



ЧИСЛЕННЫЕ МЕТОДЫ,
ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

МОСКВА 2008

Российская академия наук
Институт вычислительной математики

Московский государственный университет им. М. В. Ломоносова
Научно-исследовательский вычислительный центр

**ЧИСЛЕННЫЕ МЕТОДЫ,
ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

Сборник научных трудов
под редакцией Вл. В. Воеводина и Е. Е. Тыртышникова

Москва 2008

УДК 519.6
ББК 22.20

Численные методы, параллельные вычисления и информационные технологии: Сборник научных трудов / Под ред. Вл. В. Воеводина и Е. Е. Тыртышникова. — М.: Издательство Московского Университета, 2008. — 310 с. ISBN 978-5-211-05503-3

В сборнике представлены работы по численным методам, параллельным вычислениям и информационным технологиям — по трем направлениям, составляющим круг научных интересов академика Валентина Васильевича Воеводина и созданной им научной школы.

Для научных работников, аспирантов и студентов, специализирующихся в области численного анализа и разработки современного программного обеспечения.

The book presents a collection of works on numerical methods, parallel computations and information technologies — the three directions comprising the area of scientific interests of the academician Valentin Vasilievich Voevodin and of the scientific school founded by him.

The book is addressed to researchers, post graduates and students majorizing in numerical analysis and modern software development.

УДК 519.6
ББК 22.20

ISBN 978-5-211-05503-3

© НИВЦ МГУ, 2008.

*Посвящается светлой памяти академика
Валентина Васильевича Воеводина*

ПРЕДИСЛОВИЕ

В сборнике представлены работы по численным методам, параллельным вычислениям и информационным технологиям — по трем направлениям, составляющим круг научных интересов академика Валентина Васильевича Воеводина и созданной им научной школы, включающей уже фактически два научных поколения. Можно с удовлетворением отметить, что основная часть собранных здесь работ принадлежит «научным внукам» Валентина Васильевича.

Валентин Васильевич ушел от нас 27 января 2007 года, и уже нет возможности обратиться к нему за советом по какому-либо научному или житейскому вопросу. Он всегда умел правильно определить, в какое дело нужно впрягаться, а что можно и отложить. Но самое главное, он научил нас тому, как следует размышлять при принятии подобных решений. Мы очень надеемся, что эти решения и связанные с ними результаты, представленные в сборнике, будут достойны светлой памяти Валентина Васильевича Воеводина.

Вл. В. Воеводин
Е. Е. Тыртышников

НИВЦ МГУ: широким фронтом совместных дел

Вл. В. Воеводин*

Удивительно тесно переплелась жизнь Валентина Васильевича Воеводина с Вычислительным центром Московского университета. Без какого-либо преувеличения — 50 лет НИВЦ МГУ и Валентин Васильевич были вместе! Сначала в 1953 году именно рассказ будущих сотрудников ВЦ И.С.Березина и Е.А.Жоголева (сам ВЦ будет образован только в 1955 году) перед студентами 2-го курса мехмата повлиял на его выбор кафедры вычислительной математики в качестве основы для формирования будущей профессии. Осенью 1956 года, еще будучи студентом 5-го курса, Валентин Васильевич приходит на работу в вычислительный центр, а со следующего года становится его постоянным сотрудником, пройдя в результате путь от старшего лаборанта до директора. С 1969 года он назначается исполняющим обязанности заведующего ВЦ, а с 1970 — директором, проработав в этой должности до 1978 года. С 1981 года Валентин Васильевич переходит на работу в Отдел вычислительной математики при Президиуме АН (сейчас — это Институт вычислительной математики РАН), однако мысли о необходимости создания системы подготовки высококлассных специалистов, способных поднимать и развивать вычислительное дело, опять возвращают его в лоно Московского университета. В 1990 году коллектив единомышленников на базе НИВЦ МГУ создает Высшую компьютерную школу, работу которой многие годы возглавляет Валентин Васильевич. И до самого последнего момента он являлся членом Ученого Совета НИВЦ, членом диссертационного совета НИВЦ, членом редколлегии журнала НИВЦ «Вычислительные методы и программирование»...

В начале 90-х формируется и наша лаборатория, которая стала веточкой его научной школы, что предопределило исключительно

*Научно-исследовательский вычислительный центр МГУ

тесные не только человеческие, но и научные контакты на все эти годы. Лаборатория параллельных информационных технологий появилась в НИВЦ МГУ в 1999 году. Она стала полноправной преемницей лаборатории нелинейных вычислений, организованной в 1992 году по инициативе В.М.Репина, возглавлявшего в те годы вычислительный центр. Область научных интересов сотрудников лаборатории обширна и включает такие области, как параллельные вычисления, математические методы исследования тонкой структуры программ, методы описания и анализа архитектуры компьютеров, технологии параллельного программирования, методы оптимизации программ для суперкомпьютеров и параллельных вычислительных систем, Интернет-технологии и Интернет-проекты в науке, организация распределенных вычислений, метакомпьютинг, электронные системы в образовании.

В данной работе кратко описаны те направления фундаментальных исследований и конкретные прикладные проекты, которые выполняются силами нашего коллектива либо же в которые наши сотрудники активно вовлечены, работая в тесном контакте с коллегами из других областей науки. При этом не ставилась цель дать подробное описание каждого проекта или направления, скорее хотелось дать представление о спектре работ, выполняемых в коллективе. Вместе с этим, отметим, что часть упомянутых здесь проектов более подробно представлена в настоящем сборнике в виде отдельных статей.

ИНФОРМАЦИОННО-АНАЛИТИЧЕСКИЙ ЦЕНТР ПО ПАРАЛЛЕЛЬНЫМ ВЫЧИСЛЕНИЯМ PARALLEL.RU

Всё о мире суперкомпьютеров и параллельных вычислений... Именно такой подзаголовок лучше всего подходит к описанию тематики данного центра в сети Интернет. Первый вариант Web-сервера, посвящённого основным вопросам параллельных вычислений, был создан в мае 1998 года. Активно развиваясь, проект очень быстро вышел за рамки, характерные для традиционного тематического Web-ресурса, преобразовавшись в специализированный Информационно-аналитический Центр по параллельным вычисле-

ниям в сети Интернет. Сервер перестал быть статическим образованием, объединив вокруг себя большое число профессионалов, участвующих в обсуждении актуальных вопросов, предлагающих свою помощь по наполнению отдельных специализированных разделов, подготовке новых материалов, помогающих лучше понять особенности данной предметной области.

Тематика Центра охватывает следующие разделы: обзор новостей мира высокопроизводительных вычислений, описание архитектур компьютеров, технологии параллельного программирования, обзор источников информации в данной области, списки конференций по данной тематике, списки крупнейших суперкомпьютерных центров, информация о российских ресурсах в данной области, очерки по истории высокопроизводительных вычислений и многое другое.

Вокруг Центра сформировалось и активно функционирует сетевое сообщество. Собирается информация о персоналиях и организациях, вовлечённых в данную деятельность, выполняется регулярная рассылка по электронной почте как новостей Центра, так и новостей мира высокопроизводительных вычислений (на 2007 год это более 3000 подписчиков, представляющих все основные коллективы России и ближнего зарубежья, работающие в данной области). В Центре организован Дискуссионный клуб, в котором активно обсуждаются различные вопросы суперкомпьютерной тематики. Оказывается содействие в организации и информационной поддержке научных конференций, семинаров и других мероприятий, затрагивающих проблематику параллельных вычислений.

Интернет-центр одновременно решает задачи организации эффективного доступа к вычислительным ресурсам и консультаций для пользователей. Для поддержки этого направления был создан раздел Суперкомпьютерного комплекса Московского университета (<http://parallel.ru/cluster/>), реализована pilotная версия вычислительного и процессорного полигонов. Подготовлен большой объём учебно-методического материала, а основные издания объединены в серию «Библиотека учебных материалов Parallel.ru».

Адрес проекта в сети Интернет — <http://www.parallel.ru/>.

СПИСОК ТОР50 САМЫХ МОЩНЫХ КОМПЬЮТЕРОВ СНГ

Большие задачи появились одновременно с рождением вычислительной техники, и будут существовать всегда. Они всегда вызывают повышенный интерес, поскольку являются отражением новых компьютерных методов проведения исследований в различных областях науки, расширяя и дополняя традиционные подходы. Вычислительная аэро- и гидродинамика, молекулярное моделирование, квантовохимические методы, вычислительные технологии биоинформатики и биоинженерии имеют колоссальный потенциал для решения реальных задач, однако все они предъявляют исключительно жесткие требования к параметрам используемых компьютерных систем. Сформировалось даже специальное понятие, отражающее компьютеры с предельно высокой производительностью — суперкомпьютеры.

Чтобы помочь правильно сориентироваться в мире высокопроизводительных вычислительных систем и иметь возможность оперативно отслеживать тенденции развития данной области, НИВЦ МГУ и МСЦ РАН в мае 2004 года начали совместный проект по формированию списка 50 наиболее мощных компьютеров СНГ. С тех пор список обновляется два раза в год, а правила его ведения определены в положении, опубликованном на сайте проекта.

Несмотря на небольшую продолжительность проекта, опубликованные к настоящему времени семь редакций списка Тор50 уже содержат много интересного материала, на основе которого можно проводить содержательный анализ динамики развития суперкомпьютерной отрасли на территории СНГ. На сайте проекта доступны все редакции списка, подробная информация по отдельным вычислительным системам, статистика, новости, ведется архив. Предоставлена возможность гибкого формирования собственной выборки по индивидуальным критериям для проведения более детального анализа, который дает исключительно богатую пищу для размышлений и помогает определить множество нетривиальных закономерностей в развитии суперкомпьютерной техники [1].

Адрес проекта Тор50 в сети Интернет —
<http://www.supercomputers.ru/>.

СУПЕРКОМПЬЮТЕРНЫЙ КОМПЛЕКС МОСКОВСКОГО УНИВЕРСИТЕТА

Собственно вычислительный центр Московского государственного университета имени М.В.Ломоносова был создан в 1955 году и с момента основания был сразу оснащен самой передовой вычислительной техникой. Уже в декабре 1956 года в ВЦ была установлена первая серийная отечественная машина — «Стрела». В 1961 году была введена в строй машина М-20, в 1966 БЭСМ-4. К 1981 году в ВЦ функционировали четыре БЭСМ-6, две ЕС-1022, Минск-32, две ЭВМ Мир-2 и разработанная в самом ВЦ первая в мире ЭВМ «Сетунь» с троичной системой счисления.

Современный НИВЦ входит в число крупнейших суперкомпьютерных центров России. Приняв за основу исследования лаборатории и выбрав в 1999 году в качестве главного направления развития средств вычислительной техники использование кластерных систем, НИВЦ МГУ стал первой российской организацией, которая построила суперкомпьютерный центр коллективного пользования на данном типе архитектуры. Сегодня основной базой для проведения высокопроизводительных вычислений являются мощные вычислительные кластеры, суммарная производительность которых достигла двух триллионов операций в секунду [2]. Вопросы мониторинга функционирования суперкомпьютерного комплекса, сопровождения программной инфраструктуры, обеспечение поддержки пользователей лежат на сотрудниках лаборатории.

Среди задач комплекса — поддержка фундаментальных научных исследований и учебного процесса Московского университета. Доступ к суперкомпьютерным ресурсам предоставлен сотрудникам всех подразделений МГУ, а также ряда ведущих учебных и научных организаций России. Создано мощное централизованное хранилище данных. Спроектирована и уже применяется на практике технология глобальных вычислений с использованием разработанной в НИВЦ системы метакомпьютинга X-Сом. В состав суперкомпьютерного комплекса НИВЦ входит процессорный полигон с единой системой управления, объединяющий различные вычислительные платформы на базе основных типов современных процессоров. Основная цель полигона — предоставить пользователям инструмент для исследования и освоения новых процессорных технологий и си-

стем компиляции.

Для развития возможностей суперкомпьютерного комплекса, сотрудники лаборатории ведут активные исследования по методам решения задач с использованием компьютеров с реконфигурируемой архитектурой (FPGA-компьютеры). Исключительно интересное направление, которое стало развиваться в последнее время в лаборатории — это разработка технологий использования графических процессоров в качестве универсальных устройств для решения вычислительно сложных задач.

Сегодня идет качественное расширение возможностей суперкомпьютерного комплекса. Руководством Московского университета принято решение об установке суперкомпьютерной кластерной системы с рекордно высокой производительностью в 60 Тфлопс — самой мощной вычислительной системы на территории СНГ.

Описание состава и пользовательской инфраструктуры суперкомпьютерного комплекса представлено в сети Интернет на странице <http://parallel.ru/cluster/>.

CLEO — СИСТЕМА УПРАВЛЕНИЯ ПРОХОЖДЕНИЕМ ЗАДАНИЙ НА ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРАХ

Одной из важнейших компонент современного вычислительного кластера является система управления прохождением заданий. Система Cleo, разработанная в лаборатории, предназначена для управления программами на кластерах различных конфигураций, с различными требованиями к заданиям и режимам использования вычислительных ресурсов. Поддерживаются все основные среды параллельного программирования, такие как mpich, mchapich, Intel MPI и другие.

Система переносима на большинство UNIX-платформ, что позволяет использовать её практически на любых кластерных установках. Поддерживается работа с несколькими кластерами одновременно, а также с отдельными разделами внутри кластеров. Гибкая настройка параметров системы, политик использования ресурсов и планировщика запуска заданий делает её удобной для администрирования и планирования режимов использования кластера.

Возможности автоматической и ручной блокировки узлов и заданий упрощают процесс проведения профилактических работ кластеров без необходимости полной остановки работы пользователей. Система приоритетов и предсказания загрузки позволяют минимальными усилиями эффективно управлять загрузкой кластера.

Cleo легко расширяема. Интерфейс модулей документирован и проиллюстрирован примерами, что позволяет быстро создавать собственные планировщики или дополнять возможности системы новой функциональностью. Все модули Cleo работают в защищённой среде, что повышает защищённость системы в целом.

Состояние системы доступно в XML-формате и может быть использовано любыми внешними программами. Средства сбора статистики предоставляют как комплексные, так и детальные отчёты о работе пользователей на кластере.

В настоящее время под управлением Cleo работают кластеры суперкомпьютерного комплекса НИВЦ МГУ и ряда других организаций.

Исходные тексты Cleo доступны по адресу:
<http://sf.net/projects/cleo-bs/>.

СЕРТИФИКАЦИЯ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Для эффективной оптимизации вычислительно сложных программ необходимо изучение их поведения на различных компьютерных платформах и конфигурациях, что мы называем сертификацией эффективности работы программ в программно-аппаратных средах. Для проведения подобного исследования недостаточно иметь только аппаратные средства и собственно готовые варианты программ или же их исходные тексты. Необходима как методика проведения тестирования, так и средства, позволяющие проводить оценку эффективности работы тестируемых программ и оценивать влияние аппаратных особенностей на их поведение. Программный комплекс, позволяющий проводить такие исследования, должен включать в себя систему мониторинга для сбора информации о работе исследуемой задачи, средства для привязки времени работы тестовой задачи и узлов к параметрам запуска, а также средства визуализации и анализа.

Требования к системе мониторинга особые. Данные должны собираться с высокой степенью детализации, но иметь минимальные накладные расходы и не мешать работе исследуемых задач. Именно с учётом этих требований была разработана система мониторинга AntMon. Модульная архитектура позволяет быстро наращивать функциональность системы, а применённые при ее разработке специальные технологии и протоколы взаимодействия призваны снизить нагрузку на вычислительные узлы и коммуникационную сеть.

Помимо системы мониторинга необходимы средства, позволяющие сопоставить собранные данные со временем и параметрами запускаемых задач. Система управления прохождением заданий Cleo обладает широкими возможностями расширения, поэтому с ее помощью легко получить данные о работе задач как после проведения серии запусков, так и в процессе их выполнения.

Для изучения программы недостаточно лишь получить данные мониторинга и сопоставить их с данными о задаче: необходим последующий анализ полученных данных, позволяющий сравнить результаты запусков и выделить узкие места, мешающие работе программы. На решение этого круга вопросов нацелен программный комплекс ParCon, который объединяет данные AntMon и Cleo в единую базу и позволяет проводить анализ запущенных или уже завершённых задач.

Используя возможности ParCon, Cleo и AntMon в совокупности со специальной методикой тестирования, удается проводить комплексную сертификацию эффективности работы пользовательских программ на суперкомпьютерных кластерных системах.

Адрес проекта в сети Интернет — <http://parcon.parallel.ru/>.

СИСТЕМА МЕТАКОМПЬЮТИНГА Х-СОМ

Идея метакомпьютинга состоит в объединении доступных распределенных вычислительных ресурсов для решения различных прикладных задач. Исходные предпосылки просты. Рассмотрим локальную сеть какого-либо современного предприятия или же учебный компьютерный класс. Компьютеры, составляющие такие классы или сети, в большинстве случаев простоявают по ночам и в выходные дни, да и в рабочее время редко используются с полной загрузкой. Объединив эти компьютеры для работы над единой

задачей в те моменты, когда они не заняты другой работой, можно достичь производительности, сравнимой с производительностью кластерной установки с аналогичным числом узлов. Другой пример: вычислительная сложность прикладной задачи может быть столь велика, что с ее обработкой за разумное время не справляется имеющийся в распоряжении высокопроизводительный комплекс. Однако задача поддается решению, если к расчетам подключаются дополнительные вычислительные мощности из других организаций, городов, стран.

Особый интерес с этой точки зрения представляет сеть Интернет. В самом деле, все компьютеры, подключенные к глобальной сети, потенциально можно использовать для решения какой-либо одной задачи. Но только в том случае, если будет решен целый ряд технических и организационных проблем.

К настоящему моменту разработаны основы программирования распределенных вычислительных сред [3] и реализована система метакомпьютерных расчетов X-Сом, предназначенная для организации распределенных вычислений с использованием разнородных вычислительных ресурсов. Для решения прикладных задач система X-Сом задействует уже имеющиеся компьютеры и каналы связи. Отличительные черты системы X-Сом — это поддержка большинства современных программно-аппаратных платформ, возможность быстрого разворачивания вычислительных экспериментов, использование вычислительных ресурсов в различных режимах без необходимости вмешательства в их локальные политики администрирования, учет динамичности распределенных вычислительных сред, их крайней неоднородности и потенциальной ненадежности составляющих их отдельных вычислительных узлов.

К настоящему времени система X-Сом успешно прошла апробацию в ходе решения большого числа реальных прикладных задач биоинженерии, биоинформатики, проектирования лекарственных препаратов, электродинамики. Для решения этих задач задействовались вычислительные ресурсы НИВЦ МГУ, ВМиК МГУ, МСЦ РАН, суперкомпьютерные установки в Челябинске, Уфе, Томске, Дубне, Чернологовке, Пущино и других городах. В частности, задача поиска молекул-ингибиторов для заданных белков-мишеней (проект выполнялся совместно с Гематологическим научным центром

РАМН) была решена за 11 дней с использованием 270 процессоров трех кластеров и одного учебного класса, расположенных в НИВЦ МГУ (Москва) и ЮУрГУ (Челябинск). Эффективность расчета составил более 97%, хотя среда формировалась из самых разных процессоров: Intel Pentium III 500MHz, Intel Xeon 2.6 GHz, Intel Xeon EM64T 3.2 GHz, AMD Opteron 2.2 GHz и других.

Сайт системы X-Com: <http://X-Com.parallel.ru/>.

ПРОЦЕССОРНЫЙ ПОЛИГОН

В рамках данного проекта в лаборатории создается процессорный полигон с единой системой управления, объединяющий различные вычислительные платформы на базе современных процессоров. Основная цель — дать инструмент для исследования и освоения новых процессорных технологий. Пользователям предоставляется не только подробная информация о результатах предварительного комплексного тестирования серверов, входящих в состав полигона, но и возможность запуска на вычислительных серверах стандартных тестов и фрагментов программ.

Как сориентироваться в многообразии современных вычислительных платформ? Как подобрать оптимальную конфигурацию компьютера для проведения научных исследований в конкретной прикладной области? На какой тип процессоров ориентироваться? На основе какой вычислительной платформы имеет смысл делать высокопроизводительные кластерные системы? Каков реальный эффект от использования многопроцессорных платформ и многоядерных процессоров на конкретных приложениях? На подобный спектр вопросов и помогает ответить процессорный полигон.

Модернизация компьютерного парка проходит регулярно, поэтому и отвечать на схожие вопросы приходится постоянно. Как решаются эти вопросы сейчас? Чаще всего, на основе сложившихся традиций или же предпочтений системного администратора. А как нужно бы делать выбор? Конечно же, на основе предварительного испытания различных вычислительных платформ, на основе их тестирования на типичных алгоритмических конструкциях или программах из конкретной предметной области пользователя. Только аккуратный сравнительный анализ, опирающийся на данные реальных прогонов и испытаний в различных вычислительных средах,

поможет сделать обоснованный выбор.

На сентябрь 2007 года процессорный полигон НИВЦ МГУ содержал 20 серверов. Это 5 серверов на базе процессоров AMD Opteron, один сервер на базе процессора IBM Power5 и 14 серверов на базе процессоров Intel. Суммарная пиковая производительность серверов процессорного полигона составляет 475 GFlop/s, суммарный объём оперативной памяти 118 Гбайт. Процессорный полигон реализован в стоечном исполнении, 19" на 42U, все узлы имеют форм-фактор 1–2U. Конфигурации вычислительных узлов в рамках полигона описываются такими характеристиками, как тип и частота процессора, число процессоров/ядер в узле, структура памяти, частота системной шины и многими другими. Для полноценного сравнения платформ сформирован, протестирован и установлен набор наиболее эффективных современных компиляторов: Intel, PGI, Pathscale, Absoft, GNU.

Формируемый в результате комплексного тестирования производительности вычислительных серверов документ Performance Guide содержит сравнительные характеристики производительности серверов процессорного полигона, позволяет оценить их реальные характеристики, найти узкие места при выполнении тех или иных операций, тестов, вычислительных ядер и приложений.

ЭКСКУРСИИ ДЛЯ ШКОЛЬНИКОВ НА СУПЕРКОМПЬЮТЕРНЫЙ КОМПЛЕКС НИВЦ МГУ

Окружающий нас мир быстро меняется. Компьютеры и информационные технологии проникают в нашу жизнь, предлагая все новые и новые возможности для общения, работы, учебы. Если еще 10 лет назад не все знали, что такое электронная почта, то сегодня такие понятия, как КПК, Bluetooth, сенсорные сети или же grid-технологии у многих удивления не вызывают.

Меняется и сам компьютерный мир, который за последнее время стал «параллельным»: компьютеры соединяются в мощные кластеры, процессоры становятся многоядерными, вычислительные системы разных организаций могут объединяться в распределенные вычислительные среды для совместного решения особо сложных задач.

Это особенно ярко проявляется в суперкомпьютерных вычислительных системах, работающих на предельных скоростях и обладающих рекордной производительностью. Подобные системы уникальны, их не много, и доступ к ним имеет лишь небольшое число специалистов. Однако все те технологии, которые сначала отрабатываются на суперкомпьютерных системах, через какое-то время становятся массовыми и доступными для широкого использования.

Мы предлагаем школьникам и их наставникам заглянуть в наше будущее и приглашаем на экскурсию в суперкомпьютерный комплекс НИВЦ МГУ.

Интересных и неожиданных вопросов в суперкомпьютерном мире очень много. Мы поговорим о том,

- почему наш мир стал компьютерным, а компьютерный мир — параллельным,
- почему для создания и хорошего автомобиля, и даже хороших кроссовок, обычного компьютера недостаточно, и нужен суперкомпьютер,
- почему современный компьютер может занимать целый зал, весить 20 тонн, и это считается вполне нормальным явлением,
- почему Интернет является самым большим компьютером мира,
- как можно сделать суперкомпьютерную систему из школьных компьютеров или домашних персоналок...

При подготовке экскурсии мы исходим из того, что слушатели имеют базовую компьютерную подготовку, владеют навыками программирования и уже имеют опыт работы на персональных компьютерах в объеме учебных программ лицеев и школ, специализирующихся в области информационных технологий. Практика показала, что ребята прекрасно воспринимают новый и весьма необычный для них материал, а увиденные суперкомпьютерные системы никого из них не оставляют равнодушными.

ЛИНЕАЛ: ЭЛЕКТРОННАЯ ЭНЦИКЛОПЕДИЯ ПО ЛИНЕЙНОЙ АЛГЕБРЕ

В проблеме обучения любому предмету можно выделить два основных аспекта. Первый связан с тем, какую совокупность знаний надо освоить. Второй определяет то, как эта совокупность знаний должна быть освоена. Традиционно совокупность знаний записывается на бумажных носителях в форме книг. При всех достоинствах такой способ записи имеет существенный недостаток — с его помощью практически невозможно описать причинно-следственные связи на всем множестве фактов, определений, комментариев и т.п., имеющихся в изучаемой области. Однако это можно реализовать, используя электронную форму записи знаний и коренное преобразование формы представления самих знаний.

Подобный подход апробирован в системе ЛИНЕАЛ. На примере курса линейной алгебры реализованы общие принципы новой организации баз знаний. Основные из этих принципов два. Первый — это разбиение конкретной предметной области на отдельные статьи, представляющие определения, факты, комментарии и т.п. Второй принцип заключается в установлении причинно-следственных связей между статьями. Система ЛИНЕАЛ включает сведения, охватывающие все известные курсы по линейной алгебре. Она открыта для расширения и ориентирована на использование в сети Интернет. Материал структурно разбит на 13 разделов, 84 главы и более 1500 статей. В качестве начальных знаний для пользования системой требуется иметь лишь самые общие представления о вещественных числах и знать некоторые самые элементарные факты из школьной математики. Для быстрого нахождения нужной информации можно воспользоваться структурным и предметным указателями. Реализованы различные способы поиска информации и представления в графическом виде причинно-следственных связей в отобранным материале. Кроме подключения к Интернету и наличия стандартного браузера для работы с энциклопедией не требуется никакого дополнительного инструментария.

Программная оболочка системы ЛИНЕАЛ не зависит от предметной области и может быть использована для создания других баз знаний. Продолжением работ в данном направлении станет выпуск системы ПАРАЛЛЕЛЬ, опирающейся на те же технологии, использу-

зующей ту же самую оболочку, что и ЛИНЕАЛ, но посвященной параллельным вычислениям [4]. Первый вариант системы уже создан, и идет процесс уточнения причинно-следственных связей между отдельными статьями.

Систему ЛИНЕАЛ дополняет монография [5], которая распространяется вместе с компакт-диском, позволяющим работать с системой без необходимости выхода в Интернет.

Адрес системы в сети Интернет — <http://lineal.guru.ru/>.

АГОРА – СИСТЕМА ИНТЕРНЕТ-ПОДДЕРЖКИ ПРОВЕДЕНИЯ НАУЧНЫХ МЕРОПРИЯТИЙ

Проект АГОРА направлен на создание комплексной системы поддержки проведения конференций, симпозиумов, семинаров и других научных мероприятий в сети Интернет. Цель проекта состоит в разработке универсального средства, позволяющего быстро создавать полноценные веб-представительства научных мероприятий самого широкого спектра. Основное исходное требование на систему — быть ориентированной на самый широкий круг пользователей, не всегда знающих детали и тонкости информационных технологий, однако простота использования не должна идти в ущерб функциональности.

Предлагаемые системой АГОРА сервисы ориентированы на три категории пользователей: участников мероприятий, организаторов и экспертов. Организаторам предоставляется инструментарий для создания и поддержки веб-сайта мероприятия, обработки регистрационных данных участников, организации распределенного рецензирования присланных материалов, для помощи в выполнении многих других функций оргкомитета. Участники на веб-сайте мероприятия могут ознакомиться с необходимой текущей информацией, зарегистрироваться, отправить свои материалы в оргкомитет, подписатьсь на почтовую рассылку. Эксперты имеют возможность провести оценку выделенных оргкомитетом работ участников мероприятия в удаленном режиме.

Гибкость хорошо зарекомендовавшей себя на практике платформы (PHP+MySQL+Apache) позволяет легко наращивать набор предлагаемых сервисов. Перспективным направлением развития является реализация в рамках системы АГОРА расширенных

возможностей для перекрестного поиска по различным мероприятиям. Это, в частности, сразу дает возможность увидеть научную программу конференций, проводимых по конкретным направлениям науки, поддержанных различными научными фондами, организованные конкретными учеными или организациями и выполнять другие выборки. Аналогичные механизмы предполагается создать для проведения электронных on-line конференций и для поддержки серий мероприятий.

В настоящее время АГОРА является официальным сервисом Российского Гуманитарного Научного Фонда (<http://www.rfh.ru>, раздел «Партнеры РГНФ»), который рекомендует ее использование научным коллективам для поддержки выполнения работ по грантам фонда.

Система находится в свободном доступе по адресу <http://agora.guru.ru>.

ИНТЕРНЕТ-МУЗЕЙ «ИСТОРИЯ ИМПЕРАТОРСКОГО МОСКОВСКОГО УНИВЕРСИТЕТА»

К 250-летнему юбилею Московского университета Исторический факультет и НИВЦ МГУ выполнили совместный научно-исследовательский проект по созданию Интернет-музея «История Императорского Московского университета». В рамках данного проекта подготовлены уникальные сведения о возникновении и развитии Московского университета, его структуре, преподавании, профессорах и замечательных воспитанниках за период с основания до 1917 года.

Все подготовленные материалы представлены в сети Интернет в виде справочной системы по истории дореволюционного университета. Пользователи имеют возможность получить информацию об истории Московского университета как со статических страниц, представляющих собой иллюстрированные очерки, тексты документов и других исторических источников, так и сформулировав запрос к базе данных, объединяющей записи о персоналиях (профессорах и студентах), предметах или событиях из летописи МГУ.

