.

Keywords: parallel computing, distributed computing, grid, hybrid tasks.

References

1. Stepanova, M.M., Stesik, O.L. 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 P. 383–387.
2. Ilyin, V.A., Korenkov, V.V., Kryukov, A.P. GridNNS: sostoyanie i perspektivy [GridNNN: Status and the Perspectives]. Trudy 5-i mezhdunarodnoi konferencii «Raspredelennye vychisleniya i Grid-tehnologii v nauke i obrazovanii» (Dubna, 16–21 iyulya 2012). [Distributed Computing and Grid-Technologies in Science and Education: Pro-

ceedings of the 5th International.Conference. (Dubna, 16-21 July, 2012)]. Dubna, JINR, 2012. P. 332-336.

3. Burke S., Campana S., Lanciotti E. et al. gLite 3.2 User Guide. Manual Series. CERN-LCG-GDEIS-722398. 2012. URL: http://edms.cern.ch/document/722398 (accessed 03.04.2013).
4. MPI-Start. URL: http://grid.ifca.es/wiki/Middleware/MpiStart (accessed: 22.02.2013).
5. Extended Resource Specification Language. Reference Manual for ARC versions 0.8 and above. NORDUGRID_MANUAL-4. 2013. URL: http:www.nordugrid./documents/arc-ui.pdf (accessed 03.04.2013).
6. MPI: A Message-Passing Interface Standard. Version 2.2. – 2009. URL: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf (accessed: 22.02.2013).
7. OpenMP Application Program Interface. Version 3.1. 2011 URL: http://www.openmp.org/mp-documents/OpenMP3.1.pdf (accessed: 22.02.2013).
8. Makris I. Mixed Mode Programming on Clustered SMP Systems M.Sc. Thesis in High Performance Computing. The University Of Edinburgh, 2005. 108 p.
9. Rabenseifner R., Hager G., Jost G. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. Proceedings of 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. Nov. 2009. P. 427–236.
10. Rane A., Stanzione D. Experiences in tuning performance of hybrid MPI/OpenMP applications on quad-core systems. Proceedings of 10th LCI International Conference on High-Performance Clustered Computing – Jan. 2009. – P. 1–10.
11. Glazkova E.A., Popova N.N. Analiz effektivnosti gibridnogo parallelnogo programmirovaniya na primere sistemy BLUE GENE/P [The Analysis of Efficiency of Hybrid Parallel Programming on the Example of the System BLUE GENE/P]. Materialy Vserossiskoi nauchnoi konferencii «Nauchnyi servis v seti Internet» (Novorossiisk, 21–26 sentyabrya 2009) [Proccedings of the All-Russia Scientific Conference "Scientific Service in the Internet" (Novorossiisk, Russia, September, 21–26, 2009)] Moscow, Moscow University Publishing, 2009, P. 36–39.
12. Balaji P. MPI Forum Hybrid Programming Working Group. URL: http://meetings.mpi-forum.org/mpi3.0 hybrid.php (accessed: 22.02.2013).
13. Kryukov A.P., Demichev A.P., Ilyin V.A., Shamardin L.V. Osnovnye podhody k postroeniyu grid-infrastruktury nacionalnoi nanotehnologicheskoi seti [Basic approaches to creation of grid-infrastructure of national nanotechnological network]. Vychislitelnye tehnologii v estestvennyh naukah. Perspektivnye komputernye sistemy: ustroistva, metody i koncepcii: Trudy seminara (Tarusa, 24.03.2011) [Computing Technologies in Natural Sciences. Perspective computer systems: devices, methods and concepts: Seminar works (Tarusa, 24.03.2011)]. – Moscow, Publishing of ICR RAS, 2011. P. 51–68.
14. Demichev A.P., Ilyin V.A., Kryukov A.P., Shamardin L.V. Realizaciya programmnogo interfeisa grid-servisa Pilot na osnove arkhitekturnogo stilya REST [Implementation of the program interface of the Pilot grid-service on the basis of architectural style REST]. Vychislitelnye metody i programmirovanie [Numerical methods and programming. Advanced Computing.]. 2010. Vol. 11. P. 62–64.

Поступила в редакцию 23 марта 2013 г.

УДК 519.6

О ВОПРОСАХ РАСПАРАЛЛЕЛИВАНИЯ КРЫЛОВСКИХ ИТЕРАЦИОННЫХ МЕТОДОВ[1]

В.П. Ильин

В работе рассматриваются математические вопросы многообразных вычислительных технологий методов распараллеливания итерационных процессов крыловского типа для решения больших разреженных симметричных и несимметричных СЛАУ, возникающих при сеточных аппроксимациях многомерных краевых задач для систем дифференциальных уравнений. Характерным примером являются конечно-элементные приближения в газогидродинамических 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.

Для повышения производительности алгебраических решателей автором предлагается применение различных подходов: декомпозиции расчетной области с различными топологиями, типами краевых условий на смежных границах и размерами пересечений подобластей, методов 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 систем линейных алгебраических уравнений.

Ключевые слова: итерационные методы, подпространства Крылова, предобусловленные матрицы, декомпозиция областей, параллельные алгоритмы, программные и вычислительные технологии.

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

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии 2013».

эффективности с ростом порядков СЛАУ и/или количества вычислительных устройств. Архитектура 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 [2–8] и итерационные процессы в подпространствах Крылова [5, 1]. 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 разделе 1 мы даем постановку и особенности рассматриваемых задач, а также сравнительный анализ методов их решения. Раздел 2 посвящен как общим, так и специальным технологическим аспектам распараллеливания алгоритмов, а в последнем разделе описываются примеры технологических требований к программным реализациям алгебраических решателей и, в частности, к библиотеке KRYLOV, предварительные сообщения о которой опубликованы в [6, 8]. В заключении отмечается комплексность взаимосвязанных теоретических, алгоритмических и технологических вопросов, определяющих уровень высокопроизводительных вычислений математического и программного обеспечения для решения актуальных алгебраических систем.

1. 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 [3] 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.

1.1. О построении параллельных предобуславливателей

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 выполнять-

ся 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), [10].

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

$$\begin{aligned} &u^n = u^{n-1} + B^{-1}(f - Au^{n-1}) = u^{n-1} + B^{-1}r^{n-1} = \\ &= Tu^{n-1} + \bar{f},\ \ T = I - B^{-1}A,\ \ \bar{f} = B^{-1}f, \end{aligned} \tag{7}$$

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 (coarse grid correction, CGC), или агрегирование, а также идейно примыкающие сюда алгебраические многосеточные методы (algebraic multi-grid, 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 [9].

Развитие идей грубосеточной коррекции активно продолжается в различных направлениях, см. [11, 12] 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

$$\begin{array}{l} u^0 = u^{-1} + W_d A_d^{-1} W_d^T r^{-1}, \;\; r^0 = f - Au^0, \\ p^0 = [I - W_d A_d^{-1}(AW_d^T)]r^0, \;\; A_d = W_d^T A W_d. \end{array} \tag{12}$$

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.

В заключение данного раздела отметим, что если в матрице исходной СЛАУ (1) выделить главную блочную диагональ ($A = D - C, \;\; D = \{A_{p,p}\}$), то при условии ее разложения на блочно-треугольные множители систему можно переписать в следующем виде:

$$\begin{array}{l} Du = Cu + f, \;\; D = L_D U_D = (L_D L_D^t), \\ U_D u = L_D^{-1} C U_D^{-1} U_D u + L_D^{-1} f. \end{array} \tag{13}$$

Отсюда можно ввести формально предобусловленную СЛАУ

$$\begin{array}{l} \bar{A}\bar{u} \equiv (I - \bar{T})\bar{u} = \bar{f}, \;\; u = U_D^{-1}\bar{u}, \\ \bar{T} = L_D^{-1} C U_D^{-1} (= \bar{T}^t), \;\; \bar{f} = L_D^{-1} f, \end{array} \tag{14}$$

для решения которой естественно применять блочный (двусторонне предобусловленный) метод Якоби

$$\bar{u}^n = \bar{T}\bar{u}^{n-1} + \bar{f}, \;\; \bar{u}^n = U_D u^n. \tag{15}$$

Заметим, что если матрица A симметрична, то этим же свойством обладают и матрицы $T, \bar{A}$. Отметим, что в формулах (13) – (15) можно положить

$$L_D = D, \;\; U_D = I, \text{ или } \;\; L_D = I, \;\; U_D = D,$$

что будет соответствовать одностороннему предобуславливанию (левому или правому соответственно).

1.2. Некоторые особенности методов в подпространствах Крылова

Как видно из предыдущего раздела, итерационные методы (мы подразумеваем «в подпространствах Крылова», как оптимальные) можно применять для решения как «глобальной» СЛАУ, т.е. для ускорения блочного метода Якоби, так и для «локальных» систем в подобластях. В дальнейшем крыловские алгоритмы мы рассмотрим в единообразной форме, независимо от того, применяются они на внешних или внутренних итерациях, основой которой является A^s-ортогонализация различных векторов, $s = 0$ или $s = 1$, см. [1].

Если предобусловленная СЛАУ $\bar{A}\bar{u} = \bar{f}$ симметрична, то наиболее экономичными являются методы сопряженных градиентов или сопряженных невязок (CG или CR, для $s = 0$ или $s = 1$ соответственно), описываемые следующими формулами:

$$\begin{array}{l} r^0 = \bar{f} - \bar{A}\bar{u}^0 = \hat{u}^1 - \bar{u}^0, \;\; \hat{u}^1 = T\bar{u}^0 + \bar{f}, \;\; p^0 = r^0, \\ \bar{u}^{n+1} = \bar{u}^n + \alpha_n^{(s)} p^n, \;\; \alpha_n^{(s)} = \rho_n^{(s)}/\delta_n^{(s)}, \;\; \rho_n^{(s)} = (\bar{A}^s r^n, r^n), \\ \delta_n^{(s)} = (\bar{A}p^n, \bar{A}^{(s)}p^n), \;\; r^{n+1} = r^n - \alpha_n^{(s)}\bar{A}p^n, \\ p^{n+1} = r^{n+1} + \beta_n^{(s)} p^n, \;\; \beta_n^{(s)} = \rho_{n+1}^{(s)}/\rho_n^{(s)}. \end{array} \tag{16}$$

Различные методы решения несимметричных систем базируются на A^s-ортогонализация Арнольди с предобуславливанием. Поскольку в общем случае предобуславливающие матрицы B_n могут отличаться на разных итерациях, то мы рассматриваем «гибкую» (flexible) ортогонализацию в следующем виде:

$$\begin{array}{l} u^n = u^0 + y_1 v^1 + ... + y_n v^n, \;\; (v^n, A^s v^k) = d_n^{(s)}\delta_{k,n}, \\ d_n^{(s)} = (v^n, A^s v^n), \\ v^{n+1} = AB_n^{-1}v^n - \sum_{k=1}^n h_{k,n}^{(s)} v^k, \;\; v^1 = r^0 = f - Au^0, \;\; n = 1, 2, ... \\ h_{k,n}^{(s)} = \frac{(Av^n, A^s v^k)}{(A^s v^k, v^k)}, \;\; k = 1, ..., n+1, \;\; V_{n+1} = (v^1, ..., v^{n+1}) \\ \bar{H}_n = \{h_{k,n}\} = \left[\begin{array}{c} H_n \\ e_n^t \end{array} \right] \in \mathcal{R}^{n+1,n}, \;\; H_n \in \mathcal{R}^{n,n}. \end{array} \tag{17}$$

Получаемые предобусловленные векторы $B_k^{-1}v_k$ образуют базис подпространства Крылова

$$\mathcal{K}_{n+1}(r^0, A) = \text{span}\{B_1^{-1}v^1, ..., B_{n+1}^{-1}v^{n+1}\} = \text{span}\{B_1^{-1}r^0, AB_2^{-1}r^0, ..., A^n B_{n+1}^{-1}r^0\}, \tag{18}$$

и на их основе формируются алгоритмы полной A^s-ортогонализации или методы обобщенных A^s-минимальных невязок (FOM, или A-FOM, GMRES, или A-GMRES), см. [1, 5].

Внешние итерации оканчиваются по выполнению условия

$$(r^n, r^n) \le \varepsilon_{ex}^2 (\bar{f}, \bar{f}),$$

где $\varepsilon_{ex} \ll 1$ характеризует точность решения. Оптимизация критериев останова внутренних итераций — это непростой вопрос, и здесь параметры точности ε_{in}^n в принципе должны зависеть от номера n внешней итерации.

2. Технологические вопросы реализации параллельных итерационных алгоритмов

Рассматриваемые в данном разделе 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 [13]. 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 [14]. 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 [15] 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 [16] 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 [17] 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 [18] (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 [10]. В этом случае исходная СЛАУ формально редуциру-

ется 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 количества 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.

3. 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 [19], PETSc [20] и Saad Software [21]. 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.

Заключение

В работе рассмотрены актуальные математические и технологические проблемы решения сверхбольших плохо обусловленных СЛАУ, принципиально влияющие на эффективность параллельных программных реализаций алгоритмов для МВС сложной современной архитектуры. Основное положение заключается в комплексности тесно взаимосвязанных вопросов: обоснование, сходимость и оптимизация крыловских процессов для широкого многообразия типов и свойств матриц; алгебро-геометрические аспекты предобуславливания СЛАУ; алгоритмические и технологические особенности различных видов декомпозиции областей; основные подходы к ускорению крыловских итерационных методов на базе грубосеточной коррекции, дефляции и других приемов улучшения или расширения базисных векторов для подавления ошибки; программные реализации параллельных алгебраических решателей на вычислительных системах с многоуровневой распределенной и общей памятью.

Работа поддержана грантом РФФИ №11-01-00205, а также грантами Президиума РАН №15.9-4 и ОМН РАН №1.3.3-4.

Литература

1. Ильин, В.П. Методы и технологии конечных элементов / В.П. Ильин. — Новосибирск: Изд-во ИВМиМГ СО РАН, 2007.
2. Лебедев, В.И. Вариационные алгоритмы метода разделения области / В.И. Лебедев, В.И. Агошков. — М., Препр. ОВМ РАН; № 54. — 1983.
3. Bramble, J.H. Convergence estimates for product iterative methods with applications to domain decomposition / J. Bramble, J. Pasciak, J. Wang, J. Xu // Math. Comp. — 1991. — Vol. 57, № 195. — P. 1–21.
4. Ильин, В.П. Параллельные методы и технологии декомпозиции областей. / В.П. Ильин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». — 2012. — Вып. 1. — №. 46(305). — С. 31–44.
5. Saad, Y. Iterative Methods for Sparse Linear Systems, Second Edition / Y. Saad. — SIAM, 2003.
6. Krylov: библиотека алгоритмов и программ для решения СЛАУ / Д.С. Бутюгин, В.П. Ильин, Е.А. Ицкович и др. // Современные проблемы математического модели-

рования. 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. Ильин, В.П. Проблемы высокопроизводительных технологий решения больших разреженных СЛАУ / В.П. Ильин // Вычислительные методы и программирование. — М., МГУ. — 2009. — Т. 10, № 1. — С. 141–147.

8. Бутюгин, Д.С. Методы параллельного решения СЛАУ на системах с распределенной памятью в библиотеке Krylov / Д.С. Бутюгин, В.П. Ильин, Д.В. Перевозкин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». — 2012. — Т. 47, № 306. — С. 5–19.

9. Intel Math Kernel Library from Intel. URL: `http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mklman/index.htm` (дата обращения: 15.02.2013).

10. Ильин, В.П. Параллельные методы декомпозиции в пространствах следов / В.П. Ильин, Д.В. Кныш // Вычислительные методы и программирование. — 2011. — Т. 12, № 1. — С. 100–109.

11. Brezina, M. An improved convergence analysis of smoothed aggregation algebraic multigrid / M. Brezina, P. Vanek, P.S. Vassilevsky // Numer. Linear Algebra Appl. — 2012. — Vol. 19. — P. 441–469.