В некоторых разделах Интернет-музея поисковая система позволяет максимально гибко реагировать на запросы пользователей,

касающиеся событий истории Московского университета, преподавания или персоналий. В самом начале разработки музея было сформулировано требование, чтобы система умела отвечать на вопросы типа: «Какие предметы преподавались на физико-математическом факультете в 1855 году»? или «На каких факультетах университета в разные годы преподавалась химия»? или «В каком году возникла кафедра российской истории»? или «В какой период времени в университете существовал философский факультет»? или «Какие предметы и в какие годы в университете преподавал К.А.Тимирязев»? и т.п. Поисковые задачи такого типа никогда ранее не ставились по отношению к истории высшего учебного заведения со сложной меняющейся во времени структурой. Постоянное изменение структуры, переходы предметов между кафедрами, а кафедр — с факультета на факультет, возможность одного ученого вести несколько предметов на разных кафедрах, а иногда и на разных факультетах — все это нашло отражение на страницах музея.

В качестве технологической основы проекта используется традиционная связка PHP+MySQL+Apache. Визуализация результатов производится с помощью динамического HTML и языка сценариев JavaScript.

Адрес музея в сети Интернет — <http://museum.guru.ru/>.

Данный проект, выполненный при поддержке РГНФ, оказался исключительно успешным, показав преимущества совместной работы профессионалов из разных областей науки: информационных технологий и истории. Он стал отправной точкой нового направления работы лаборатории, дальнейшее развитие которого кратко описано в следующем проекте.

ИНТЕРНЕТ-ЦЕНТР «МУЗЕЙ-КВАРТИРА АНДРЕЯ БЕЛОГО»

Специалисты в области лингвистики и компьютерных технологий НИВЦ МГУ и сотрудники музея «Мемориальная квартира Андрея Белого на Арбате» выполнили совместный научно-исследовательский проект по созданию Интернет-музея «Мемориальная квартира Андрея Белого на Арбате». Этот сайт посвящен литературе и культуре России начала XX века — «Серебряному веку» российской культуры. Он представляет интерес для всех, кто интересуется историей

интеллектуальной жизни указанной эпохи, и может использоваться как для углубления дальнейших исследований в данной области, так и в преподавании соответствующих дисциплин в высшей и средней школе. В сочетании с уже созданным Интернет-музеем Московского университета данный проект является предпосылкой для создания виртуального сообщества, объединяющего профессиональных филологов и любителей российской словесности и российской истории. Безусловно, сайт полезен и тем, кто просто планирует посетить музей-квартиру Андрея Белого.

Для создания сайта было использовано целое множество мультимедийных и Интернет-технологий, современных протоколов взаимодействия компонентов системы, технологий интерактивных запросов и удаленного доступа к базам данных. Постоянное дополнение и обновление информационных данных идет за счет контактов с научными коллективами, общения ученых в рамках виртуальных конференций и семинаров, периодических изданий Центра.

Адрес музея в сети Интернет — <http://kvartira-belogo.guru.ru/>.

Заключение

На первый взгляд представленные направления работ сотрудников лаборатории могут показаться разрозненными и не очень-то связанными друг с другом общими идеями или технологиями. Например, как можно объяснить соседство параллельных вычислений со столь тесными связями с проектами из области гуманитарных наук? Но это только на первый взгляд. Имея значительную суперкомпьютерную историю в исследовательской работе, с появлением сети Интернет возникла идея создания информационно-аналитического Интернет-центра Parallel.ru. Стали активно разбираться с Интернет-технологиями, появились первые наработки, которые позволили, в частности, организовать и провести в 1999 году первый в России массовый Интернет-конкурс «Московской университет: история, люди, факты». Понимая перспективность и востребованность данного направления, в этом же году совместно с РГУ и ИВМ РАН организовали и провели первую всероссийскую научную конференцию серии «Научный сервис в сети Интернет», которая с тех пор проводится нами ежегодно. Возникла масса контактов с учеными из самых разных областей, которые привели к возможности успешной рабо-

ты ИТ-профессионалов с профессиональными историками, филологами, лингвистами и выполнению целого ряда уникальных совместных проектов, о чём и было рассказано выше. Исследуя потенциал Интернет-технологий для организации распределенных вычислений, мы быстро пришли к возможности создания специализированных сервисов и систем таких, как ЛИНЕАЛ и АГОРА. Разрабатывая методику мониторинга параллельных программ, получили сначала средство для комплексного контроля за работой больших суперкомпьютерных комплексов, а затем именно его адаптировали для обеспечения круглосуточного мониторинга состояния серверного хозяйства РГНФ с возможностью автоматического оперативного оповещения системных администраторов фонда через Интернет-страницы, электронную почту или же мобильную связь.

Не все проекты лаборатории нашли свое отражение в данной статье, однако часть из них представлена в сборнике в виде отдельных работ. Исключительно интересным является создание коллективного банка тестов по параллельным вычислениям, для анализа эффективности программ важен вычислительный полигон и каталог Performance Guide реальной производительности вычислительных платформ, безусловно перспективным является исследование возможности использования графических процессоров для решения вычислительно сложных задач.

В целом же, чем дальше входим в тематику, тем шире становиться круг работ и разнообразнее выполняемые проекты. Наверное, это естественный процесс, при котором не всегда удается сразу оценить перспективность идей и возможности их реализации в смежных областях, но на то и существуют поисковые исследования. Главное — есть энергичная молодая группа, есть интерес, есть потенциал и желание работать, а база в наш коллектив была заложена Учителем основательная.

Список литературы

- [1] Воеводин Вл.В. Топ500: числом или уменьем? // Открытые системы. 2005, N10, С.12–15.
- [2] Воеводин Вл.В., Жуматий С.А. Вычислительное дело и кластерные системы. -М.: Изд-во МГУ, 2007. — 150 с.

- [3] Воеводин Вл.В. Решение больших задач в распределенных вычислительных средах// Автоматика и Телемеханика. 2007, N5, С.32–45.
- [4] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.
- [5] Воеводин В.В., Воеводин Вл.В. Энциклопедия линейной алгебры. Электронная система ЛИНЕАЛ. — СПб.: БХВ-Петербург, 2006. — 544 с.

О системе программирования вычислений общего назначения на графических процессорах

А. В. Адинец^{*}, Н. А. Сахарных*

Статья посвящена обсуждению вопросов программирования современных графических процессоров для решения с их помощью общих вычислительных задач. Статья начинается с общего понятия графических процессоров и вычислений на них. Далее приводится обзор типичных архитектур и основных ее особенностей, которые определяют как класс решаемых задач, так и возможности программирования. После этого дается краткий обзор существующих средств программирования для графических процессоров и описание системы, созданной авторами данной работы.

Система C\$ состоит из двух частей: языка C\$ и среды исполнения, выполняющей трансляцию на графический процессор. Язык C\$ представляет собой Java (C#)-подобный язык, расширенный конструкциями для операций с функциями и массивами. На этапе компиляции параллельный код отделяется от непараллельного, после чего транслируется в вызовы к среде исполнения. Среда исполнения работает по принципу ленивых вычислений, строя изначально граф массивных операций и исполняя его на графическом процессоре; подобный алгоритм обусловлен особенностями целевой архитектуры. В заключение даются результаты работы алгоритма, анализ и возможные направления дальнейшего развития разработанной системы.

^{*}Научно-исследовательский вычислительный центр МГУ

1. Введение

Начиная с 2003 года, активные исследования ведутся в области использования современных графических процессорных устройств (ГПУ) для решения вычислительных задач [1]. В зарубежной литературе это направление получило название GPGPU (сокращение от General Purpose Graphics Processor Unit, графическое процессорное устройство общего назначения), в настоящей статье в качестве его перевода на русский язык будет использоваться ОВГПУ (общие вычисления на графических процессорах). В настоящее время ГПУ используются для решения ряда вычислительных задач, для которых традиционно использовались суперкомпьютерные архитектуры, например, для выполнения матричных операций [2], решения уравнений в частных производных при помощи сеточных методов [3, 4], решения задач машинного зрения [5], выполнения операций обработки изображений и звука, в т.ч. преобразования Фурье [6], трассировки лучей [7] и многих других. При грамотном использовании ресурсов графического процессора удается добиться прироста реальной производительности во много раз по сравнению с использованием ресурсов только центрального процессора (речь идет о сравнении наиболее передовых образцов графических и центральных процессоров в определенный период времени). На некоторых задачах достигается реальная производительность в сотни ГФлопс [8], которая до недавнего времени была доступна лишь на компьютерных кластерах и суперкомпьютерах. Кроме того, графические процессоры обладают меньшей стоимостью в расчете на ГФлопс пиковой производительности (порядка 1–2\$/ГФлопс), а также, что особенно становится актуальным, меньшим энергопотреблением (порядка 0.5 Вт/ГФлопс) по сравнению с кластерными системами. Поэтому написание эффективных программ для графических процессоров является актуальной задачей.

В настоящее время наблюдается существенный дисбаланс между возможностями графических процессоров и их реальным использованием, вызванный прежде всего отсутствием адекватных средств программирования для данной архитектуры. В то время как для кластеров и суперкомпьютеров существует огромное количество языков программирования (прежде всего Си и ФОРТРАН [9]) и библиотек, до недавнего времени программы на ГПУ писались ис-

ключительно с использованием интерфейсов для работы с трехмерной полигональной графикой — таких как OpenGL [10]. Поскольку эти интерфейсы предназначены для эффективной работы с графикой, писать с их помощью программы для графических процессоров неудобно. В дополнение к этому программы получаются громоздкими, сложными в отладке, и непереносимыми на архитектуры, отличные от графических процессоров. В таких условиях программирование графических процессоров требует не только понимания модели программирования, но и знания всех тонкостей интерфейса для работы с трехмерной графикой и изучения языка шейдеров, например GLSL [11], что препятствует распространению ОВГПУ среди специалистов в области параллельного программирования.

В подобных условиях становится важной задача разработки инструментария для написания программ для графических процессоров. В первую очередь это именно средства программирования, затем — средства отладки, профилировки и т.д. При этом создаваемое средство должно быть сравнительно простым в освоении для специалистов в области параллельного программирования и позволять создавать эффективные и переносимые программы.

В настоящее время направление ОВГПУ получило поддержку не только в академических кругах, но и среди производителей графических процессоров. Ведущие производители графических процессоров, AMD [12] и NVIDIA [13] уже выпустили как программные средства для взаимодействия с графическими процессорами в обход интерфейсов для работы с графикой (соответственно, Data Parallel Virtual Machine (DPVM) [14] и NVIDIA CUDA [15]), так и аппаратные ускорители, ориентированные на общие вычисления. Последние представляют собой те же графические процессоры (соответственно, ATI Radeon X1900 и NVIDIA GeForce 8800), но продаваемые как решения для ускорения вычислений общего назначения.

Данная статья организована следующим образом. В разделе «Особенности архитектуры графических процессоров» дается обзор архитектуры графических процессоров и их основных возможностей. В разделе «Существующие подходы программирования» рассматриваются уже существующие подходы программирования ГПУ, дается их анализ. В разделе «Система C\$: идеи и архитектура» рассматриваются идеи, положенные в основу системы C\$ и их



Рис. 1. Программируемый графический конвейер

влияние на архитектуру создаваемой системы. В разделе «Трансляция операций над массивами в системе C\$» рассказывается, как по дереву вычислений строится программа для ГПУ, учитывая особенности архитектуры конкретного ГПУ. В разделе «Результаты и направления дальнейшей работы» даются результаты работы и направления, в котором в дальнейшем будет развиваться система.

2. Особенности архитектуры графических процессоров

Архитектура современных графических процессоров определяется их основной задачей — возможностью создавать реалистичные изображения в реальном масштабе времени. Традиционно архитектура графических процессоров изображается в виде графического конвейера, изображенного на Рис. 1.

Примитивы для обработки (в виде вершин, индексных буферов, задающих принадлежность вершин, а также дополнительных атрибутов вершин, таких как текстурные координаты или нормали) задаются при помощи интерфейса трехмерного программирования (например, OpenGL). После этого они передаются на графический процессор, где отдельные вершины обрабатываются при помощи вершинных шейдеров. Шейдер — это программа для графического процессора. Вершинные шейдеры получают на вход атрибуты вершины и выдают новые атрибуты вершины; чаще всего они преобразовывают координаты и нормаль в соответствии с видовой матрицей. На этапе сборки примитивов из вершин собираются прими-

тивы (на более современных ГПУ этот этап заменен геометрическим шейдером — который позволяет осуществлять более гибкий набор операций с примитивами, например, разбиение). Растеризация преобразует примитив (чаще всего это треугольник) в набор фрагментов — проекций этого треугольника на пиксели экрана. Для каждого фрагмента выполняется фрагментный шейдер. Основная его задача — это вычисление цвета фрагмента, и, возможно, его глубины. После этого фрагмент подается на заключительный этап — пиксельных операций. Наиболее важные из пиксельных операций — это тест глубины (выполняет аппаратное удаление невидимых поверхностей; очень эффективно реализован в большинстве современных ГПУ) и тест трафарета. Следует заметить, что если фрагментный шейдер не вычисляет значение глубины, то отсечение фрагмента по глубине происходит до работы фрагментного шейдера для этого фрагмента, экономя тем самым вычислительные ресурсы ГПУ и предоставляя довольно эффективный механизм для маскирования неактивных фрагментов в ОВГПУ.

Более подробно о графическом конвейере см., например, в [16]. Здесь же отметим только, что наибольшее значение с точки зрения ОВГПУ имеют фрагментные (или пиксельные) шейдеры. Это обусловлено рядом особенностей архитектуры графического процессора:

1. Фрагментные шейдеры позволяют осуществлять произвольный доступ к определенным областям памяти (при помощи текстур), в то время как вершинные шейдеры этого не могут.
2. Результат работы фрагментных шейдеров доступен непосредственно, в то время как результат работы вершинных шейдеров доступен только после интерполяции.
3. Фрагментные шейдеры являются узким местом работы большинства современных приложений трехмерной визуализации, поэтому архитектура графических процессоров разрабатывается прежде всего для оптимальной работы фрагментных шейдеров. Как следствие, для них обычно выделяется больше шейдерных функциональных устройств (ФУ), и они обладают более гибкими возможностями, чем вершинные шейдеры. В последних графических процессорах с унифицированной шей-

дерной архитектурой функциональные устройства разделяются между различными видами шейдеров, поэтому их возможности примерно одинаковы.

В настоящее время на рынке производства дискретных, т.е. не интегрированных в системную плату, ГПУ доминируют две компании — AMD (ATI) и NVIDIA. И если архитектура ГПУ предыдущего поколения (ATI X1K и NVIDIA GeForce 7) у них во многом была похожей, то текущее поколение их архитектур (ATI HD 2K и NVIDIA GeForce 8) значительно отличаются. И хотя графические процессоры обоих производителей поддерживают работу через интерфейсы программирования трехмерной графики, внутренние реализации их отличаются настолько, что для каждой приходится писать свою эффективную реализацию одного и того же фрагментного шейдера. Ниже сначала дается описание общих характеристик современных ГПУ, а затем приводится описание особенностей различных архитектур графических процессоров.

2.1. Общие характеристики современных ГПУ. Для эффективной работы фрагментных шейдеров современные графические процессоры имеют большое количество шейдерных ФУ. Их число лежит в диапазоне от 4 до 320, причем сами ФУ могут быть скалярными или работающими с 4-ками вещественных чисел. Каждое шейдерное ФУ имеет не менее 32 регистров и работает с вещественными числами одинарной точности; самые новые ГПУ поддерживают также работу с целыми числами. Поток управления либо один на все шейдерные ФУ, либо один на группу из достаточно большого (до 8) числа шейдерных ФУ (в этом случае программа у всех одна и та же). Таким образом, современные ГПУ являются параллельной архитектурой типа ОКМД.

Все ФУ выполняют один и тот же шейдер для всех целочисленных пар координат заданного прямоугольника на плоскости со сторонами, параллельными осям координат, и результат исполнения для каждого элемента записывается в буфер видеопамяти. Современные ГПУ позволяют записывать эти данные в промежуточный буфер без вывода на экран, равно как и поддерживать вывод в 8 буферов одновременно. Шейдер может быть сменен только после того, как текущий шейдер проработает для всех элементов прямоугольника; смена шейдера считается дорогой операцией. Для каждого ис-

полнения шейдера координаты его выхода жестко заданы и не могут быть изменены, что позволяет оптимизировать запись результатов в видеопамять. Исполнения шейдера для различных пикселей полностью логически независимы, могут выполняться параллельно, и не существует никаких специальных средств синхронизации между исполнениями шейдера для различных пикселей. Современные ГПУ предоставляют альтернативный режим работы, в котором запись может быть осуществлена в произвольное место выходного буфера, однако не предоставляют никаких средств синхронизации подобной записи. В этом случае, однако, кэширование записи не производится, и для достижения высокой производительности приходится моделировать режим регулярной записи, только с большим количеством записываемых элементов.

Каждое ФУ может выдавать команды на чтение данных из оперативной памяти. Чтения полностью асинхронны и исполняются на специальных устройствах, именуемых текстурными устройствами. В зависимости от типа и режима работы, доступна либо вся память, либо набор из не более чем 32 сегментов (буферов). В последнем случае загрузка сегментных регистров производится до выполнения шейдера, и во время его выполнения они изменены быть не могут. Адресация может быть либо линейной, либо двумерной. Чтение может быть кэшированным или некэшированным (см. описания ГПУ от конкретных производителей).

Области памяти, адресуемые сегментными регистрами, могут располагаться как в собственной памяти графического процессора (видеоОЗУ), так и в ОЗУ. ВидеоОЗУ современных ГПУ имеет размер от 128 до 1024 МБ. Доступ к видеоОЗУ осуществляется со скоростью от 25 до 80 ГБ/сек, доступ к ОЗУ ЦПУ на порядок медленнее. Выгрузка данных обратно из видеоОЗУ в ОЗУ ЦПУ еще медленнее и осуществляется на скорости до 1 ГБ/сек. Таким образом, обработку данных следует организовывать таким образом, чтобы программы работали только с данными в видеоОЗУ.

При программировании ГПУ операции доступа к памяти считаются медленными операциями: обмен данными с памятью может занимать до 400 тактов, в то время как за 1 такт одно ФУ может выполнить до 8 арифметических операций. Для скрытия расходов на взаимодействие с памятью используется аппаратная многопоточ-

ность. Потоки ГПУ намного менее гибки, чем традиционные потоки; каждый из них состоит из определенного числа одновременно обрабатываемых фрагментов (от 4 до 48), и хранит весь контекст внутри ГПУ, что позволяет переключать потоки без потери времени (реально — за 1 такт). Как следствие, количество потоков ограничено размером массива регистров общего назначения (РОН) внутри ГПУ и обратно пропорционально числу используемых в шейдере регистров. Еще одним следствием такого механизма является отсутствие программного контроля переключения потоков, так как все операции переключения осуществляются аппаратно с целью повышения быстродействия. Аппаратура выполняет переключение потоков лишь в том случае, если один из них заблокировался на операции с памятью.

За счет полной независимости исполнений шейдера для различных фрагментов ГПУ достигают достаточно большой степени параллелизма и значительной производительности в терминах вещественных вычислений. Например, графический процессор X1950 XTX с 48 шейдерными ФУ имеет пиковую производительность 300 ГФлопс при вещественных вычислениях с одинарной точностью, что сравнимо с производительностью современных многомашинных комплексов и на порядок превосходит производительность самого мощного современного центрального процессора.

2.2. ГПУ компании AMD (ATI). Общая схема архитектуры графического процессора компании AMD приведена на рисунке 2. Данная схема описывает как ГПУ серии X1K, так и ГПУ серии HD 2K. Хотя HD 2K на уровне шейдерных ФУ имеет иную архитектуру, сходство между семействами значительно больше, чем различий.

ГПУ семейства X1K имеют от 4 до 48 шейдерных ФУ, каждое из которых обрабатывает четверки вещественных чисел (ОКМД ФУ). ФУ способны выполнять вещественные арифметические операции, а также вычисление элементарных функций (только для скалярных величин) и команды управления; кроме того, они могут выдавать запросы на чтение/запись контроллеру памяти. Каждому исполнению шейдера доступны до 128 128-битных вещественных регистров. Чтение данных может производиться из 16 двумерных буферов размера до 4096×4096 элементов, расположенных в видеоОЗУ или ОЗУ. Каждый элемент может быть либо вещественным числом, либо чет-

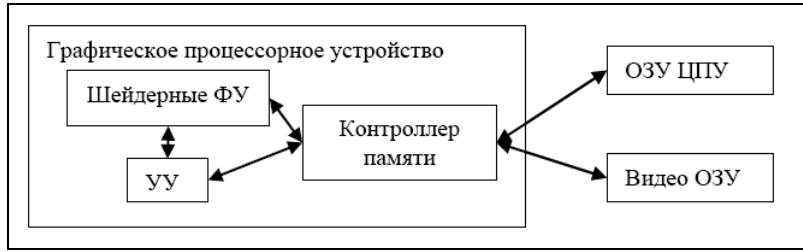


Рис. 2. Архитектура графического процессора компании AMD

веркой вещественных чисел; в качестве буфера глубины поддерживаются буфера из 16-битных беззнаковых целых чисел. Запись возможна в один из 4-х целевых буферов, также поддерживается режим произвольной записи. Чтение и запись кэшируются; однако даже в случае попадания в кэш стоимость одного чтения может составлять до 4-х тактов, что делает понижение общего количества доступов в память важной задачей оптимизации.

ГПУ семейства HD 2K [17] имеют от 4 до 64 шейдерных ФУ, построенных по архитектуре с длинным командным словом. Каждое шейдерное ФУ имеет в себе 5 подустройств, каждое из которых за такт способно выполнить 2 арифметические команды с вещественным числом. Помимо этого, 5-ое ФУ также может вычислять элементарные функции и выполнять специальные команды. Поддерживаются команды для работы с целыми числами, а также команды управления. Количество регистров осталось прежним, однако появилась возможность косвенной адресации как самих регистров, так и номера текущего буфера чтения. Размеры буферов увеличились до 8192×8192 , число входных буферов — до 32, а число выходных буферов — до 8. Появились новые возможности, поддерживающие интерфейс программирования трехмерной графики DirectX 10. Однако в остальном архитектура изменилась мало, и длинное командное слово, хотя и является более гибким, чаще используется для работы с теми же четверками вещественных чисел.

2.3. ГПУ компании NVIDIA. В отличие от AMD, шейдерные ФУ в ГПУ серии GeForce 8 от NVIDIA являются суперскалярными, а не ОКМД, и работают они с отдельными вещественными и

целыми числами, а не с четверками чисел. С одной стороны, такое аппаратное решение позволяет повысить тактовую частоту ФУ. С другой стороны, это облегчает написание шейдеров для работы на таких процессорах, и как следствие — упрощает программирование ФУ. Еще одним плюсом, не относящимся напрямую к архитектуре, является возможность использовать диалект С CUDA для программирования ГПУ NVIDIA. И хотя программирование ведется по старой схеме загрузки данных и исполнения шейдеров, написание шейдеров на языке С значительно облегчает задачу.

Второй особенностью ГПУ NVIDIA является замена кэш-памяти на явно адресуемую пользователем статическую память, скорость обращения к которой примерно соответствует скорости обращения к регистру. Подобное решение также применяется, например, в процессоре CELL [18]. Работа с видеоОЗУ, таким образом, становится некэшируемой. С одной стороны, такое решение дает дополнительные возможности по оптимизации, но с другой — усложняет создание эффективных программ на подобных архитектурах.

Подводя итог по архитектурам, можно сказать, что задача создания эффективных средств построения высокоуровневых и переносимых программ на ГПУ является, с одной стороны, актуальной, а с другой — достаточно сложной и интересной. Ведь при создании эффективного кода требуется учитывать различные особенности различных архитектур ГПУ — такие как статическую память, ОКМД ФУ и другие, при этом эти детали желательно по возможности скрыть от конечного пользователя, или предоставить в понятных ему терминах. Кроме того, возникает вопрос о переносимости подобных программ на другие архитектуры — например, на многоядерные процессоры, процессоры типа CELL и т.д., тем более что во многом эти архитектуры похожи и во многих случаях они используются для решения одних и тех же классов задач (например, обработка изображений).

3. Существующие подходы программирования

При рассмотрении ОВГПУ шейдер можно рассматривать как элемент программы графического процессора, а текстуры и буфера, с которыми он работает — просто как двумерные массивы данных. Таким образом, шейдер можно рассматривать как:

- как функцию («ядро»), которая перерабатывает порцию элементов входного массива, а на выходе дает элемент другого массива,
- как функцию, которая применяется к входным массивам и в результате дает выходной массив,

причем в обоих случаях эта функция не имеет побочного эффекта.

Традиционно для описания ГПУ применялся первый подход, который получил название «потоковое программирование» (streaming programming) [19]. Идея этого подхода состоит в следующем. Программа разбивается на ядра, т.е. небольшие программные элементы со строго определенными входами и выходами. Ядра обрабатывают потоки входных элементов. Одно ядро может иметь один или несколько потоков входных элементов. На каждый входной элемент ядро генерирует столько-то выходных элементов. При отображении на графический процессор потоки отображаются на текстуры, а ядра — на шейдеры. Для ядер с условной генерацией выходных элементов используются возможности маскирования ГПУ за счет буфера глубины. Доступ к нескольким элементам одного и того же потока осуществляется при помощи операций типа scatter и gather; при этом операция gather отображается в чтение соседних элементов текстуры, а операция scatter — в дополнительный проход, в котором результаты работы фрагментного шейдера подаются на вход вершинному шейдеру (который может изменять позицию записи).

Этот подход хорошо работал, когда размеры шейдеров были ограничены (не более 20 команд), и накладывались ограничения на чтение из текстуры. Это подходило для простых алгоритмов; для более сложных алгоритмов и для более сложных шейдеров (например, с циклами), подход не работает. Например, для того, чтобы сформулировать в нем алгоритм трассировки лучей, его приходится разбивать на элементарные операции [20]. Непонятно в том числе и то, как в терминах таких операций выразить, например, операцию умножения матриц.

Поэтому со временем более популярным стал второй подход — шейдер есть функция, которая применяется к набору входных массивов и выдает выходной массив. Эта функция не имеет побочного

эффекта и может быть вычислена независимо (как следствие, параллельно) для различных элементов выходного массива.

Но как определить такую функцию? Самый простой способ заключается в следующем: есть элементарные операции с массивами (например, сложение, умножение и другие), они комбинируются, и полученное арифметическое выражение рассматривается как функция. Поскольку смена шейдера влечет за собой накладные расходы, появляется стремление поместить как можно больше операций обработки в один шейдер. Вычисления массивных операций в таком случае осуществляется ленивым образом: пока результат явно не нужен, для него строится дерево, а когда результат реально требуется (например, он приводится к типу языкового массива из специального класса, реализующего массивы на ГПУ), то по дереву строится шейдер, он компилируется, исполняется, а результат загружается обратно.

По такому принципу устроены библиотеки Accelerator [21], Peak Stream [22] и Rapid Mind [23]. По сути, они предлагают некоторую библиотеку массивных операций над вещественными числами, которая использует ленивые вычисления и динамическую компиляцию. Кроме того, подобные библиотеки можно делать переносимыми между различными системами — например, Rapid Mind работает также на CELL [18] и на многоядерных системах.

4. Система C\$: идеи и архитектура

Система C\$ [24] заимствует идеи ленивого исполнения и динамической трансляции у уже существующих систем, и одновременно предлагает целый ряд нововведений.

Во-первых, для программирования ГПУ предлагается использовать не библиотеку, а язык программирования. При этом, хотя на архитектуру языка и оказала влияние архитектура современных ГПУ, сам язык является машинно-независимым и в принципе может быть перенесен на другие архитектуры, в том числе CELL и многоядерные системы. Синтаксис нового языка и большинство конструкций заимствованы из языка C# (и Java), что должно облегчить освоение. А разработка языка с использованием среды исполнения .NET облегчает интеграцию его в уже существующие системы. Специфические объекты языка (такие как функции или массивы) в других

.NET-языках, таких как C#, будут видны просто как классы, а специфические языковые конструкции — как вызовы методов.

Процесс вычисления на графическом процессоре можно представить как *параллельное независимое вычисление функции без побочных эффектов на ее области определения*. Ведь один вызов шейдера является вычислением функции (шейдера) на его области определения (экранном прямоугольнике). Функция может быть достаточно простой, как в случае сложения двух массивов, или, наоборот, достаточно сложной, как в случае трассировки лучей [7]. Кроме того, для вычисления одного элемента выходного массива может использоваться много элементов входного массива — например, целая строка или столбец, как в случае умножения матриц.

Поэтому в язык C\$ введено как понятие функции без побочного эффекта, так и понятие типа функции без побочного эффекта, или функционального типа. Сам по себе функциональный тип является абстрактным классом. Поэтому значением переменной функционального типа может быть ссылка на объект одного из производных классов: массив, метод класса, элементарная операция или уже сконструированное, но еще не вычисленное выражение. Любое такое значение называется *функциональным объектом*. Таким образом, массив в языке C\$ представляет собой частный случай функции.

Программа на языке C\$ состоит из этапов, на каждом из которых выполняется построение такой функции. Параллельное вычисление осуществляется за счет вычисления функции на всей ее области определения.

Основной операцией, используемой для конструирования новых функций в C\$, является операция суперпозиции функций. В отличие от других языков, где для суперпозиции используется специальная функция или обозначение, в C\$ суперпозиция осуществляется за счет конструкции функционального продвижения. Она работает следующим образом: пусть имеется функция h типа $T5(T1, T3)$, т.е. функция h принимает на вход одно значение типа $T1$ и одно значение типа $T3$, и возвращает значение типа $T5$. Пусть также имеются функции g типа $T3(T4)$ и f типа $T1(T2)$. Тогда запись $h(f, g)$ в языке C\$ задает функция типа $T5(T2, T4)$ (если $T2$ и $T4$ - различные типы) или $T5(T2)$ (если это — один и тот же тип). При этом значение этой функции может быть вычислено следующим образом: функция пе-

редает свои аргументы в функции *f* и *g*, результаты их работы, в свою очередь, передает в функцию *h*, после чего возвращает значение, полученное из *h*.

Функциональное продвижение используется как дополнительное средство разрешения перегрузки функций — когда прочие средства, такие как приведение типов, не могут найти подходящую функцию.

Таким образом, любая функция может применяться не только к скалярным значениям, но и к другим функциям — становясь функцией более высокого порядка (в терминах функционального программирования).

Поскольку массивы в языке C\$ являются частным случаем функций, за счет суперпозиции к массивам можно применять арифметические операции и скалярные функции. Таким образом, в языке C\$ появляются возможности массивно-ориентированного программирования.

Пример использования функционального продвижения в языке C\$ приведен в листинге 1.

```
float sin(float x) { ... } // функция синус
float[] g, l; // целочисленный массив
float(float) m; // функция float -> float; может быть как методом,
так и ассоциативным массивом
...
var h = g + sin; // var выполняет автоматический вывод типа
переменной; тип float(int, float)
var t = 1 * g; // тип float (int)
var w = m + sin; // тип float (float)
```

Листинг 1. Пример функционального продвижения в C\$

Элементы объектно-ориентированного и функционального программирования в языке C\$ существуют. При этом обращения к полям или к свойствам трактуются как функции, которые принимают один параметр (объект класса) и возвращают одно значение (объект типа поля класса). Как следствие, операция «*.*» участвует в суперпозиции функций и может применяться к функциям. Пример этого приведен в листинге 2.

```
final class float3 {float x, y, z;} // класс трехмерного вектора
```

```
float3 [] vs; // массив вершин
var tt = vs.x; // функция, которая принимает на вход целое число, а возвращает элементы x векторов
var ll = tt * vs.y + g; // см. выше; тип float (int)
Листинг 2. Пример функционального преобразования с использованием доступа к члену в C$
```

В языке C\$ вводится понятие **простого неизменяемого класса** — это класс, от которого нельзя наследовать, поля которого нельзя изменять после создания объекта класса и ссылка на который не может быть `null`. Также требуется, чтобы поля объекта этого класса не могли ссылаться на объекты этого же класса, прямо или косвенно. Такие классы могут использоваться совместно с функциями без побочных эффектов. Кроме того, массивы объектов таких классов можно хранить не как массивы ссылок, а как массивы самих объектов. Для объявления такого класса перед ключевым словом `class` нужно поставить модификатор `final`. Если при этом требуется, чтобы класс был обычным ненаследуемым, а не простым неизменяемым, то перед ним нужно поставить модификатор `mutable`.

Идея введения подобного класса не нова — в .NET, например, уже существуют типы-структуры (в противовес типам-классам), хотя они не являются неизменяемыми. Идея неизменяемых классов, состоящих только из простых типов, также используется в языках X10 [25] и Titanium [26], и для тех же целей — обеспечить возможность хранения массивов объектов как массивов самих объектов, а не массивов ссылок.

В языке также есть возможность вводить классы, которые наследуют от функциональных типов. Путем переопределения метода вычисления функции можно, например, связать такой класс с файлом — тогда собственно чтение из файла не произойдет до того момента, как эта функция будет достигнута при ленивом вычислении. Если операция чтения дополнительно переопределена, например, для случая, когда на вход подается массив подряд идущих индексов, это позволит выполнять ее более эффективно и прочитать только ту часть файла, которая действительно необходима.

Операция редукции реализована как применение двуместной функции с типом $T \times T$ к функции, которая возвращает тип T . Если такая функция помечена как коммутативная или ассоциативная,

то при редукции может изменяться порядок вычислений. В дальнейшем возможно введение в язык операции обобщенной редукции — любой операции, которая берет на вход массив и возвращает скаляр, при этом может быть вычислена в цикле за время $O(n)$ (n — размер массива) с $O(1)$ памяти.

С точки зрения компилятора, редукция, как и суперпозиция, является способом разрешения перегрузки функций.

В языке C\$ любой функциональный объект имеет ряд атрибутов — одним из них является домен, или область определения. В качестве домена может выступать любая функция (хотя чаще всего это декартово произведение целочисленных промежутков). Операция редукции может быть выполнена над функцией только в том случае, функция имеет конечный домен — в противном случае генерируется ошибка. Проверка конечности производится на этапе выполнения.

Для того, чтобы показать, что при вычислении 2 измерения массива должны пробегаться параллельно, а не независимо, может быть использована конструкция, называемая *связанной переменной*. Связанная переменная — это специальная переменная, которая имеет тип, но не имеет определенного значения; она связывает измерения (параметры) различных функций. Область, пробегаемая переменной, вычисляется исходя из доменов функций, в которых она применяется, и явных ограничений на нее (вида $i:f$, где i — связанная переменная, а f — функция). Связанные переменные не обязательно объявлять явно — если они не объявлены, то их тип выводится из их первого применения. Если связанная переменная присутствует только внутри редукционной конструкции, то по ней осуществляется редукция. Пример использования операции редукции и связанных переменных для умножения матриц приведен в листинге 3.

```
type matrix = float(int, int); // псевдоним типа
matrix mul(matrix a, matrix b) {
    var c(j, k) = + (a(j, l) * b(l, k));
    return c;
}
```

Листинг 3. Пример умножения матриц в языке C\$

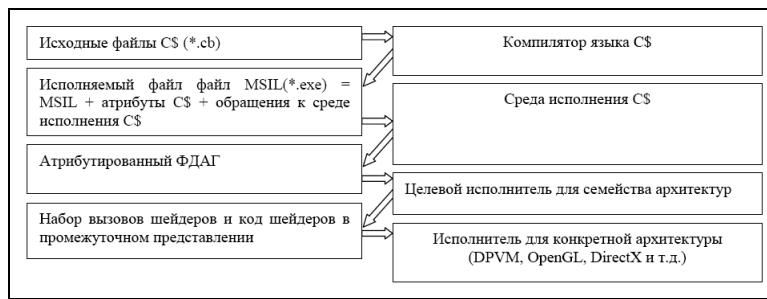


Рис. 3. Архитектура графического процессора компании AMD

В данном случае унарный + выполняет редукцию (в C\$ нет стандартного унарного оператора +) по связанной переменной l, а переменные j и k остаются, т.к. они объявлены вне оператора редукции. Для введения какой-либо переменной в выражение (функцию) может использоваться конструкция типа let. Для удобства введены операции sum() вместо редукционного + и prod() вместо редукционного *. Кроме того, в языке в качестве операций представлены min и max, которые также используются как редукционные.

В терминах императивного программирования связанные переменные представляют собой цикл без заголовка; заголовок им не нужен, поскольку область определения связанной переменной вычисляется автоматически. В терминах функционального программирования связанные переменные аналогичны comprehension-конструкциям — с той разницей, что в C\$ нет специальных списочных типов.

Использование связанных переменных позволяет программировать задачи с большей вычислительной мощностью, а значит, более полно использовать возможности графического процессора. Например, операция сложения двух больших массивов, скорее всего, не позволит достигнуть более 2% пиковой производительности ГПУ, т.к. она имеет малую вычислительную мощность. С другой стороны, операция умножения матриц позволяет выполнить $O(n^3)$ вычислительных операций на $O(n^2)$ входных данных, что позволит достигать до 50–60% пиковой производительности ГПУ.

Общая архитектура системы C\$ изображена на рисунке 3.

Компилятор языка преобразует языковые конструкции в вызовы к среде исполнения C\$. При этом исходный код делится на обычный и параллельный в терминах C\$. Код считается параллельным в терминах C\$, если он содержит использование функционального движения, редукции или связанных переменных. Таким образом, решается задача выделения параллельных элементов программы, которые можно отображать в код для ГПУ. Операция вычисления функции на всем домене выражается как приведение функции к массивному типу (т.е. типу с квадратными скобками). Исходный код на языке C\$ транслируется в промежуточное представление — в настоящее время это специальное разработанное нами представление, однако планируется для этой цели использовать Microsoft Intermediate Language (MSIL) — представление промежуточного кода в .NET.

Обращения к среде выполнения C\$ создают направленный ациклический граф (ДАГ) функциональных операций. Он также называется функциональным ДАГ (ФДАГ). В настоящее время этот граф имеет 3 типа листовых вершин: скаляры, функции, связанные переменные (в т.ч. и пустые), и 3 типа внутренних вершин: применения функции, редукции и введения новых связанных переменных.

После машинно-независимых оптимизирующих преобразований ФДАГ передается целевому исполнителю для семейства архитектур, например, для ГПУ. На этом этапе выбираются способы хранения данных и отображения кода на целевую архитектуру, а также производится основной набор машинно-зависимых оптимизаций. Сама программа из ФДАГ переводится в набор шейдеров на промежуточном языке. И хотя этот набор шейдеров еще не является программой, исполняемой на аппаратуре, он настолько близок к ней, что работа, выполняемая на следующем этапе (исполнителя для конкретной архитектуры) является скорее механической.

Наконец, самым нижним уровнем нашей системы является исполнитель для конкретной архитектуры. Основной его задачей является предоставление стандартного интерфейса уровню семейства архитектур. Этот интерфейс включает в себя механизм управления памятью, механизм трансляции кода из промежуточного представления в машинный, и механизм сообщения о возможностях устройства. К последним относится, например, максимальное количество доступных для чтения или записи буферов, максимальное количе-

ство доступных регистров и другие особенности.

Работа с памятью тоже распределена по уровням системы, как и работа с трансляцией программы. Уровень системы времени выполнения работает с функциями и прямоугольными массивами — например, с массивами вещественных чисел. Он знает ранг и домен массива, но ничего не знает о том, как он хранится в памяти. Наиболее важным уровнем является уровень исполнителя для семейства архитектур. Его задача состоит в том, чтобы эффективно представить исходный массив произвольной размерности в памяти устройства для организации в дальнейшем его эффективной обработки. На этом уровне, например, решаются вопросы отображения исходных массивов в массивы размерности, поддерживаемой аппаратурой, например, размерности 2, как на большинстве ГПУ. Кроме того, решается задача синхронизации данных; это позволяет минимизировать количество дорогостоящих операций обмена между ОЗУ и видеоОЗУ. Наконец, уровень исполнителя для конкретной архитектуры решает задачу управления физической памятью.

5. Трансляция операций над массивами в системе C\$

Трансляция массивных операций осуществляется в 3 этапа: средой времени выполнения, целевым исполнителем для семейства и целевым исполнителем для конкретной архитектуры. При этом среда времени выполнения осуществляет машинно-независимую оптимизацию — такую как удаление повторяющихся выражений и свертку констант. Также она определяет списки измерений в каждой вершине ФДАГ. Самое важное, на этом этапе для каждой вершины графа вычисляется ее домен. Впоследствии он используется для выделения память под целевую функцию и определения области ее вычисления.

Размеченный таким образом граф поступает на вход целевому исполнителю для семейства архитектур ГПУ, который преобразует его в последовательность шейдеров. Его работа состоит из нескольких этапов:

1. Деление исходного графа на проходы. Каждый проход соответствует исполнению одного шейдера.

2. Выбор отображения данных и кода на целевую архитектуру. Здесь выбирается схема представления данных в физической памяти устройства, а способ физической реализации массивных операций.
3. Генерация индексов для операций чтения из памяти. Поскольку при этом должны учитываться отображения исходного и целевого массива, а также промежуточных операций, эта генерация вынесена в отдельный этап.
4. Генерация шейдерного кода в промежуточном представлении по ФДАГ.

При этом массивные операции отображаются на арифметические операции в шейдере, а операции редукции — либо на редукцию вдоль измерений буфера памяти, если ее результатом является скаляр, либо на цикл внутри шейдера, если результатом ее является другая функция.

Сгенерированный шейдерный код подается на вход уровню исполнителя для конкретной архитектуры. Его задача — преобразовать код из этого представления в код, пригодный для исполнения на целевом ГПУ.

На рис. 4 приведено дерево, которое строится по коду умножения матриц, приведенному в листинге 3. Используемые типы вершин: применения функции \otimes , редукции R . При этом в скобках у вершин редукции указаны связанные переменные, по которым идет редукция. Связанные переменные и массивы помечены буквами. На рис. 5 приведен код, полученный из этого графа выражений для умножения матриц размером 1024 на 1024. Обратите внимание, что код был сгенерирован с учетом того, что ГПУ работает с четверками вещественных чисел. Операция редукции была автоматически преобразована в двойной цикл, т.к. максимальное число итераций цикла на ГПУ (255) меньше, чем половина размера матрицы. Матрица представлена буфером из 512×512 четверок вещественных чисел, при этом каждая из четверок — это подматрица размером 2×2 . Следует сказать, что более эффективный (в 2 раза) код, генерируемый нашей системой, еще в два раза длиннее, и не приведен здесь по причине экономии места.

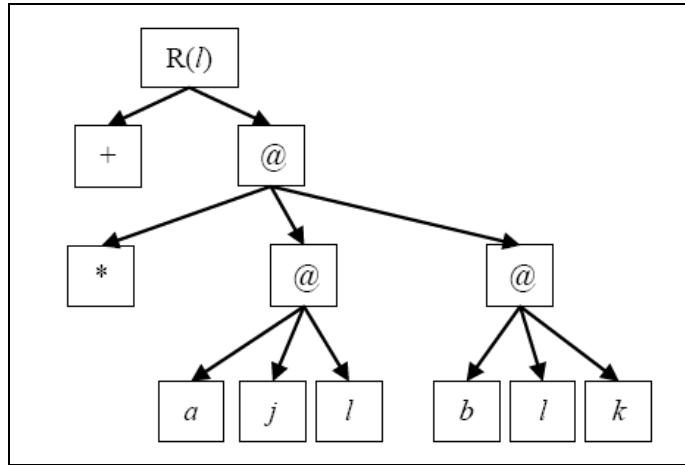


Рис. 4. ДАГ операций, построенный по коду перемножения матриц

6. Результаты и направления дальнейшей работы

Часть системы C\$ была реализована на языке C# 2.0 для работы в среде выполнения .NET 2.0. Доступ к графическому процессору осуществлялся средствами ATI DPVM [14]. Текущая реализация системы имеет следующие ограничения по сравнению с описанными выше идеями:

- Поддерживаются только простые функции (т.е. операции с простыми типами — float, int) и массивы.
- Поддерживаются только домены, представляющие собой декартово произведение целочисленных промежутков, либо полные домены.
- Трансляция осуществляется в специфическое промежуточное представление, а не в код MSIL. Это, однако, планируется исправить.
- Не поддерживаются вершины введения дополнительных переменных (let-вершины). Однако поддерживаются редукционные вершины и вершины суперпозиции.

```

ps_3_0
dcl vPos.xy
dcl_2d s0
dcl_2d s1
mov r1.x, vPos.x
mov r1.y, c1.x
mov r2.x, c1.x
mov r2.y, vPos.y
mov r0, c0.zzzz
rep il
rep i0
texld r3, r1, s0
texld r4, r2, s1
add r1.y, r1.y, c0.y
add r2.x, r2.x, c0.y
mul r5, r3.xxzz, r4.xyxy
mad r6, r3.yww, r4.zwzw, r5
add r0, r0, r6
endrep
endrep
mov oC0, r0

```

Рис. 5. Шейдерный код, сгенерированный по ДАГ на рис. 4

В остальном же функциональность текущей системы реализована в соответствии с высказанными выше идеями.

Система была протестирована на коде умножения матриц, приведенном в листинге 3. Тестирование проводилось на матрицах различных размеров и с различными настройками оптимизации. В таблице 1 приведены результаты (производительность в Гфлопс) для различных настроек оптимизации. При этом матрица представлялась в виде простого двумерного массива (1×1), в виде массива подматриц 2×2 и массива подматриц 4×4 . Кроме того, для сравнения приводится производительность ручной реализации процедуры sgemm на ассемблере из примеров AMD для DPVM.

Размер матрицы	128×128	256×256	512×512	1024×1024
Настройки оптимизации				
float, 1×1	3.94	5.00	5.16	4.61
float4, 2×2	8.10	14.20	15.66	15.81
float4, 4×4	9.03	24.13	29.16	29.35
ручная оптимизация	—	—	30	31

Табл. 1. Производительность программы умножения матриц на матрицах различных размеров при различных настройках оптимизации

На основании результатов, приведенных в таблице 1, можно сделать ряд выводов. Во-первых, с увеличением размера матрицы производительность вычислений растет. Это можно объяснить двумя причинами. С одной стороны, запуск шейдера на ГПУ требует фиксированных накладных расходов, таких как коммуникации между ЦПУ и ГПУ, установки регистров данных и команд, и других. С другой стороны, и это более важно, в самом шейдере есть участок начальной инициализации (инициализация индексов, переменных ре-дукции) и циклический участок. С ростом размера матрицы растет доля циклического участка в общем числе команд, а значит, возрастает производительность.

Во-вторых, оптимизации, связанные с повышением вычислительной мощности генерированного кода, приносят значительный выигрыш. Трехкратный рост производительности наблюдается при переходе от тривиального алгоритма к алгоритму с блоками 2×2 . Двукратный рост наблюдается при переходе к блокам размера 4×4 . Следует заметить, что максимальный размер блока ограничен общим числом выходных элементов, поэтому увеличивать размер блока дальше нет возможности. При этом рост производительности при переходе к блокам 2×2 обусловлен как повышением отношения количества арифметических операций к количеству чтений, так и повышением эффективности использования арифметических операций. Рост же при переходе от 2×2 к 4×4 обусловлен только повы-

шением отношения количества арифметических операций к чтению из памяти, потому он и меньше.

Результаты, приведенные в таблице 1, уже сами по себе неплохи. В лучшем случае достигнута примерно половина пиковой производительности, и, что самое важное - почти 100% производительности, достигнутой при помощи ручной оптимизации. Дальнейшее развитие системы может идти по нескольким направлениям.

Во-первых, это расширение гибкости системы C\$. Прежде всего это возможность использовать определенные пользователем функции, а также простые неизменяемые классы. Это позволит применять систему для решения более широкого класса задач.

Во-вторых, это расширение набора целевых архитектур, на которых система C\$ сможет работать. Помимо поддержки OpenGL и DirectX, которые хороши прежде всего возможностью работать с практически любыми ГПУ, большой интерес представляют процессоры семейства NVIDIA GeForce 8. Для их эффективного программирования необходимо разработать методы работы с промежуточной статической памятью. В более далекой перспективе интересны многоядерные системы и процессоры CELL. Кроме этого, потребуется более широкий набор оптимизаций для нового поколения ГПУ AMD.

В-третьих, это расширение системы C\$ на целевые архитектуры с несколькими графическими процессорами. Эта задача становится актуальной с возрастанием распространенности подобных систем, созданных на основе ГПУ от AMD и NVIDIA. Кроме того, специализированные решения для высокопроизводительных вычислений от этих компаний чаще всего оснащаются ГПУ от AMD и NVIDIA.

Список литературы

- [1] GPGPU.org. <http://www.gpgpu.org/>.
- [2] Fatahalian, K., Sugerman, J. и Hanrahan, P. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication// ACM Press, 2004. Р. 133–137. ISBN 1727-3471, 3-905673-15-0.

- [3] Bolz, Jeff, etc. Sparse matrix solvers on the GPU: conjugate gradients and multigrid// ACM Press, 2005. ACM SIGGRAPH 2005.
- [4] Goodnight, Nolan, etc. A multigrid solver for boundary value problems using programmable graphics hardware// Proceedings of SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware. P.102–111. ISBN:1727-3471, 1-58113-739-7.
- [5] Farrugia, J.P., etc. GPUCV: A framework for image processing acceleration with graphics processors // Proceedings of IEEE International Conference on Multimedia & Expo (ICME 2006).
- [6] Mitchel, Jason L., Ansari, Marvan Y., Hart, Evan. Advanced Image Processing with DirectX 9 Pixel Shaders. ATI Technologies Inc. Shader X2. 2004.
- [7] Adinetz, Andrew V., Berezin, Sergey B. Implementing Classical Ray Tracing on GPU - a Case Study of GPU Programming// Proceedings of Graphicon'06. P.1–7. ISBN 5-89407-262-X.
- [8] Belleman, Robert G., Bedorf, Jeroen, Portegies Zwart, Simon F. High Performance Direct Gravitational N-Body Simulations on Graphics Processing Units. Elsevier Preprint. 16 июля 2007 г.
- [9] Wikipedia. Fortran. <http://en.wikipedia.org/wiki/Fortran>.
- [10] OpenGL.org. <http://www.opengl.org>.
- [11] Kessenich, John, Baldwin, Dave, Rost, Randi. The OpenGL Shading Language. 7 September 2006.
- [12] AMD Inc. ATI Web Site. <http://ati.amd.com/>.
- [13] NVIDIA Corporation. <http://www.nvidia.com/page/home.html>.
- [14] Peercy, Mark, Segal, Mark, Gerstmann, Derek. A performance-oriented data parallel virtual machine for GPUs// Proceedings of ACM SIGGRAPH 2006 Sketches. ISBN: 1-59593-364-6.
- [15] NVIDIA Corporation. NVIDIA CUDA Complete Unified Device Architecture. 12 February 2007.

- [16] Segal, Mark, Akeley, Kurt. The OpenGL (R) Graphics System: A Specification (Version 2.1). Edited by Pat Brown. 1 December 2006.
- [17] Persson, Emil. ATI Radeon TM HD 2000 Programming Guide. June 2007.
- [18] IBM Corporation. Cell Broadband Engine Architecture. 3 October 2006.
- [19] Buck, Ian, etc. Brook for GPUs: stream computing on graphics hardware// ACM Press, 2004. P.777–786.
- [20] Foley, Tim, Sugerman, Jeremy. KD-tree acceleration structures for a GPU raytracer// Proceedings of SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware. P.15–22. ISBN:1-59593-086-8.
- [21] Tarditi, David, Puri, Sidd, Oglesby, Jose. Accelerator: using data parallelism to program GPUs for general-purpose uses// Proceedings of the 12th international conference on Architectural support for programming languages and operating systems. P.325–335. SESSION: Embedded and special-purpose systems.
- [22] PeakStream Inc. <http://www.peakstreaminc.com/>.
- [23] Rapid Mind Inc. <http://www.rapidmind.net/>.
- [24] Adinetz, Andrew V. C\$ Project Web Site. <http://www.codeplex.com/cbucks>.
- [25] Saraswat, Vijay. Report on the Experimental Language X10. 8 декабря 2006.
- [26] Bonachea, Dan, etc. Titanium Language Reference Manual. Berkeley, California, USA. Август 2006 г.

Влияние характеристик программно-аппаратной среды на производительность приложений

A. С. Антонов*

Создаваемый в НИВЦ МГУ имени М.В.Ломоносова процессорный полигон позволяет исследовать влияние базовых характеристик программно-аппаратной среды на производительность пользовательских приложений. Такой анализ необходим для получения на реальных программах максимально возможной производительности.

1. Введение

Практика показывает, что для реальных приложений, написанных на языке высокого уровня, весьма трудно получить производительность процессорного ядра, превышающую 15-20% от пиковой. Причём это касается всех наиболее распространённых на данный момент серийных микропроцессоров. Применяя достаточно сложные ухищрения или используя низкоуровневую оптимизацию, иногда удается существенно приблизиться к пиковым характеристикам [1], но это обычно неприменимо для реальных пользовательских задач.

Для получения высокой эффективности пользовательского приложения нужно учитывать большое количество факторов, влияющих на производительность конкретного процессора. Эти факторы определяются как внутренней архитектурой вычислительного узла, так и влиянием всего комплекса программного обеспечения. При этом всё множество факторов действует одновременно, в результате чего и получается столь существенное уменьшение производительности.

*Научно-исследовательский вычислительный центр МГУ

Исследовать влияние базовых характеристик программно-аппаратной среды на производительность приложений можно при наличии подходящей аппаратной базы, максимально полного комплекса системного программного обеспечения и разработанной системы тестов, позволяющей проводить комплексное исследование максимального количества факторов, отражающихся на производительности.

2. Аппаратная база и системное программное обеспечение

В НИВЦ МГУ имени М.В.Ломоносова создаётся уникальный процессорный полигон из вычислительных узлов на основе современных серийных микропроцессоров. К настоящему моменту он содержит 20 серверов. Это вычислительные сервера на базе:

- 5 серверов на базе процессоров AMD Opteron со следующими характеристиками:

Название узла	Тип процессора	Тактовая частота, ГГц	Число проц.	Число ядер на проц.	Объём ОП, ГБ	Пиковая производительность узла, GFlop/s
opteron1	Opteron 244	1.8	2	1	2	7.2
opteron2	Opteron 244	1.8	2	1	2	7.2
opteron3	Opteron 280	2.4	2	2	4	19.2
opteron4	Opteron 265	1.8	2	2	4	14.4
opteron5	Opteron 265	1.8	1	2	2	7.2

- 1 сервер на базе процессора IBM Power5 со следующими характеристиками:

Название узла	Тип процессора	Тактовая частота, ГГц	Число проц.	Число ядер на проц.	Объём ОП, ГБ	Пиковая производительность узла, GFlop/s
power1	POWER5	1.65	2	2	4	26.4

- 14 серверов на базе процессоров Intel со следующими характеристиками:

Название узла	Тип процессора	Тактовая частота, ГГц	Число проц.	Число ядер на проц.	Объём ОП, ГБ	Пиковая производительность узла, GFlop/s
dempsey1	Xeon 5050	3	1	2	4	12
xeon1	Xeon EM64T	3.2	2	1	4	12.8
pentiumd1	PentiumD 945	3.4	1	2	4	13.6
pentium41	Pentium 4 531	3.0	1	1	2	6
woodcrest1	Xeon 5150	2.66	1	2	16	21.28
woodcrest2	Xeon 5150	2.66	2	2	8	42.56
woodcrest3	Xeon 5150	2.66	1	2	4	21.28
woodcrest4	Xeon 5150	2.66	1	2	4	21.28
woodcrest5	Xeon 5160	3	2	2	10	48
woodcrest6	Xeon 5130	2	2	2	8	32
woodcrest7	Xeon 5130	2	2	2	8	32
clovertown1	Xeon 5310	1.6	2	4	16	51.2
clovertown2	Xeon 5345	2.33	1	4	6	37.28
clovertown3	Xeon 5355	2.66	1	4	6	42.56

Суммарная пиковая производительность серверов процессорного полигона составляет 475.44 GFlop/s, суммарный объём оперативной памяти 118 Гбайт. Сервера в процессорный полигон подбираются таким образом, чтобы отследить основные тенденции развития компьютерного рынка и предоставить возможность исследования максимального количества факторов, влияющих на производительность современных серийных микропроцессоров. Все сервера установлены в стойку, имеют форм-фактор 1–2U и доступны посредством системы очередей Cleo [3].

На узлах процессорного полигона установлена ОС Linux, дистрибутив SUSE 10.1, на узле power1 — AIX 5.3.

На всех узлах процессорного полигона (кроме узла power1) установлены следующие компиляторы:

- GNU (C, C++, Fortran 77/90/95) 4.1.2 (4.0.2);
- Intel Compilers (C, C++, Fortran 77/90/95) 9.1;
- Portland Group Inc. Compilers (C, C++, Fortran 77/90/95) 6.2-5;
- PathScale EKOPath Compiler Suite (C, C++, Fortran 90/95) 2.5;
- Absoft Fortran (Fortran77/90/95) 9.0.

Для корректного сравнения различных компиляторов, если не указано дополнительно, использовалась только опция стандартной

оптимизации -O3. Использованию более специфичных для каждого компилятора опций посвящено отдельное исследование.

Для задач, требующих параллельного исполнения, использовалась библиотека Intel MPI 3.0. Для компиляции теста HPL были установлены альтернативные библиотеки ATLAS 3.7.30, MKL 5.2 и GotoBLAS 1.11.

3. Комплексное тестирование вычислительных серверов

Для решения задач комплексного тестирования вычислительных серверов процессорного полигона предназначена система тестов, включающих как широко известные тесты, так и тесты, разработанные в НИВЦ МГУ имени М.В.Ломоносова.

HPL (High Performance Linpack) — стандартный тест, реализующий решение больших систем линейных алгебраических уравнений методом LU-разложения. Вычисления производятся с помощью вызовов процедур BLAS. Тест HPL традиционно используется для упорядочивания списка наиболее мощных компьютеров мира Top500 (<http://www.top500.org/>) и списка наиболее мощных компьютеров СНГ Top50 (<http://supercomputers.ru>).

Peak — небольшой тест, адаптированный из [4], предназначенный для достижения максимально возможной производительности процессорного ядра на фрагменте программы, написанном на языке высокого уровня (Фортран). Для этого используются пары операций сложение+умножение, а все данные потенциально могут быть расположены на регистрах.

Operations — тест, определяющий производительность процессорного ядра на ряде простых арифметических операций и их комбинаций.

STREAM — стандартный тест, измеряющий производительность на операциях $a(i) = b(i)$; $a(i) = q * b(i)$; $a(i) = b(i) + c(i)$; $a(i) = b(i) + q * c(i)$ и производящий замер скорости передачи данных из оперативной памяти в процессор.

Flo52 — программа из пакета тестов Perfect Club Benchmarks, реализующая упрощённый вариант одной из задач вычислительной гидродинамики [5].

Применяется также ряд других тестов, и этот набор постоянно пополняется с расширением множества анализируемых факторов, влияющих на производительность приложений.