12. Farhat, C. FETI-DP: A dual-primal unified FETI method. Part I: A faster alternative to the two-level FETI method / C. Farhat, M. Lesoinne, P. LeTollei, K. Pierson, D. Rixen // Int. J. Numer. Math. Engrg. — 2001. — Vol. 50. — P. 1523–1544.

13. Кластер НКС-30Т: URL: `http://www2.sscc.ru/HKC-30T/HKC-30T.htm` (дата обращения: 15.02.2013).

14. Message Passing Interface at Open Directory Project: URL: `http://www.dmoz.org/Computers/Parallel_Computing/Programming/Libraries/MPI/` (дата обращения: 15.02.2013).

15. Малышкин, В.Э. Параллельное программирование мультикомпьютеров / В.Э. Малышкин, В.Д. Корнеев // Новосибирск: Изд. НГТУ, 2006.

16. CUDA Tools & Ecosystem: URL: `http://developer.nvidia.com/cuda-tools-ecosystem` (дата обращения: 15.02.2013).

17. Bell, N. CUSP: Generic parallel algorithms for sparse matrix and graph computations / N. Bell, M. Garland // URL: `http://cusp-library.googlecode.com` (дата обращения: 15.02.2013).

18. Karypis, G. A fast and high quality multilevel scheme for partitioning irregular graphs / G. Karypis, V. Kumar // SIAM J. Sci. Comp. — 1999. — Vol. 20, № 1. — P. 359–392.

19. Hypre: URL: `http://acts.nersc.gov/hypre/` (дата обращения: 15.02.2013).

20. PETSc: Home Page: URL: `http://www.mcs.anl.gov/petsc/` (дата обращения: 15.02.2013).

21. Yousef Saad — SOFTWARE: URL: `http://www-users.cs.umn.edu/~saad/software/` (дата обращения: 15.02.2013).

Ильин Валерий Павлович, г.н.с. ИВМиМГ СО РАН, д.ф.-м.н., профессор НГУ, e-mail: ilin@sscc.ru

ON THE QUESTIONS OF PARALLELIZED KRYLOV'S ITERATIVE METHODS

V.P. Il'in, Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of the Russian Academy of Sciences (Novosibirsk, Russian Federation)

Mathematical questions of various computational technologies of parallelelized iterative processes of Krylov's type for solving large sparse symmetric and non-symmetric SLAEs, obtained in grid approximations of multi-dimensional boundary value problems for PDEs, are considered. Example are presented by finite approximations in gas-hydrodynamical applications, where five unknowns in each node are defined and corresponding SLAEs have small-block structure. The base of used algorithms is flexible generalized minimal residual, FGMRES, method with dynamical preconditioners of additive type, which presents an upper level of two-step iterarive Swartz algorithm.

High performance of algebraic solvers is provided by using different approaches: domain decompositions of various topologies, boundary conditions and sizes of subdomain overlapping, coarse grid correction, deflation and aggregation, and incomplete factorizations of matrices. The unified formulations of using algorithms as well as the questions of computational efficiency and scalable parallelization at the geterogenous supercomputers are described. The examples of technical requirements for peculiarities of program implementation of the libraries of parallel algorithms for solving systems of linear algebraic equation, are presented.

Keywords: iterative methods, Krylov subspaces, preconditioned matrices, domain decomposition, parallel algorithms, program and computational technologies.

References

1. Il'in V.P. Metody i tekhnologii konechnykh elementov [Finite element methods and technologies]. Novosibirsk, NSC Publ., 2007.
2. Lebedev V.I., Agoshkov V.I. Variazionnyi method dekompozitsii oblastei [Variational domain decomposition method]. Moscow, DCMRAS, preprint No 54, 1984.
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, No 195. P. 1–21.
4. Il'in V.P. Parallelnye methody i tehnologii dekompozitsii oblastei [Parallel domain decomposition methods and technologies]. Vestnik YUURGU. Seriya "Vychislitelnaya matematika i informatika"[Bulletin of South Ural State University. Seriers: Computational Mathematics and Software Engineering]. 2012. Vol. 46, No 305. P. 31–44.
5. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, 2003.
6. Butyugin D.S., Il'in V.P., Itskovich E.A. et all. Krylov: biblioteka algoritmov i program dlya teshenia SLAU.[Krylov: the library of algorithms and programs for solving SLAEs]. Modern problems of math. modeling. –Rostov-Don, YUFU Publ., 2009. P. 110–128.
7. Il'in V.P. Problemy vysokoproizvoditelnyh tehnologiy reshenia bolshih redkih SLAU [Problems of high performance technologies of solving large sparse SLAEs] Vychislitelnye

metody i programmirovanie [Computational methods and programming]. 2009. Vol. 10, No 1. P. 141–147.

8. Butyugin D.S., Il'in V.P., Perevozkin D.V. Metodu parallelnogo resheniya SLAU na systemah s raspredelennoi pamyatyu [Methods of parallel solving SLAEs on the systems with distributed memory]. Vestnik YUURGU. Seriya "Vychislitelnaya matematika i informatika"[Bulletin of South Ural State University. Seriers: Computational Mathematics and Software Engineering]. 2012. Vol. 47, No 306. P. 5–19.
9. Intel (R) Math Kernel Library from Intel. URL: `http://software.intel.com/en-us/intel-mkl` (accessed 12 February 2013).
10. Il'in V.P., Knysh L.V. Parallelnye metody dekompozitsii v prostranstve sledov [Parallel domain decomposition methods in trance subspaces]. Computational methods and programming. 2011. Vol. 12, No 1. P. 100–109.
11. 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.
12. 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.
13. Cluster HKC-30T: URL: `http://www2.sscc.ru/HKC-30T/HKC-30T.htm` (accessed 12 February 2013).
14. Message Passing Interface at Open Directory Project: URL: `http://www.dmoz.org/Computers/Parallel_Computing/Programming/Libraries/MPI/` (accessed 12 February 2013).
15. Malyshkin V.E., Korneev V.D. Parallelnoe programmirovanie multikompyuterov [Parallel programming the multi-computers]. Novosibirsk, NSTU Publ., 2006, 310 p.
16. CUDA Tools & Ecosystem: URL: `https://developer.nvidia.com/cuda-tools-ecosystem` (accessed 12 February 2013).
17. Bell N., Garland M. CUSP: Generic parallel algorithms for sparse matrix and graph computations: URL: `http://cusp-library.googlecode.com` (accessed 12 February 2013).
18. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comp. 1999. Vol. 20, No 1. P. 359–392.
19. Hypre: URL: `http://acts.nersc.gov/hypre/` (accessed 12 February 2013).
20. PETSc: Home Page: URL: `http://www.mcs.anl.gov/petsc/` (accessed 12 February 2013).
21. Yousef Saad – SOFTWARE: URL: `http://www-users.cs.umn.edu/~saad/software/` (accessed 12 February 2013).

Поступила в редакцию 13 марта 2013 г.

УДК 519.67

О РАСПАРАЛЛЕЛИВАНИИ РЕШЕНИЯ КРАЕВЫХ ЗАДАЧ НА КВАЗИСТРУКТУРИРОВАННЫХ СЕТКАХ

В.М. Свешников, Б.Д. Рыбдылов

Рассматриваются технологические аспекты решения краевых задач на предлагаемых квазиструктурированных сетках специального вида. Их особенностью является то, что и макросетка в расчетной области, и подсетки в подобластях являются структурированными и прямоугольными сетками, что обеспечивает создание экономичных структур данных и эффективное применение численных алгоритмов. В то же время, результирующая квазиструктурированная сетка является адаптивной к неоднородностям внутри области и к сложной конфигурации внешней границы, что достигается путем регулировки плотности узлов подсеток и локальной модификации сетки вблизи криволинейной границы. Существенным является то, что подсетки могут быть несогласованными. Решение ищется предлагаемым вариантом метода декомпозиции, который основан на отдельной аппроксимации краевой задачи на интерфейсе и в подобластях. Распараллеливание проводится путем группировки подобластей в объединения с целью балансировки загрузки процессоров. Приводятся оценки эффективности распараллеливания на примере решения модельной задачи на различном числе вычислительных ядер, различных сетках и объединениях.

Ключевые слова: краевые задачи, параллельные алгоритмы и технологии, декомпозиция области, квазиструктурированная сетка.

Введение

Адаптивные квазиструктурированные сетки, рассматриваемые в настоящей статье, имеют ряд преимуществ по сравнению со структурированными и неструктурированными сетками. По отношению к первым они дают возможность избавиться от лишнего числа узлов, которые зачастую вводятся лишь для поддержки структурированности, а по отношению ко вторым имеют гораздо более простую структуру данных небольшого объема, что позволяет более эффективно строить численные алгоритмы. Важным является то, что подсетки, из которых состоит результирующая квазиструктурированная сетка, могут быть несогласованными. Это дает возможность автономной аппроксимации решаемого дифференциального уравнения в подобластях, на которые разбивается расчетная область. Сетки, обладающие данным свойством, рассматривались в работах Ю.А. Кузнецова [1], Ю.В. Василевского [2], К. Бернарди [3] и других авторов. Характерным для этих работ является применение метода конечных элементов и множителей Лагранжа для поиска решений в подобластях и на границе их сопряжения (интерфейсе). В настоящей работе рассматривается конечно-разностный подход [4], основанный на непосредственной аппроксимации уравнения Пуанкаре–Стеклова на интерфейсе при помощи дискретных функций Грина, который дает простой параллельный алгоритм для поиска решения.

Процесс распараллеливания алгоритмов решения краевых задач на квазиструктурированных сетках имеет некоторые особенности, связанные с равномерной загрузкой процессоров. Обычно используемое при распараллеливании отображение одна подобласть – один процессор в этом случае недопустимо вследствие того, что подсетки в подобластях могут быть несогласованными, то есть иметь существенно различное число

узлов. Это приводит к разбалансировке загрузки процессоров. Для её выравнивания предлагается группировать подобласти в объединения, имеющие приблизительно одинаковое число узлов, что привносит некоторые изменения в технологию распараллеливания, так как в объединение могут входить как подобласти, не требующие межпроцессорных пересылок, так и подобласти, для которых они необходимы. В последнем случае обмены происходят с процессорами-соседями, номера которых должны быть предварительно определены и сохранены. С целью исследования эффективности данного нового подхода был проведен цикл численных экспериментов, результаты которых приводятся ниже.

1. Алгоритмы построения квазиструктурированных сеток и решения на них краевых задач

В этом разделе мы кратко приведем сведения по алгоритмам построения квазиструктурированных сеток и решения на них краевых задач, которые необходимы для изложения дальнейшего материала. Более подробную информацию по данному вопросу можно почерпнуть в работах [4, 5].

Расчетной областью, в которой ищется решение $\varphi(x)$ краевой задачи, будем называть замыкание $\overline{G} = G \bigcup \Gamma$, где G – открытая, односвязная, ограниченная область в двумерном вещественном пространстве, в которой задано решаемое уравнение

$$Lu = f, \quad x \in G, \tag{1}$$

а Γ – граница, на которой заданы краевые условия

$$lu = g, \quad x \in \Gamma. \tag{2}$$

Здесь L, l – соответственно операторы уравнения и граничных условий, а f, g – задаваемые функции. Отметим, что задача (1), (2) рассматривается как в декартовых x, y, так и в цилиндрических $x = r, y = z$ координатах соответственно для плоских и осесимметричных областей.

В прямоугольнике $\overline{R} = \left\{0 \le x \le D_{x,}\ 0 \le y \le D_y\right\}$, где D_x, D_y – заданы, описанном вокруг расчетной области ($\overline{G} \subset \overline{R}$), построим прямоугольную равномерную макросетку

$$\overline{\Omega}_H = \left\{ X_I = IH_x, Y_J = JH_y, \quad I = \overline{0, N_x}, J = \overline{0, N_y}, \quad H_x = \frac{D_x}{N_x}, H_y = \frac{D_y}{N_y} \right\},$$

где N_x, N_y – заданные целые числа, с шагами $H_x, H_y >> h$ (h – максимальный шаг сетки, на которой аппроксимируется задача (1),(2)). Фактически мы тем самым проводим декомпозицию G на подобласти $G_{I,J}$. Среди них будем различать $G_{I,J}^{(0)}$ – *внешние*, $G_{I,J}^{(1)}$ – *внутренние*, $G_{I,J}^{(2)}$ – *граничные* подобласти, которые соответственно не содержат точек G, содержат только точки G, содержат точки G и Γ. Точки пересечения координатных линий $x = X_I$, $y = Y_J$ макросетки являются макроузлами, которые обозначим как T_p, а отрезки координатных линий, принадлежащие расчетной области, образуют границу сопряжения подобластей γ или интерфейс.

В замкнутых подобластях $\overline{G}_{I,J}^{(1)}$, $\overline{G}_{I,J}^{(2)}$ построим равномерные прямоугольные подсетки

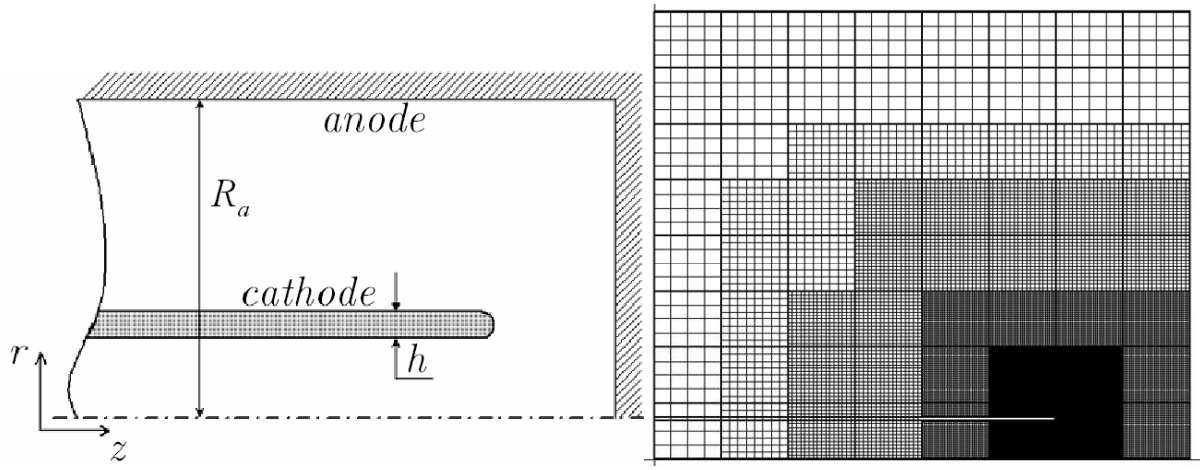
$$\overline{\Omega}_{h,k} = \left\{ x_{i_k} = X_I + i_k h_{x,k},\ y_{j_k} = Y_J + j_k h_{y,k},\quad i_k = \overline{0, n_{x,k}},\ j_k = \overline{0, n_{y,k}} \right\}$$

с шагами $h_{x,k} = \dfrac{X_{I+1} - X_I}{n_{x,k}}$, $h_{y,k} = \dfrac{Y_{J+1} - Y_J}{n_{y,k}}$. Здесь введена единая нумерация подобластей и подсеток по индексу $k = \overline{1,K}$. По аналогии с подобластями будем различать подсетки двух типов: *внутренние* $\overline{\Omega}^{(1)}_{h,k}$ и *граничные* $\overline{\Omega}^{(2)}_{h,k}$.

Граничные подсетки $\overline{\Omega}^{(2)}_{h,k}$ подвергаются локальной модификации, состоящей в сдвиге приграничных узлов на границу Γ, в результате которой они преобразуются в подсетки $\overline{\Omega}^{(\mu)}_{h,k}$. Объединение подсеток $\overline{\Omega}^{(1)}_{h,k}$ и $\overline{\Omega}^{(\mu)}_{h,k}$ образует результирующую квазиструктурированную сетку $\overline{\Omega}_h$.