Формируемый в результате комплексного тестирования производительности вычислительных серверов процессорного полигона документ Performance Guide [6] содержит информацию о сравнительных характеристиках серверов процессорного полигона, позволяет оценить их пиковые характеристики, найти узкие места при выполнении тех или иных операций, определить эффективность базового программного обеспечения (операционных систем, компиляторов, оптимизированных низкоуровневых библиотек и т.д.), а также решать многие другие задачи.

4. Результаты тестирования

Рассмотрим некоторые результаты, полученные в ходе тестирования серверов процессорного полигона. Целью тестирования являлось определение влияния различных характеристик программно-аппаратной среды на производительность приложений.

Сначала посмотрим, какой производительности можно достичь в принципе на процессорных ядрах серверов полигона на программе, написанной на языке высокого уровня. Рис. 1 демонстрирует производительность теста peak в сравнении с пиковой производительностью процессорных ядер. Для компиляции на всех узлах был использован компилятор gfortran с опцией оптимизации -O3.

На графике видно, что тест peak позволяет значительно лучше приблизиться к пиковой производительности на процессорах AMD, чем на процессорах Intel или IBM. На процессорах Intel можно добиться большего при задействовании команд из наборов SSE2 и SSE3, но эта операция требует дальнейшего преобразования текста используемого теста.

Программа HPL традиционно используется в высокопроизводительных вычислениях как некоторая мера производительности вычислительного узла. Она позволяет оценить возможность достижения максимальной производительности вычислительного узла с использованием низкоуровневых библиотек. Результаты теста HPL приведены на рис. 2. Для компиляции на всех узлах был использован компилятор gcc с опцией оптимизации -O3. В качестве реализа-

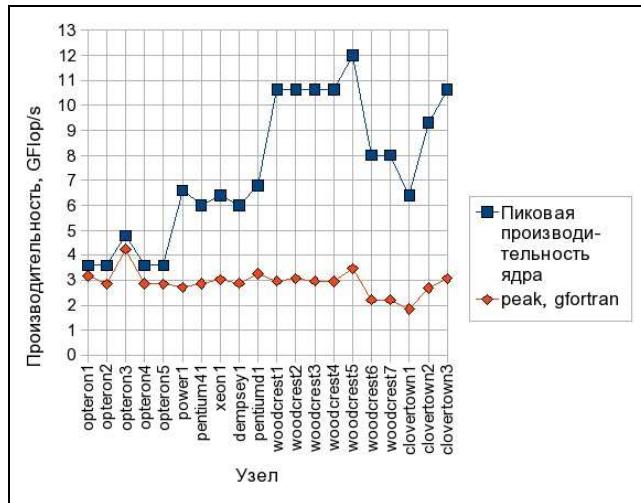


Рис. 1. Производительность теста peak на узлах процессорного полигона

ции BLAS была взята библиотека ATLAS. Тест запускался на одном ядре с матрицей 15000×15000 .

График показывает, что с использованием оптимизированной под конкретную архитектуру низкоуровневой библиотеки можно значительно ближе подойти к пиковой производительности процессорного узла, особенно это касается процессоров Intel и IBM.

Многие стандартные алгоритмы многократно реализованы на доступных платформах. Зачастую бывает гораздо проще использовать существующую реализацию, а не писать свой вариант. Но насколько эффективны те или иные реализации стандартных библиотек на конкретной платформе, нужно показать при помощи набора тестов. Посмотрим, как влияет выбор конкретной специализированной библиотеки на производительность приложения. Для теста HPL сравним производительность при использовании библиотек ATLAS, MKL и GotoBLAS (рис. 3). Для компиляции на всех узлах был использован компилятор gcc с опцией оптимизации -O3. Тест запускался на одном ядре узла opteron1 с матрицей 15000×15000 .

Наиболее эффективной библиотекой в данном случае является

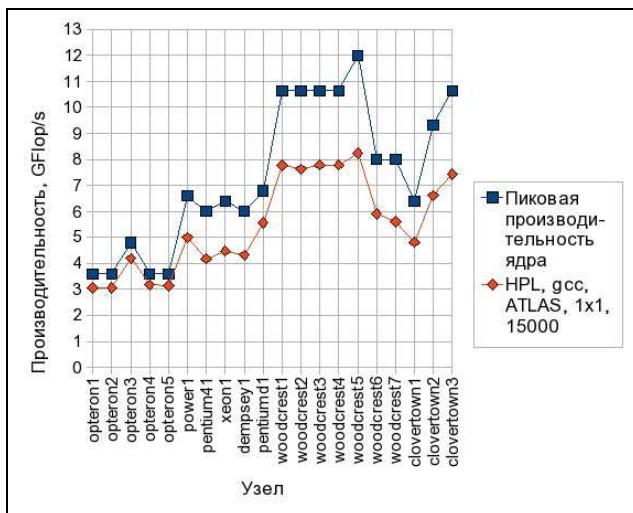


Рис. 2. Производительность теста HPL на узлах полигона

GotoBLAS. Её использование даёт наилучшие результаты и на процессорах Intel, но на них библиотека MKL эффективнее, чем библиотека ATLAS.

Теперь посмотрим, какова производительность процессорных ядер полигона на вычислительном ядре реального алгоритма. Для этого возьмём большой (large) вариант программы flo52 (рис. 4).

Очевидно, что производительность процессоров на реальном приложении значительно ниже, чем на специально разработанном тесте или на тесте, использующем оптимизированные библиотеки.

Оценим теперь, какой вклад в производительность процессорных ядер на программах, написанных на языках высокого уровня, вносят различные базовые арифметические операции. Посмотрим часть результатов теста operations на нескольких серверах процессорного полигона (рис. 5). Для компиляции на всех узлах был использован компилятор gfortran с опцией оптимизации -O3.

Результаты теста показывают, что даже на, казалось бы, «хороших» с точки зрения архитектуры процессоров базовых операциях получается производительность, составляющая лишь небольшую долю от пиковой производительности.

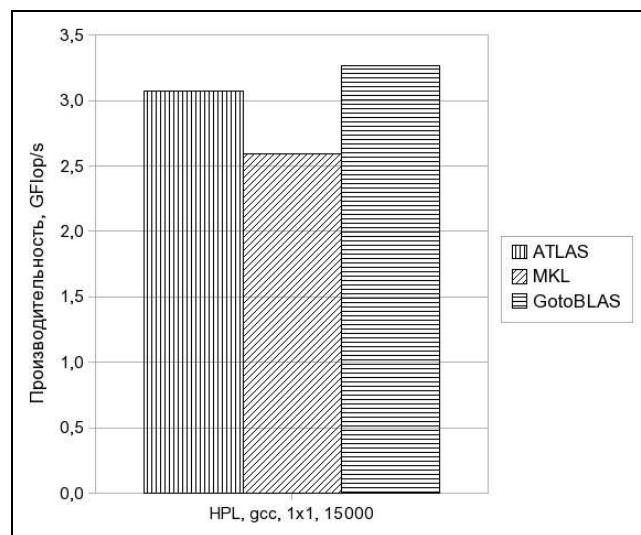


Рис. 3. HPL на узле opteron1 в зависимости от используемой реализации BLAS

Теперь хотелось бы проверить, насколько велико влияние компилятора на эффективность использования аппаратных средств. Посмотрим, как зависит производительность большого варианта теста flo52 от использования различных компиляторов (рис. 6). Для компиляции на всех узлах были использованы компиляторы с опцией оптимизации -O3.

Видно, что для flo52 лучшим вариантом на всех узлах является использование компилятора PathScale. Это касается как процессоров AMD, так и процессоров Intel.

На рис. 7 приведено сравнение эффективности различных компиляторов на наборе из нескольких тестов. Для компиляции на узле woodcrest6 были использованы компиляторы с опцией оптимизации -O3.

На всех тестах кроме peak снова лучшим является компилятор PathScale. С тестом peak чуть лучше других справляется компилятор Portland Group.

Каждый компилятор является сложной программой. Для его эф-

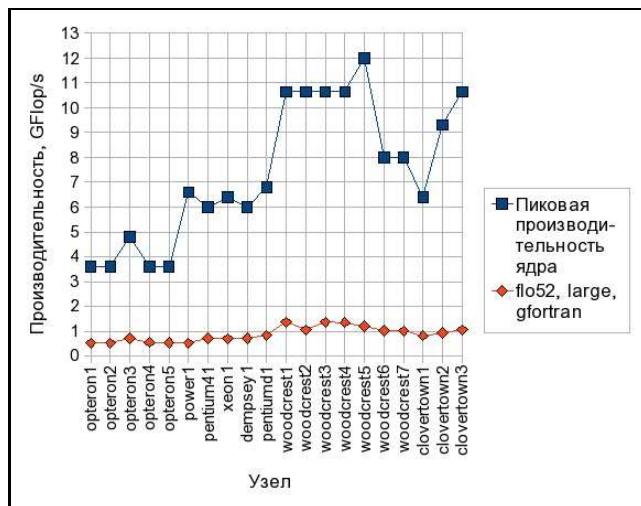


Рис. 4. Производительность теста flo52 на узлах полигона

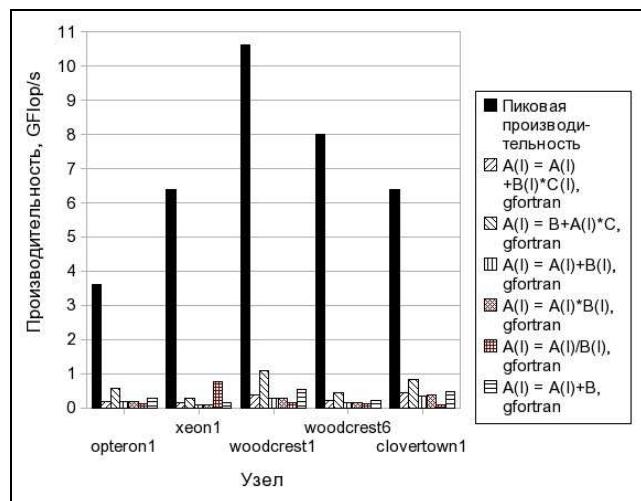


Рис. 5. Производительность арифметических операций на узлах полигона

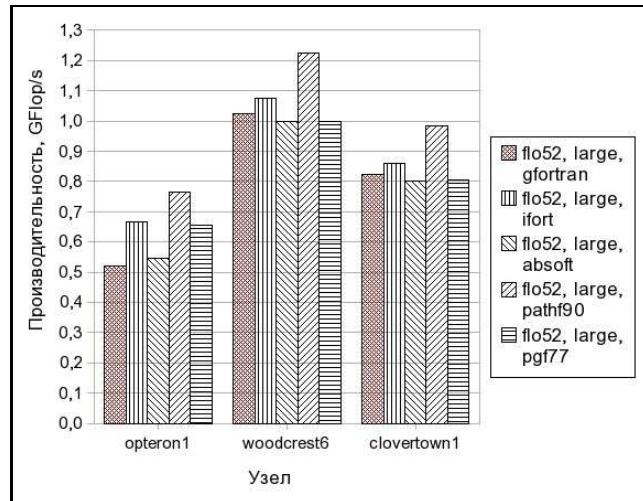


Рис. 6. flo52 на узлах полигона в зависимости от используемого компилятора

фективного использования на каждой конкретной платформе нужно знать много тонких вещей. Все предыдущие эксперименты проводились со стандартным уровнем оптимизации -O3. Посмотрим, чего можно добиться при использовании других специфических опций оптимизации. На рис. 8 показана зависимость производительности набора тестов от используемых опций компилятора ifort. Результаты приводятся для узла woodcrest6.

Видно, что для компилятора Intel Fortran на всех тестах кроме peak лучшие результаты получаются с опцией -fast, которая является сокращением для набора опций -O3 -ipo -no-prec-div -static -xP. На teste peak лишние оптимизации только ухудшают ситуацию, и оптимальным остаётся вариант с уровнем оптимизации -O1. Дальнейшие исследования специфических опций каждого компилятора для каждой конкретной архитектуры могут дать дополнительный эффект, но вряд ли очень существенно изменят общую картину.

До сих пор речь шла о производительности одного процессорного ядра. В настоящее время почти все процессоры делаются многоядерными, и эта тенденция находит отражение в составе процессор-

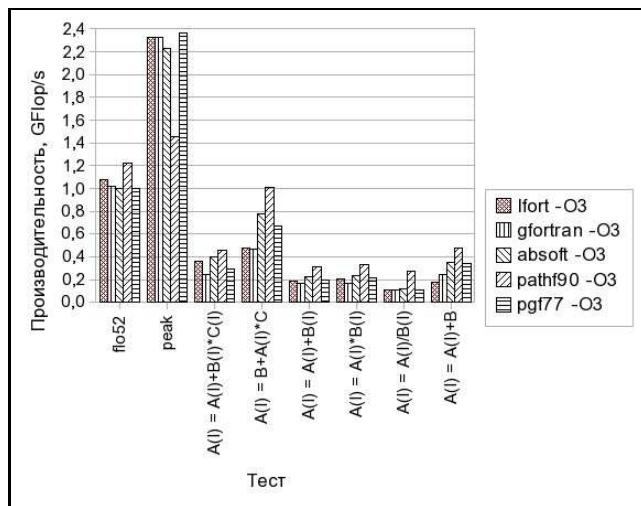


Рис. 7. Разные тесты на узле woodcrest6 в зависимости от используемого компилятора

ного полигона. На многоядерных и многопроцессорных узлах можно проверить масштабируемость параллельных программ, то есть изменение производительности тестовых приложений при увеличении количества процессов. График на рис. 9 демонстрирует масштабируемость теста HPL при увеличении количества задействованных ядер узла clovertown1. Для компиляции на всех узлах был использован компилятор gcc с опцией оптимизации -O3. В качестве реализации BLAS была взята библиотека GotoBLAS. Тест запускался с матрицей 15000×15000 .

При увеличении количества задействованных процессорных ядер большее влияние начинают оказывать каналы межъядерного и межпроцессорного взаимодействия. Поэтому производительность теста всё более удаляется от пиковой. Здесь показана масштабируемость только внутри одного сервера, при задействовании нескольких серверов всё большее влияние начнёт оказывать и используемая коммуникационная сеть.

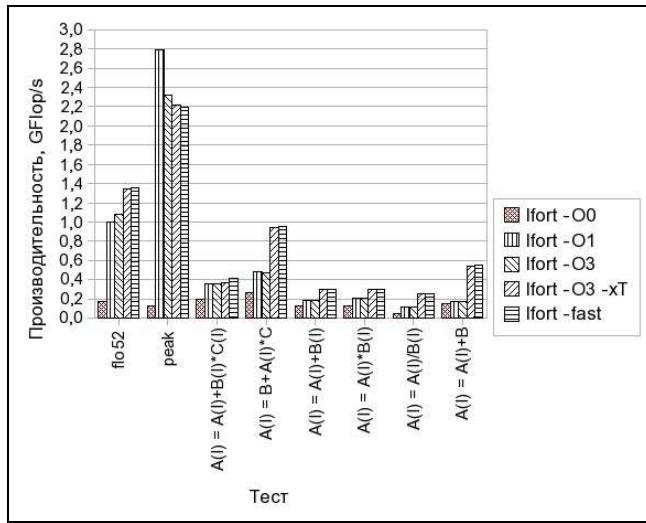


Рис. 8. Разные тесты на узле woodcrest6 в зависимости от используемых опций компилятора ifort

5. Заключение

Часть факторов, влияющих на производительность приложений, осталась за рамками данной статьи. Это, например, оценка скорости работы с различными уровнями памяти, оценка влияния на производительность объёма оперативной памяти, частоты системной шины, эффективность операционной системы и другие. Для каждой такой задачи требуются свои подходы и формируются свои наборы тестов. Часто влияние одного фактора очень трудно отделить от влияния другого, а в процессе работы программы сказываются почти все факторы. Важно, что проанализировав результаты комплексного тестирования пользователь может оценить важность для его задачи тех или иных факторов и попытаться найти лучшее соответствие реализуемого алгоритма и целевой программно-аппаратной среды.

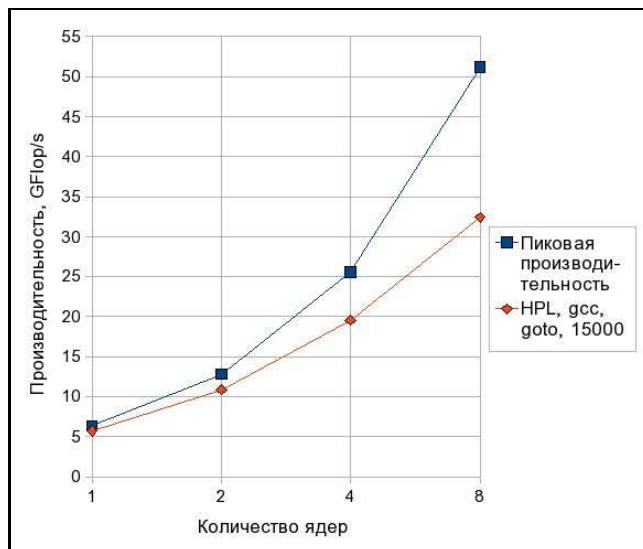


Рис. 9. HPL на узле clovertown1 в зависимости от количества ядер

Список литературы

- [1] Антонов А.С. Далеко ли до пика?// Открытые системы, N 6, 2006. С. 64–66.
- [2] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002.
- [3] Жуматий С.А. Система анализа производительности параллельных программ на кластерных установках// Вычислительные методы и программирование. 2005. Т 6, N 2. С. 57–64.
- [4] AIX Version 3.2 for RISC System/6000. Optimization and Tuning Guide for Fortran, C, and C++.
- [5] Воеводин Вл.В., Антонов А.С. Эффективная адаптация последовательных программ для современных векторно-конвейерных и массивно-параллельных супер-ЭВМ// Программирование, 1996, N 4, С.37–51.

- [6] Антонов А.С. Комплексное тестирование производительности вычислительных серверов// Научный сервис в сети Интернет: многоядерный компьютерный мир. 15 лет РФФИ: Труды Всероссийской научной конференции (24-29 сентября 2007 г., г.Новороссийск).-М.: Изд-во МГУ, 2007. С. 135–138.

Численные алгоритмы анализа чувствительности и сложности описания в задачах идентификации моделей математической иммунологии [§]

Г. А. БОЧАРОВ[®], Н. А. МЕДВЕДЕВА^{*}

Аннотация

Развитие эффективных вычислительных подходов к реализации задач системного анализа иммунных процессов в норме и при вирусных заболеваниях, с учетом блочной структуры организма, является единственным средством интеграции различных данных наблюдений и теоретических гипотез в единую математическую теорию иммунной системы. Модульный подход к построению уравнений моделей на основе балансных соотношений позволяет сформировать некоторое семейство возможных математических моделей. Проверка адекватности структуры уравнений данным наблюдений с целью выбора оптимальной модели является чрезвычайно проблемным и плохо исследованным этапом формирования количественных математических теорий (гипотез) иммунных процессов. Это связано с трудностями многократного решения обратных задач, численной реализации алгоритмов анализа идентифицируемости моделей, оценивания информационной сложности моделей. В данной работе рассматриваются ключевые элементы численной технологии построения оптимального описания динамики иммунных процессов на примере противовирусной реакции системы интерферона. В основе нашего подхода лежит использование принципа максимального правдоподобия для идентификации параметров модели, использование информационной матрицы Фишера для оценки степени неопределенности параметров, локальный и глобальный анализ чувствительности в рамках детерминистского и стохастического подходов, оценивание качества моделей на основе информационно-теоретических подходов.

[§]Данная работа проводилась при поддержке Российского фонда фундаментальных исследований (грант РФФИ №05-01-00732).

[®]Институт вычислительной математики РАН

^{*}Московский государственный университет им. М. В. Ломоносова

1. Уравнения математической иммунологии

Одной из центральных задач математической иммунологии является разработка эффективной вычислительной методологии построения оптимальных, в смысле информативности, моделей динамики сложных систем. Это связано с решением задач усвоения данных, получаемых с помощью современных высокопроизводительных лабораторных и клинических методов, таких как, проточная цитофлуорометрия, протеомика, геномика и др. В отличие от классической физики или химии, при моделировании живых систем используется феноменологический подход к конструированию уравнений моделей. При этом, модели сложных систем строятся из блоков описывающих элементарные процессы. Так модели популяционной динамики иммунных реакций строятся на основе балансных соотношений процессов рождения и гибели клеток и патогенов. Этот подход позволяет сформировать некоторое семейство возможных математических моделей, различающихся функциональными зависимостями, используемыми при описании одного и того же элементарного процесса, числом параметров, структурной сложностью. Анализ адекватности структуры моделей данным наблюдений с целью выбора наиболее информативной предполагает необходимость развития эффективных, адаптированных к конкретным типам уравнений, численных методов решения задач оценивания параметров моделей, исследования чувствительности и информационной сложности моделей. В данной работе излагаются ключевые элементы единого подхода, на основе метода максимального правдоподобия, к построению оптимального описания иммунных процессов. В качестве конкретного объекта моделирования будет рассмотрена динамика реакции системы интреферона на уровне базового блока, описывающего активность дендритных клеток. В качестве средства математического описания мы рассматриваем модели на основе систем дифференциальных уравнений с запаздывающим аргументом:

$$y'(t) = f(y(t), y(t-\tau), p) \quad (\tau \geq 0), \quad t \in [t_0, T], \quad (1)$$

где $y(t, p) \in \mathbb{R}^M$ и $p \in \mathbb{R}^L$. Обозначим компоненты вектора параметров в уравнениях p через p_ℓ . Величина запаздывания, $\tau \geq 0$, является дополнительным параметром, анализ которого является более сложным, чем для обычных параметров p_ℓ . Для данного

класса систем требуется задать начальные условия $[t_0 - \tau, t_0]$, в общем случае, также зависящие от параметров:

$$y(t, p) := \psi_0(t, p), \quad t \in [t_0 - \tau, t_0]. \quad (2)$$

Решение модели в моменты времени наблюдений $t_j \in [t_0, T]$, $y(t_j, p)$ должно соответствовать по выбранной мере данным измерений $\{y_j\}$. Детальное изложение различных аспектов построения уравнений моделей в иммунологии рассматривается в работах [1, 2].

2. Математическая модель системы интерферона

В модели реакции системы интерферона в ответ на вирусную инфекцию [3] рассматривается популяционная динамика численностей вирусов $V(t)$, молекул интерферона $I(t)$, неинфицированных клеток $C(t)$ и инфицированных клеток, продуцирующих интерферон $C_V(t)$. Уравнения модели содержат описание кинетики процессов, изображенных на рис. 1.

Система уравнения модели содержит описание скорости изменения численности популяций вирусов, молекул интреферона и клеток

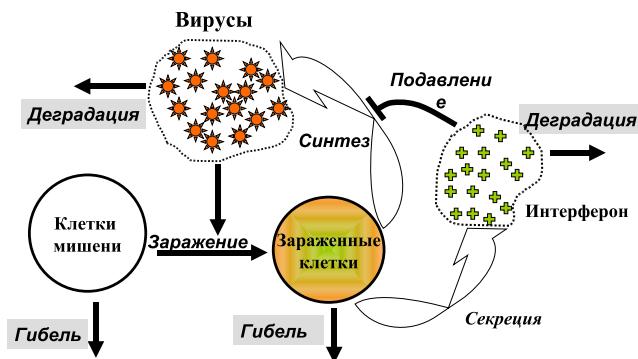


Рис. 1. Биологическая схема, лежащая в основе математической модели (3) синтеза интерферона антигенпрезентирующими клетками в ответ на вирусную инфекцию.

с помощью обыкновенных дифференциальных уравнений (ОДУ) и дифференциальных уравнений с запаздывающим аргументом (ДУЗА) следующего вида:

$$\frac{dV}{dt}(t) = \frac{\rho_V C_V(t - \tau_V)}{1 + I(t)/\theta} - d_V V(t), \quad (3a)$$

$$\frac{dI}{dt}(t) = \rho_I C_V(t - \tau_I) - d_I I(t), \quad (3b)$$

$$\frac{dC}{dt}(t) = -\sigma V(t)C(t) - d_C(t)C(t), \quad (3c)$$

$$\frac{dC_V}{dt}(t) = \sigma V(t)C(t) - d_{CV}(t)C_V(t). \quad (3d)$$

Одной из особенностей данной модели является использование закона Гомпертца, при описании продолжительности жизни клеток. С учетом этого, кинетика гибели клеток определяется двумя параметрами — начальной скоростью гибели и темпом увеличения данной скорости. Обоснованность данного описания будет исследована позже с помощью информационных критериев.

Начальные данные для задачи Коши ($t = 0$) имеют вид: $V(0) = V_0$, $I(0) = I_0$, $C(0) = C_0$, $C_V(t) = 0$, $t \in [-\max(\tau_V, \tau_I), 0]$. Параметры модели описаны в табл. 1. Для идентификации параметров моделей использовался подход на основе метода максимального правдоподобия.

3. Обратные задачи и критерий правдоподобия

Задача приближения модели к данным наблюдений связана, прежде всего, с выбором критерия близости или целевого функционала невязки. В идеале, этот выбор должен определяться характером распределения и статистическими свойствами погрешностей измерений наблюдаемых переменных модели. Ранее, данный вопрос исследовался нами в работе [4]. В частности, путем анализа большого массива экспериментальных данных по динамике численности лимфоцитов при иммунном ответе было показано, что по критерию Колмогорова-Смирнова наиболее адекватной моделью ошибок является нормальное или лог-нормальное распределения. Функционал невязки $\Phi(p)$, $\Phi(p) \geq 0$, зависит от данных наблюдений $\{t_j; y_j^i\}_{j=1}^N$ (для $i = 1, \dots, M$) и значениями соответствующих

Параметр	Физический смысл	Размерность	Оптим. оценка
ρ_V	Скорость продукции вирусов одной клеткой	вир/час	1.1
ρ_I	Скорость продукции интерферона одной клеткой	пг/час	0.00091
θ	Порог 50% ингибиования продукции вирусов интерфероном	пг/мл	11.6
σ	Скорость инфицирования клеток	кл/вир/час	2.1×10^{-6}
τ_V	Задержка продукции вирусов клеткой	час	4.9
τ_I	Задержка продукции интерферона клеткой	час	4.5
d_{0C_V}	Начальная скорость гибели инфицированных клеток	1/час	0.1
k_{CV}	Ускорение темпа гибели клеток инфицированных клеток	1/час	0.13
f_i	Начальная доля инфицированных клеток при кратности инфекции 1		0.0077
d_V	Скорость гибели вирусов	1/час	0.155
d_I	Скорость распада интерферона	1/час	0.012
d_{0C}	Начальная скорость гибели неинфицированных клеток	1/час	0.0055
k_C	Ускорение темпа гибели клеток неинфицированных клеток	1/час	0.089

Таблица 1. Параметры модели и их оценки полученные по методу максимального правдоподобия.

переменных $\{y^i(t_j; p)\}_{j=1:N}^{i=1:M}$ решения модели $y(t; p)$ (3), неявно зависящего от параметров уравнений. Тем самым, задача сводится к поиску таких значений параметров p^* для (3), при которых решение $\{y^i(t_j; p^*)\}_{j=1:N}^{i=1:M}$, наилучшим образом описывает данные $\{y_j^i\}_{j=1:N}^{i=1:M}$:

$$\Phi(p^*) = \min_{p \in \mathbb{R}_+^L} \Phi(p). \quad (4)$$

Данная обратная задача, в общем случае, является некорректной в силу того, что с учетом ошибок измерений, статистически допустимым решениями являются все те значения параметров, при которых решение уклоняется от данных наблюдений не более чем на некоторую величину. Решение обратной задачи, с вычислительной точ-

ки зрения, сводится к поиску глобального минимума функционала $\Phi(\cdot)$.

Сделаем следующие предположения:

- 1) ошибки измерений в различные моменты времени независимы;
- 2) ошибки измерений распределены по нормальному закону

$$\mathbf{y}_j \sim \mathcal{N}(\mathbf{y}(t_j; \mathbf{p}), \Sigma_j),$$

где $\{\mathbf{y}(t_j; \mathbf{p})\}_{j=1}^N$ являются средними определяемыми моделью, а Σ_j является j -ой ковариационной матрицей (матрицей ошибок);

- 3) ошибки измерений компонент вектор-функции решения являются некоррелированными, т.е. ковариационная матрица является диагональной

$$\Sigma_j = \sigma^2 \{ \text{diag}[\omega_1^{[j]}, \omega_2^{[j]}, \dots, \omega_M^{[j]}] \}, \quad (5)$$

(где σ^2 — коэффициент вариации).

Для решения обратной задачи будем следовать принципу *максимального правдоподобия* (МП). Распределение вероятности наблюдения данных выборки объема N определяется произведением функций [5, 7]:

$$\left\{ \mathcal{H}(\mathbf{y}_j; \mathbf{p}) = \frac{1}{\sqrt{(2\pi)^M \det \Sigma_j}} \exp\left\{-\frac{1}{2} [\mathbf{y}(t_j; \mathbf{p}) - \mathbf{y}_j]^T \Sigma_j^{-1} [\mathbf{y}(t_j; \mathbf{p}) - \mathbf{y}_j]\right\} \right\}_{j=1}^N. \quad (6)$$

Полная функция правдоподобия или вероятность получения данной выборки как функции вектора параметров модели \mathbf{p} имеет вид

$$\mathcal{L}(\mathbf{p}) = \prod_{j=1}^N \mathcal{H}(\mathbf{y}_j; \mathbf{p}). \quad (7)$$

Вектор параметров \mathbf{p}^* , при котором эта функция достигает максимума и является оценкой максимального правдоподобия.

Рассмотрим функционал взвешенных наименьших квадратов

$$\Phi_{WLS}(\mathbf{p}) \equiv [\mathbf{y}(t_j; \mathbf{p}) - \mathbf{y}_j]^T \Sigma_j^{-1} [\mathbf{y}(t_j; \mathbf{p}) - \mathbf{y}_j]. \quad (8)$$

С учетом предположения 3)

$$\Phi_{WLS}(\mathbf{p}) \equiv \sigma^{-2} \Phi_{OLS}(\mathbf{p}), \quad (9)$$

где

$$\Phi_{OLS}(\mathbf{p}) = \sum_j \| \text{diag}^{-1}[\omega_1^{[j]}, \omega_2^{[j]}, \dots, \omega_M^{[j]}] [\mathbf{y}(t_j; \mathbf{p}) - \mathbf{y}_j] \|^2. \quad (10)$$

Логарифмическая функция максимального правдоподобия имеет вид

$$\begin{aligned} \ln \mathcal{L}(\mathbf{p}) &= -\frac{1}{2} \left\{ NM \ln (2\pi) + NM + 2 \sum_{i,j} \ln (\omega_i^{[j]}) \right\} \\ &\quad - \frac{1}{2} [NM \ln (\Phi_{OLS}(\mathbf{p})) - NM \ln (NM)]. \end{aligned} \quad (11)$$

Нетрудно видеть, что максимизация функции правдоподобия $\mathcal{L}(\mathbf{p})$ эквивалентна минимизации $\Phi_{OLS}(\mathbf{p})$ (или $\Phi_{WLS}(\mathbf{p})$), при этом оценка МП дисперсии имеет вид

$$\begin{aligned} \sigma^{2*} &= \frac{1}{NM} \sum_j \| \text{diag}^{-1}[\omega_1^{[j]}, \omega_2^{[j]}, \dots, \omega_M^{[j]}] [\mathbf{y}(t_j, \mathbf{p}^*) - \mathbf{y}_j] \|^2 \\ &= \frac{1}{NM} \Phi_{OLS}(\mathbf{p}^*). \end{aligned} \quad (12)$$

где \mathbf{p}^* обозначает оптимальную в смысле МП оценку параметров.

Задача поиска точечных оценок максимально правдоподобных параметров сводится к минимизации функционала наименьших квадратов. Корректный выбор конкретного вида функционала наименьших квадратов предполагает известным закон распределения ошибок измерений. В приложениях встречаются три варианта функционала невязки, соответствующих

- нормальному закону с дисперсией, не зависящей от моментов измерений и одинаковой для всех наблюдаемых переменных – обычный метод наименьших квадратов:

$$\Phi_{OLS}(\mathbf{p}) = \sum_{j=1}^N \sum_{i=1}^M [\mathbf{y}^i(t_j; \mathbf{p}) - \mathbf{y}_j^i]^2 = \sum_{j=1}^N \|\mathbf{y}(t_j, \mathbf{p}) - \mathbf{y}_j\|^2; \quad (13a)$$

- нормальному закону с дисперсией, не зависящей от моментов измерений, но различной для каждой переменных – взвешенный метод наименьших квадратов (при этом, может быть иметь место ситуация, когда относительная величина дисперсии одинакова для всех переменных):

$$\Phi_{WLS}(p) \equiv \sigma^{-2} \sum_{j=1}^N \sum_{i=1}^M \{ \omega_i^{[j]} [y^i(t_j, p) - y_j^i] \}^2; \quad (13b)$$

- лог-нормальному закону (предполагается, что $y_j^i > 0$ и $y^i(t_j; p) > 0$) с дисперсией не зависящей от моментов измерений и одинаковой (случай различных дисперсий приводит к взвешенному варианту, аналогично предыдущему случаю) для всех наблюдаемых переменных – логарифмический метод наименьших квадратов:

$$\Phi_{LogLS}(p) = \sum_{j=1}^N \sum_{i=1}^M [\ln(y^i(t_j, p)) - \ln(y_j^i)]^2. \quad (13c)$$

Нормальный и лог-нормальный законы распределения отражают, соответственно, арифметическую и геометрическую зависимость среднего от значений данных наблюдений. Качественно, различие между арифметической и геометрической нормальностью ошибок измерений, проявляется в том, в первом случае отклонения от ожидаемых средних значений в обе стороны (увеличения или уменьшения) вносят одинаковый вклад в значение функционала, если их абсолютные значения равны, а во втором варианте, только, если их относительные значения равны. Последнее вариант предполагает, что если масштаб переменных модели сильно варьирует, то следует использовать логарифмический метод наименьших квадратов. Для идентификации параметров модели 3 мы использовали логарифмический метод наименьших квадратов, с учетом того, что переменные модели существенно различаются по своим абсолютным значениям. Часть фундаментальных параметров модели (последние 4 в табл. 1, в частности, скорости деградации вирусов, интерферона и неинфицированных клеток, определялась по независимым экспериментальным данным. Оптимальным оценкам параметров табл. 1

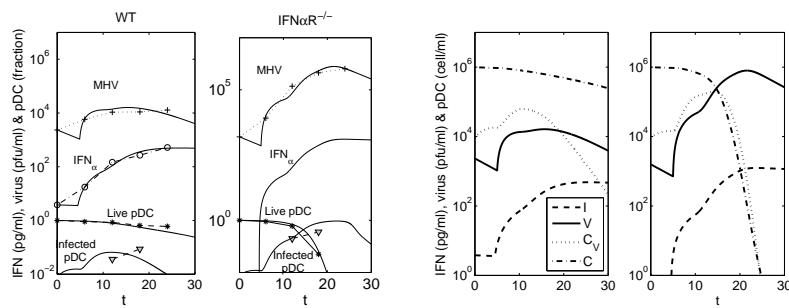


Рис. 2. Данные наблюдений и решения модели, соответствующие минимуму логарифмического функционала наименьших квадратов. Два левых графика описывают кинетику экспериментально наблюдавшихся процессов в случае инфекции нормальных дендритных клеток и клеток, у которых отсутствует рецептор к интерферону. Два правых графика характеризуют динамику переменных модели, соответственно, при наличии ингибирующего эффекта интерферона на продукцию вирусов, и в случае, когда его нет.

соответствует решение модели приведённое на рис. 2. Заметим, что часть переменных модели ($V(t)$ и $I(t)$) являются непосредственно наблюдаемыми, а оставшиеся должны быть преобразованы в наблюдаемые некоторым нелинейным образом

$$\% \text{Live pDC}(t) = \frac{C(t) + C_V(t)}{C_0}, \quad (14a)$$

$$\% \text{Infected pDC}(t) = \frac{C_V(t)}{C(t) + C_V(t)}. \quad (14b)$$

4. Уравнения чувствительности и информационная матрица Фишера

4.1. Чувствительность по параметрам и запаздыванию. В основе анализа чувствительности моделей к вариациям параметров лежит исследование элементарных коэффициентов чувствительности, являющихся частными производными от компонент вектор-функции решения модели по параметрам. Предполагая гладкость решения $y(t; p)$ по вектору параметров p , рассмотрим разложение

$$y(t; p + \delta p) = y(t; p) + S(t, p)\delta p + O(\|\delta p\|^2).$$

Матрица $S(t) \equiv S(t; p)$ является $M \times L$ матрицей коэффициентов чувствительности, i -я строка которой имеет вид $s_i(t; p) = \left[\frac{\partial y^i(t, p)}{\partial p_1}, \frac{\partial y^i(t, p)}{\partial p_2}, \dots, \frac{\partial y^i(t, p)}{\partial p_L} \right]^T$. Используя обозначение $\left\{ \frac{\partial}{\partial p} \right\}^T$, матрицу коэффициентов чувствительности $S(t; p)$ можно записать в виде

$$S(t; p) \equiv \left\{ \frac{\partial}{\partial p} \right\}^T y(t; p) \in \mathbb{R}^{M \times L}. \quad (15)$$

Таким образом, матрица $S(t; p)$ характеризует локальную чувствительность решения модели $y(t, p)$ к малым изменениям компонент параметра p ; при этом, строка $s_i(t; p)$ соответствует первым производным i -ой компоненты $y_i(t; p)$ по параметрам p_ℓ ($\ell \in \{1, 2, \dots, L\}$).

Особенностью ДУЗА является разрывность первых производных решения по времени. Если начальная функция имеет разрыв первого рода в точке t_0 , то соответствующее решение имеет разрывы производных в моменты $t \in \bigcup_{n \in \mathbb{N}} n\tau$. При этом, имеет место увеличение гладкости решений с ростом t . Воспользуемся обозначениями

$$y := y(t; p), \quad y_\tau := y(t - \tau; p), \quad y'_{-\tau} := y'(t - \tau; p). \quad (16)$$

Для системы (1) зависящие от времени коэффициенты чувствительности решения по компонентам вектора параметров p удовлетворяют на отрезках $t \in \bigcup_{n \in \mathbb{N}} [t_0 + n\tau, t_0 + (n+1)\tau]$ следующей системе уравнений чувствительности

$$\begin{aligned} S'(t) &= \frac{\partial}{\partial y} f(y, y_\tau; p) S(t) + \frac{\partial}{\partial y_\tau} f(y, y_\tau; p) S(t - \tau) \\ &\quad + \frac{\partial}{\partial p} f(y, y_\tau; p). \end{aligned} \quad (17)$$

На рис. 3 приведены графики поведения элементарных функций (коэффициентов) чувствительности решения модели 3 для нормальных клеток и клеток, не имеющих рецепторы к интерферону, по параметрам ρ_I и ρ_V , характеризующим скорости продукции одной клеткой интерферона и вирусных частиц.

Производная $s_\tau(t) \equiv s_\tau(t, p)$ вектор-функции решения $y(t; p)$ по параметру запаздывания τ ($s_\tau(t, p) = \frac{\partial}{\partial \tau} y(t; p) \in \mathbb{R}^M$) удовлетво-

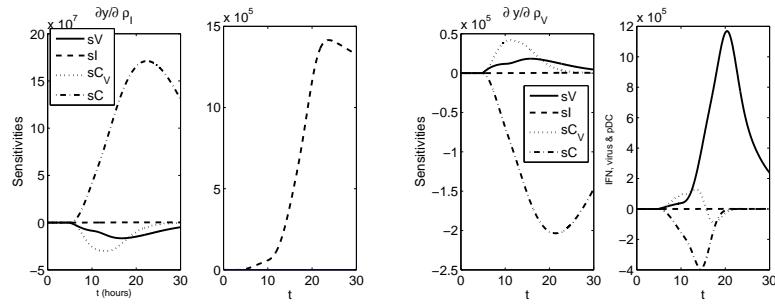


Рис. 3. Динамика производных решения модели по параметрам — коэффициенты чувствительности. Два левых графика описывают чувствительность решений к изменению скорости синтеза интерферона в случае нормальных дендритных клеток и клеток, у которых отсутствует рецептор к интерферону, соответственно. Аналогично, два правых графика характеризуют чувствительность решений модели к вариации скорости синтеза вирусов при наличии ингибирующего эффекта интерферона на продукцию вирусов, и в случае, когда его нет

ряет уравнениям с запаздывающим аргументом нейтрального типа следующего вида:

$$\begin{aligned} s_\tau'(t) = & \frac{\partial}{\partial y} f(y, y_\tau, y'_\tau; p) s_\tau(t) + \frac{\partial}{\partial y_\tau} f(y, y_\tau, y'_\tau; p) s_\tau(t - \tau) \\ & - \frac{\partial}{\partial y_\tau} f(y, y_\tau, y'_\tau; p) y'(t - \tau) + \frac{\partial}{\partial \tau} f(y, y_\tau, y'_\tau; p). \end{aligned} \quad (18)$$

Численное решение таких уравнений представляет более сложную задачу, т.к. увеличения гладкости решений с ростом t в случае уравнений нейтрального типа не имеет места [6].

4.2. Информационная матрица Фишера. При оценивании точности решения обратных задач фундаментальная роль принадлежит матрице Фишера, которая характеризует количество информации о параметрах модели содержащейся в конкретной выборке данных наблюдений [8, 9]. Информационная матрица позволяет определить предельную точность, с которой можно оценить параметра модели. Информационная матрица Фишера F , определяется как математическое ожидание матрицы вторых производных функции

максимального правдоподобия по элементам вектора параметров \mathbf{p}

$$\mathbf{F}(\mathbf{p}^*, \tau^*) \equiv E \left\{ \frac{\partial^2}{\partial \mathbf{p}^2} \mathcal{L}(\mathbf{p}, \tau) \right\}_{\mathbf{p}^*, \tau^*}. \quad (19)$$

Используя решение уравнений чувствительности S, s_τ , для множества времен данных наблюдений $t_n, n=1^N$ построим расширенную матрицу чувствительности \mathbf{Z} , строки которой содержат коэффициенты чувствительности по параметрам для всех моментов (выборки) измерений

$$\mathbf{Z} := \begin{bmatrix} S(t_1; \mathbf{p}^*, \tau^*) & s_\tau(t_1; \mathbf{p}^*, \tau^*) \\ S(t_2; \mathbf{p}^*, \tau^*) & s_\tau(t_2; \mathbf{p}^*, \tau^*) \\ \vdots & \vdots \\ S(t_N; \mathbf{p}^*, \tau^*) & s_\tau(t_N; \mathbf{p}^*, \tau^*) \end{bmatrix} \quad (20)$$

В случае, когда ошибки наблюдений распределены по нормальному закону, то для матрицы Фишера справедливо следующее представление через выборочную матрицу чувствительности \mathbf{Z} :

$$\mathbf{F} := \mathbf{Z}^\top \Sigma_y \mathbf{Z}. \quad (21)$$

Согласно неравенству информации Крамера–Рао нижняя оценка элементов матрицы рассеяния параметров Σ_p (ковариационной матрицы параметров) от истинных значений определяется элементами матрицы, обратной информационной матрице Фишера:

$$\Sigma_p \geq \mathbf{F}^{-1}. \quad (22)$$

Результаты расчета матрицы Фишера для анализа предельной оценки точности важнейших параметров модели интерферона (за исключением запаздываний и начальной доли инфицированных клеток) приведены в табл. 2. Полученные оценки снизу для стандартных отклонений оценок параметров показывают, что по имеющемуся массиву данных измерений параметры, модели нельзя определить точнее, чем 50% от их номинальных значений.

5. Анализ идентифицируемости параметров

Выборочная матрица чувствительности позволяет ранжировать параметры по их вкладу в измеренные значения наблюдений. Поскольку масштаб переменных модели и параметров сильно варьирует, мы провели анализ перескалированной выборочной матрицы

чувствительности, составленной из блоков

$$\mathbf{S}_{\log}(t; \mathbf{p}) \equiv \left\{ \frac{\partial}{\partial \log(\mathbf{p})} \right\}^T \log(\mathbf{y}(t; \mathbf{p})) \in \mathbb{R}^{M \times L}. \quad (23)$$

Алгоритм анализа, приведённый в [10] для исследования задач химической кинетики, аналогичен методу линейной пошаговой регрессии: факторы последовательно включаются в регрессионную модель в порядке убывания корреляционной связи с откликом; при этом, остаточная дисперсия, показывающая степень расхождения между данными и прогнозируемыми моделью значениями отклика, может быть неприемлемо велика, для преодоления чего и пополняется модель. Схема численной реализации алгоритма можно записать в виде

- S1. Вычислить сумму квадратов элементов для каждого столбца \mathbf{Z} ;
- S2. Выбрать столбец Z_l с максимальной суммой, $1 \leq l \leq L$;
- S3. Вычислить коэффициенты линейной регрессии столбцов матрицы \mathbf{Z} относительно столбцов расширенной матрицы \mathbf{P} , составленной следующим образом $\mathbf{P} = [\mathbf{P}, Z_l]$ (при первой итерации $\mathbf{P} = Z_l$, где l номер выбранного столбца) по методу наименьших квадратов и осуществить прогноз:

$$\hat{\mathbf{Z}} = \mathbf{P}(\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P} Z_l; \quad (24)$$

Таблица 2. Предельная оценка точности параметров на основе информационной матрицы Фишера.

Параметр	Оптимальная оценка	Нижняя оценка дисперсии
ρ_V	1.1	0.52
ρ_I	0.00091	0.0004
θ	11.6	6.5
σ	2.1×10^{-6}	8.9×10^{-7}
d_{0C_V}	0.1	0.1
k_{C_V}	0.13	0.055

- S4. Вычислить матрицу невязки (ошибку регрессионного прогноза) $\mathbf{R} = \mathbf{Z} - \widehat{\mathbf{Z}}$;
- S5. Переопределить анализируемую матрицу $\mathbf{Z} := \mathbf{R}$;
- S6. Если число столбцов матрицы \mathbf{P} меньше L , то повторить шаги 1–5.

Применив данный алгоритм для анализа для матрицы чувствительности, параметры модели можно упорядочить по их относительному вкладу в наблюдаемые переменные модели следующим образом

$$k_{C_V} > \sigma > \rho_V > \rho_I > \theta > d_{0C_V}.$$

Результаты анализа позволяют более точно оценить вклад параметров в динамику наблюдаемых переменных, с учетом особенностей эксперимента.

6. Информационные критерии сложности моделей

Поскольку для описания количественных закономерностей одного и того же динамического процесса в иммунологии можно предложить несколько различных моделей, важнейшей задачей анализа является выбор наиболее адекватной модели. Величина функционала невязки $\Phi(\mathbf{p}^*)$ в точке минимума является простейшей характеристикой близости конкретной модели $y(t_j; \mathbf{p})$ к данным y_j . В общем случае, данный критерий не является обоснованным, поскольку модели более сложной структуры, например, с большим числом параметров, могут более точно приблизить данные. При этом, поскольку информационное содержание массива данных остается неизменным, естественно ожидать, что начиная с некоторого уровня сложности, точность оценивания параметров будет уменьшаться. Это, в свою очередь, ухудшит качество прогноза на основе модели. Следуя классической концепции интерпретации математической модели как предполагаемой функции плотности распределения вероятности наблюдений, целью задачи идентификации является отыскание модели, минимизирующей информационное расстояние до истинной системы [9]. Подход к идентификации связанный с максимизацией

функции правдоподобия позволяет оценить среднее информационное расстояние между данной моделью и истинной системой. Широко известен информационный критерий Акаике ранжирования моделей [11]:

$$\mu = -2 \ln \mathcal{L}(\hat{\mathbf{p}}) + 2(L+1), \quad (25a)$$

$$\mu_c = -2 \ln \mathcal{L}(\hat{\mathbf{p}}) + 2(L+1) + \frac{2(L+1)(L+2)}{v-L-2}, \quad (25b)$$

$$v = NM.$$

Модели с меньшей величиной критерия находятся ближе к истинной системе по информационной мере Кульбака-Лейблера. Вопросы приложений данного критерия в задачах математической иммунологии исследовались нами, в частности в [7].

Наряду с критериями, в основе которых лежит оценивание информационного уклонения от идеальной модели данных наблюдений, перспективным представляется подход к ранжированию моделей по критерию «длины описания» [14], связанный с именем Рисснена. В основе данного критерия лежит концепция Колмогоровской сложности алгоритмов описания данных. В рамках данного подхода наилучшей является модель, которая допускает наиболее компактное описание данных, т.е. наиболее сильное сжатие информации в данных [9].

$$\Delta_{MDL} = -\ln \mathcal{L}(\hat{\mathbf{p}}) + \frac{L}{2} \log\left(\frac{v}{2\pi}\right) + \log\left(\int_{\Omega} \sqrt{\det[\mathbf{F}(\mathbf{p})]} d\mathbf{p}\right) \quad (26)$$

Реализация алгоритма вычисления длины описания для многопараметрических моделей является вычислительно трудоёмкой задачей, т.к. связана с расчетом многомерных интегралов, зависящих от детерминанта матрицы Фишера, в свою очередь, определяемой решением системы уравнений чувствительности модели по параметрам.

Для иллюстрации эффективности данных критериев, рассмотрим простейшую задачу кинетики персистенции клеток в норме, т.е. при отсутствии возмущений в виде инфекции. Наиболее распространённым способом описания кинетики является экспоненциальная модель (обозначим её индексом E)

$$\frac{d}{dt} C(t) = -d_C \cdot C(t), \quad C(0) = C_0. \quad (27)$$

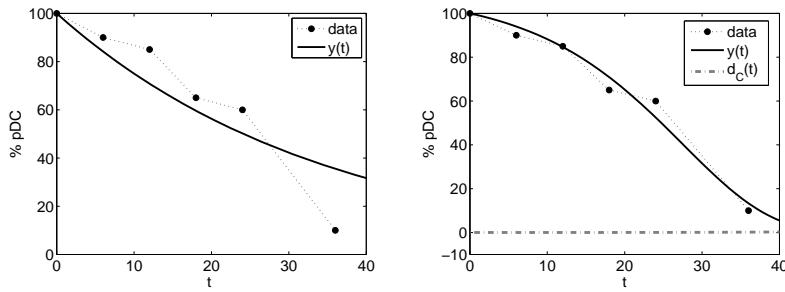


Рис. 4. Данные наблюдений и решения моделей экспоненциальной гибели клеток (слева) и гибели по закону Гомпертца (справа).

В то же время, многие задачи биологии продолжительности жизни моделируются с помощью закона Гомпертца (обозначим G), задаваемого системой уравнений для гибели клеток и изменения параметра скорости гибели:

$$\frac{d}{dt} C(t) = -d_C(t) \cdot C(t), \quad (28a)$$

$$\frac{d}{dt} d_C(t) = k_C \cdot d_C(t). \quad (28b)$$

На рис. 4 представлены экспериментальные данные и решения модели соответствующие оптимальным оценкам параметров этих двух моделей. Величины функционалов невязки в точке минимума, и соответствующих критериев Акаике и длины описания для данных моделей имеют следующие значения:

$$\Phi_E = 1012, \Phi_G = 1012, \mu_E = 38, \mu_G = 30, \Delta_E = 22.1, \Delta_G = 19.5.$$

Таким образом, сочетание критериев близости к данным наблюдений, с информационными критериями, позволяет однозначно идентифицировать модель Гомпертца, как наиболее адекватный закон описания процесса гибели клеток в данной системе.

7. Глобальный анализ чувствительности

Изложенные выше методы анализа чувствительности моделей являлись локальными. В реальных приложениях важным является

ся исследование вариаций глобального поведения модели при одновременном изменении параметров. Глобальный анализ чувствительности многопараметрических моделей можно провести, используя совокупность двух приемов: выборки на основе Латинского гиперкуба и рангового критерия оценки параметров [12, 13]. Результатом такого анализа является ранжирование параметров модели по степени их влияния на переменные модели или функционалы от таковых. Способ выборки по схеме Латинского гиперкуба является разновидностью метода Монте Карло. Каждый анализируемый параметр модели трактуется как случайная величина. Для всех параметров определяется функция распределения, область значения делится особым способом, и затем, параметр выбирается случайным образом. С помощью такой выборки можно получить множество сочетаний значений параметров, таких, что выборочное значение каждого параметра используется только один раз. Далее, модель просчитывается для всех наборов параметров, и полученные значения переменных модели используются для анализа чувствительности (корреляции) каждой полученной переменной и каждого параметра с применением рангового критерия.

7.1. Построение выборки на основе Латинского гиперкуба. Следуя [12, 13], для каждого исследуемого параметра из вектора параметров модели \mathbf{p} необходимо определить область значений и функцию плотности распределения. Далее, выбирается число испытаний N — количество наборов параметров и соответственно число прогонок модели. Строгие критерии для выбора этого числа не существуют, однако опытным путем ранее было получено, что $N > \frac{3}{4}L$. Соответственно, область значений каждого из L параметров делится на N неперекрывающихся, равновероятных отрезков. Далее, составляется таблица выборки на основе Латинского гиперкуба размером $N \times L$ по следующему правилу: на каждом из N интервалов случайным образом, но в соответствии с законом распределения, выбирается значение каждого параметра, после чего, снова случайным образом составляются пары из интервалов для первого и второго параметров, далее из этих пар опять случайным образом получают тройки с третьим параметром и так далее, пока не исчерпаны все параметры. В результате, получаем некоторую матрицу размером $N \times L$, строки которой содержат случайный вектор параметров

\mathbf{r} , а каждый столбец содержит случайные реализации каждого параметра r_ℓ . Последним этапом реализации алгоритма реализации случайной выборки является численный расчет решений модели для всех векторов \mathbf{r} из построенной выборки параметров, с целью получения набора значений исследуемых переменных модели.

Для исследуемой модели системы интерферона область неопределенности значений параметров задавалась в следующем виде: $\frac{1}{2}r_\ell^* \leq r_\ell \leq 2r_\ell^*$ и предполагалось, что параметры распределены по треугольному закону с пиком в точке $\frac{r_\ell^{\max} + r_\ell^{\min}}{2}$. В ходе численного эксперимента N равнялось 500. В качестве исследуемой переменной (целевой функционал) бралось суммарное количество интерферона $I_t \equiv \int_0^T I(t)dt$.

7.2. Ранговый критерий в оценке значимости параметров..
Чтобы ранжировать параметры по степени их влияния на целевой функционал модели, будем использовать следующий статистический критерий. Для множества векторов параметров \mathbf{r} и значений зависимой переменной I_t , полученных в ходе реализации выборки на основе Латинского гиперкуба, составим матрицу $N \times (L+1)$ с данными расчетов по модели $X = \{X_l\}_{l=1}^{L+1}$. Далее, вычислим корреляционную матрицу C , элементы которой C_{ij} являются коэффициентами корреляции между переменными столбцов X_i и X_j :

$$C_{ij} = \frac{(L+1) \sum X_i^k X_j^k - \sum X_i^k \sum X_j^k}{\sqrt{(L+1) \sum (X_i^k)^2 - (\sum X_i^k)^2} \sqrt{(L+1) \sum (X_j^k)^2 - (\sum X_j^k)^2}} \quad (29)$$

Поскольку нас интересует корреляция функционала I_t и параметров модели, определим частичный коэффициент корреляции между I_t и r_ℓ по формуле (C^{-1} – обратная матрица):

$$\sigma_{r_\ell I_t} = -C_{1(L+1)} / (C_{11} C_{(L+1)(L+1)})^{\frac{1}{2}}. \quad (30)$$

По результатам анализа данных коэффициентов параметры модели можно упорядочить по их степени их влияния на изменчивость I_t .

Применение изложенного алгоритма к модели системы интерферона позволяет оценить неопределенность в значении функции от решения модели и случайными значениями параметров. Результаты расчетов для некоторых параметров приведены на рис. 5-6. Сплошные линии, представленные на графиках соответствуют уравнениям регрессии. Интересно, что зависимость I_t от параметра

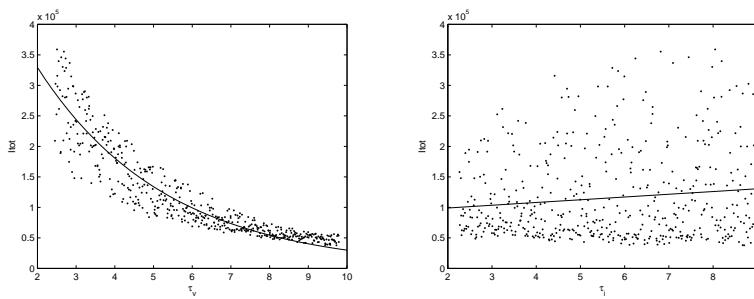


Рис. 5. Результаты численных экспериментов по исследованию чувствительности модели интерферона (зависимая переменная – I_t) к параметрам модели с построением случайной выборки по схеме Латинского гиперкуба. Слева: корреляция I_t с величиной запаздывания τ_V , уравнение регрессии имеет вид $I_t = 6 \times 10^5 \exp^{-0.3 \cdot \tau_V}$. Справа: корреляция I_t с величиной запаздывания τ_I , уравнение регрессии имеет вид $I_t = 90000 + 4500 \cdot \tau_V$.

запаздывания τ_V может быть неплохо описана экспоненциальной кривой, в то время как по остальным параметрам такой явной зависимости нет. Количественно, анализ корреляции приводит к следующим оценкам коэффициентов ранговой корреляции между целевой функцией и параметрами модели, представленными в табл. 3.

Величина коэффициента корреляции характеризует степень влияния неопределенности значения параметра на неточность в оценке целевой функции. Для рассматриваемой модели можно сделать вывод о том, что наиболее значимыми с точки зрения глобальной корреляции суммарного количества интреферона произведенного в ходе инфекции является скорость синтеза интерферона ρ_I и число

Таблица 3. Выборочные коэффициенты ранговой корреляции между I_t и параметрами модели интерферона.

	I_t		I_t		I_t
ρ_I	0.3917	σ_V	0.0684	τ_I	-0.0464
ρ_V	-0.0007	d_{oc}	0.0151	τ_V	-0.0947
θ	0.0286	k_c	0.0266	f_i	0.3154

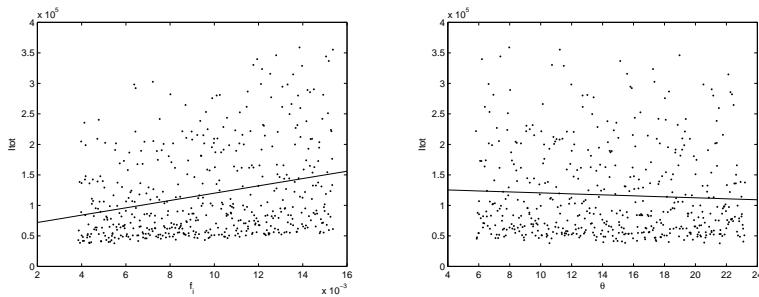


Рис. 6. Результаты численных экспериментов по исследованию чувствительности модели интерферона (зависимая переменная — I_t) к параметрам модели с построением случайной выборки по схеме Латинского гиперкуба. Слева: зависимость I_t от доли первично инфицированных клеток f_i , уравнение регрессии имеет вид $I_t = 60000 + 6 \times 10^6 \cdot f_i$. Справа: зависимость I_t от параметров ингибиции синтеза вирусов θ , уравнение регрессии имеет вид $I_t = 128500 - 803 \cdot \theta$.