На границе сопряжения подобластей γ введем сетку ω_h, не содержащую макроузлов T_p: $\omega_h = \{(x_i, y_i) \in \gamma, (x_i, y_i) \neq T_p,\ i = \overline{1, N_\omega}\}$, где N_ω – известное целое число. Дополнив ω_h макроузлами T_p, образуем сетку $\bar{\omega}_h = \{\omega_h, T_1, T_2, \ldots T_P\}$.

На рис. 1а приведен пример геометрии практической задачи, которая представляет собой так называемый сильноперенапряженный промежуток, предназначенный для формирования мощных электронных пучков в приборах сильноточной электроники. Ее особенностью является то, что отношение максимального размера к минимальному составляет 35000 (h=0,0001, R_a=3,5). На рис. 1б показана квазиструктурированная сетка для расчета данного прибора, которая стягивается к острию катода, то есть подобласти, где требуется получить детальное решение.



а) б)

Рис. 1. Пример геометрии сильноперенапряженного промежутка и квазиструктурированная сетка

Краевая задача (1),(2) в подобластях $\overline{G}^{(1)}_{I,J}$, $\overline{G}^{(2)}_{I,J}$ методом конечных разностей, конечных элементов или конечных объемов на сетках $\overline{\Omega}^{(1)}_{h,k}$, $\overline{\Omega}^{(\mu)}_{h,k}$ заменяется приближенной задачей, причем на интерфейсе ставится граничное условие Дирихле

$$\varphi_h\big|_\gamma = v_h,$$

где φ_h, v_h – приближенные значения функций u, v, (v – след φ на γ).

Для отыскания v_h применяется метод декомпозиции расчетной области на подобласти, сопрягаемые без наложения, предложенный и подробно изложенный в работе [4]. Он основан на непосредственной аппроксимации уравнения Пуанкаре–Стеклова

$$\left(\frac{\partial\varphi}{\partial\vec{n}}\right)^{+}-\left(\frac{\partial\varphi}{\partial\vec{n}}\right)^{-}=0 \tag{3}$$

при помощи дискретных функций Грина в узлах сетки ω_h и исходного уравнения (1) в макроузлах. Здесь $\vec{n}$ – нормаль к γ, верхние индексы показывают принадлежность объекта к различным подобластям. Результатом является система линейных алгебраических уравнений

$$Av_h+b=0$$

относительно функции v_h с невырожденной квадратной матрицей A и известным вектором b. Ее решение осуществляется итерационным методом вида

$$v_h^{(n+1)}=\Lambda\left(Av_h^{(n)},v_h^{(n)}\right), \tag{4}$$

где n=0,1,... – номер итерации, а Λ – матрица, определяющая конкретный алгоритм, то есть здесь используется лишь сам вектор $v_h^{(n)}$ и действие $Av_h^{(n)}$ матрицы на вектор. Этим условиям удовлетворяет, например, семейство быстросходящихся итерационных методов в подпространствах Крылова [6]. На каждом шаге данного итерационного процесса необходимо вычислять величины

$$f_{h,i}^{(n)}=\left(d_h^{(+)}\varphi_h^{(n,+)}\right)_i-\left(d_h^{(-)}\varphi_h^{(n,-)}\right)_i, \tag{5}$$

в узлах сетки ω_h, где $d_h^{(+)}, d_h^{(-)}$ – конечно-разностные операторы, аппроксимирующие нормальные производные в уравнении (3).

2. Параллельные технологии решения задач

Распараллеливанию подлежит итерационный процесс (4) по подобластям, который занимает подавляющую часть времени решения всей задачи. Важным звеном в технологической цепи решений по достижению эффективности распараллеливания является отображение сеточных данных на вычислительную сеть. Обычно принятый способ отображения «одна подобласть – один процессор» в данном случае не эффективен, так как подобласти могут содержать существенно различное число узлов, в которых вычисляются значения искомой функции (в дальнейшем – счетных узлов), что приводит к дисбалансу загрузки процессоров.

Поэтому подобласти группируются в объединения U_m, $m=\overline{1,M}$, где M – известное число, с целью обеспечения приблизительно равной загрузки процессоров. Для этого в каждое объединение $U_m=\bigcup\limits_k\overline{\Omega}_{h,k}^{(m)}$ включаются такие подсетки $\overline{\Omega}_{h,k}^{(m)}$, которые давали бы в сумме $N_m=\sum\limits_k N_k^{(m)}$ число счетных узлов такое, что $N_m\approx N_U$, где N_U – заданная величина, а $N_k^{(m)}$ – число счетных узлов в подобластях.

Алгоритм группировки подобластей для идеальной балансировки должен строиться на решении задачи линейного программирования. При этом может оказаться, что в объединение включаются не только соседние подобласти, но и подобласти, разделенные подобластями из других объединений. Последнее обстоятельство может привести к

значительному увеличению объема самых медленных операций, а именно операций пересылки в системах с разделенной памятью.

Обмены не происходят между соседними подобластями одного объединения, в связи с чем был принят следующий *алгоритм построения объединений*. Задается число N_U, которое должно быть больше или равно максимальному числу узлов в подобласти. Для каждой свободной, то есть не включенной ни в одно объединение, подобласти просматриваются свободные соседи. Количество узлов просмотренных подсеток суммируется. Если после просмотра k-й подсетки в m-м объединении окажется, что $N_m > N_U$, то процесс группировки заканчивается, причем при $N_m - N_U > 0.5N_k^{(m)}$ последняя подсетка не включается в данное объединение.

Строится отображение «одно объединение – один процессор». На текущем процессоре на каждой n-й итерации выполняются следующие вычислительные работы:

1. Вычисление искомой функции в подобластях, то есть решение приближенной задачи в подобластях с заданным значением $v_h^{(n)}$ на интерфейсе.

2. Расчет нормальных производных на сторонах подобластей, входящих в интерфейс. По окончании расчетов массивы производных пересылаются на процессоры-соседи, номера которых определяются до проведения итераций.

3. Вычисление разностей производных (5) на смежных сторонах подобластей. Перед проведением вычислений с каждого процессора-соседа читаются необходимые массивы производных.

4. Реализация очередного шага итерационного процесса по подобластям, то есть вычисление нового значения $v_h^{(n+1)}$. При этом могут потребоваться обмены информацией с процессорами-соседями, необходимой для реализации конкретного итерационного процесса, определяемого формулой (4).

Рассмотренные алгоритмы и технологии были реализованы с использованием системы параллельного программирования MPI [7]. Расчеты проводились на кластере НКС-30Т Сибирского суперкомпьютерного центра СО РАН.

3. Численные эксперименты

Цель численных экспериментов – исследование эффективности распараллеливания решения краевых задач на квазистуктурированных сетках. Наибольший интерес при этом представляют несогласованные сетки, так как одно из основных практических применений предлагаемого подхода – это решение задач с сильными неоднородностям на несогласованных сетках.

Рассматривалась модельная краевая задача для уравнения Лапласа:

$$\begin{cases} \Delta u = 0 \quad в \quad G = \{0.1 \le x \le 0.6,\ 0 \le y \le 0.5\} \\ u|_\Gamma = f \end{cases}$$

с точным решением $\overline{u} = 1 + \lg\sqrt{x^2 + y^2}$ и значениями на границе $f = \overline{u}|_\Gamma$. Расчетная область покрывалась квазиструктурированной сеткой, которая представляла собой совокупность подсеток $\Omega_{h,k}$ в подобластях G_k. Подсетки $\Omega_{h,k}$ строились в шахматном порядке: соседние подобласти имели по каждому направлению число узлов, отличающееся в два раза, то есть были несогласованными. На рис. 2 приведен пример такой сетки, в

которой число узлов макросетки равно $N_x = N_y = 4$ и число узлов подсеток – 8×8, 16×16.

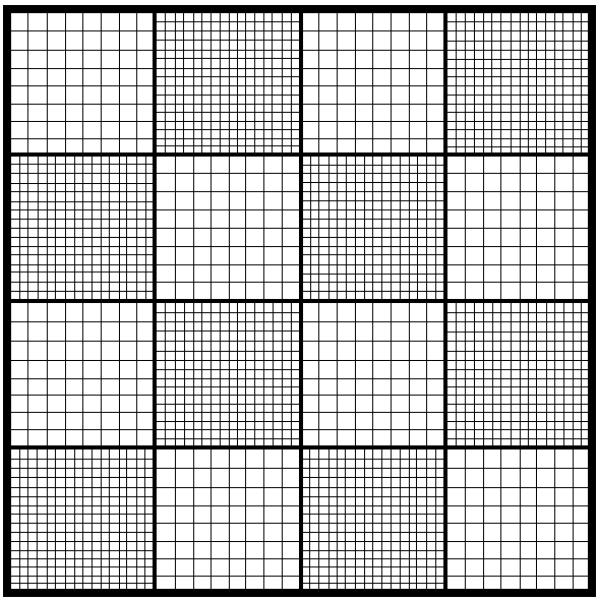


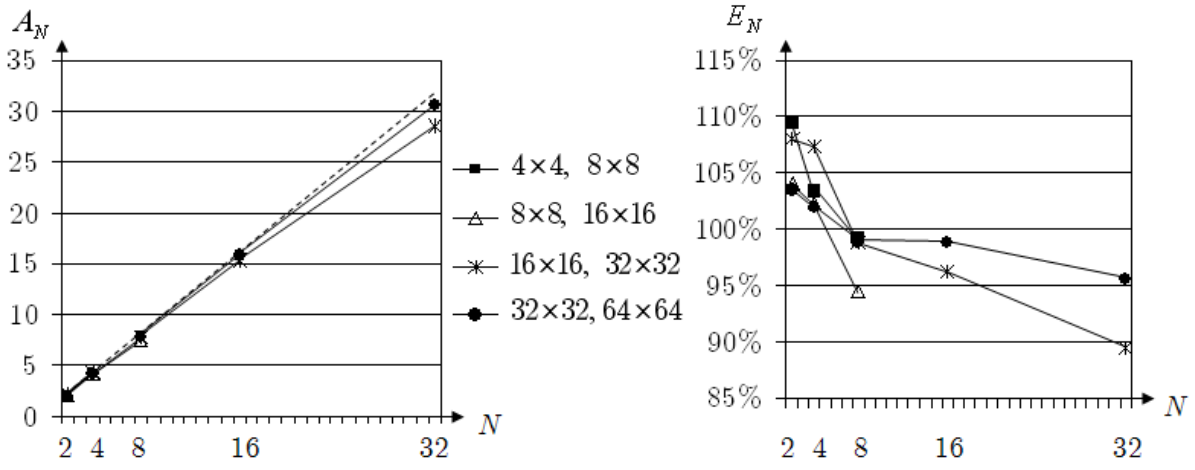
Рис. 2. Декомпозиция на 4×4 подобласти

Подсетки группировались в объединения естественным образом: слева направо, снизу вверх в зависимости от числа процессоров N, на которых производится расчет. Все объединения имели одинаковое число узлов. Так, при N=2 в объединения включались подобласти, лежащие снизу-сверху от горизонтальной линии, при N=4 – снизу-сверху и слева-справа от горизонтальной и вертикальной средних линий и т.д. Расчеты проводились для числа подобластей $N_x \times N_y = 32\times 32,\ 64\times 64$ и при числе процессоров N=2, 4, 8, 16, 32. Подсетки при этом имели следующее число узлов: $\Omega_{h,k} = \{4\times 4, 8\times 8\}$, $\{8\times 8, 16\times 16\}$ $\{16\times 16, 32\times 32\}$, $\{32\times 32, 64\times 64\}$. Таким образом, самая густая квазиструктурированная сетка имела количество узлов, приблизительно равное $((32\times 32)\times 32 + (64\times 64)\times 32)\times 64$, что составляет более 10 миллионов узлов. Уравнение Лапласа аппроксимировалось обычной пятиточечной схемой. Производные, входящие в формулу (5), вычислялись по трехточечным схемам второго порядка. В качестве итерационного процесса брался метод сопряженных градиентов. Сначала были проведены численные эксперименты, которые показали, что относительная точность решения данной задачи составляет приемлемую величину порядка долей процента. Затем, для исследования эффективности распараллеливания проводилось заданное число итераций, равное 100.

Результаты экспериментов представлены на рис. 3, 4, где ускорение вычислений A_N и эффективность процесса распараллеливания E_N (последнее – в процентах) вычисляются по формулам

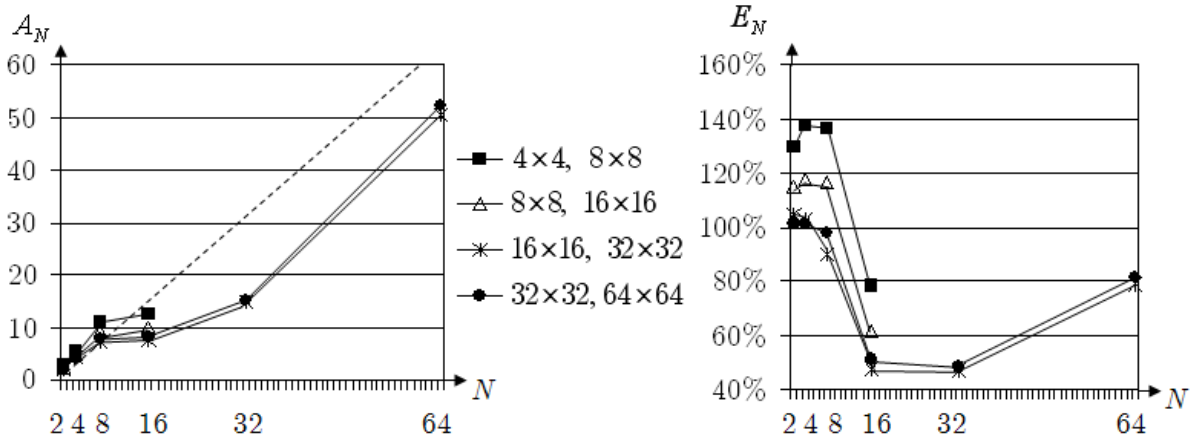
$$A_N = \frac{T_1}{T_N},\ \ E_N = \frac{A_N}{N}100,$$

при различных вычислительных параметрах (пунктирной линией изображено линейное ускорение).



а) Ускорение A_N б) Эффективность E_N

Рис. 3. Результаты для числа подобластей 32×32



а) Ускорение A_N б) Эффективность E_N

Рис. 4. Результаты для числа подобластей 64×64

Из приведенных результатов можно сделать следующие выводы:

1. Ускорение вычислений A_N растет с увеличением числа процессоров, что говорит о масштабируемости предлагаемого подхода.

2. Эффективность E_N падает с увеличением числа процессоров, что объясняется увеличением числа межпроцессорных обменов.

3. При малом числе процессоров (не более 8) наблюдается явление сверхлинейного ускорения, при котором эффективность превышает 100 %, которое особенно заметно на сетках с большим числом подобластей, то есть при густой макросетке. Объяснение этому следующее. Подобласти группируются в объединения. Каждое объединение обрабатывается одним процессором. Расчет подобластей, входящих в объединение, на одном процессоре проводится последовательно, то есть одна подобласть рассчитывается автономно, что приводит к эффективному использованию сверхбыстрой кэш-памяти компьютера. Кроме того, малое число процессоров (или малое число объединений) говорит о малых затратах на обмены. Эти два эффекта: ускорение за счет кэш-памяти, с одной стороны, и уменьшение числа обменов, с другой стороны, – дают явление сверхлинейного ускорения.

Заключение

В работе рассматривался вопрос о распараллеливании решения двумерных краевых задач на предлагаемых квазиструктурированных сетках. Данные сетки просты в использовании и адаптивны к решению, которое ищется итерационным методом декомпозиции. Проведены численные эксперименты по распараллеливанию решения модельной краевой задачи на несогласованных сетках. Результаты численных экспериментов показали, во-первых, работоспособность и адекватность предлагаемого подхода, и, во-вторых, наличие сверхлинейного ускорения за счет влияния сверхбыстрой кэш-памяти. В дальнейшем планируется проведение исследований по эффективному использованию данного явления в предлагаемых алгоритмах и технологиях.