первично инфицированных клеток f_i .

8. Заключение: матричный анализ в задачах идентификации

Оценка параметров и идентификация оптимальных моделей является важнейшей прикладной задачей моделирования в биологии. Для её успешного решения требуется развитие эффективных методов решения обратных задач, оценивания параметрической идентифицируемости моделей и информационной сложности моделей. В данной работе представлен набор базовых алгоритмов анализа параметрических моделей в контексте данных наблюдений. Развивающаяся нами вычислительная технология моделирования в иммунологии может быть использована для изучения гораздо более широкого спектра прикладных задач математической биологии. Вопросы эффективной численной реализации соответствующих алгоритмов базируются на широким применением методов матричного анализа и линейной алгебры [15, 16] и требуют дальнейших исследований.

Список литературы

- [1] Бочаров Г.А., Марчук Г.И. Прикладные проблемы математического моделирования в иммунологии. *ЖВМиМФ.* // 2000. 40. 1905–1920.
- [2] Andrew S.M., Baker C.T.H., Bocharov G.A. Rival approaches to mathematical modelling in immunology. // *J. Comput. Appl. Math.* 2007. **205**. 669–686.
- [3] G. Bocharov, L. Cervantes-Barragan, R. Zust, K. Eriksson, V. Thiel, B. Ludewig. Mathematical modeling of the antiviral type I interferon response. // In: Proceedings of the FOSBE 2007 Eds. F. Allgower and M. Reuss. Fraunhofer IRB Verlag. 2007. 325–330.
- [4] L.K. Babadzhanjan, A.A. Voitylov, P. Krebs, B. Ludewig, D.R. Sarkissian, G.A. Bocharov. On primary statistical data processing of experimental measurements of lymphocytes using C57BL/6 mouse line. // В сб. «Устойчивость и процессы управления». Международной конференции посвященной 75-летию В.И. Зубова, под ред. Д.А. Овсянникова и Л.А. Петросяна. Санкт-Петербургский государственный Университет. 2005. Т. 2. С. 1227-1236.
- [5] C.T.H. Baker, G.A. Bocharov, C.A.H. Paul and F.A. Rihan. Computational modelling with functional differential equations: identification, selection and sensitivity. // *Applied Numerical Mathematics.* 2005. **53**. 107–129.
- [6] Christopher T.H. Baker and Gennady A. Bocharov. Computational aspects of time-lag models of Marchuk type that arise in immunology. // *Russ. J. Numer. Anal. Math. Modelling.* 2005. **20**. 247–262.
- [7] C.T.H. Baker, G.A. Bocharov, J.M. Ford, P.M. Lumb, S.J. Norton, C.A.H. Paul, T. Junt, P. Krebs, B. Ludewig. Computational Approaches to Parameter Estimation and Model Selection in Immunology. // *J. Comput. Appl. Math.* 2005. **184**. 50–76.
- [8] Теребиж В.Ю. Введение в статистическую теорию обратных задач. – М.: ФИЗМАТЛИТ. 2005.

- [9] Льюнг Л. Идентификация систем. Теория для пользователя. — М.: Наука. 1991.
- [10] K. Zhen Yao, Benjamin M. Shaw, Bo Kou, Kim B. McAuley, D. W. Bacon. Modeling Ethylene/Butene Copolymerization with Multisite Catalysts: Parameter Estimability and Experimental Design. // *Polymer Reaction Engineering*. 2003. **11**. pp. 563–588.
- [11] Burnham, K.P., Anderson, D.R. Model Selection and Multimodel Inference - a practical information-theoretic approach, 2nd ed. — Springer-Verlag, New York. 3rd printing. 2004.
- [12] Ronald L. Iman, Jon C. Helton. An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models. // *Risk Analysis* 1988. **8** (1). 71–90.
- [13] S. M. Blower. H. Dowlatabadi. Sensitivity and Uncertainty Analysis of Complex Models of Disease Transmission: An HIV Model, as an Example. // *International Statistical Review / Revue Internationale de Statistique*. 1994. **62**. pp. 229–243.
- [14] Hanson A.J., Fu P.C-W. Applications of MDL to selected families of models. In: Advances in Minimum Description Length Theory and Applications. Ed. by P.D. Grünwald, I.J. Myung, M.A. Pitt. The MIT Press. Cambridge MA. 195–150. 2005.
- [15] Воеводин В.В. Численные методы линейной алгебры (теория и алгоритмы). —М.: Наука. 1966.
- [16] Дж. Голуб., Ч. Ван Лоун. Матричные вычисления. —М.: Мир. 1999.

Архитектура и принципы реализации коллективного банка тестов в сети Интернет

Вад. В. Воеводин*, С. И. Соловьев*, А. В. Фролов[®]

В данной статье описываются технологические основы и архитектура системы, предназначеннной для формирования в сети Интернет коллективного банка тестов по некоторой предметной области. Главной особенностью системы является возможность объединения работы многих специалистов-профессионалов для создания в распределенном режиме высококачественного набора тестов, вопросов и упражнений, пригодных как для проведения тестирования уровня подготовки учащихся, так и для использования в режиме тренажера. Оболочка системы не зависит от предметной области и может быть использована для поддержки учебного процесса по самым разным дисциплинам, а первая версия системы будет ориентирована на параллельные вычисления.

1. Введение

Современный мир становится все более «компьютерным», а компьютерный мир — все более «параллельным». Многоядерные процессоры, кластерные системы, распределенные вычисления и grid-технологии становятся доступными, поэтому хорошие специалисты по параллельным вычислениям уже на данный момент чрезвычайно востребованы, и со временем необходимость в них будет только расти. Подготовка таких специалистов является очень важной задачей, и одной из составляющих данного процесса является проведение те-

*Московский государственный университет им. М. В. Ломоносова

*Научно-исследовательский вычислительный центр МГУ

[®]Институт вычислительной математики РАН

стирований. Данная статья содержит описание разрабатываемой системы коллективного банка тестов, целью которой является создание Интернет-ресурса, предоставляющего набор тестов, вопросов и упражнений по некоторой предметной области.

Надо отметить, что системы, позволяющие проходить тестирование по определенной тематике, существуют уже достаточно давно [1, 2]. Однако у подобных систем есть большой недостаток: созданием вопросов занимается некая определенная, и обычно небольшая, группа людей, что не позволяет в полной мере охватить представленную в тесте область знаний. В описываемой системе предлагать свои вопросы может любой квалифицированный специалист, что позволяет расширить описание предметной области силами многих специалистов. Системы, предлагающие схожий подход, то есть реализующие наполнение ресурсов силами многих пользователей, уже существуют, и самой известной среди них является набор программ Wiki, положенных в основу энциклопедии Wikipedia, на примере которой можно увидеть, насколько такие системы могут быть востребованными.

Предполагается, что описываемая в данной работе система будет апробирована на параллельных вычислениях, однако оболочка системы проектируется независимой от предметной области.

2. Общие принципы работы системы

Коллективный банк тестов является хранилищем тестов и вопросов, из которых эти тесты состоят, и предоставляет возможности для пополнения и использования хранимой информации. Данные возможности включают в себя создание и сохранение вопросов в хранилище, подбор локального теста из сохранных вопросов (при этом он может быть использован только тем пользователем, который его создал), создание так называемых эталонных тестов, которые являются типовыми для данной области знаний и доступны всем, а также прохождение созданных тестов и оценивание полученных результатов.

Перечисленные в таком порядке пункты соответствуют алгоритму использования данной системы: сначала создаются вопросы, потом из них формируются тесты, которые в конечном итоге используются в процессе тестирования.

Коллективный банк тестов также содержит описание структуры предметной области. Вся область представлена в виде трехуровневого дерева, в листьях которого хранятся вопросы. Корень этого дерева — это сама область. Она состоит из разделов (1-й уровень), те в свою очередь состоят из глав (2-й уровень), а главы — из параграфов (3-й уровень). Непосредственно сами вопросы находятся на самом нижнем уровне. Как показала практика [3, 4], многие предметные области могут быть представлены в виде указанного дерева, а жесткая структуризация на не более чем три уровня не является сильным ограничением.

Под вопросом понимается обычное задание: есть формулировка вопроса, есть варианты ответа на него, и необходимо выбрать правильный вариант. В первой версии системы вопросы могут быть одного из трех типов: выбор единственного варианта из многих, множественный выбор и ввод значений. Вместе с этим, система сразу проектируется таким образом, чтобы в дальнейшем можно было было легко добавить новые типы, например, выбор областей на графическом изображении.

Тест представляет собой упорядоченное множество вопросов, снабженное некоторым набором параметров, таких как время, отведенное на прохождение теста, или критерии оценивания результатов тестирования.

Все люди, вовлеченные в работу с системой, в зависимости от стоящих перед ними задач могут быть разделены на несколько групп: Эксперт, Редактор, Преподаватель, Студент, Администратор, Гуру.

Каждому участнику системы предоставляется некоторый набор функций для работы с системой (общая схема показана на рис. 1).

Основная задача Эксперта — это сформулировать вопрос и предложить его для включения в основную базу знаний. Экспертов может быть много, работают они независимо друг от друга через Интернет, и именно на данном этапе реализуется указанное выше важное свойство системы, позволяющее каждому прошедшему регистрацию специалисту добавлять вопросы для тестов. Эксперт составляет вопрос, определяет его параметры и отправляет на рассмотрение Редактору.

Редактор рассматривает этот вопрос и либо отказывает, либо разрешает его занесение в основную базу знаний. В последнем слу-

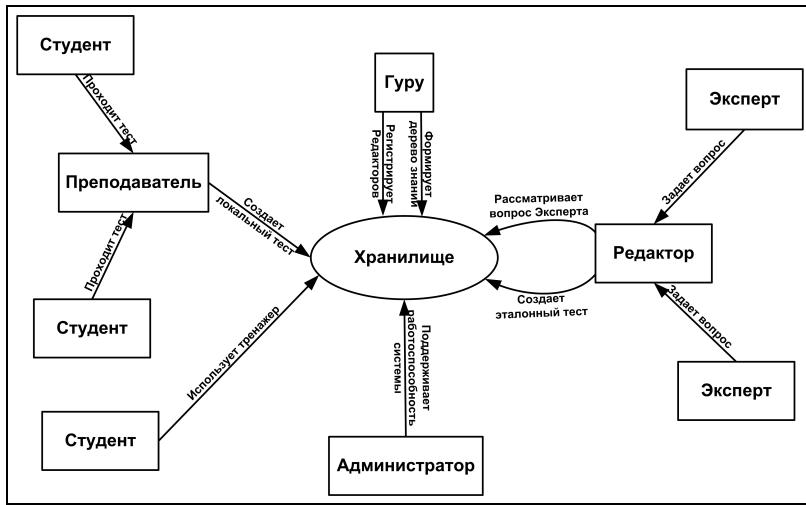


Рис. 1. Общая схема работы системы

Чае Редактор имеет возможность предварительно изменить любой атрибут, будь то формулировка самого вопроса, количество ответов, положение в дереве знаний или что-то иное. По результатам работы Редактора Эксперт получает ответ, был ли принят его вопрос или нет, и если вопрос был принят, то вносил ли Редактор в него какие-либо корректизы.

Помимо этих функций, Редактор занимается созданием эталонных тестов, то есть типовых тестов, доступных всем. Такие тесты обязаны быть аккуратно составлены, поэтому только Редакторы могут их создавать, так как эта группа действующих лиц системы состоит из наиболее квалифицированных специалистов в данной области.

После того, как вопрос был создан Экспертом, а затем принят Редактором и включен в банк, его может использовать Преподаватель. Преподаватель набирает вопросы для теста из основной базы знаний или создает свои, доступные только ему локальные вопросы, и определяет параметры прохождения этого теста. Также он описывает и регистрирует в системе группу Студентов, после чего назначает этой группе прохождение созданного им теста. Для создания сво-

их локальных вопросов ему предоставляются права Эксперта. Все созданные Преподавателем сущности видны только ему, за одним исключением: поскольку он пользуется возможностями Эксперта, то имеет возможность предлагать вопросы для занесения в основную базу знаний. Его собственные тесты при этом всегда остаются локальными и не могут быть использованы другими.

Каждый *Студент* принадлежит одной и только одной группе. После того, как Преподаватель назначил данной группе тест, Студенты в отведенное для этого теста время начинают его прохождение. По завершению тестирования введенные Студентом ответы сохраняются в хранилище, и Преподаватель может просмотреть результаты каждого участника по этому тесту.

Администратор выполняет чисто технические функции. Основной его задачей является поддержание работоспособности банка тестов и всех сопутствующих технологических компонент. Поскольку предполагается, что создаваемая система будет доступна не только в Интернете, а будет поддерживать и возможность установки в локальных сетях, то Администратор отвечает и за начальное развертывание системы.

Если Администратор отвечает за техническую составляющую системы, то *Гуру* определяет ее содержательную часть. В его компетенции находятся, прежде всего, работа с Редакторами и описание структуры предметной области. Работа с Редакторами заключается в их назначении и контроле над их деятельностью. Структура предметной области задается в виде описанного выше дерева, что позволяет каждый заносимый в банк вопрос однозначно приписать к одной из его вершин. Регистрацию Гуру в системе осуществляет Администратор.

Описанная выше схема работы системы предназначена для тестирования — режима, в котором из заданных вопросов создается тест, используемый в дальнейшем для обучения и оценки знаний Студента. Однако систему можно также использовать для тренировки: в этом режиме пользователю предоставляется набор вопросов по всем темам, на которые он может в произвольном порядке отвечать. На каждом вопросе, после того, как пользователь предложит свой вариант, ему показывается правильный ответ. Режим тренажера, в отличие от тестирования, доступен любому пользователю системы,

в том числе и незарегистрированному.

3. Технологические основы построения системы

В основу системы положена клиент-серверная технология: вся система целиком будет установлена на сервере, на который будут поступать запросы с клиентской стороны. Основной режим работы — это распределенная работа пользователей через Интернет по стандартному протоколу HTTP, однако также предусматривается установка системы в локальных сетях и распространение на CD-дисках для установки системы на отдельный компьютер. В последнем случае будет предоставляться не вся система, а только часть, необходимая для использования режима тренажера. Исходя из вышесказанного, было поставлено требование независимости системы от платформы на стороне клиента, а также отсутствия предварительной настройки и/или установки программ, что послужило причиной для использования следующих программных технологий и продуктов:

- веб-сервер Apache;
- технологии программирования PHP и JavaScript;
- СУБД MySQL.

При использовании таких технологий на стороне клиента должен быть установлен только браузер с поддержкой JavaScript, при этом не накладывается никаких ограничений на используемое аппаратное и программное обеспечение. Связка данных технологий хорошо зарекомендовала себя во множестве Интернет-проектов. Она обладает достаточной для проекта функциональностью, хорошей производительностью и масштабируемостью. Все программные компоненты, входящие в предлагаемый набор, распространяются свободно.

4. Заключение

На данный момент система находится в стадии реализации. Разработана структура предметной области, сформулированы первые наборы вопросов, проходит отладка протоколов взаимодействия действующих лиц внутри системы и определение внешних интерфейсов для их работы. Эта система будет доступна в сети Интернет и первой

областью, на которой будут апробированы описанные здесь идеи, станут параллельные вычисления.

Список литературы

- [1] Online тестирование и сертификация. Интернет-ресурс <http://tests.specialist.ru/>.
- [2] Интернет Университет Информационных Технологий. <http://intuit.ru/>
- [3] Воеводин В.В., Воеводин Вл.В. Энциклопедия линейной алгебры. Электронная система ЛИНЕАЛ - СПб.: БХВ-Петербург, 2006. - 544 с.,
- [4] Базовая электронная энциклопедия по линейной алгебре. Интернет-ресурс. <http://lineal.guru.ru>.

Структура и производительность подсистем памяти современных вычислительных платформ

П. А. Гаврилушкин*

В статье описывается архитектура подсистем памяти современных вычислительных платформ. Рассматривается взаимосвязь эффективности выполнения программ и особенности организации работы с памятью.

1. Введение

Скорость работы процессора несравненно выше скорости работы подсистемы памяти. Эта асимметричность обязывает разработчиков программного обеспечения очень тщательно подходить к налаживанию взаимодействия между процессором и памятью. На всех современных компьютерных платформах время ожидания данных из памяти на порядок больше времени исполнения операции, что не может не сказаться на времени исполнения программы в целом. Каким же образом можно повысить эффективность работы с подсистемой памяти? Во-первых, это возможность установки более мощной аппаратной базы: процессор с более быстрым и большим кэшем, материнская плата с более производительными каналами связи и контроллером памяти, высокоскоростные модули оперативной памяти и внешние источники данных. И во-вторых, это соответствие алгоритма архитектуре памяти.

2. Архитектура памяти

Запоминающие устройства могут быть разделены как по назначению, так и по физическим принципам построения. Если не брать в

*Научно-исследовательский вычислительный центр МГУ

рассмотрение специальную память, как то: постоянную (ROM), перепрограммируемую (Flash), энергонезависимую память, применяемую для хранения установок BIOS (CMOS RAM), видеопамять, то остальная память компьютера соответствует иерархической структуре (рис. 1). И для нее справедливо следующее правило: процессор и каждый из уровней иерархии может обращаться на чтение и запись только к ближайшему снизу уровню. Причем, более высокий уровень имеет меньший объём, большую стоимость и обладает большим быстродействием.

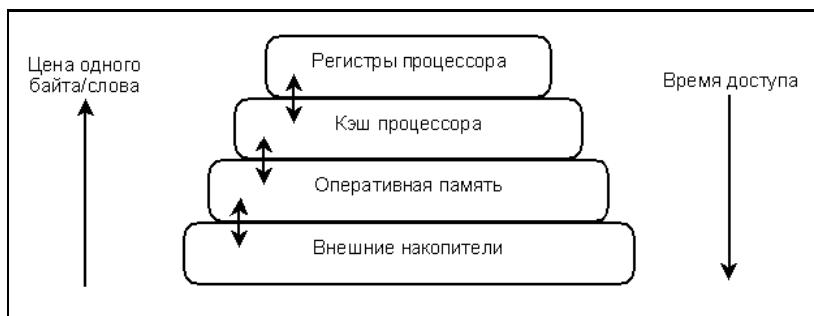


Рис. 1. Иерархия памяти

Рассмотрим основные параметры подсистемы памяти. У каждого запоминающего устройства (ЗУ) есть такие параметры, как:

- латентность — время ожидания (в мс. или тактах) реакции ЗУ на поступивший запрос;
 - объём — максимальное количество одновременно хранимой информации (байт) в ЗУ;
 - единица доступа — минимальное количество байт, которое можно получить за одно обращение к ЗУ;
 - адресуемость — возможность обращения к данных в ЗУ по адресу;
 - продолжительность сохранения — постоянная (ПЗУ) или временная (ОЗУ);

- энергопотребление — количество потребляемой ЗУ энергии на операции чтения, хранения и записи.

В процессе развития аппаратного обеспечения компьютеров латентность постепенно уменьшается, а объём увеличивается. С точки зрения программного обеспечения, необходимо отметить свойства локальности исполняемых команд и используемых данных. На практике, это означает что выборка очередной команды (порция данных) из всех команд (данных) происходит не равновероятно, а в зависимости от предыдущей. Балансировка соотношения стоимости/производительности и принцип локальности обращений привели к иерархии памяти (быстрая память маленького объёма, а более медленная — большого). Рассмотрим подробнее каждый из уровней иерархии, начиная с ближайшего к процессору.

3. Регистровая память

На вершине иерархии стоит регистровая память. Чтение и запись в нее происходит на частоте процессора и с нулевой латентностью. Каждому регистру соответствует уникальное имя, по которому можно обратиться как по обычному адресу. Количество регистров и их структура отличаются для разных архитектур процессоров: x86, x86-64, PowerPC и другие. Современные процессоры с архитектурой x86-64 обладают 64-битными регистрами данных, регистрами указателей, сегментными регистрами, регистрами флагов, а также служебными регистрами. Общий объем регистровой памяти исчисляется сотнями байт. Для повышения производительности программ используются расширенные наборы регистров, например восемь MMX-регистров размером по 64 бита.

4. Кэш-память

Следующий уровень в иерархии занимает кэш-память или просто кэш. Он может быть подразделён на уровни (L1, L2, L3), также состоящие в иерархии. L1-кэш является неотъемлемой частью процессора, работает с ним на одной частоте (частоте ядра), обладает небольшим объёмом (до 128Кб) и латентностью в 2-4 такта. Обычно он состоит из кэша для данных (L1D) и кэша для инструкций (L1I). Их объемы и латентности могут различаться между собой. Кэши

L2 и L3 имеют больший размер (от мегабайта), доступ к ним характеризуется ещё большей латентностью (десятки тактов) и, в отличие от L1, могут быть отчуждены от процессора (не присутствовать вообще, не присутствовать на кристалле, иметь возможность программного отключения). Кэш L3 используется, в основном, в серверном сегменте (процессоры серии Xeon, PowerPC, UltraSPARC T). Если в одном процессоре присутствует несколько ядер или в системе несколько процессоров, то L1 всегда свой для каждого ядра. Кэши L2 также могут присутствовать по одному на каждое ядро, а могут быть динамически разделяемыми между несколькими ядрами.

Кэш-память формируется из строк (lines). Каждая строка, с учётом локальности использования данных, соответствует группе соседних байт основной памяти. Это соответствие определяется адресными полями. Адресные поля хранятся для строк целиком, а не для отдельных байт, тем самым достигается экономия по занимаемому ими месту.

В отличие от остальных запоминающих устройств, кэш не является адресуемым. Обращение к данным кэша происходит неявно и не по внутреннему адресу, а по адресу в оперативной памяти. Для чтения из ячейки, имеющей некоторый адрес в оперативной памяти, необходимо осуществить поиск по соответствующему адресу и в кэше. Успешный поиск называется попаданием (*cache hit*), а неуспешный — промахом (*cache miss*). Латентность при чтении данных, уже находящихся в нём, отличается на порядок от латентности чтения данных, требующих предварительной подкачки из кэшей более низких уровней или основной памяти. Количество неуспешных поисков напрямую влияет на эффективность исполнения программы, так как за каждый промах приходится платить увеличением латентности (*miss penalty*).

Существует три типа [1] кэш-памяти, отличающихся по способу размещения данных основной памяти:

- Кэш прямой адресации (Direct Mapped Cache). Кэш, в котором каждому возможному значению поля виртуального адреса соответствует ровно одна строка, определяемая по значению этого поля (виртуальный адрес = младшие биты адреса в памяти). Такая организация неизбежно влечёт отображение различных ячеек памяти с одинаковыми младшими частями

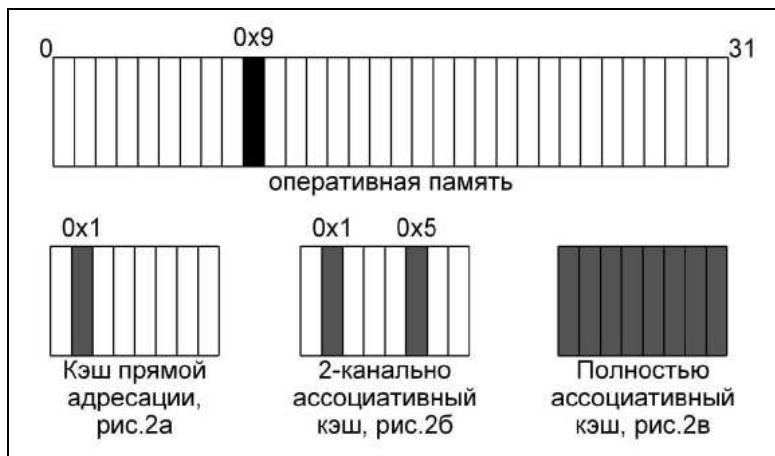


Рис. 2. Ассоциативность кэша. В верхней части рисунка изображен участок основной памяти размером в 32 байта. А в нижней части различные варианты организации кэш-памяти

адресов в одну строку кэша, то есть коллизия (collision). Этот пример иллюстрирован на рис.2а: адреса 0x9 и 0x17 имеют три одинаковых младших бита, и поэтому оба отобразятся в виртуальный адрес 0x1.

- Полностью ассоциативный кэш (Fully Associative Cache). Этот тип кэш-памяти решает проблему коллизий, которая характерна для кэша прямой адресации. В случае полностью ассоциативного кэша блок данных из любого адреса оперативной памяти может храниться в любой строке кэша, а сам адрес используется как тег кэша; при проверке на совпадение все теги должны одновременно сравниваться с адресом запроса, что требует дополнительного усложнения аппаратуры. Для устранения проблемы некогерентности используется ещё один тег — тег актуальности данных. На рис.2в серым цветом отмечены ячейки, в которые может попасть ячейка с адресом 0x9 при занесении. Для полностью ассоциативного кэша — это все доступные ячейки.

- N-канально ассоциативный кэш (N-Way Set Associative Cache). Строки N-канально ассоциативного кэша делятся на секторы, или секции (sets). Информация по некоторому адресу оперативной памяти может храниться в N местах кэш-памяти. Этот способ адресации позволяет получить значительное преимущество по скорости перед кэшем прямой адресации, но имеет гораздо более простую аппаратную реализацию, нежели чем полностью ассоциативный кэш. На рис. 26 серым цветом показаны два возможных места хранения ячейки с адресом 0x9 при занесении. Это адреса 0x1 и 0x5, обладающие смещением 0x1 в каждом из каналов кэша.

Запись ячейки в кэш, по своей сути, никогда не требует подкачки данных из основной памяти или других уровней кэша, поэтому выполняется за гарантированное время. Но и здесь есть свои тонкости, а именно: необходимость поддержания актуальности (ко-герентности) данных как на различных уровнях кэша, так и в основной памяти. Обеспечение когерентности решается аппаратно на уровне контроллера кэша. Либо когерентность соблюдается всегда, либо при возникновении конфликта, всё зависит от политики записи при кэшировании.

Существуют две основные [2] политики записи при кэшировании:

- Сквозная запись (write-through). Подразумевает, что при изменении содержимого ячейки памяти, запись происходит синхронно и в кэш, и в основную память. Такая политика используется в L1 кэше процессоров Intel Pentium 4, Intel Pentium D.
- Отложенная запись (write-back). Подразумевает, что можно отложить момент записи циркованных данных в основную память, записав их только в кэш. При этом модифицированные данные будут выгружены в оперативную память только в случае обращения к ним какого либо другого устройства (другое ядро ЦП, контроллер DMA) либо нехватки места в кэше для размещения других данных. Такая политика используется в L1 кэше процессоров семейства Intel Core, AMD Athlon64.

Современные алгоритмы вытеснения малоиспользуемых данных и предвыборки следующих команд позволяют достичь уровня попа-

даний порядка 90% при соблюдении локальности обращений. Примером снижения (сильного снижения) производительности при отсутствии свойств локальности обращений может служить последовательная выборка (выборка через интервал, на единицу больший длины строки) из памяти элементов массива, целиком не умещающегося в кэш. Типичные характеристики кэшей процессоров серии Intel Core 2 Duo: 32+32КБ (кэш данных+кэш инструкций) 8-канально ассоциативного кэша первого уровня с размером строки равным 64 байта и политикой сквозной записи в менее скоростную память, 4096КБ 16-канально ассоциативного кэша второго уровня с размером строки равным 64 байта и политикой отложенной записи в менее скоростную память, кэш третьего уровня не поддерживается.

5. Оперативная память

Рассмотрим в хронологическом порядке технологии, применяемые в современной оперативной памяти DDR SDRAM (синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных).

Память с произвольным доступом (RAM). Один из видов памяти, позволяющий обращаться к данным на чтение и запись, не учитывая порядок их расположения.

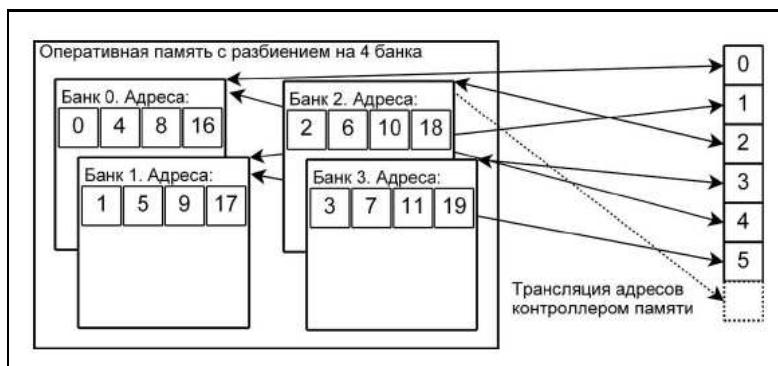


Рис. 3. Разбиение на банки

Динамическая память с произвольным доступом (DRAM) — это один из видов памяти с произвольным доступом. Динамическая па-

№ сервера	Размер L1 (Кб)	Ассоциативность L1	Размер строки L1	Размер L2 (Мб)	N-канальная ассоциативность L2	Размер строки L2	Частота памяти (МГц)	Частота шины памяти (МГц)
1	64+64	2	64	1+1	16	64	200	200
2	64+64	2	64	1+1	16	64	200	200
3	32+32	8	64	4	16	64	133	266
4	32+32	8	64	4	16	64	133	266
5	32+32	8	64	4	16	64	166	333
6	32+32	8	64	4+4	16	64	166	333
7	64+64	2	64	1	16	64	166	166
8	16+16	8	64	2	8	64	133	266

Таблица 1. Характеристики и структура памяти современных платформ

память является энергозависимой и нуждается в периодической регенерации данных.

Структура динамической памяти с произвольным доступом следующая: группы битовых ячеек образуют строки, группы строк образуют страницы, группы страниц образуют банки. Для ускорения доступа к данным запоминающего устройства ячейки с соседними адресами хранятся в различных банках, обеспечивая снижение влияния задержек регенерации.