Литература

1. Kuznetsov, Yu. Efficient iterative solvers for elliptic problems on nonmatching grids / Yu. Kuznetsov Yu. // Russian Journal of Numerical Analysis and Mathematical Modeling. – 1995. – Vol. 10, No 3. – P. 187–211.
2. Василевский, Ю.В. Методы решения краевых задач с использованием нестыкующихся сеток / Ю.В. Василевский // Труды Математического центра им. Н.И. Лобачевского. – Казань: УНИПРЕСС, 1999. – Т 2. – С. 94–121.
3. Bernardi, C. A new nonconforming approach to domain decomposition: the mortar element method / C. Bernardi, Y. Maday, A. Patera // Nonlinear partial differential equations and their applications. – Paris: College de France Seminar, 1994. – Vol. 11. – P. 13–51.
4. Свешников, В.М. Построение прямых и итерационных методов декомпозиции / В.М. Свешников // Сибирский журнал индустриальной математики. – 2009. – Т. 12, № 3(39). – С. 99–109.
5. Свешников, В.М., Построение квазиструктурированных локально-модифицированных сеток для решения задач сильноточной электроники / В.М. Свешников, Д.О. Беляев // Вестник ЮУрГУ, серия «Математическое моделирование и программирование». – 2012. – Вып. 14. – № 40(299). – С. 118–128.
6. Ильин, В.П. Методы конечных разностей и конечных объемов для эллиптических уравнений / В.П. Ильин – Новосибирск: Изд-во ИВМиМГ (ВЦ) СО РАН, 2001. – 318 с.
7. Корнеев, В.Д. Параллельное программирование в MPI / В.Д. Корнеев – Новосибирск: Изд-во ИВМиМГ (ВЦ) СО РАН, 2002. – 215 с.

Свешников Виктор Митрофанович, д.ф.-м.н., заведующий лабораторией вычислительной физики, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), victor@lapasrv.sscc.ru.

Рыбдылов Батор Доржиевич, аспирант, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), rybdylovbd@mail.ru.

ABOUT PARALLELIZATION OF SOLVING OF BOUNDARY VALUE PROBLEMS ON QUASISTRUCTURED GRIDS

V.M. Sveshnikov, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
B.D. Rybdylov, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation)

Technological components of boundary problems solution on offered quasi-structured grids of special kind is considered. The feature of these grids is that both macrogrid (coarse grid) in a whole domain and subgrids (local grids) in subdomains are structured and rectangular, it provides efficient structure of data and effective using of computational algorithms. At the same time, resulting quasi-structured grid is adaptive to irregularities within a domain and to complicated shape of domain boundary. It is essential that subgrids can be unmatched. One variant of domain decomposition methods for solving boundary problems is offered, this one is based on separate approximation of boundary problem on the interface and within the subdomains. In order to balance utilization of processors whole set of subdomains is divided into unions (groups) of subdomains. Estimates of paralellization efficiency was obtained for model problem using different number of processors, different grids and different unions of subdomains.

Keywords: boundary value problems, parallel algorithms and technologies, domain decomposition, quasistructured grids.

References

1. Kuznetsov Yu. Efficient iterative solvers for elliptic problems on nonmatching grids // Russian Journal of Numerical Analysis and Mathematical Modeling. 1995. Vol. 10, No 3. P. 187–211.
2. Vasilevskij Yu.V. Metody reshenija krajevykh zadach s ispolzovaniem nestykujuschikhsja setok [Methods for solving of boundary value problems using unmatched grids]. Trudy Matematicheskogo tsentra imeni N.I. Lobachevskogo [Proceedings of Mathematical Center of N.I. Lobachevski]. 1999. Vol. 2. Kazan: UNIPRESS. P. 94–121.
3. Bernardi C., Maday Y., Patera A. A new nonconforming approach to domain decomposition: the mortar element method // Nonlinear partial differential equations and their applications. Paris: College de France Seminar, 1994. Vol. 11. P. 13–51.
4. Sveshnikov V.M. Postroenie pryamykh i iteratsionnykh metodov decompozitsii [Construction of direct and iterative decomposition methods]. Sibirskij Zhurnal Industrial'noj Matematiki [Journal of Applied and Industrial Mathematics]. 2009. Vol. 12, No. 3(39). P. 99–109.
5. Sveshnikov V.M., Belyaev D.O. Postroenie kvazistrukturirovannykh lokalno-modifitsirovannykh setok dlya resheniya zadach sil'notochnoj elektroniki [Construction of quasi-structured locally modified grids for solving problems of high current electronics]. Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta, serija "Matematicheskoe modelirovanie i programmirovanie" [Journal of South-Ural State University, series "Mathematical modeling and programming"]. 2012. No. 40(299). P. 118–128.

6. Il'in V.P. Metody konechnykh raznostej i konechnykh ob'emov dlya ellipticheskikh uravnenij [Finite differences and finite volumes methods for elliptic equations]. Novosibirsk: ICM&MG SB RAS, 2001. 318 p.
7. Korneev V.D. Parallelnoe programmirovanie v MPI [Parallel programming in MPI]. Novosibirsk: Publ. of ICM&MG SB RAS, 2002. 215 p.

Поступила в редакцию 9 апреля 2013 г.

УДК 004.056.55

ИСПОЛЬЗОВАНИЕ ДЕТЕРМИНИРОВАННОЙ ФУНКЦИИ РАЗБИЕНИЯ НА МНОЖЕСТВА ДЛЯ РАСПАРАЛЛЕЛИВАНИЯ ρ-МЕТОДА ПОЛЛАРДА[1]

Е.Г. Качко, К.А. Погребняк

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.

Ключевые слова: дискретный логарифм, метод Полларда, эллиптическая кривая.

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.

[1]Статья рекомендована к публикации программным комитетом международной научной конференции «Параллельные вычислительные технологии 2013»

1. 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 $1 < x < n$.

1.1. 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

$$R_{i+1} = f(R_i) = \begin{cases} Q + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ P + R_i, & R_i \in S_3 \end{cases}$$

Пусть $R_{i+1} = a_i P + b_i Q$, тогда коэффициенты определяются следующим образом

$$a_{i+1} = \begin{cases} a_i \mod n, & R_i \in S_1 \\ 2a_i \mod n, & R_i \in S_2 \\ a_i + 1 \mod n, & R_i \in S_3 \end{cases}$$

$$b_{i+1} = \begin{cases} b_i + 1 \mod n, & R_i \in S_1 \\ 2b_i \mod n, & R_i \in S_2 \\ b_i \mod n, & R_i \in S_3 \end{cases}$$

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

$$\begin{aligned} R_i &= f\left(R_{i-1}\right) \\ R_{2i} &= f\left(f\left(R_{i-1}\right)\right) \end{aligned} \tag{1}$$

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

$$\begin{aligned} R_{i_0} &= a_{i_0} P + b_{i_0} Q \\ R_{2i_0} &= a_{2i_0} P + b_{2i_0} Q \end{aligned}$$

можно вычислить

$$x = \frac{a_{2i_0} - a_{i_0}}{b_{i_0} - b_{2i_0}}$$

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, $1 \le j \le N$ на практике определяется либо младшими, либо старшими разрядами R_i.

Например:

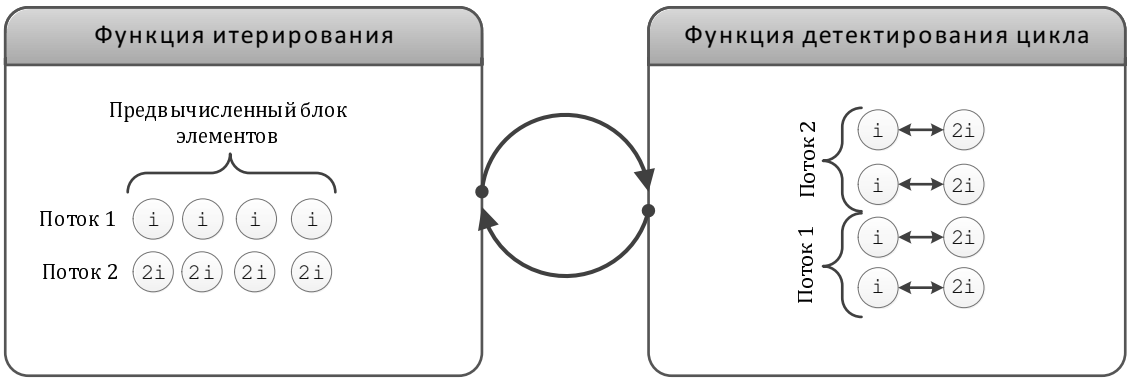
$$j = \nu(R_i) = x_i \mod N + 1,$$

где $\nu : G \to \{1, \dots N\}$ – функция разбиения группы на множества.

1.2. 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.



Параллельный алгоритм Полларда с предвычисленным блоком

Функция итерирования распараллеливается на этапе вычисления последовательностей $\{R_i\}_{i=0}^{m}$ и $\{R_{2i}\}_{i=0}^{m}$. Это достигается организацией вычислений последовательностей вида

$$\{R_{iw+j}\}_{i=0}^{l} \text{ и } \{R_{2(iw+j)}\}_{i=0}^{l}$$

где w – это размер блока вычислений, $0 \le j < w$, $l = \left\lceil \frac{m}{w} \right\rceil$.

Результатом выполнения функции итерирования точек являются два множества точек $\{R_i\}_{i=0}^{w}$ и $\{R_{2i}\}_{i=0}^{w}$, сравнение которых осуществляется по блокам, то есть

$$R_i = R_{2i}, 1 \le i \le \frac{w}{2}$$
$$R_i = R_{2i}, \frac{w}{2} < i \le w$$

Следует отметить, что вычисление точки R_{2i}, согласно формуле (1), 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

$$j = \xi(R_i) = i \mod N + 1. \tag{2}$$

Такая функция разбиения абелевой группы на множества позволяет сделать предсказание траектории перехода и построить композиционную функцию.

Предложение 1. Пусть задана детерминированная функция разбиения группы на множества $\xi(R_i) = i \mod N + 1$, тогда вычисление R_{2i} может быть представлено как

$$R_{2i} = g(S_{i-1}),$$

где $g : G \to G$, $S_{i-1} = R_{2(i-1)}$.

Доказательство. Определим функцию итерирования g следующим образом:

$$S_{i+1} = g(S_i) = \begin{cases} 2(Q + S_i), & S_i \in S_1 \\ S_i + P + Q, & S_i \in S_2 \\ 2S_i + P, & S_i \in S_3 \end{cases}$$

По определению $R_{2i} = f\left(f\left(R_{i-1}\right)\right)$. Так как функция разбиения группы на множества представлена уравнением (2), то

$$R_{2i} = \begin{cases} 2(R_{2(i-1)} + Q), & R_{2(i-1)} \in S_1 \\ R_{2(i-1)} + P + Q, & R_{2(i-1)} \in S_2 \\ 2R_{2(i-1)} + P, & R_{2(i-1)} \in S_3 \end{cases}$$

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.

1.3. 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:

$$R_{i+1} = f_{PG}(R_i) = \begin{cases} W_1 + R_i, & R_i \in S_1 \\ 2R_i, & R_i \in S_2 \\ W_2 + R_i, & R_i \in S_3 \end{cases}$$

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:

$$R_{i+1} = f_{TA}(R_i) = R_i \cdot W_{\nu(R_i)}, \nu(R_i) \in 1..r$$

Смешанная функция итерирования Теске записывается как:

$$R_{i+1} = f_{TM}(R_i) = \begin{cases} R_i \cdot W_{\nu(R_i)}, & \nu(R_i) \in 1..r \\ 2R_i, & \nu(R_i) \notin 1..r \end{cases}$$

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.

2. 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 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.

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

Таблица

Временные показатели методов Полларда

№ п/п	Методы Полларда	δ_t (с)	
		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

точки R_i множеству S_i, что позволило минимизировать потери, связанные с неправильным предсказанием переходов и в полной мере использовать параллельные вычисления.

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 двухъядерных 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. On the efficiency of Pollard's rho method for discrete logarithms / S. Bai, R. P. Brent // 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. Параллельный метод Полларда решения задачи дискретного логарифмирования в группе точек эллиптической кривой / Е.Г. Качко, К.А. Погребняк // 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. Методы распараллеливания алгоритма Полларда решения задачи дискретного логарифмирования для систем с общей памятью / И.Д. Горбенко, Е.Г. Качко, К.А. Погребняк // 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. Параллельный метод Полларда решения задачи дискретного логарифмирования в мультипликативной группе поля Галуа / И.Д. Горбенко, Е.Г. Качко, К.А. Погребняк // 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.

Елена Григорьевна Качко, кандидат технических наук, профессор, кафедра «Программная инженерия», Харьковский национальный университет радиоэлектроники (г. Харьков, Украина), ekachko@gmail.com.

Константин Анатольевич Погребняк, кандидат технических наук, кафедра «Безопасность информационных технологий», Харьковский национальный университет радиоэлектроники (г. Харьков, Украина), iitkostya@gmail.com.

USING A DETERMINISTIC PARTITIONING FUNCTION FOR POLLARD'S RHO METHOD PARALELLIZATION

E.G. Kachko, Kharkov National University of Radioelectronics (Kharkov, Ukraine)**,**
K.A. Pogrebnyak, Kharkov National University of Radioelectronics (Kharkov, Ukraine)

An improved method for parallelization of Pollard's algorithm for solving the discrete logarithm problem in a group of elliptic curve points and in a multiplicative group of a Galois field for shared memory systems is suggested in the paper. Improvement of the method is achieved by constructing a deterministic partitioning function. Such a function allows to organize two independent load balancing computational threads for building a block of group elements of fixed length. Also we analyze advanced iteration functions for Pollard's algorithm and build generic deterministic partitioning function.

Keywords: discrete logarithm, Pollard's rho method, elliptic curve.

References

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. Kachko E.G., Pogrebnyak K.A. Parallelniy metod Pollarda resheniya zadachi diskretnogo logarifmirovaniya v gruppe tochek elipticheskoy krivoy [Parallelized Pollard's Method for Solving the Elliptic Curve Discrete Logatrithm Problem] // Parallelniye vichislitelniye tekhnologii (PAVT–2012): trudy mezhdunarodnoy nauchnoy konferentsii [Proceedings of the International Scientific Conference «Parallel Computational Technologies (PCT–2012)»], March 26–30, 2012, Novosibirsk, *Proceedings.* – Chelyabinsk: YUrGU Publ., 2012. – P. 723.
3. Gorbenko I.D., Kachko E.G., Pogrebnyak K.A. Metody rasparallelivaniya algoritma Pollarda resheniya zadachi diskretnogo logarifmirivaniya dlya sistem s obshey pamyatyu [Methods of Parallelization of Pollard's Algorithm for Solving the Discrete Logarithm Problem for Shared Memory Systems] // Vysokoproductivniye vichisleniya (HPC–UA'2012): trudy mezhdunarodnoy nauchnoy konferentsii [Proceedings of the International Scientific Conference «High performance computing (HPC–UA'2012)»], October 8–10, 2012, Kiev, *Proceedings.* – Kiev: NANU, 2012. P. 152–157.
4. Gorbenko I.D., Kachko E.G., Pogrebnyak K.A. Parallelniy metod Pollarda resheniya zadachi diskretnogo logarifmirovaniya v multiplikativnoy gruppe polya Galua [Parallelized Pollard's Method for Solving the Discrete Logatrithm Problem in the Multiplicative Group of a Galois Field] // Sovremenniye problemy informatsionnoy bezopasnosti na transporte (SPIBT–2012): materialy vseukrainskoy nauchno-tekhnicheskoy konferentsii s mezhdunarodnym uchastiem [Proceedings of the Scientific Conference «Modern problems of information security on transport»], November 29–30, 2012, Nikolaev, *Proceedings.* – Nikolaev: NUK, 2012. – P. 9–11.

Поступила в редакцию 19 апреля 2013 г.

УДК 004.65

ПАРАЛЛЕЛЬНАЯ СУБД С ОТКРЫТЫМ ИСХОДНЫМ КОДОМ ДЛЯ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Е.В. Гавриш, А.В. Колтаков, А.А. Медведев, Л.Б. Соколинский

Статья посвящена вопросам разработки параллельной СУБД с открытым исходным кодом для кластерных вычислительных систем. Дан обзор известных решений в этой области. Рассмотрена новая параллельная СУБД «Омега» с открытым исходным кодом, ориентированная на кластерные вычислительные системы. Приведена общая архитектура системы «Омега». Представлены диаграмма размещения и диаграмма классов. Описаны основные подсистемы СУБД «Омега» и принципы их взаимодействия при выполнении запросов.

Ключевые слова: параллельная СУБД, обработка сверхбольших баз данных, программное обеспечение с открытым исходным кодом, кластерные вычислительные системы.

Введение

Проблема больших данных становится все более актуальной в наше время. К 2020 г. объем цифровой информации, хранимой в базах данных, достигнет отметки в 40 зеттабайт [13]. Наиболее эффективным решением проблемы хранения и обработки больших баз данных является использование параллельных систем управления базами данных (СУБД), обеспечивающих параллельную обработку запросов на многопроцессорных вычислительных системах. Среди многопроцессорных вычислительных систем сегодня выделяются кластеры, занимающие в списке Top500 [16] более 80 %. Кластеры обладают хорошим соотношением цена/производительность. Это объясняется тем, что кластеры состоят из компонент, которые массово продаются на рынке.

Известно несколько коммерческих решений, которые позволяют обрабатывать большие объемы данных, но они до последнего времени были ориентированы на ту или иную специфическую аппаратную платформу (DB2 Parallel Edition, NonStop SQL, NCR Teradata, Oracle RAC, Greenplum и др.) и не подходили для массового использования на кластерных системах. Кроме того, указанные коммерческие решения являются дорогостоящими. Так, аппаратно-программные решения компании Teradata предлагаются по ценам от $41 000 за 1 терабайт обрабатываемой базы данных [15]. Сегодня появляются на рынке коммерческие СУБД, ориентированные на кластерные системы. Однако эти продукты также отличаются высокой стоимостью. Например, компания Oracle предлагает СУБД Oracle RAC, предназначенную для обработки запросов на кластерных системах, по цене $10 000 за процессор при бессрочной лицензии [9].

С другой стороны, набирают популярность расширения для свободно распространяемых СУБД с открытым исходным кодом, которые обеспечивают параллельную обработку транзакций [7, 11, 20].

Данная работа посвящена вопросам разработки свободно распространяемой параллельной СУБД для кластерных систем, которая должна обладать высокой масштабируемостью, однако может проигрывать коммерческим аналогам в производительности. В статье описывается подобная параллельная СУБД, основанная на реляционной модели данных. Раздел 1 посвящен обзору известных СУБД с открытым исходным кодом,

ориентированных на обработку сверхбольших баз данных. В разделе 2 представлена архитектура разрабатываемой параллельной СУБД и описана общая схема обработки запросов. В разделе 3 представлены диаграммы классов основных подсистем СУБД «Омега». Раздел 4 содержит описание протокола взаимодействия пользовательского приложения и СУБД «Омега». В заключении представлены основные полученные результаты и направления дальнейших исследований.

Обзор известных решений

Рассмотрим известные параллельные СУБД с открытым исходным кодом, ориентированные на хранение и обработку сверхбольших баз данных.

СУБД SciDB [3] ориентирована на обработку научных данных, полученных в результате экспериментов и наблюдений. Данная СУБД оптимизирована для обработки и анализа «сырых» данных, которые интенсивно читаются, но почти не изменяются. SciDB не рассчитана на обработку транзакций в реальном времени (OLTP), не поддерживает ACID-транзакции и не является реляционной, так как хранение данных организовано в виде многомерных вложенных массивов, для обработки которых вместо языка SQL задействованы языки AQL (Array Query Language) и AFL (Array Functional Language).

Система HadoopDB [1] основана на концепции связывания нескольких одноузловых СУБД (например, MySQL или PostgreSQL) с помощью технологий Hadoop в единую параллельную СУБД. Среда Hadoop имеет два уровня системной иерархии: уровень хранения данных и уровень обработки данных. Уровень хранения данных представлен распределенной файловой системой HDFS (Hadoop Distributed File System), а уровень обработки данных реализуется с помощью программного каркаса Hadoop MapReduce. Основным недостатком HadoopDB является низкая производительность по сравнению с реляционными параллельными СУБД [12, 18]. Это объясняется тем, что изначально система HadoopDB разрабатывалась как прототип параллельной системы управления аналитическими (научными) данными.

СУБД MongoDB [2] является документо-ориентированной СУБД класса NoSQL. В MongoDB отсутствует полноценная поддержка ACID-транзакций. Существенным недостатком данной СУБД является полное отсутствие поддержки изолированности операций над данными.

Свободная параллельная СУБД MySQL Cluster [8, 14] обладает хорошей производительностью и масштабируемостью на простых запросах, но эффективность СУБД существенно падает при обработке сложных запросов с условиями [4]. Недостатком MySQL Cluster является использование полной репликации данных.

В рамках научного проекта ParGRES [10] разрабатывается параллельная СУБД, предназначенная для обработки OLAP-запросов. СУБД ParGRES представляет собой промежуточное программное обеспечение, которое управляет экземплярами свободной последовательной СУБД PostgreSQL, запускаемыми на узлах кластерной системы. Недостатком СУБД ParGRES является использование полной репликации всех таблиц базы данных на узлах кластерной системы, что приводит к общему снижению производительности СУБД. Развитием данной разработки является СУБД GParGRES [6], предназначенная для грид-сред. СУБД GParGRES использует репликацию базы данных, межзапросный и внутризапросный параллелизм для эффективной обработки OLAP-

запросов в грид. Предложенный в СУБД GParGRES подход подразумевает распараллеливание запроса на двух уровнях: на уровне грид (реализовано в GParGRES) и на уровне узлов (реализовано в ParGRES). Однако СУБД GParGRES также требует полной репликации базы данных на всех узлах всех кластеров, объединенных в грид.

Параллельная СУБД VoltDB [17] с открытым исходным кодом предназначена для кластерных систем и обладает сравнительно высокой производительностью. СУБД является представителем класса NoSQL. Существенным недостатком данной СУБД с точки зрения обработки сверхбольших баз данных является необходимость хранить всю базу данных в оперативной памяти.

Проведенный анализ показывает, что на данный момент отсутствуют реляционные параллельные СУБД с открытым исходным кодом, ориентированные на кластерную архитектуру и обладающие высокой эффективностью при обработке запросов.

Общая архитектура СУБД «Омега»

Целью проекта «Омега», выполняемого в Лаборатории суперкомпьютерного моделирования Южно-Уральского государственного университета, является разработка новых методов и алгоритмов параллельной обработки запросов к базам данных, ориентированных на современные кластерные вычислительные системы, а также реализация этих методов и алгоритмов в виде параллельной СУБД «Омега» с открытым исходным кодом. Фундаментом разрабатываемой параллельной СУБД с одноименным названием «Омега» является архитектура, представленная на рис. 1.

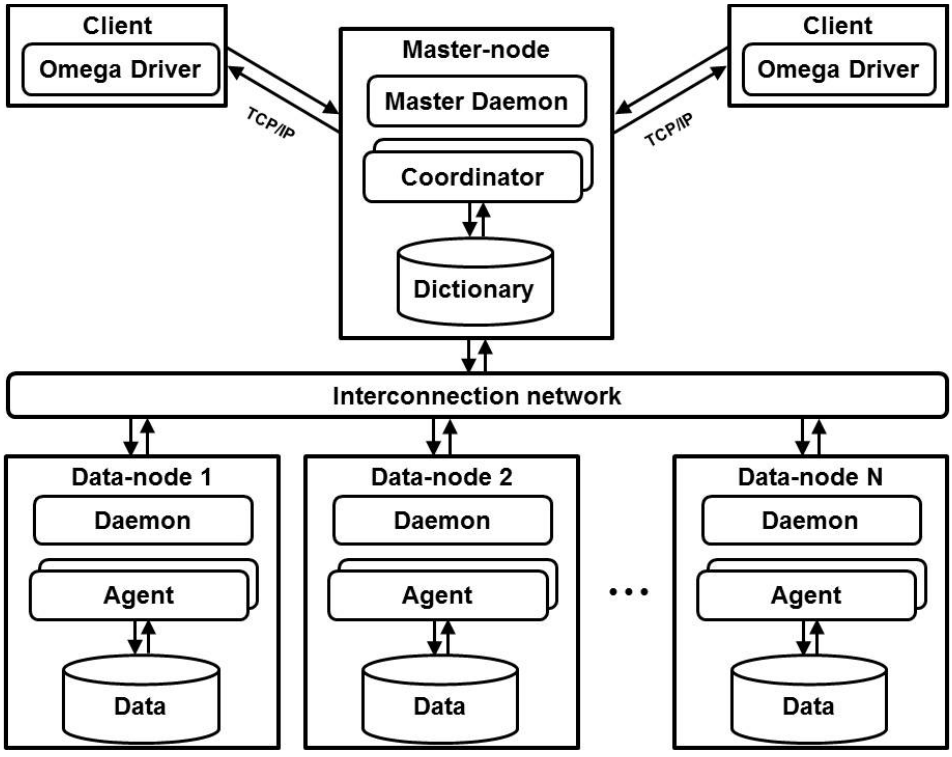


Рис. 1. Общая архитектура СУБД «Омега»

В соответствии с данной архитектурой в СУБД выделяются следующие основные подсистемы.

Управляющий узел (Master-node) – это совокупность процессов и подсистем, которые отвечают за взаимодействие СУБД с клиентами, скрывая от них все детали распределенной обработки запросов. На данном узле размещается *словарь базы данных* (*Dictionary*). На управляющем узле выполняется *серверный фоновый процесс* (*MasterDaemon*), который отвечает за прием соединений, устанавливаемых клиентами. Для каждого клиента серверный фоновый процесс порождает специальный процесс – *координатор* (*Coordinator*) для обработки запросов. Координатор выполняет построение плана запроса и его рассылку на вычислительные узлы, сбор промежуточных результатов выполнения запроса и отправку конечного результата клиенту. Данный процесс также управляет балансировкой загрузки СУБД во время обработки запроса.

Поскольку для каждого клиента создается отдельный координатор, для однозначной идентификации координатора внутри системы вводится понятие сессии. *Сессия* – это совокупность процессов СУБД, выполняемых одним клиентом. Каждому отдельному клиенту присваивается уникальный номер сессии.

Вычислительный узел (Data-node) – это совокупность процессов и подсистем, которые непосредственно отвечают за исполнение запроса и хранение *данных* (*Data*). Количество таких узлов теоретически ограничено производственными мощностями вычислительной системы. На данном узле выполняется *фоновый процесс* (*Daemon*), который взаимодействует с серверным фоновым процессом (MasterDaemon) и при получении соответствующей команды создает отдельный процесс – параллельный *агент* (*Agent*) [19] для обработки запроса. Агент получает от координатора план запроса, исполняет его и отсылает координатору промежуточный результат. Агенты однозначно сопоставляются с координатором по номеру сессии.

Клиент (Client) – это пользовательское (клиентское) приложение на языке Java, которое для работы с СУБД использует библиотеку *Omega Driver*. *Omega Driver* представляет собой JDBC-драйвер [5], который обеспечивает доступ к СУБД по протоколу TCP/IP через стандартные API-интерфейсы JDBC, доступные на платформе Java.

Взаимодействие между узлами происходит с использованием высокоскоростной вычислительной сети – *interconnection network*.

Рассмотрим порядок взаимодействия основных модулей СУБД при выполнении запроса (рис. 2). Порядок взаимодействия включает в себя следующие шаги.

1. Connect (соединение). Клиент, используя Omega Driver, устанавливает соединение с серверным процессом.
2. Accept (прием). Серверный процесс принимает запрос клиента на соединение и связывает себя с конкретным адресом, для того, чтобы иметь возможность в дальнейшем обрабатывать запросы клиента.
3. CreateCoordinator (создание координатора). Серверный процесс создает отдельный процесс-координатор для данного клиента. На данном шаге координатору присваивается номер сессии.
4. SendSession (отправка сессии). Серверный процесс отправляет фоновым процессам на вычислительных узлах номер сессии данного клиента.
5. CreateAgent (создание агента). Каждый фоновый процесс создает параллельного агента и присваивает ему номер сессии.
6. Execute (выполнение запроса). Клиент отсылает координатору запрос на языке SQL.

7. SendMetadata (отправка метаданных). Координатор обрабатывает запрос, строит план выполнения запроса, вычисляет метаданные результирующего отношения и отправляет их клиенту.
8. SendPlan (отправка плана). Координатор производит пересылку плана выполнения запроса его агентам.
9. Exchange (обмен). Агенты исполняют план запроса. Во время исполнения плана агенты могут осуществлять межпроцессорные обмены (пересылку кортежей по сети).
10. SendTuple (отправка кортежа). Агенты отсылают результирующие кортежи координатору.
11. SendResult (отправка результата). Координатор производит слияние и обработку кортежей, полученных от агентов, и отсылает конечный результат клиенту.

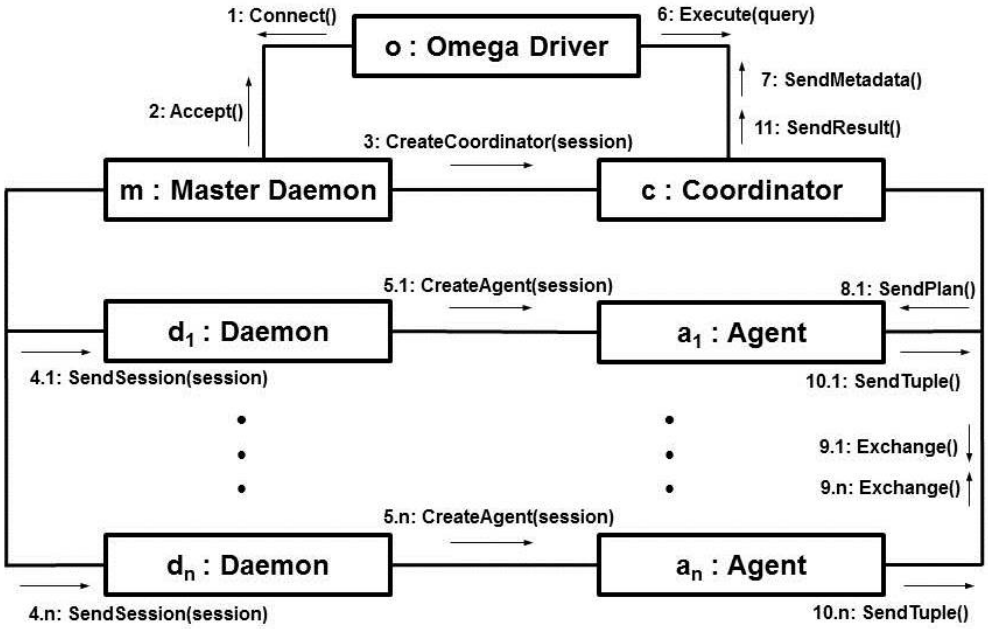


Рис. 2. Схема обработки запроса

Структура СУБД «Омега»

Ключевым элементом СУБД «Омега» является управляющий узел. Управляющий узел включает в себя 4 подсистемы: серверный фоновый процесс, координатор, менеджер сессий и словарь баз данных. Соответствующая диаграмма классов представлена на рис. 3.

Серверный фоновый процесс включает в себя следующие основные методы:

- run – запускает основной цикл работы серверного процесса;
- accept – принимает запрос на соединение от клиента;
- createCoordinator – создает процесс-координатор и предает ему номер сессии;
- sendSession – передает фоновым процессам на вычислительных узлах номер сессии;
- finish – вызывается при поступлении команды завершения СУБД, корректно завершает все запущенные процессы.