Основными характеристиками памяти DRAM являются тайминги (timings) и рабочая частота. Для обращения к ячейке контроллер задаёт номер банка, номер страницы в нём, номер строки и номер столбца. Каждое из этих действий исполняется за некоторое время — тайминг. Основными таймингами DRAM [3] являются:

1. Задержка между подачей номера столбца и получением содержимого ячейки, называемая временем рабочего цикла (CAS).

№ сервера	Латентность L1 (такты)	Латентность L2 (такты)	Латентность памяти(такты)	Тайминги памяти (CAS, RAS to CAS, RAS, tRAS)	Средняя пропускная способность памяти при чтении (ГБ/с)	Максимальная пропускная способность памяти при чтении (ГБ/с)
6	3	14	140	5-5-5-15	5,27	6,47
7	3	17	144	2,5-3-3-7	3,03	5,03
8	4	28,5	106	4-3-3-8	5,06	6,37

Таблица 2. Временные характеристики памяти и реальная пропускная способность по результатам тестов [4]

2. Задержка между подачей номера строки и номера столбца, называемая временем полного доступа (RAS to CAS).
3. Задержка между чтением последней ячейки и подачей номера новой строки (RAS precharge).
4. Задержка для регенерации строки памяти (RAS Active to Precharge Delay, tRAS).

Эти величины измеряются в тактах, и чем они меньше, тем быстрее работает оперативная память. Записываются все четыре значения через дефис, например 3-4-4-8.

Синхронная динамическая память с произвольным доступом (SDRAM). В отличие от других типов DRAM, использовавших асинхронный обмен данными, ответ на поступивший в устройство управляющий сигнал возвращается не сразу, а лишь при получении следующего тактового сигнала. Тактовые сигналы позволяют организовать работу SDRAM в виде конечного автомата, исполняющего

команды. При этом команды могут поступать в виде непрерывного потока, не дожидаясь, пока будет завершено выполнение предыдущих инструкций (конвейерная обработка): сразу после команды записи может поступить следующая команда, не ожидая, когда данные окажутся записаны. Поступление команды чтения приведёт к тому, что на выходе данные появятся с некоторой задержкой.

Синхронная динамическая память с произвольным доступом и удвоенной скоростью передачи данных (Double Data Rate SDRAM). При использовании DDR SDRAM достигается большая полоса пропускания, нежели в обыкновенной SDRAM. Передача данных осуществляется по обоим фронтам синхросигнала, за счёт этого фактически удваивается производительность, не увеличивая при этом частоты шины памяти. Следующие поколения технологии DDR, такие как DDR2, DDR3, позволяют снизить энергопотребление наряду с увеличением рабочих частот, ценой увеличения таймингов. При этом никаких принципиально новых решений они не привносят. В связи с этим, нельзя однозначно показать превосходство чипов памяти DDR3, работающих на высоких частотах — нужно в каждом конкретном случае производить тестовые замеры производительности, учитывая задержки и рабочие частоты.

6. Характеристики подсистем памяти современных серверов

Приведем характеристики подсистем памяти нескольких серверов, находящихся в процессорном полигоне НИВЦ МГУ:

1. Процессор: AMD Opteron 265 1.8 ГГц (два ядра, FSB 1000, 1МБ L2); Материнская плата: Thunder K8SD Pro на чипсете AMD 8131; Память: 2Гб PC3200;
2. Процессор: AMD Opteron 280 2.4 ГГц (два ядра, FSB 1000, 1МБ L2); Материнская плата: Thunder K8SD Pro на чипсете AMD 8131; Память: 4Гб PC3200;
3. Процессор: Intel Xeon 5050 3.0 ГГц (два ядра Dempsey, FSB 667, 4МБ L2); Материнская плата: Intel «Alcolu» 5000P; Память: 4Гб PC2-4200;

4. Процессор: Intel Xeon 5150 2.66 ГГц (два ядра Woodcrest, FSB 1333, 4МБ L2); Материнская плата: Intel «Alcolu» 5000P; Память: 16Гб PC2-4200;
5. Процессор: Intel Xeon 5355 2.66 ГГц (четыре ядра Clowertown, FSB 1333, 8МБ L2); Материнская плата: Intel «Alcolu» 5000P; Память: 6Гб PC2-5300;

Также рассмотрим тесты пропускной способности подсистем памяти следующих серверов:

6. Процессор: Intel Xeon 2.0 ГГц (два ядра Woodcrest, FSB 1333, 4МБ L2); Материнская плата: Intel «Alcolu» 5000P; Память: 8Гб PC2-5300;
7. Процессор: AMD Opteron 244 (FSB 1800, 1МБ L2); Материнская плата: ASUS SK8N на чипсете nForce3; Память: 2Гб PC2700;
8. Процессор: Intel Pentium 4 3.6 ГГц (ядро Prescott, FSB 800, 2МБ L2); Материнская плата: Gigabyte 8AENXP-D на чипсете Intel 925XE; Память: 1Гб PC2-4200.

Для тестирования использовалась программа RightMark Memory Analyzer [5], осуществляющая чтение данных в двух вариантах: измерение предельной пропускной способности памяти с использованием оптимизаций чтения типа предвыборки (последний столбец) и измерение средней реальной пропускной способности без оптимизаций (предпоследний столбец). Как видно из сравнительной таблицы, отличия по строкам в пропускной способности памяти при чтении обуславливаются различиями в аппаратных конфигурациях. Но, что более важно, отличия в двух последних столбцах является следствием программного улучшения, которое даёт сравнимый прирост в рамках одной конфигурации!

Список литературы

- [1] Cache mapping and associativity,
<http://www.pcguide.com/ref/mbsys/cache/funcMapping-c.html>.
- [2] Cache write policy,
<http://www.pcguide.com/ref/mbsys/cache/funcWrite-c.html>.

- [3] DDR SDRAM, http://en.wikipedia.org/wiki/DDR_SDRAM.
- [4] Детальное исследование платформ с помощью тестового пакета RightMark Memory Analyzer,
<http://www.ixbt.com/cpu/rmma-k7-k8.shtml>,
<http://www.ixbt.com/cpu/rmma-prescott2.shtml>.
- [5] RightMark Memory Analyzer,
http://cpu.rightmark.org/products/rmma_rus.shtml.

Об оценке сходимости метода бидиагонализации[§]

С. А. ГОРЕЙНОВ[®]

Получена оценка погрешности матричной билинейной аппроксимации, получаемой методом бидиагонализации в условиях работы в точной арифметике, являющаяся более точной, чем применение к данной задаче аппроксимации известных результатов о сходимости метода Ланцоша.

Пусть $A \in \mathbb{C}^{m \times n}$ — заданная матрица. Рассмотрим задачу отыскания $\min_{U, V} k$, где $U \in \mathbb{C}^{m \times k}$, $V \in \mathbb{C}^{n \times k}$ удовлетворяют условию $\|A - UV^*\|_F \leq \varepsilon$. Матричные приближения такого сорта, называемые билинейными — весьма полезный инструмент при аппроксимации больших плотных матриц, имеющих отношение к так называемым асимптотически гладким функциям и нелокальным (интегральным, интегро-дифференциальным) операторам, возникающим в задачах математической физики [11, 3]. Для „хороших“ матриц A ранг билинейной аппроксимации k значительно меньше m и n . В связи с этим были развиты методы, позволяющие получать билинейные аппроксимации таких матриц за линейное по совокупности m и n время [12, 9]. В данной работе обсуждается метод бидиагонализации [4], который применим к более широкому классу матриц, но имеет время работы $O(mn)$. Основной результат — оценка погрешности получаемой аппроксимации в условиях работы в точной арифметике, являющаяся более точной, чем применение известных результатов о сходимости метода Ланцоша [7, 8] к нашей задаче.

[§]Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (гранты РФФИ №№05-01-00721 и 06-01-08052) и программы фундаментальных исследований отделения математических наук РАН «Вычислительные и информационные проблемы решения больших задач» по проекту «Матричные методы и технологии для задач со сверхбольшим числом неизвестных».

[®]Институт вычислительной математики РАН

Вот решение нашей задачи в терминах сингулярного разложения исходной матрицы A .

Теорема 1 (Шмидт, Мирский [10]). *Рассмотрим сингулярное разложение матрицы $A \in \mathbb{C}^{m \times n}$:*

$$A = U\Sigma V^*, \quad U^*U = V^*V = I, \quad \Sigma = \text{diag}(\sigma_1, \dots, \sigma_p),$$

где $p = \min(m, n)$, $U \in \mathbb{C}^{m \times p}$, $V \in \mathbb{C}^{n \times p}$, и сингулярные числа $\{\sigma_k\}$ упорядочены по невозрастанию. Обозначим символами U_k , V_k матрицы, составленные из первых k столбцов U и V . Пусть также Σ_k обозначает ведущую подматрицу Σ порядка k . Тогда

$$\min_{B: \text{rank } B=k} \|A - B\|_F^2 = \|A - U_k \Sigma_k V_k^*\|_F^2 = \sigma_{k+1}^2 + \dots + \sigma_p^2. \quad (1)$$

Таким образом, первые k сингулярных векторов и чисел, где k — минимальный индекс, обеспечивающий неравенство $\sigma_{k+1}^2 + \dots + \sigma_p^2 \leq \varepsilon$, образуют искомую билинейную аппроксимацию A .

Один из известных способов (приближенного) отыскания нескольких старших собственных векторов и чисел эрмитовой матрицы — итерационный алгоритм Ланцша [2]. Если применить его к матрице A^*A и преобразовать с тем, чтобы в расчетных формулах выражение A^*A не возникало¹, получится алгоритм бидиагонализации, используемый для (приближенного) отыскания нескольких старших сингулярных векторов и чисел A . Выпишем управляющие соотношения обоих методов.

Пусть $q_1, q_2, \dots, q_k, \dots$ — векторы процесса Ланцша, пусть Q_k обозначает матрицу, составленную из первых k этих векторов, и пусть

$$T_k = \begin{pmatrix} \omega_1 & t_1^* & & \\ t_1 & \ddots & \ddots & \\ & \ddots & \ddots & t_{k-1}^* \\ & & t_{k-1} & \omega_k \end{pmatrix}$$

обозначает соответствующую трехдиагональную матрицу. Тогда имеет место соотношение

$$A^* A Q_k = Q_k T_k + q_{k+1} t_k e_k^T, \quad (2)$$

¹ В противном случае имеется нежелательный эффект при реальных расчетах, характеризуемый фразами „возвведение обусловленности в квадрат“ или „извлечение квадратного корня из машинного ε “.

причем $Q_k^* Q_k = I$, $Q_k^* q_{k+1} = 0$.

Чтобы перейти к алгоритму бидиагонализации, введем разложение Холецкого $T_k = R_k^* R_k$, где матрица

$$R_k = \begin{pmatrix} \rho_1 & \tau_1^* \\ \ddots & \ddots \\ & \ddots & \tau_{k-1}^* \\ & & \rho_k \end{pmatrix}$$

верхняя треугольная (бидиагональная), и обозначим $P_k = A Q_k R_k^{-1}$. В силу (2) матрица P_k имеет ортонормированные столбцы. Подставляя P_k в (2) и умножая полученное равенство на R_k^{-1} , получаем соотношения алгоритма бидиагонализации:

$$\begin{cases} A Q_k = P_k R_k, \\ A^* P_k = Q_k R_k^* + q_{k+1} \tau_k e_k^*, \end{cases} \quad (3)$$

где $\tau_k = t_k \rho_k^{-1}$.

Соотношения (3) „наращиваются“ точно так же, как (2): скалярные величины τ_k вместе с q_{k+1} можно вычислять нормализацией вектора $A^* p_k - q_k \rho_k^*$, а ρ_k вместе с p_k получаются нормализацией вектора $A q_k - p_{k-1} \tau_{k-1}^*$.

Знаки комплексного матричного сопряжения, стоящие кое-где в наших формулах при скалярных величинах t_k , τ_k , ρ_k , означают, что формулы верны и для блочных вариантов рассматриваемых методов [13, 4].

Сходимость метода бидиагонализации, используемого для нахождения билинейного приближения A , контролируется на практике весьма просто.

Лемма 1 ([5]).

$$\|A - P_k R_k Q_k^*\|_F^2 = \|A\|_F^2 - \|R_k\|_F^2. \quad (4)$$

Доказательство. В силу первой формулы (3)

$$A - P_k R_k Q_k^* = A - A Q_k Q_k^* = A Q_{kO} Q_{kO}^*,$$

где Q_{kO} — дополнение Q_k до квадратной унитарной матрицы Q порядка n . Дополнив таким же образом P_k , запишем

$$A [Q_k \quad Q_{kO}] = [P_k \quad P_{kO}] \begin{bmatrix} R_k & S \\ 0 & T \end{bmatrix},$$

где $S = P_k^* A Q_{kO}$ и $T = P_{kO}^* A Q_{kO}$. Поэтому, в силу унитарной инвариантности и аддитивности по подматрицам фробениусовой нормы,

$$\begin{aligned} \|A - P_k R_k Q_k^*\|_F^2 &= \|A Q_{kO} Q_{kO}^*\|_F^2 = \text{tr}(A Q_{kO} Q_{kO}^* Q_{kO} Q_{kO}^* A^*) \\ &= \|A Q_{kO}\|_F^2 = \|P_k S + P_{kO} T\|_F^2 \\ &= \text{tr}(P_k S + P_{kO} T)^*(P_k S + P_{kO} T) \\ &= \|S\|_F^2 + \|T\|_F^2 = \left\| \begin{bmatrix} R_k & S \\ 0 & T \end{bmatrix} \right\|_F^2 - \|R_k\|_F^2. \quad \square \end{aligned}$$

Следствие 1.

$$\|A - P_k R_k Q_k^*\|_F^2 = \text{tr}(A^* A) - \text{tr} T_k. \quad (4')$$

Если известны оценки чисел Ритца (собственных чисел матрицы T_k), то, ввиду (4'), легко получить и теоретическую оценку сходимости. С этой целью приведем здесь известный результат.

Теорема 2 ([13], Theorem 2.6.1, p.37). Пусть $\lambda_1, \lambda_2, \dots, \lambda_n$ — собственные значения матрицы $A^* A$, упорядоченные по невозрастанию. Пусть $\mu_1, \mu_2, \dots, \mu_p$ — так же упорядоченные собственные значения матрицы T_s процесса Ланцоша с размером блока p . Пусть блочный вектор $v_1 \in \mathbb{C}^{n \times p}$ содержит первые p собственных векторов $A^* A$. Предположим, что $\lambda_p > \lambda_{p+1}$ и что $\cos \theta := \sigma_{\min}(q_1^* v_1) > 0$. Тогда для собственных значений матрицы T_s имеют место неравенства

$$\begin{aligned} \lambda_k &\geq \mu_k \geq \lambda_k - \varepsilon_k^2, \quad k = 1, \dots, p, \\ \varepsilon_k^2 &= (\lambda_k - \lambda_n) \frac{\operatorname{tg}^2 \theta}{T_{s-1}^2(\frac{1+\gamma_k}{1-\gamma_k})}, \quad \gamma_k = \frac{\lambda_k - \lambda_{p+1}}{\lambda_k - \lambda_n}. \end{aligned} \quad (5)$$

Символом T_{s-1} здесь обозначен многочлен Чебышева первого рода степени $s-1$.

Однако непосредственные оценки следов T_k приводят к более точному результату; переходим к его изложению. Наряду с ланцюшевскими векторами q_k введем и ланцюшевские многочлены p_k : по индукции легко следует, что $q_k = p_{k-1}(A^*A)q_1$. Далее, обозначим матрицу собственных векторов A^*A через V , матрицу собственных значений через Λ , и введем вектор $\varphi = V^*q_1$. Тогда

$$Q_k = V[\varphi \quad p_1(\Lambda)\varphi \quad \dots \quad p_{k-1}(\Lambda)\varphi],$$

откуда следует, что

$$\sum_{\mu=1}^n |\varphi_\mu|^2 p_j(\lambda_\mu) p_k(\lambda_\mu) = q_j^* q_k = \delta_{kj}.$$

Определение 1 ([1]). *Аддитивное и однородное отображение \mathfrak{S} пространства многочленов $\{p(x)\}$, рассматриваемых на отрезке K , на множество вещественных чисел назовем позитивным функционалом, если всякий раз из $p(x) \geq 0, \forall x \in K$, и $p(x) \not\equiv 0$ следует, что $\mathfrak{S}(p) > 0$.*

Поскольку $|\varphi_1|^2 + \dots + |\varphi_n|^2 = 1$, ясно, что функционал $\mathfrak{S}(p) \equiv \sum_{\mu=1}^n |\varphi_\mu|^2 p(\lambda_\mu)$ позитивен на отрезке $[\lambda_n, \lambda_1]$ для многочленов степени меньшей $2n - 2$.

Теорема 3 ([1], с. 80). *Функция $\rho_k(z)$, определяемая по последовательности многочленов $\{p_k\}$, ортогональных относительно позитивного функционала \mathfrak{S} , следующим образом: $\rho_k(z)^{-1} = \sum_{j=1}^k |\varphi_j(z)|^2$, удовлетворяет соотношению*

$$\rho_k(z) = \min_{\substack{\deg P_k \leq k \\ P_k(z)=1}} \mathfrak{S}(|P_k|^2).$$

Следствие 2. $\rho_j(\lambda) \leq \sum_{\mu=1}^n |\varphi_\mu|^2 P_j^2(\lambda_\mu)$ для всякого многочлена P_j степени не выше j с вещественными коэффициентами, принимающим значение 1 в точке λ .

Таким образом,

$$\begin{aligned} \operatorname{tr} T_k &= \sum_{\mu=1}^k \sum_{j=1}^n |\varphi_j|^2 \lambda_j |p_{\mu-1}(\lambda_j)|^2 = \sum_{j=1}^n \frac{|\varphi_j|^2 \lambda_j}{\rho_{k-1}(\lambda_j)} \\ &\geq \sum_{j=1}^k \frac{|\varphi_j|^2 \lambda_j}{\rho_{k-1}(\lambda_j)} \geq \sum_{j=1}^k \frac{|\varphi_j|^2 \lambda_j}{\sum_{\mu=1}^n |\varphi_\mu|^2 P_{k,j}^2(\lambda_\mu)}, \end{aligned}$$

где $P_{k,j}(\lambda)$, $j = 1, \dots, k$ — многочлен степени не выше $k - 1$, принимающий в точке λ_j значение 1.

Далее, преобразуем знаменатель в последней сумме:

$$\sum_{\mu=1}^n |\varphi_\mu|^2 P_{k,j}^2(\lambda_\mu) \leq |\varphi_j|^2 + \sum_{\mu=1, \mu \neq j}^n |\varphi_\mu|^2 \left(\max_{\lambda \in K_j} |P_{k,j}(\lambda)| \right)^2,$$

где множество $K_j = [\lambda_n, \lambda_{j+1}] \cup [\lambda_{j-1}, \lambda_1]$.

В связи с последним максимумом выберем в качестве $P_{k,j}$ многочлен степени не выше k , равный 1 в точке λ_j и наименее уклоняющийся от нуля на указанном объединении отрезков, а значение максимума для него обозначим $Z_{k,j,n}$.

Эти многочлены были исследованы Золотаревым, носят его имя, и допускают явное представление через тета-функции Якоби [6, Lemma 2.1, Problem BZ13].

Наконец, имеем

$$\operatorname{tr} T_k \geq \sum_{j=1}^k \lambda_j \frac{|\varphi_j|^2}{|\varphi_j|^2 + (1 - |\varphi_j|^2) Z_{k,j,n}^2}.$$

Обозначая через $\varepsilon_k^* = \sum_{j=k+1}^n \lambda_j$ — квадрат погрешности наилучшей k -ранговой аппроксимации A во фробениусовой норме, имеем

$$\|A - P_k R_k Q_k\|_F^2 \leq \varepsilon_k^* + \sum_{j=1}^k \lambda_j \frac{Z_{k,j,n}^2}{|\varphi_j|^2}. \quad (6)$$

Мы пришли к следующей теореме.

Теорема 4. Пусть все проекции стартового вектора процесса бидиагонализации q_1 на правые сингулярные векторы матрицы A ненулевые и все сингулярные числа A различны. Тогда погрешность k -ранговой аппроксимации матрицы A , порождаемой процессом би-диагонализации, оценивается по формуле (6).

Список литературы

- [1] АХИЕЗЕР Н. И. Классическая проблема моментов и некоторые вопросы анализа, связанные с ней. М.: Физматлит, 1961.

- [2] ПАРЛЕТТ Б. Симметрическая проблема собственных значений. М.: Мир, 1983.
- [3] Тыртышников Е. Е. Тензорные аппроксимации матриц, порожденных асимптотически гладкими функциями. // Матем. сб. 2003. Т. 194. №6. С. 147–160.
- [4] GOLUB G., LUK F., OVERTON M. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. // ACM Transactions on Math. Soft. 1981. V. 7. №2. P. 149–169.
- [5] GOREINOV S. A., TYRTYSHNIKOV E. E., YEREMIN A. Yu. Matrix-free iterative solution strategies for large dense linear systems. Numer. Linear Algebra Appl. 1997. V. 4. №4. P. 273–294.
- [6] LEBEDEV V. I. Zolotarev polynomials and extremum problems. // Russian J. Numer. Anal. Math. Modelling. 1994. V. 9. №3. P. 231–263.
- [7] PAIGE C. C. The computation of eigenvalues and eigenvectors of very large sparse matrices. Ph.D.thesis. The University of London. 1971.
- [8] SAAD Y. Numerical methods for large eigenvalue problems. Manchester University Press, 1992.
- [9] OSELEDETS I. V., SAVOSTIANOV D. V., TYRTYSHNIKOV E. E. Tucker dimensionality reduction of three-dimensional arrays in linear time. // SIAM J. Matr. Anal. 2008. In press.
- [10] G. W. STEWART, J. SUN. Matrix Perturbation Theory. Academic Press, 1990.
- [11] E. E. TYRTYSHNIKOV. Mosaic-skeleton approximations. // Calcolo. 1996. V. 33. №1–2. P. 47–57.
- [12] E. E. TYRTYSHNIKOV. Incomplete cross approximation in the mosaic-skeleton method. // Computing. 2000. V. 4. P. 367–380.
- [13] R. R. UNDERWOOD. An iterative block Lanczos method for the solution of large sparse symmetric eigenproblems. Ph.D.thesis. Stanford University. 1975.

Построение тетраэдральных сеток для областей с заданными поверхностными триангуляциями

А. А. Данилов[®]

В работе рассмотрена задача построения тетраэдральной сетки для произвольной области с заданной триангуляцией на границе. Предложен метод решения этой задачи, основанный на алгоритме продвигаемого фронта и надежном алгоритме построения тетраэдральных сеток для многогранников.

1. Введение

Построение сеток является одним из шагов при решении задач компьютерного моделирования. В приложениях часто используют неструктурированные тетраэдральные сетки, так как с их помощью проще дискретизировать сложные геометрические области. Одним из способов задания геометрии области может служить использование дискретизации ее границы. В качестве дискретизации границы может выступать поверхностьная триангуляция. Во многих приложениях важной задачей является построение тетраэдральной сетки с заданной фиксированной поверхностью триангуляцией. Например, это может быть связано с особенностями задания граничных условий в рассматриваемой модели, или с необходимостью иметь одинаковую дискретизацию на интерфейсе между двумя областями.

В данной статье рассматривается задача построения конформной тетраэдральной сетки для произвольной области с заданной триангуляцией на границе. Границная триангуляция является конформной, т.е. любые два треугольника либо не пересекаются, либо имеют

[®]Институт вычислительной математики РАН

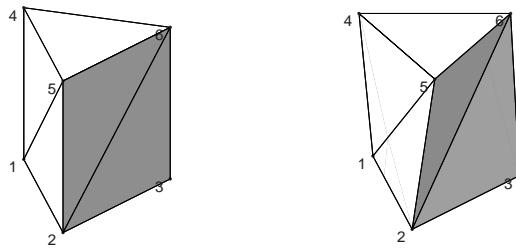


Рис. 1. Призма Schönhardt'a. Для построения тетраэдральной сетки с заданной поверхностной триангуляцией необходимо добавить хотя бы одну вершину внутри призмы. (на рис. скрыты ребра 3-4)

ровно одну общую вершину, либо имеют целое общее ребро. Построенная тетраэдральная сетка также должна быть конформной — любые два тетраэдра либо не пересекаются, либо имеют ровно одну общую вершину, либо целое общее ребро, либо целую общую грань. В аналогичной задаче для плоскости произвольный простой многоугольник может быть разбит на треугольники с помощью диагоналей без добавления дополнительных внутренних точек [1]. В трехмерном пространстве даже для выпуклых многогранников аналогичное утверждение неверно [2] (см. Рис. 1), однако наличие хотя бы одной внутренней точки решает проблему для выпуклых многогранников [3]. В общем случае доказано существование тетраэдральной сетки при условии возможности добавления новых вершин внутри области [4].

В последнее время было предложено много методов для построения неструктурированных сеток, основанных на тетраэдральном разбиении Делоне, и на методе продвигаемого фронта. Алгоритмы, основанные на тетраэдральном разбиении Делоне, гарантируют сохранение заданной границы области только в частных случаях. Если на границе выпуклой области любые два соседних треугольника лежат в разных плоскостях, то тетраэдральное разбиение Делоне будет иметь заданный след на границе области. В общем случае для восстановления триангуляции на границе могут быть использованы локальные перестроения сетки [5]. Предложены способы измельчения сетки для восстановления границы [6], а также методы построения тетраэдральных разбиений Делоне с ограничениями [7]. Метод

продвигаемого фронта [8], напротив, отлично работает около произвольной границы области, но может столкнуться с проблемами внутри области. Перспективным направлением видится комбинация методов Делоне и продвигаемого фронта [9].

Во многих практических приложениях важна регулярность построенной сетки, то есть малое отклонение построенных тетраэдров от равносторонних. Если поверхностная триангуляция области уже является сильно нерегулярной, то построение регулярной тетраэдральной сетки с заданной триангуляцией на границе является неразрешимой задачей. В подобных случаях более практическим подходом может быть предварительное перестроение и улучшение поверхностной триангуляции [10].

В данной статье предлагается метод, состоящий из двух частей. Первая часть основана на алгоритме продвигаемого фронта [10] и используется для построения большей части тетраэдральной сетки. Вторая часть метода основана на конструктивном доказательстве существования тетраэдральной сетки для произвольного многогранника [4, 11]. Вместе эти два алгоритма представляют надежный способ построения тетраэдральных сеток для областей с заданными поверхностными триангуляциями. В разделе 2 будет кратко описан алгоритм продвигаемого фронта. В разделе 3 будет описан второй, надежный алгоритм. В разделе 4 обсуждаются способы получения регулярных сеток.

2. Алгоритм продвигаемого фронта

В качестве основного алгоритма для построения тетраэдральной сетки был выбран алгоритм продвигаемого фронта [10]. Этот алгоритм подразумевает наличие поверхностной триангуляции, называемой начальным фронтом. У треугольников фронта имеется ориентация, она определяет направление для дальнейшего продвижения фронта. Алгоритм является многошаговым. На каждом его шаге строится один тетраэдр на границе фронта, при этом фронт сдвигается внутрь области. После каждого шага текущий фронт отделяет построенную тетраэдральную сетку от оставшейся области. Алгоритм продвигаемого фронта не гарантирует построение тетраэдральной сетки для всей заданной области. Для разбиения оставшейся части используется второй, более надежный и медленный ал-

горитм, рассмотренный в следующем разделе.

Алгоритм продвигаемого фронта строит по одному тетраэдру на каждом шаге. Из фронта выбирается треугольник с самой маленькой площадью. Из его центра масс проводится нормаль внутрь области. На нормали выбирается четвертая точка будущего тетраэдра в соответствии с желаемым размером элементов сетки в данной точке. Если построенный тетраэдр пересекает текущий фронт, то начинается перебор возможного положения четвертой вершины тетраэдра среди соседних вершин фронта. Если после перебора всех возможных кандидатов тетраэдр так и не будет построен, то весь шаг повторяется с самого начала, где из фронта выбирается следующий треугольник. Когда все треугольники фронта будут рассмотрены, и ни на одном из них не получится построить тетраэдр, алгоритм останавливается. Оставшийся фронт будет ограничивать нераэбитую область. Тетраэдральная сетка внутри этой области будет построена с помощью надежного алгоритма.

При построении новых вершин поддерживается следующее дополнительное условие: добавление новой вершины не должно уменьшать наименьшее из попарных расстояний между вершинами фронта. Если это условие будет выполняться при продвижении фронта, то расстояние между любыми двумя вершинами сетки будет ограничено снизу наименьшим расстоянием между вершинами начального фронта. В конечной области, ограниченной начальным фронтом, может быть расположено лишь конечное число вершин сетки. Следовательно, количество тетраэдров также будет ограничено, что гарантирует конечность времени работы алгоритма. В частности поддерживаемое условие на расстояние между вершинами накладывает ограничение на возможный размер тетраэдров в сетке. Выполнение этого условия является более приоритетным, чем поддержание желаемого размера элементов в сетке. Предложенный алгоритм может быть использован для построения равномерных или разгрубляющихся сеток. Для получения сгущающихся сеток может быть необходимо использование других алгоритмов для перестроения полученной сетки [14, 15].

В конце работы алгоритма проводится разглаживание сетки. Для каждой внутренней вершины сетки вычисляется центр масс ее соседей, и внутренние вершины сдвигаются в новые посчитанные положения.