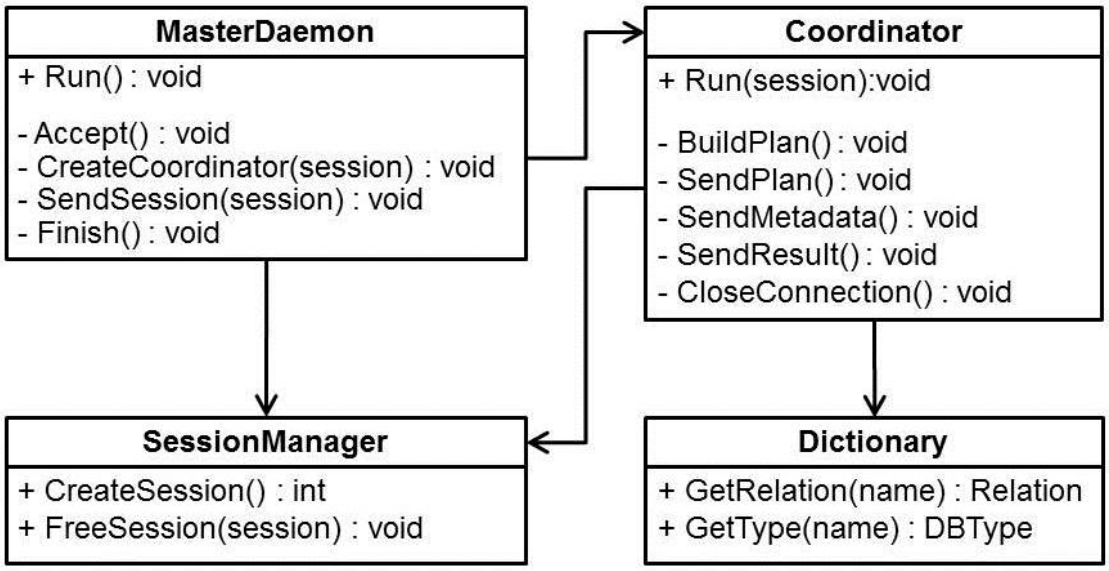


Рис. 3. Диаграмма классов управляющего узла

Координатор включает в себя следующие основные методы:

- run – запускает основной цикл работы серверного процесса;
- buildPlan – производит обработку запроса на языке SQL и выполняет построение плана запроса;
- sendPlan – отправляет план запроса соответствующим параллельным агентам;
- sendMetadata – отправляет клиенту метаданные результирующего отношения;
- sendResult – отправляет клиенту результат выполнения запроса;
- closeConnection – разрывает соединение с клиентом, освобождает сессию.

Менеджер сессий включает в себя следующие основные методы:

- createSession – осуществляет создание новой сессии, возвращает ее номер;
- freeSession – осуществляет освобождение сессии;

Словарь баз данных включает в себя следующие основные методы:

- getRelation – возвращает информацию об отношении;
- getType – возвращает информацию о типе.

Взаимодействие серверного процесса СУБД и клиентского приложения осуществляется с помощью стандартных API-интерфейсов JDBC, доступных на платформе Java. В рамках проекта реализовано подмножество функций стандарта JDBC. Диаграмма классов клиента представлена на рис. 4.

В классах DriverManager и Driver предусмотрены методы getConnection и connect соответственно для установки соединения между клиентом и управляющим узлом СУБД. Параметром данных методов является символьная строка, которая имеет формат «jdbc:<тип драйвера>://<ip-адрес сервера>:<порт>».

Основные методы класса Connection:

- createStatement – возвращает объекты типа Statement, служащие для исполнения запросов к базе данных на языке SQL;
- close – осуществляет закрытие соединения.

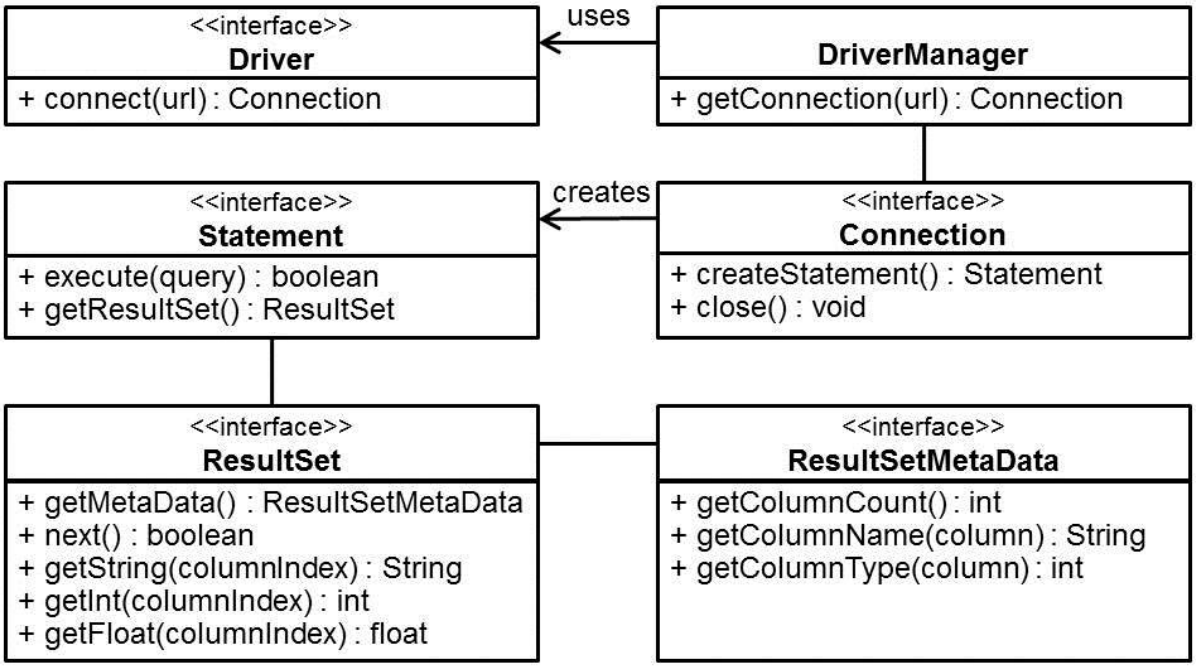


Рис. 4. Диаграмма классов клиента

Основные методы класса Statement:

- execute – возвращает истину, если запрос является запросом типа SELECT (существует ResultSet), иначе возвращает ложь, то есть имеет место быть запрос типа INSERT или UPDATE (отсутствует ResultSet);
- getResultSet – возвращает множество строк, удовлетворяющих выражению на языке SQL.

Основные методы класса ResultSet:

- getMetadata – возвращает метаданные;
- next – возвращает истину, если существует следующий кортеж, иначе возвращает ложь;
- getString, getInt и getFloat – возвращают значение конкретного поля текущего кортежа.

Основные методы класса ResultSetMetaData:

- getColumnCount – возвращает количество столбцов (колонок);
- getColumnName – возвращает название поля столбца (колонки);
- getColumnType – возвращает идентификатор типа данных столбца (колонки), где для типа int соответствует значение 4, float – 6, а string – 15.

Протокол взаимодействия клиента и СУБД

Взаимодействие клиента (пользовательского приложения) и СУБД (управляющего узла) осуществляется с помощью протокола на основе специальных команд. Протокол определяет формат команд и их последовательность при обработке запросов от клиента.

Формат команды следующий: <идентификатор><размер данных><данные>.

Идентификатор команды определяет семантику команды, наличие передаваемых данных и их назначение. Расшифровка идентификаторов, используемых в протоколе, представлена в таблице 1.

Таблица 1.

Идентификаторы протокола

Идентификаторы клиента			
№	id	Команда	Семантика
1	S	start	Установить соединение.
2	Q	query	Отправить запрос.
3	C	close	Закрыть соединение.
Идентификаторы управляющего узла			
№	id	Команда	Семантика
1	R	ready	Готовность к обработке запросов.
2	I	info	Отправить информацию о выполнении запроса.
3	M	metadata	Отправить метаданные.
4	T	tuple	Отправить кортеж.
5	E	error	Уведомить об ошибке.
6	F	finish	Завершить обработку запроса.

Заключение

В работе рассмотрены основные современные свободные решения в сфере обработки сверхбольших баз данных, представлена архитектура разрабатываемой авторами СУБД, приведены диаграммы классов основных подсистем и описан процесс выполнения запросов в рамках данной архитектуры.

На данный момент авторами работы разработан прототип, который успешно развертывается на узлах вычислительного кластера и осуществляет взаимодействие между узлами СУБД (основные системные команды).

В рамках продолжения исследований планируется реализация основных компонентов параллельной СУБД с использованием методов и алгоритмов параллельной обработки запросов в СУБД.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 12-07-00443-а.

Литература

1. Abouzeid, A. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads / A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, A. Rasin // VLDB'2009, Proceedings of 35th International Conference on Very Large Data Bases, August 24-28, 2009, Lyon, France. – VLDB Endowment, 2009. – P. 922–933.
2. Boicea, A. MongoDB vs Oracle – database comparison / A. Boicea, F. Radulescu, L.I. Agapin // Proceedings of the Third International Conference on Emerging Intelligent Data and Web Technologies, September 19-21, 2012, Bucharst, Romania. – P. 330–335.
3. Brown, P.G. Overview of sciDB: Large Scale Array Storage, processing and analysis / P.G. Brown // Proceedings of the ACM SIGMOD International Conference on Man-

agement of Data, June 6-10, 2010, Indianapolis, Indiana, USA. – ACM, 2010. – P. 963–968.

4. Hubel, M. Technical Comparison of DB2 and MySQL / M. Hubel – Martin Hubel Consulting Inc., 2004. – 32 p.
5. Java SE Documentation. URL: http://www.oracle.com/technetwork/java/javase/jdbc/ (дата обращения: 10.03.2013).
6. Kotowski, N. Parallel query processing for OLAP in grids / N. Kotowski, A.A.B. Lima, E. Pacitti, P. Valduriez, M. Mattoso // Concurrency and Computation: Practice and Experience, – 2008. – Vol. 20, No. 17. – P. 2039–2048.
7. Lee, R. Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing / R. Lee, M. Zhou // Proceedings of the 12th International Conference on Database Systems for Advanced Applications, April 9-12, 2007, Bangkok, Thailand. – Proceedings. Lecture Notes in Computer Science, Springer, 2007. – Vol. 4443. – P. 1086–1097.
8. MySQL Cluster Information.
   URL: http://www.mysql.com/products/cluster/resources.html (дата обращения: 10.03.2013).
9. Oracle Store. URL: http://shop.oracle.com (дата обращения: 10.03.2013).
10. Paes, M. High-Performance Query Processing of a Real-World OLAP Database with ParGRES / M. Paes, A.A.B. Lima, P. Valduriez, M. Mattoso // High Performance Computing for Computational Science – VECPAR 2008: 8th International Conference, June 24-27, 2008, Toulouse, France. – Revised Selected Papers. Springer, 2008. – P. 188–200.
11. Paulson, L.D. Open Source Databases Move into the Marketplace / L.D. Paulson // Computer, – 2004. – Vol. 37, No. 7. – P. 13–15.
12. Pavlo, A. Comparison of Approaches to Large Scale Data Analysis / A. Pavlo, A. Rasin, S. Madden, M. Stonebraker, D. DeWitt, E. Paulson, L. Shrinivas, D.J. Abadi // Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, June 29 – July 2, 2009, Providence, Rhode Island, USA. – ACM, 2009. – P. 165–178.
13. Press Release EMC2. URL: http://www.emc.com/about/news/press/2012/20121211-01.htm (дата обращения: 10.03.2013).
14. Ronstrom, M. Recovery Principles in MySQL Cluster 5.1 / M. Ronstrom, J. Oreland // Proceedings of the 31st International Conference on Very Large Data Bases, August 30 – September 2, 2005, Trondheim, Norway. – ACM, 2005. – P. 1108–1115.
15. Teradata Purpose-Built Platform Pricing.
   URL: http://www.teradata.com/t/WorkArea/DownloadAsset.aspx?id=4682 (дата обращения: 10.03.2013).
16. Top500 List. URL: http://www.top500.org (дата обращения: 10.03.2013).
17. VoltDB Documentation. URL: http://voltdb.com/community/documentation (дата обращения: 10.03.2013).
18. Кузнецов, С.Д. MapReduce: внутри, снаружи или сбоку от параллельных СУБД? / С.Д. Кузнецов // Труды Института системного программирования РАН. – 2010. – Т. 19. – С. 35–70.
19. Лепихов, А.В. Обработка запросов в СУБД для кластерных систем / А.В. Лепихов, Л.Б. Соколинский // Программирование. – 2010. – № 4. – С. 25–39.

20. Пан, К.С. Разработка параллельной СУБД на основе последовательной СУБД PostgreSQL с открытым исходным кодом / К.С. Пан, М.Л. Цымблер // Вестник ЮУрГУ. Серия «Математическое моделирование и программирование». – 2012. – № 18(277). – Вып. 12. – С. 112–120.

Гавриш Евгений Владимирович, аспирант кафедры системного программирования Южно-Уральского государственного университета (Челябинск, Российская Федерация), evgeniy.gavrish@gmail.com

Колтаков Алексей Владимирович, аспирант кафедры системного программирования Южно-Уральского государственного университета (Челябинск, Российская Федерация), aleksey.koltakov@gmail.com

Медведев Александр Андреевич, аспирант кафедры системного программирования Южно-Уральского государственного университета (Челябинск, Российская Федерация), medbedalex@gmail.com

Соколинский Леонид Борисович, доктор физ.-мат. наук, профессор кафедры системного программирования Южно-Уральского государственного университета (Челябинск, Российская Федерация), sokolinsky@acm.org

---

PARALLEL OPEN SOURCE DBMS FOR CLUSTER COMPUTING SYSTEMS

E.V. Gavrish, A.V. Koltakov, A.A. Medvedev, L.B. Sokolinsky

The article deals with problem of development open source parallel DBMS for cluster computing systems. The article describes overview of well-known solves in this area. It is considered a new parallel open source DMBS named Omega, which is oriented on cluster computing systems. The article shows the general architecture of system Omega, its deployment and class diagrams. The article describes Omega's main subsystems and principles of their interaction during query processing.

Keywords: parallel DBMS, big data processing, open source software, cluster computing systems.

References

1. Abouzeid A., Bajda-Pawlikowski K., Abadi D., Silberschatz A., Rasin A. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. VLDB'2009, Proceedings of 35th International Conference on Very Large Data Bases, August 24-28, 2009, Lyon, France. – VLDB Endowment, 2009. P. 922–933.
2. Boicea A., Radulescu F., Agapin L.I. MongoDB vs Oracle – database comparison. Proceedings of the Third International Conference on Emerging Intelligent Data and Web Technologies, September 19-21, 2012, Bucharst, Romania. P. 330–335.
3. Brown P.G. Overview of sciDB: Large Scale Array Storage, processing and analysis. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 6-10, 2010, Indianapolis, Indiana, USA. ACM, 2010. P. 963–968.
4. Hubel, M. Technical Comparison of DB2 and MySQL. Martin Hubel Consulting Inc., 2004. 32 p.

5. Java SE Documentation. URL: http://www.oracle.com/technetwork/java/javase/jdbc/ (accessed: 10.03.2013).
6. Kotowski N., Lima A.A.B., Pacitti E., Valduriez P., Mattoso M. Parallel query processing for OLAP in grids. Concurrency and Computation: Practice and Experience, 2008. Vol. 20, No. 17. P. 2039–2048.
7. Lee R., Zhou M. Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing. Proceedings of the 12th International Conference on Database Systems for Advanced Applications, April 9-12, 2007, Bangkok, Thailand. – Proceedings. Lecture Notes in Computer Science, Springer, 2007. Vol. 4443. P. 1086–1097.
8. MySQL Cluster Information.
URL: http://www.mysql.com/products/cluster/resources.html (accessed: 10.03.2013).
9. Oracle Store. URL: http://shop.oracle.com (accessed: 10.03.2013).
10. Paes M., Lima A.A.B., Valduriez P., Mattoso M. High-Performance Query Processing of a Real-World OLAP Database with ParGRES. High Performance Computing for Computational Science – VECPAR 2008: 8th International Conference, June 24–27, 2008, Toulouse, France. – Revised Selected Papers. Springer, 2008. P. 188–200.
11. Paulson L.D. Open Source Databases Move into the Marketplace. Computer, – 2004. – Vol. 37, No. 7. – P. 13–15.
12. Pavlo A., Rasin A., Madden S., Stonebraker M., DeWitt D., Paulson E., Shrinivas L., Abadi D.J. Comparison of Approaches to Large Scale Data Analysis. Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, June 29 – July 2, 2009, Providence, Rhode Island, USA. ACM, 2009. P. 165–178.
13. Press Release EMC2. URL: http://www.emc.com/about/news/press/2012/20121211-01.htm (accessed: 10.03.2013).
14. Ronstrom M., Oreland J. Recovery Principles in MySQL Cluster 5.1. Proceedings of the 31st International Conference on Very Large Data Bases, August 30 – September 2, 2005, Trondheim, Norway. ACM, 2005. P. 1108–1115.
15. Teradata Purpose-Built Platform Pricing.
URL: http://www.teradata.com/t/WorkArea/DownloadAsset.aspx?id=4682 (accessed: 10.03.2013).
16. Top500 List. URL: http://www.top500.org (accessed: 10.03.2013).
17. VoltDB Documentation. URL: http://voltdb.com/community/documentation (accessed: 10.03.2013).
18. Kuznetcov S.D. MapReduce: vnutri, snarugi ili sboku ot parallelnih SUBD? [MapReduce: inside, outside or beside parallel DMBS]. Trudi Instituta sistemnogo programmirovania RAN [Proceedings of the Institute of System Programming]. 2010. Vol. 19. P. 35–70.
19. Lepikhov A.V., Sokolinsky L.B. Query Processing in a DBMS for Cluster Systems // Programming and Computer Software. – 2010. – Vol. 30, No. 4. – P. 205–215.
20. Pan C.S., Zymbler M.L. Razrabotka parallelnoj SUBD na osnove posledovatelnoj SUBD PostgreSQL s otkrytym ishodnym kodom [Development of a Parallel Database Management System on the Basis of Open-Source PostgreSQL DBMS]. Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2012. No. 18(277). Vol. 12. P. 112–120.

Поступила в редакцию 14 июня 2013 г.

УДК 519.612

ФУНКЦИОНАЛЬНОСТЬ И ТЕХНОЛОГИИ АЛГЕБРАИЧЕСКИХ РЕШАТЕЛЕЙ В БИБЛИОТЕКЕ KRYLOV[1]

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

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. Программные реализации выполнены на типовых сжатых разреженных форматах матричных данных. Приводятся результаты численных экспериментов с демонстрацией эффективности распараллеливания для характерных плохо обусловленных задач. *Ключевые слова: предобусловленные итерационные алгоритмы, подпространства Крылова, методы декомпозиции областей, разреженные алгебраические системы, численные эксперименты.*

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 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 Intel MKL [8], или авторскими

[1]Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии 2013»

версиями алгоритмов BiCGStab и GMRES [7, 9], предобусловленных с помощью покомпонентной или эффективной «мелкоблочной» модификациями Айзенштата неполной факторизации. В последнем случае имеются в виду СЛАУ, полученные, например, при сеточных аппроксимациях систем дифференциальных уравнений, когда одному узлу сетки соответствует несколько неизвестных функций (в задачах тепломассопереноса это могут быть три компоненты вектора скорости, давление и температура). Отличительной чертой соответствующих матриц является то, что они представимы в блочном виде с диагональными матрицами малого порядка, который не зависит от числа узлов сетки и, следовательно, от порядка СЛАУ.

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 [10], 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 [11], Hypre [12], HIPS и pARMS [13] и др., однако в Krylov реализован также ряд оригинальных методов, обсуждаемых в работе.

В разделе 1 мы описываем функциональное наполнение библиотеки Krylov, в разделе 2 излагаются особенности программных реализаций алгебраических решателей, последний же раздел посвящен численным экспериментам для характерных практических задач. В заключении подводятся итоги работы и делаются выводы о применимости изложенных методов.

1. 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 меньших порядков:

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

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 $\sum\limits_{i=1}^{p} N_i$.

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 хотя бы 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

$$\bar{\Omega}_p \equiv \Omega_p \bigcup \Gamma_p^1 ... \bigcup \Gamma_p^{\Delta} \tag{4}$$

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

$$(\bar{A}_{p,p} + \theta\bar{D}_p)\bar{u}_p = \bar{f}_p - \sum_{\substack{q=1 \\ q\neq p}}^{P} \bar{A}_{p,q}\bar{u}_q + \theta\bar{D}_p\bar{u}_p = \bar{r}_p, \tag{5}$$

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

$$\bar{D}_p e = \sum_{\substack{q=1 \\ q\neq p}}^{P} \bar{A}_{p,q} e, \;\; e = (1, ..., 1)^T \in \mathcal{R}^{\bar{N}_p}. \tag{6}$$

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

$$B_p = \bar{A}_{p,p} + \theta\bar{D}_p, \tag{7}$$

из (5) получаем итерационный аддитивный процесс Шварца в виде

$$u^n = u^{n-1} + B^{-1}(f - Au^{n-1}) = u^{n-1} + B^{-1}r^n, \tag{8}$$

где B — предобуславливающая матрица, определяемая следующим образом:

$$B^{-1} = B_{AS}^{-1} = \sum_{p=1}^{P} \bar{B}_p^{-1}, \;\; \bar{B}_p^{-1} = W_p^T B_p^{-1} W_p. \tag{9}$$

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 (8) 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 (8) используется универсальный «гибкий» метод обобщенных минимальных невязок FGMRES [7], оптимизирующий невязку в подпространствах Крылова.

2. 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.

2.1. 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 представлен на рисунке.

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

$$\begin{bmatrix} H_{k,n} \\ H_{k+1,n} \end{bmatrix} \leftarrow \begin{bmatrix} c_k & s_k \\ -\bar{s}_k & c_k \end{bmatrix} \begin{bmatrix} H_{k,n} \\ H_{k+1,n} \end{bmatrix}$$

  EndFor
  $c_n \leftarrow |H_{n,n}|/\sqrt{|H_{n,n}|^2 + |H_{n+1,n}|^2}$
  $\bar{s}_n \leftarrow c_n H_{n+1,n}/H_{n,n}$

$$\begin{bmatrix} \xi_n \\ \xi_{n+1} \end{bmatrix} \leftarrow \begin{bmatrix} c_n & s_n \\ -\bar{s}_n & c_n \end{bmatrix} \begin{bmatrix} \xi_n \\ \xi_{n+1} \end{bmatrix}$$

  $H_{n,n} \leftarrow c_n H_{n,n} + s_n H_{n+1,n}$, $H_{n+1,n} \leftarrow 0$
  $||r_{n+1}|| \leftarrow \beta|\xi_{n+1}|$
EndFor
$y_n \leftarrow \beta H^{-1} \xi$
$x_n \leftarrow x_0 + [z_0 \dots z_{n-1}] y_n$.

Метод FGMRES

Анализ вычислительной схемы алгоритма показывает, что основными в методе FGMRES являются следующие операции:

- 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 (метод Шварца);
- 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 матрицы Хессенберга во время работы решателей также можно игнорировать в предположении, что порядок исходной системы много больше количества итераций. Такое разложение также не требует

большого количества памяти, поэтому с целью уменьшения коммуникационных затрат оно выполняется в каждом MPI-процессе.

Векторно-векторные операции в итерационных решателях распараллеливаются естественным образом за счет разбиения векторов, порождаемого декомпозицией матрицы на расширенные 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 элементы вектора вместо того, чтобы пересылать вектор целиком. В случае, если разбиение на подобласти было построено подходящим способом, количество граничных вершин N_Γ будет существенно меньше размера подобласти (например, для двумерных задач N_Γ 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 (8), где предобуславливатель B составлен из блоков (7). Отсюда видно, что применение предобуславливателя B^{-1} в подобласти с номером p не требует знания переменных, не принадлежащих ей.

2.2. Реализация внутренних итераций в подобластях

Для решения задач $A_{p,p}v_p = b_p$ в подобластях 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 $A = D - L - U$, а B — предобуславливающая матрица, записанная в форме

$$B = (G - L)G^{-1}(G - U) = G - L - U + LG^{-1}U, \tag{10}$$

где $G = \text{block-diag}\{G_k\}$, $G_k \in \mathcal{R}^{m,m}$, $k = 1, ..., M = N/m$ 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 невырожденными множителями L_G и U_G.

С помощью обозначений

$$\bar{D} = L_G^{-1} D U_G^{-1},\ \bar{L} = L_G^{-1} L U_G^{-1},\ \bar{U} = L_G^{-1} U U_G^{-1} \tag{11}$$

предобусловленная матрица системы записывается в форме

$$\begin{gathered} \bar{A} = (I - \bar{L})^{-1} L_G^{-1} A U_G^{-1} (I - \bar{U})^{-1} = \\ = (I - \bar{L})^{-1} + (I - \bar{U})^{-1} + (I - \bar{L})^{-1} (\bar{D} - 2I)(I - \bar{U})^{-1}, \end{gathered} \tag{12}$$

где I есть единичная матрица.

Отсюда предобусловленную СЛАУ можно представить как

$$\bar{A}\bar{u} = \bar{f} \equiv L_G^{-1} (I - \bar{L})^{-1} f,\ \ \bar{u} = (I - \bar{U}) U_G u. \tag{13}$$

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

$$\bar{A}v = (I - \bar{L})^{-1} [v + (\bar{D} - 2I)w] + w,\ \ w = (I - \bar{U})^{-1} v, \tag{14}$$

составляющей суть модификации Айзенштата.

Конкретизируем теперь выбор матрицы G. Мы используем

$$G = \frac{1}{\omega} D, \tag{15}$$

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:

$$G = \text{block-diag}\{G_k = L_{G,k} U_{G,k}\}, \tag{16}$$

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

$$\begin{gathered} \bar{L} = \{\bar{L}_{k,l} = L_{G,k}^{-1} L_{k,l} U_{G,l}^{-1}\},\ \ \bar{U} = \{\bar{U}_{k,l} = L_{G,k}^{-1} U_{k,l} U_{G,l}^{-1}\}, \\ k, l = 1, ..., M. \end{gathered} \tag{17}$$

Точнее, при использовании блочного представления векторов $v = \{v_k\}$, $w = \{w_k\}$, $k = 1, ..., M$, решения участвующих в (14) треугольных систем

$$(I - \bar{U})w = v,\ \ (I - \bar{L})y = z \equiv v + (\bar{D} - 2I)w \tag{18}$$

осуществляются фактически по следующим рекуррентным формулам:

$$\begin{gathered} w_M = v_M,\ \ w_k = v_k + \sum_{l=k+1}^{M} \bar{U}_{k,l} v_l,\ \ k = M - 1, ..., 1, \\ y_1 = z_1,\ \ y_k = z_k + \sum_{l=1}^{k-1} \bar{L}_{k,l} z_l,\ \ k = 2, ..., M. \end{gathered} \tag{19}$$

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.

3. Результаты численных экспериментов

Мы приведем результаты некоторых численных экспериментов по решению ряда практических задач с помощью библиотеки Krylov, выполненных на кластере ИВМиМГ СО РАН [14]. В данных расчетах внешние итерации проводились с помощью метода FGMRES с критерием окончания итераций по условию на евклидовую норму невязки

$$||r^n||_2 \le \varepsilon||f||_2, \;\; \varepsilon = 10^{-7}. \tag{20}$$

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 узлов, с формированием такого же числа подобластей и соответствующих им 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.

Таблица 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 для данных СЛАУ с использованием PARDISO в качестве решателя в подобластях. В каждой из клеток 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 областей (количество расширений сеточной подобласти); N_{th} — количество формируемых вычислительных потоков в одном MPI-процессе; T_1 — время (в секундах) решения СЛАУ без распараллеливания, т.е. при $N_{mpi} = N_{th} = 1$.

Таблица 2

Решение 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

Таблица 3

Решение СЛАУ Г3 с помощью PARDISO, $N_{th} = 4$ ($T_1 = 10,1$)

$P \setminus \Delta$	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

Таблица 4

Решение 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

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 [15], 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 $\Delta = 0$.

Таблица 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 клетках таблицы указано общее время счета t (слева), 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.

Таблица 6

Сравнение прямых и итерационных решателей для СЛАУ в подобластях

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

Полученные результаты позволяют 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 внут-

ренних итераций n^i_{max} с 10 до 20 число внешних итераций несколько падает, но каждая из них делается «дороже» и общее время решения увеличивается.

Заключение

В статье рассмотрен параллельный двухуровневый итерационный решатель, входящий в состав библиотеки Krylov, и приведены ключевые моменты его реализации: организация двухуровневых итераций с использованием методов Шварца и FGMRES, решение СЛАУ в подобластях итерационными и прямыми методами. Произведены эксперименты, показывающие применимость изложенных подходов к решению характерных задач. В частности, продемонстрирована возможность использования итерационных методов с невысокой точностью для решения СЛАУ в подобластях, а также показано преимущество таких методов в случае явного учета блочной структуры матриц.

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.

Литература

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

2. Бутюгин, Д.С. Методы параллельного решения СЛАУ на системах с распределенной памятью в библиотеке Krylov / Д.С. Бутюгин, В.П. Ильин, Д.В. Перевозкин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». — 2012. — №. 47(306). — С. 5–19.

3. Ильин, В.П. Методы конечных разностей и конечных объемов для эллиптических уравнений / В.П. Ильин — Новосибирск: Изд-во ИВМиМГ СО РАН, 2001. — 345 с.

4. Ильин, В.П. Методы и технологии конечных элементов / В.П. Ильин — Новосибирск: Изд-во ИВМиМГ СО РАН, 2007. — 371 с.

5. Ильин, В.П. Параллельные методы и технологии декомпозиции областей / В.П. Ильин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». — 2012. — No. 46(305). — С. 31–44.

6. Ильин, В.П. Параллельные методы декомпозиции в пространствах следов / В.П. Ильин, Д.В. Кныш // Вычислительные методы и программирование. — 2011. — Т. 12, No. 1. — С. 100–109.

7. Saad, Y. Iterative Methods for Sparse Linear Systems, Second Edition / Y. Saad — SIAM, 2003. — 528 p.

8. Intel Math Kernel Library. Reference Manual. URL: `http://software.intel.com/sites/products/documentation/doclib/mkl_sa/11/mklman/index.htm` (дата обращения: 12.02.2013).

9. Ильин, В.П. Методы бисопряженных направлений в подпространствах Крылова / В.П. Ильин // СибЖИМ. — 2008. — Т. 11, No. 4(36). — С. 47–60.

10. Bell, N. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations / N. Bell, M. Garland. URL: `http://cusp-library.googlecode.com` (дата обращения: 15.10.2012).

11. PETSc: Home Page. URL: `http://www.mcs.anl.gov/petsc/` (дата обращения: 12.02.2013).

12. Hypre. URL: `http://acts.nersc.gov/hypre/` (дата обращения: 12.02.2013).

13. Yousef Saad – Software. URL: `http://www-users.cs.umn.edu/~saad/software/` (дата обращения: 12.02.2013).

14. Кластер НКС-30Т. URL: `http://www2.sscc.ru/HKC-30T/HKC-30T.htm` (дата обращения: 12.02.2013).

15. Nabben, R. A comparison of abstract versions of deflation, balancing and additive coarse grid correction preconditioners / R. Nabben, C. Vuik // Numerical Linear Algebra with Applications. — 2008. — Vol. 15, No. 4. — P. 355–372.

Дмитрий Сергеевич Бутюгин, младший научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН, аспирант, Новосибирский государственный университет (Новосибирск, Российская Федерация), dm.butyugin@gmail.com.

Яна Леонидовна Гурьева, к.ф.-м.н., старший научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), yana@lapasrv.sscc.ru.

Валерий Павлович Ильин, д.ф.-м.н., профессор, главный научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), ilin@sscc.ru.

Данил Валерьевич Перевозкин, младший научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), foxillys@gmail.com.

Артем Владимирович Петухов, младший научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), petukhov@lapasrv.sscc.ru.

Игорь Николавевич Скопин, к.ф.-м.н., научный сотрудник, Институт вычислительной математики и математической геофизики СО РАН (Новосибирск, Российская Федерация), iskopin@gmail.com.

PARALLEL ALGEBRAIC SOLVERS LIBRARY KRYLOV

D.S. Butyugin, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
Y.L. Guryeva, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
V.P. Il'in, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
D.V. Perevozkin, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
A.V. Petukhov, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation),
I.N. Skopin, Institute of Computational Mathematics and Mathematical Geophysics SB RAS (Novosibirsk, Russian Federation)

Article describes functional capabilities and software implementation peculiarities of parallel algorithms library Krylov, which is oriented on the solution of large systems of linear algebraic equations with sparse symmetric and unsymmetric matrices (positive definite and semi-definite) obtained from discrete approximations of multidimensional boundary value problems for partial differential equations on unstructured meshes. The library includes two-level iterative methods in Krylov subspaces; preconditioning of the latter is based on the balanced decomposition of the computational domain with variable sizes of subdomain overlapping and different boundary conditions on interfacing boundaries. Program implementations use typical compressed sparse matrix data formats. Results of numerical experiments are presented which demonstrate the efficiency of parallelization for typical ill-conditioned problems.

Keywords: preconditioned iterative algorithms, Krylov subspaces, domain decomposition methods, sparse algebraic systems, numerical experiments.

References

1. Butyugin D.S., Il'in V.P., Itskovich Y.A., et al. Krylov: biblioteka algoritmov i programm dlya resheniya SLAU [Krylov: library of algorithms and programs for SLAEs solution]. Sovremennye problemy matematicheskogo modelirovaniya. Matematicheskoe modelirovaniye, chislennye metody i compleksy programm. Sbornik trudov Vserossiyskikh nauchnykh molodezhnykh shkol [Modern problems of mathematical simulation. Mathematical simulation, numerical methods and program complexes. All-Russian young scientists schools proceedings]. Rostov-na-Donu, Publishing of South Federal University, 2009. P. 110–128.
2. Butyugin D.S. Metody parallelnogo resheniya SLAU na systemakh s raspredelennoy pamatyu v biblioteke Krylov [Methods of parallel SLAEs solution on the systems with distributed memory in Krylov library]. Vestnik YUURGU. Seriya "Vychislitelnaya matematika i informatika" [Bulletin of South Ural State University. Seriers: Computational Mathematics and Software Engineering], 2012. No. 47(306). P. 5–19.
3. Il'in V.P. Metody konechnykh raznostey i konechnykh objemov dlya ellipticheskikh uravnenij [Methods of finite differences and finite volumes for elliptic equations] [Methods and technologies of finite elements]. Novosibirsk, ICM&MG SBRAS Publishing, 2001.
4. Il'in V.P. Metody i tekhnologii konechnykh elementov [Methods and technologies of finite elements]. Novosibirsk, ICM&MG SBRAS Publishing, 2007.

5. Il'in V.P. Parallelnye metody i tekhnologii decompozitsii oblastey [Parallel methods and technologies of domain decomposition]. Vestnik YUURGU. Seriya "Vychislitelnaya matematika i informatika" [Bulletin of South Ural State University. Seriers: Computational Mathematics and Software Engineering], 2012. No. 46(305). P. 31–44.

6. Il'in V.P., Knysh D.V. Parallelnye metody decompozitsii v prostranstvakh sledov [Parallel decomposition methods in trace spaces]. Vychislitelnye metody i programmirovanie [Computational methods and programming], 2011. Vol. 12, No. 1. P. 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/doclib/mkl_sa/11/mklman/index.htm` (accessed: 12.02.2013).

9. Il'in V.P. Metody bisopryazhennykh napravleniy v podprostranstvakh Krylova [Bi-conjugate directions methods in Krylov subspaces]. SibZHIM [Syberian Journal of Industrial Mathematics], 2008. Vol. 11, No. 4(36). P. 47–60.

10. Bell, N. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations / N. Bell, M. Garland. URL: `http://cusp-library.googlecode.com` (accessed: 15.10.2012).

11. PETSc: Home Page. URL: `http://www.mcs.anl.gov/petsc/` (accessed: 12.02.2013).

12. Hypre. URL: `http://acts.nersc.gov/hypre/` (accessed: 12.02.2013).

13. Yousef Saad – Software. URL: `http://www-users.cs.umn.edu/~saad/software/` (accessed: 12.02.2013).

14. Klaster HKC-30T [Cluster HKC-30T].
URL: `http://www2.sscc.ru/HKC-30T/HKC-30T.htm` (accessed: 12.02.2013).

15. Nabben R., Vuik C. A comparison of abstract versions of deflation, balancing and additive coarse grid correction preconditioners // Numerical Linear Algebra with Applicati 2008. Vol. 15, No 4. P. 355–372.

Поступила в редакцию 14 июня 2013 г.

УДК 004.432

ИСПОЛЬЗОВАНИЕ ЯЗЫКА FORTRAN DVMH ДЛЯ РЕШЕНИЯ ЗАДАЧ ГИДРОДИНАМИКИ НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ[1]

В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, А.А. Смирнов

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.

Ключевые слова: DVM for Heterogeneous systems, Fortran DVMH, гибридные системы с ускорителями, графические процессоры, CUDA.

Введение

Будущее высокопроизводительных компьютерных технологий неразрывно связано с массивным параллелизмом и с гетерогенностью. Создаются процессоры, содержащие все 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 наукоемкая 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.

Цель данной работы – исследование эффективности DVMH-подхода к созданию прикладного программного обеспечения на примере задач гидродинамики.

В разделе 1 описаны основные возможности языка Fortran DVMH. В разделе 2 приведены сведения о задачах гидродинамики «Каверна» и «Контейнер», описаны преобразования, которые потребовались при их распараллеливании с использованием языка Fortran DVMH. В разделе 3 приводятся экспериментальные данные об эффективности

[1] Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии – 2013».

выполнения полученных параллельных программ на суперкомпьютерном комплексе МГУ «Ломоносов», выполнено сравнение использования высокоуровневых и низкоуровневых средств программирования для реализации одной и той же прикладной задачи при использовании до 1024 GPU.

1. 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.

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.

1.1. 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.

При помощи клауз вычислительного региона может быть задано:

1) Направление использования подмассивов и скаляров в регионе:

`IN(subarray_or_scalar {, subarray_or_scalar})` – по входу в регион нужны актуальные данные;

`OUT(subarray_or_scalar {, subarray_or_scalar})` – значения указанных переменных в регионе изменяются и могут быть использованы далее;

`LOCAL(subarray_or_scalar {, subarray_or_scalar})` – значения указанных переменных в регионе изменяются, но эти изменения не будут использованы далее;

`INOUT(subarray_or_scalar {, subarray_or_scalar})` – сокращенная запись одновременно двух клауз `IN` и `OUT`;

`INLOCAL(subarray_or_scalar {, subarray_or_scalar})` – сокращенная запись одновременно двух клауз `IN` и `LOCAL`.

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) Список типов вычислителей, на которых предполагается исполнять регион:

`TARGETS(target_name {, target_name})`

где `target_name` – это `CUDA` | `HOST`.

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 – возможность 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 имеющим указанную размерность.

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.

1.2. 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`).

1.3. 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 – это 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), инициализацию редукционных переменных, а после выполнения порции передает 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 и параллельных ЭВМ различной конфигурации.

1.4. Пример программы Якоби на языке Fortan DVMH

Проиллюстрируем возможности языка Fortan DVMH на примере программы для алгоритма Якоби (см. рисунок).

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 память.

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)`.

```
      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$    ACTUAL(EPS)
!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
```

Программа Якоби на языке Fortan DVMH

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

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

2.1. 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.

2.2. 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.

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

3.1. 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 3200×3200 на разном числе ядер и GPU (в секундах).

Таблица 1

Время выполнения программы «Каверна» на сетке 3200×3200 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

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.

Таблица 2

Время выполнения программы «Каверна» на сетке 3200×3200 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

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

Таблица 4

Время выполнения программы «Контейнер» на разном числе GPU

Сетка, кол-во итераций	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**1280**
200×200×200 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
400×400×400 itmax=100	–	–	168,80	89,17	47,15	26,09	14,17	8,39	4,86	3,20	2,88	3,26
800×800×800 itmax=50	–	–	–	–	–	92,20	51,80	30,32	13,58	7,56	4,67	4,17
1600×1600×1600 itmax=20	–	–	–	–	–	–	–	–	37,38	20,14	10,74	8,95

Для сеток 200×200×200 и 400×400×400 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 200×200×200 при использовании 4 GPU программа ускоряется в 16,47 раз по сравнению с выполнением на 4-х ядрах.

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

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

Для сетки 1600×1600×1600 при использовании 512 GPU программа ускоряется в 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 150×150×150), 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	87,82	88,8	89,29	90,21	90,99	91,4	91,57	91,97	92,46	92,74
эффективность, %	100	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	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 Кб используется для L1, а 48 Кб – для общей памяти. В системе поддержки выполнения DVMH-программ задан режим cudaDeviceSetCacheConfig(cudaFuncCachePreferL1), в котором 48 Кб используется для кэша L1, а 16 Кб – 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.

Заключение

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.

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.

Литература

1. Konovalov, N.A. Fortran DVM – a Language for Portable Parallel Program Development / N.A. Konovalov, V.A. Krukov, S.N. Mihailov, A.A. Pogrebtsov // Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. – Moscow, 1994. – P. 124–133.

2. CUDA C Programming Guide. URL: http://docs.nvidia.com/cuda-c-programming-guide/index.html (дата обращения: 13.06.2013).
3. CUDA Fortran. Programming Guide and Reference. Release 2013. URL: http://www.pgroup.com/lit/whitepapers/pgicudaforug.pdf (дата обращения: 13.06.2013).
4. Бахтин, В.А. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Вестник Южно-Уральского университета. Серия «Математическое моделирование и программирование». – 2012. – № 18(277). – Выпуск 12. – С. 82–92.
5. Антонов, А.С. Практика суперкомпьютера «Ломоносов» / Вл.В. Воеводин, С.А. Жуматий, С.И. Соболев, А.С. Антонов, П.А. Брызгалов, Д.А. Никитенко, К.С. Стефанов, Вад.В Воеводин // Открытые системы. – М.: Издательский дом «Открытые системы», 2012. – № 7. – С. 36–39.
6. Давыдов, А.А. Моделирование течений несжимаемой жидкости и слабосжимаемого газа на многоядерных гибридных вычислительных системах. / А.А. Давыдов, Б.Н. Четверушкин, Е.В. Шильников // Ж. Вычисл. матем. и матем. физ. – 2010. – Т. 50, № 12. – С. 2275–2284.

Бахтин Владимир Александрович, кандидат физико-математических наук, заведующий сектором, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), bakhtin@keldysh.ru.

Клинов Максим Сергеевич, кандидат физико-математических наук, старший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), klinov@keldysh.ru.

Крюков Виктор Алексеевич, доктор физико-математических наук, профессор, заведующий отделом, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), krukov@keldysh.ru.

Поддерюгина Наталия Викторовна, кандидат физико-математических наук, старший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), konov@keldysh.ru.

Притула Михаил Николаевич, младший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), pritmick@yandex.ru.

Смирнов Александр Андреевич, младший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), as_47@mail.ru.

USAGE OF FORTRAN DVMH LANGUAGE FOR SOLVING HYDRODYNAMICS PROBLEMS ON HYBRID COMPUTING SYSTEMS

V.A. Bakhtin, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

M.S. Klinov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

V.A. Krukov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

N.V. Podderyugina, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

M.N. Pritula, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

A.A. Smirnov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation)

In the 2011 year DVMH programming model for new heterogeneous and hybrid supercomputer systems (or DVM for Heterogeneous systems) was introduced in the Keldysh Institute for Applied Mathematics of RAS. The developed high-level programming languages were based on standard Fortran and C programming languages, but extended with the directives for mapping the program onto a parallel computer. The directives are represented as special comments (or pragmas). The paper includes analysis of the efficiency of the developed programs for solving the hydrodynamics problems «Cavity» and «Container». The calculation results are gained by using several thousand CPU cores and gained by using more than 1200 GPU accelerators are presented.

Keywords: DVM for Heterogeneous systems, Fortran DVMH, hybrid computational systems with accelerators, GPU, CUDA.

References

1. Konovalov N.A., Krukov V.A., Mihailov S.N., Pogrebtsov A.A. Fortran DVM - a Language for Portable Parallel Program Development. Proceedings of Software For Multiprocessors & Supercomputers: Theory, Practice, Experience. Institute for System Programming, RAS, Moscow, 1994. P. 124–133.
2. CUDA C Programming Guide. URL: http://docs.nvidia.com/cuda-c-programming-guide/index.html (accessed: 13.06.2013).
3. CUDA Fortran. Programming Guide and Reference. Release 2013. URL: http://www.pgroup.com/lit/whitepapers/pgicudaforug.pdf (accessed: 13.06.2013).
4. Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Extension of DVM Parallel Programming Model for Clusters with Heterogeneous Nodes [Rasshirenie DVM-modeli parallel'nogo programmirovanija dlja klasterov s geterogennymi uzlami]. Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya «Matematicheskoe modelirovanie i programmirovanie» [Bulletin of South Ural State University. Series: Mathematical Modeling, Programming & Computer Software]. 2012. No. 18(277). Vol. 12. P. 82–92.
5. Sadovnichy V., Tikhonravov A., Voevodin Vl., Opanasenko V. «Lomonosov»: Supercomputing at Moscow State University. In Contemporary High Performance Computing:

From Petascale toward Exascale (Chapman & Hall/CRC Computational Science), 2013, Boca Raton, USA, CRC Press, P. 283–307.

6. Davydov A.A., Chetverushkin B.N., Shil'nikov E. V. Simulation of incompressible flow and weakly compressible gas on hybrid multi-core computing systems [Modelirovanie techenij neszhimaemoj zhidkosti i slaboszhimaemogo gaza na mnogojadernyh gibridnyh vychislitel'nyh sistemah]. J. Vychislitel'naja matematika i matematicheskaja fizika [Computational Mathematics and Mathematical Physics], 2010, Vol. 50, Issue 12, P. 2275–2284.

Поступила в редакцию 14 июня 2013 г.

СВЕДЕНИЯ ОБ ИЗДАНИИ

Серия основана в 2012 году.

Свидетельство о регистрации ПИ ФС77-26455 выдано 13 декабря 2006 г. Федеральной службой по надзору за соблюдением законодательства в сфере массовых коммуникаций и охране культурного наследия.

ПРАВИЛА ДЛЯ АВТОРОВ

1. Правила подготовки рукописей и пример оформления статей можно загрузить с сайта серии `http://vestnikvmi.susu.ru`. **Статьи, оформленные без соблюдения правил, к рассмотрению не принимаются и назад авторам не высылаются.**
2. Адрес редакции научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»:

   Россия 454080, г. Челябинск, пр. им. В.И. Ленина, 76, Южно-Уральский государственный университет, факультет Вычислительной математики и информатики, кафедра СП, ответственному секретарю, доценту Цымблеру Михаилу Леонидовичу.
3. Адрес электронной почты редакции: `vestnikvmi@gmail.com`
4. **Плата с авторов за публикацию рукописей не взимается, и гонорары авторам не выплачиваются.**
5. Подписной индекс научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»: 10244, каталог «Пресса России». Периодичность выхода — 4 выпуска в год (февраль, май, август и ноябрь).