жения. Сдвиг выполняется, если не нарушается конформность сетки. Операция повторяется еще раз, но при этом вершины сдвигаются лишь на половину вектора, затем на треть, четверть и т.д. Уменьшение множителей позволяет избежать нарушения конформности сетки при сдвиге вершин. С другой стороны, расходимость ряда $\sum \frac{1}{n}$ не ограничивает расстояние, на которое может сдвинуться вершина. Делается от 5 до 10 операций разглаживания сетки.

3. Надежный алгоритм построения тетраэдральных сеток

В этом разделе рассматривается вариант алгоритма [4, 11], предназначенный для гарантированного построения конформной тетраэдральной сетки внутри многогранника с заданным следом на границе. Алгоритм состоит из трех частей. Сначала на множестве начальных вершин строится тетраэдральное разбиение Делоне (раздел 3.1). Далее проводится поиск недостающих граней в построенной сетке, на границе области строятся точки пересечения (раздел 3.2). В конце построенные новые вершины сдвигаются внутрь области, восстанавливая требуемую граничную триангуляцию (раздел 3.3).

3.1. Тетраэдральное разбиение Делоне. Основная цель первой части алгоритма состоит в построении вспомогательной тетраэдральной сетки на наборе вершин из заданной поверхностной триангуляции. Тетраэдральная сетка Делоне для набора вершин в пространстве — это такое разбиение, что никакая вершина не попадает внутрь ни одной из сфер, описанных вокруг тетраэдров [12].

Для построения триангуляции Делоне может быть применен простейший последовательный алгоритм [13]. Вначале искусственно вводятся дополнительные 8 вершин прямоугольного параллелепипеда, полностью покрывающего исходный набор точек. Этот параллелепипед с 8 узлами разбивается на тетраэдры, удовлетворяющие условию Делоне. Далее на каждом шаге выбирается очередная вершина из исходного набора. Если вершина попала внутрь одного из тетраэдров, то этот тетраэдр делится этой вершиной на четыре части. Если вершина попала на общую грань для двух тетраэдров, то тетраэдры делятся на тройки. Если вершина попала на общее ребро для нескольких тетраэдров, то каждый тетраэдр делится этой вершиной на пару тетраэдров. Далее проводятся локальные проверки

на соответствие условию Делоне. Если условие не выполняется, то проводится перестроение тетраэдров [13].

В результате будет получено тетраэдральное разбиение с вершинами из исходного набора вершин. Из алгоритма построения следует, что полученная сетка будет конформной, и будет удовлетворять условию Делоне. Но алгоритм не гарантирует совпадение требуемой поверхности триангуляции с полученной сеткой. Часть ребер и треугольников из исходной поверхности триангуляции может отсутствовать в построенной тетраэдральной сетке Делоне. Эти недостающие элементы будут добавлены в следующей части алгоритма.

3.2. Разбиение граничной триангуляции. Во второй части алгоритма производится поиск недостающих ребер и треугольников заданной поверхности триангуляции. В первую очередь последовательно рассматриваются все недостающие ребра. Для требуемого ребра AB и имеющихся тетраэдров с вершиной A существуют следующие возможности:

1. Ребро AB является общим ребром, выходящим из A , для нескольких тетраэдров вокруг A . В этом случае AB уже целиком присутствует в построенной сетке.
2. Ребро AB проходит по одной из граней тетраэдра с вершиной в A . Иначе говоря, AB пересекается с некоторым ребром CD в построенной сетке. В этом случае добавляется новая вершина P — точка пересечения AB и CD . Каждый тетраэдр с ребром CD делится на пару тетраэдров с ребрами CP и PD . В частности делятся и тетраэдры, с вершиной в A , таким образом отрезок AP попадает в сетку (см. Рис. 2). Далее рассматривается оставшаяся часть PB и вершина P .
3. Ребро AB проходит через внутренность одного тетраэдра с вершиной A , и пересекает противоположную грань в некоторой точке. В этой точке создается новая вершина P , два тетраэдра, граничащие по этой грани, делятся каждый на три части. После этой операции отрезок AP попадает в сетку, и далее рассматривается оставшаяся часть PB и вершина P (см. Рис. 3).

Описанные операции могут измельчать ребра в тетраэдральной сетке, но не удаляют их из нее. Таким образом, при добавлении в

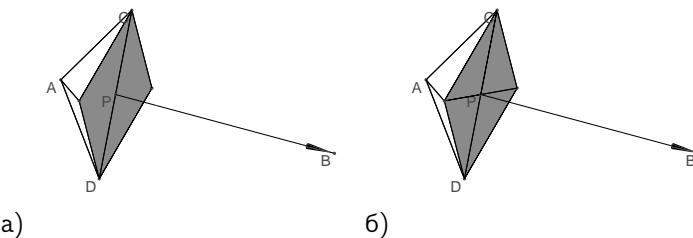


Рис. 2. Ребро проходит по грани тетраэдра: а) ребро AB пересекается с ребром CD в точке P ; б) тетраэдры разбиваются на пары тетраэдров.

сетку очередного ребра из заданной поверхностной триангуляции гарантируется сохранность уже добавленных ребер.

После добавления всех необходимых ребер полученная сетка по-прежнему является конформной. Теперь рассматриваются треугольники из исходной поверхностной триангуляции один за другим. Для каждого треугольника его ребра или их разбиения уже находятся в сетке. Однако возможна ситуация, в которой грани имеющейся сетки не покрывают требуемый треугольник полностью, в этом случае найдется одно или несколько ребер, пересекающих этот треугольник. Каждое из таких ребер должно быть разделено на два отрезка с помощью новых вершин на пересечении ребра и плоскости треугольника. Соответствующие тетраэдры при этом также разбиваются. Эта операция может еще больше измельчить ребра и треугольники в поверхностной триангуляции, но не удаляет их из тетраэдральной сетки. То есть сохранность уже рассмотренных ребер и треугольников гарантируется.

В результате будет получена более мелкая конформная сетка, причем все треугольники заданной поверхностной триангуляции будут в ней присутствовать в виде разбиения. На этом этапе можно удалить из тетраэдральной сетки те тетраэдры, которые не лежат внутри заданного многогранника. Следующим шагом будет перенос новых точек пересечения внутрь области.

3.3. Восстановление граничной триангуляции. В последней части алгоритма производится перенос новых вершин внутрь обла-

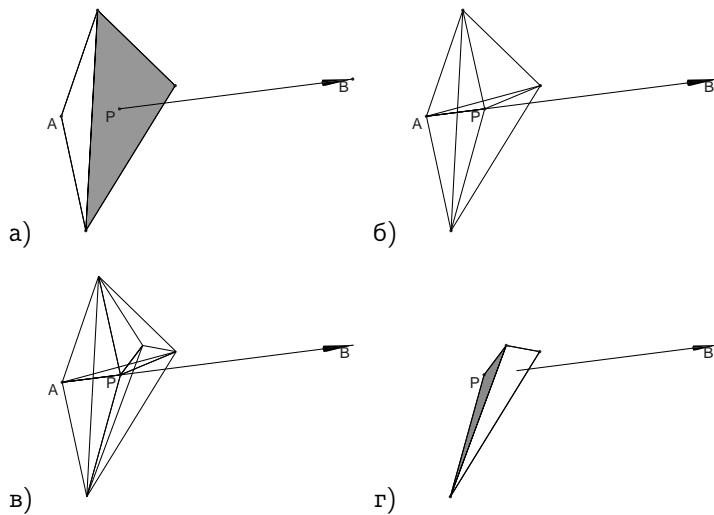


Рис. 3. Ребро проходит через тетраэдр: а) ребро AB пересекает грань в точке P ; б) тетраэдр разбивается на три тетраэдра; в) соседний тетраэдр также разбивается; г) переход к отрезку PB .

сти. Сначала переносятся вершины, лежащие внутри граней, затем вершины, лежащие на ребрах.

Пусть вершина P лежит внутри грани поверхностной триангуляции. Тетраэдры с вершиной в P образуют некоторую оболочку вокруг P , граница этой оболочки состоит из двух частей. Одна часть состоит из треугольников с вершиной P , лежащих на поверхности области, и образующих некоторый многоугольник, вторая часть состоит из остальных треугольников. Так как в рассмотренной оболочке конечное количество тетраэдров, и их ориентированные объемы строго положительны, то существует некоторая открытая окрестность точки P , внутри которой возможно перемещение вершины P с сохранением положительности ориентированных объемов у тетраэдров. После сдвига вершины внутрь области в пределах этой окрестности останется неразбитая часть области в виде пирамиды с вершиной P и основанием в виде многоугольника на поверхности области. Многоугольник можно разбить на треугольники с помощью диагоналей без добавления новых вершин [1], следовательно,

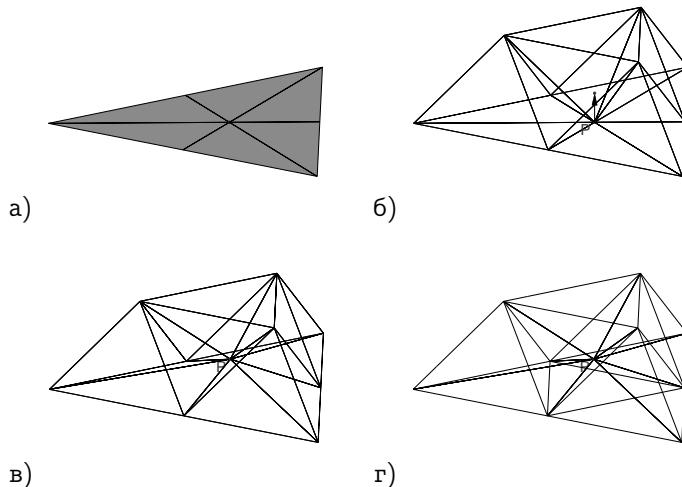


Рис. 4. Удаление вершины с грани: а) треугольники на границе области; б) оболочка из тетраэдров; в) перенос вершины внутрь области; г) разбиение пирамиды.

пирамиду можно разбить на тетраэдры без добавления новых вершин. В результате будет получена конформная тетраэдральная сетка для той же области, с тем же количеством вершин, но одна из вершин будет перемещена внутрь области (см. Рис 4). Этот процесс повторяется для всех вершин, лежащих внутри треугольников поверхностной триангуляции.

Пусть теперь вершина P попала на ребро поверхностной триангуляции. Этот случай аналогичен уже рассмотренному. Граница оболочки из тетраэдров будет состоять из трех частей: двух поверхностных и одной внутренней. Соответственно, после сдвига внутрь области останется две пирамиды, которые могут быть разбиты на тетраэдры без добавления дополнительных вершин. Сдвигая поочереди все вершины внутрь области, в конце будет получена конформная тетраэдральная сетка с заданным следом на границе области.

4. Регулярность построенных сеток

В предложенном методе в первую очередь используется алгоритм продвигаемого фронта, который строит большую часть сетки. Этот алгоритм позволяет строить разгрублющиеся и регулярные сетки, однако не гарантирует их полное построение. Второй, надежный алгоритм строит нерегулярные сетки. То есть метод не может гарантировать построение регулярных сеток. Для получения регулярных тетраэдральных сеток можно использовать два основных подхода: предварительное перестроение и улучшение поверхностной сетки [10]; и последующее перестроение полученной сетки [14, 15].

Если для некоторой задачи сохранение поверхностной триангуляции не является необходимым условием, то перестроение сетки позволит получить регулярную поверхностную сетку. Регулярность поверхностной сетки позволяет алгоритму продвигаемого фронта строить более регулярные тетраэдральные сетки. На практике применение поверхностной регуляризации позволяет избежать использования надежного алгоритма [10].

Предложенный в статье метод решает задачу построения конформной тетраэдральной сетки. Вместе с методами предобработки поверхностных сеток и методами постобработки тетраэдральных сеток они образуют полный комплекс программ для построения сеток. Примерами таких комплексов могут служить библиотеки для работы с анизотропными сетками [16, 17].

Список литературы

- [1] Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. — М.: Мир. 1989. С. 285–295.
- [2] Schönhardt E. Über die Zerlegung von Dreieckspolyedern in Tetraeder. // *Mathematische Annalen*. 1928. V. 98. P. 309–312.
- [3] Joe B. Three-dimensional boundary-constrained triangulations. // *Proc. of 13th IMACS World Congress*. 1992. P. 215–222.
- [4] George P.L., Borouchaki H. Maillage simplicial d'un polyèdre arbitraire. // *C. R. Acad. Sci. Paris. Ser. I* 338. 2004. P. 735–740.

- [5] Borouchaki H., Hecht F., Saltel E., George P.L. Reasonably efficient Delaunay based mesh generator in 3 dimensions. // *Proc. of 4th Int. Meshing Roundtable*. 1995. P. 3–14.
- [6] Du Q., Wang D. Recent progress in robust and quality Delaunay mesh generation. // *J. of Comp. and App. Math.* 2006. V. 195. P. 8–23.
- [7] Shewchuk J.R. Constrained Delaunay tetrahedralizations and provably good boundary recovery. // *Proc. of 11th Int. Meshing Roundtable*. 2002. P. 193–204.
- [8] George P.L. Automatic mesh generation and finite element computation. // *Handbook of Num. Anal.* 1996. V. 4. P. 127–148.
- [9] Yang Y.J., Yong J.H., Sun J.G. An algorithm for tetrahedral mesh generation based on conforming constrained Delaunay tetrahedralization. // *Computers & Graphics*. 2005. V. 29(4). P. 606–615.
- [10] Василевский Ю., Вершинин А., Данилов А., Пленкин А. Технология построения тетраэдральных сеток для областей, заданных в САПР. // *Матричные методы и технологии решения больших задач*. — М.: ИВМ РАН. 2005. С. 21–32.
- [11] George P.L., Borouchaki H., Saltel E. 'Ultimate' robustness in meshing an arbitrary polyhedron. // *Int. J. Numer. Meth. Eng.* 2003. V. 58. P. 1061–1089.
- [12] Делоне Б.Н. О пустоте сферы. // *Изв. АН СССР*. 1934. Т. 4, С. 793–800.
- [13] Скворцов А.В. Триангуляция Делоне и её применение. — Томск: Изд-во Том. ун-та. 2002. С. 22–43.
- [14] Zavattieri P.D., Dari E.A., Buscaglia G.C. Optimization strategies in unstructured mesh generation. // *Int. J. Numer. Meth. Eng.* 1996. V. 39. P. 2055–2071.
- [15] Agouzal A., Lipnikov K, Vassilevski Yu. Adaptive generation of quasi-optimal tetrahedral meshes. // *East-West J. Numer. Math.* 1999. V. 7, No. 4. P. 223–244, 1999.

- [16] 2D Generator of Anisotropic Meshes.
<http://sourceforge.net/projects/ani2d/>
- [17] 3D Generator of Anisotropic Meshes.
<http://sourceforge.net/projects/ani3d/>

Комплексный подход к обслуживанию вычислительных кластеров

С. А. Жуматий*

В статье приводится описание программного комплекса ParCon, предназначенного для помощи пользователям и администраторам в эффективном использовании вычислительных кластеров. Приведены примеры часто возникающих задач, успешно решаемых с помощью ParCon.

1. Введение

Вычислительные кластеры являются мощным инструментом для проведения сложных расчётов, использование которого сопряжено с рядом трудностей как для администраторов, так и для самих пользователей. Взять хотя бы поддержку работоспособности и мониторинг текущего состояния большого количества постоянно включенного оборудования — крайне непростая задача. В данной работе мы обсудим, что может быть необходимо для обеспечения эффективной работы и пользователей, и их задач на кластере.

Пользовательская программа может обладать специфическими свойствами, которые значительно снижают её производительность на кластере, причем компилятор, библиотеки вряд ли помогут исправить ситуацию. Что является причиной замедления — несоответствие размера оперативной памяти узла данным задачи, необходимость постоянно пользоваться медленными сетевыми дисками, несогласованность работы параллельных процессов или что-то ещё? Умение найти ответы на подобные вопросы может значительно облегчить работу пользователя.

Администратору кластера такая информация тоже будет очень полезна, так как он должен иметь информацию о характере использо-

*Научно-исследовательский вычислительный центр МГУ

зования кластера. В детали работы каждой задачи ему вникать, может, и не всегда надо, но иметь сводную статистику по задачам, пользователям и использованию ресурсов кластера просто необходимо.

Существуют различные средства, позволяющие помочь в решении перечисленных выше вопросов. К примеру, система Ganglia позволяет собирать данные о работе процессоров, использовании памяти и другие характеристики работы вычислительных узлов. Однако, привязать эти данные к задачам пользователей или получить комплексные характеристики о работе кластера невозможно. Другие программные комплексы, насколько нам известно, также решают лишь очень узкие задачи и не способны служить серьёзным подспорьем ни пользователю, ни администратору кластера.

2. Постановка задачи

Требуется создать программную среду для поддержки выполнения параллельных приложений на кластерных установках, позволяющую:

- управлять прохождением задач,
- автоматически распределять свободные процессоры между задачами,
- гибко управлять политикой использования кластера различными пользователями,
- предоставлять детальную информацию о работе задач и пользователей на кластере,
- предоставлять детальную статистику использования ресурсов кластера задачей как во время работы задачи, так и после её завершения,
- предоставлять суммарные отчёты о работе задач, пользователей, использовании ресурсов кластера,
- оказывать минимальное влияние на работу пользовательских задач.

3. Архитектура предложенного решения

Для решения поставленной задачи был предложен подход, предусматривающий создание двух программных компонентов, способных работать как независимо друг от друга, так и в режиме взаимодействия. Оба компонента в комплексе представляют собой интегрированную среду, получившую название ParCon.

Первый компонент — это система управления заданиями Cleo. Она была разработана в НИВЦ МГУ специально для управления использованием ресурсов вычислительных кластеров. Система ориентирована на работу с параллельными приложениями и поддерживает многие параллельные среды. Под её управлением в течении 5 лет успешно работали 4 разных кластера суперкомпьютерного комплекса НИВЦ МГУ, объединявшие в общей сложности 153 вычислительных узла (306 процессоров).

Ключевые особенности Cleo:

- изначальная ориентированность на вычислительные кластеры,
- гибкость политик управления ресурсами,
- поддержка большого числа параллельных сред,
- расширяемость за счёт дополнительных планировщиков и модулей.

Для работы с Cleo пользователю практически не нужно дополнительной подготовки, поскольку программа *trigrin* автоматически ставит задачу в очередь. Для просмотра очереди можно использовать web-интерфейс или консольные команды.

Описание системы Cleo доступно по адресу в интернете:
<http://parcon.parallel.ru/cleo.html>.

Второй компонент среды ParCon — это комплекс мониторинга AntMon. Именно он позволяет собирать детальную информацию о состоянии вычислительных узлов и передать эту информацию администратору. AntMon изначально разрабатывался таким образом, чтобы предоставить возможность сбора статистики с большого числа узлов кластера с высокой степенью детализации и низкой нагрузкой на узлы. Этот программный пакет спроектирован по модульно-

му принципу: в любой момент к нему можно добавить модуль для сбора той информации, которая существенна для данного кластера.

Сам AntMon способен не только собирать данные, но и определять критические ситуации на вычислительных узлах (если собранная информация позволяет это сделать), оповещать об этом администратора и/или выполнять набор экстренных команд. Однако для того, чтобы дать возможность провести содержательный анализ работы кластера или вычислительных программ, требуется иное средство.

Таким средством является аналитический модуль ParCon, который сохраняет данные о задачах и полученную с узлов информацию в базе данных, предоставляя возможность для исследования динамических характеристик задач и узлов. Например, с его помощью можно получить ответ на такие вопросы как «какова была средняя загрузка процессора во время работы задачи?» или «насколько согласована работа вычислительных узлов на данной задаче?».

Описание комплекса ParCon представлено на странице в интернете: <http://parcon.parallel.ru>.

4. Система управления заданиями

Cleo предоставляет пользователям прозрачную схему запуска программ, для чего используется привычный скрипт mpirun. В поставку системы входят программы для просмотра состояния очереди, удаления задач и изменения приоритета задач. Разработан web-интерфейс, позволяющий просматривать состояние узлов кластера, задач и очередей.

Администратору кластера предоставлены возможности гибкого управления политикой запуска программ пользователей. Среди таких возможностей стоит отметить:

- выделение процессорного времени для особых задач,
- временная блокировка указанных задач,
- ограничение количества одновременно занятых пользователем процессоров,
- ограничение количества запущенных пользователем задач,

- ограничение количества задач пользователя в очереди,
- управление приоритетами пользователей.

Система автоматически отслеживает состояние вычислительных узлов кластера и блокирует их в случае неполадок. Состояние задач на узлах также контролируется, и в случае некорректного завершения задания его процессы на узлах принудительно завершаются.

В систему встроена возможность автоматического определения работоспособности узлов, и в случае остановки работы вычислительного узла система автоматически блокирует его и снимает все задачи, работавшие на нём.

Администратор может в любое время блокировать и разблокировать задачи и вычислительные узлы. Реализована возможность «отложенной» блокировки узлов, когда блокировка вступает в силу только после окончания работы задач, распределённых на эти узлы.

В системе поддерживается иерархическая организация очередей, что позволяет создавать несколько разделов в рамках одного кластера, сохраняя возможность использовать его полностью, или, наоборот, объединять несколько кластеров под единой системой управления.

Cleo спроектирована расширяемой за счёт дополнительных планировщиков и внешних модулей, выполняющих по заданным условиям. В случае, если модуль или планировщик содержит ошибку и вызывает сбой в работе, то Cleo просто блокирует его и продолжает работать без данного модуля или со встроенным стандартным планировщиком.

Немаловажной особенностью Cleo является поддержка иерархии очередей в рамках одного кластера. Она позволяет использовать часть процессоров для выделенных задач (например, для отладки). При этом остаётся возможность использования всех процессоров, как выделенных, так и остальных, одновременно для одной задачи. Для такого использования не требуется переконфигурация или приостановка системы очередей.

5. Система мониторинга

Система AntMon была разработана для сбора данных с большого числа узлов и реагирования на аномальные значения полученных

данных. Она спроектирована таким образом, чтобы сбор необходимых сведений проводился с минимальной нагрузкой на сеть и процессоры, а сбой головного сервера не приводил бы к отказу всей системы.

Система имеет распределённую структуру. На узлах, где необходимо собирать данные мониторинга, работают агенты, а анализ собранной информации выполняют головные серверы.

Информация для мониторинга собирается агентами при помощи подключаемых модулей. Модули запускаются один раз при получении первого запроса на информацию от них. После запуска модули остаются в рабочем состоянии и активируются при очередном запросе. Сами запросы обслуживаются агентами системы, они же отвечают за запуск модулей. На каждом узле, где необходимо собирать информацию, запускается один агент.

Модульный принцип построения AntMon позволяет отслеживать не только те параметры, которые предусмотрены в системе изначально, но и любые другие параметры необходимые пользователю, для чего достаточно написать модуль, который будет получать эти данные и передавать в систему. Интерфейс системы с модулями прост и полностью документирован. Модуль может быть написан практически на любом языке программирования и для его написания не требуется высокой квалификации программиста.

Для повышения надежности системы в архитектуре AntMon предусмотрено использование нескольких головных серверов. В штатном режиме они работают независимо друг от друга и могут распределять между собой общую нагрузку: каждый сервер будет обслуживать свой набор агентов или отдельных параметров составных частей кластера. Для каждого параметра указывается список головных серверов, которые могут его обслуживать.

Сначала параметр обслуживается первым в списке сервером. Если связь с этим сервером обрывается, то параметр переходит под опеку следующего в списке сервера. В случае сбоя одного из головных серверов, остальные серверы перераспределяют ответственность за приписанные ему параметры и производят реагирование на сбой, запуская последовательность предопределенных команд. Когда сбойный головной сервер возвращается в рабочее состояние, остальные серверы повторно перераспределяют ответственность за



Рис. 1. Распределение времён счёта задач менее 5 минут

параметры, а также выполняют соответствующий набор команд, например уведомление администратору вычислительной системы.

6. Аналитический модуль

Модуль выполняет задачу объединения данных от Cleo и AntMon в единую базу данных и предоставления аналитических данных пользователю и администратору. С его помощью осуществляется анализ собранных данных с привязкой к конкретным задачам на кластере. Сохранение информации в базе данных позволяет проводить анализ различной степени сложности. Например, узнать средний процент загрузки процессора на задачах определённого пользователя за последние несколько месяцев.

В качестве примера приведём небольшое исследование работы параллельных задач на кластере Ant НИВЦ МГУ. На рис. 1, 2, 3 и 4 представлен пример анализа времени счёта задач за месяц. По вертикали отложено число задач, по горизонтали — время счёта. Видно, что большое число задач завершается, не отработав даже секунды. Это говорит о том, что проводилось большое число запусков неотложенных программ. С помощью аналогичного анализа с разбивкой по пользователям удалось быстро выяснить пользователя, у которого регулярно возникали такие проблемы.

Яркие «пики» в 30 и 6000 минут соответствуют лимитам времени



Рис. 2. Распределение времён счёта задач менее часа



Рис. 3. Распределение времён счёта задач до двух дней

работы задач в очередях. «Пик» в 4300 минут соответствует лимиту времени, установленному одним из пользователей кластера.

Другой интересный пример анализа конкретной задачи можно увидеть на рис 5. Здесь приведены графики минимальной, максимальной и усреднённой доли простоя всех процессоров, занятых задачей. Видно, что в начальный момент времени часть процессоров простаивает на 70%. Через какое-то время доля простоя процессоров сводится к нулю.

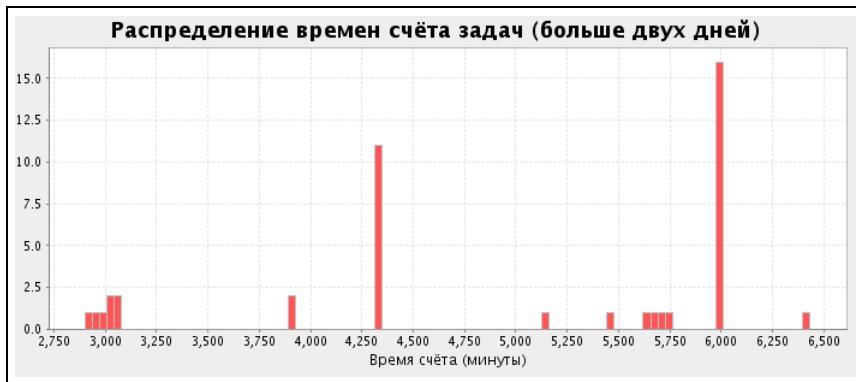


Рис. 4. Анализ времени счёта задач

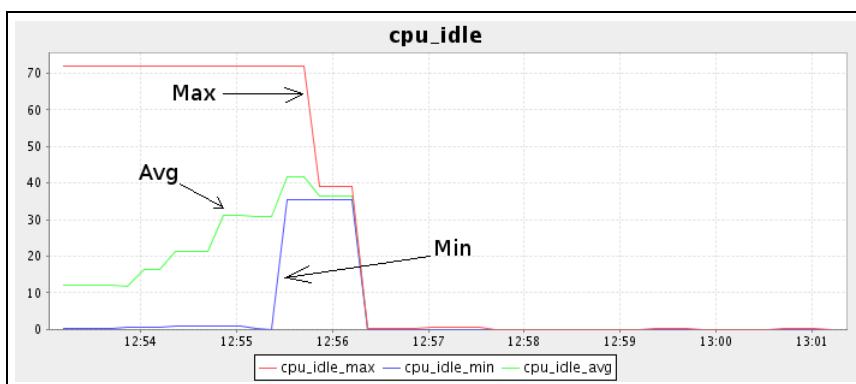


Рис. 5. Доля простоя процессора

Если сопоставить это график с графиком свободной памяти (рис. 6), то становится видна их корреляция. Обусловлена она тем, что на некоторых узлов под данные задачи не хватило памяти и начался процесс вытеснения страниц задачи на диск. В момент, когда задаче выделяется достаточное место в памяти и происходит «обнуление» доли простоя процессора.

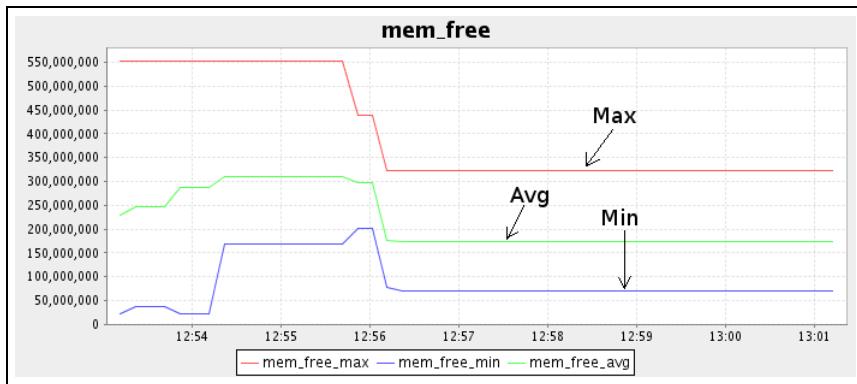


Рис. 6. Уровень загрузки памяти

7. Заключение

Обслуживание вычислительных кластеров — это большая и комплексная задача. Исключительное значение в её решении имеет информация о кластере, пользователях и их задачах и инструментальные средства для выполнения типовых действий. Чем больше администратор кластера знает о работе установки, тем продуктивнее будет её работа. Чем полнее представление пользователя о характере работы его программы на кластере, тем больше у него возможностей повысить эффективность программы.

ParCon способен предоставить пользователю и администратору средства для получения информации о работе как кластера в целом, так и отдельных задачах, пользователях, использовании ресурсов. ParCon продолжает развиваться. В первоочередные планы его развития входит исследование масштабируемости и расширение аналитических возможностей.

Список литературы

- [1] Жуматий С.А. Система ParCon — ассистент в работе пользователя и администратора вычислительного кластера // Методы и средства обработки информации. Труды Всероссийской научной конференции. 2005. Изд-во отд. факультета вычислитель-

ной математики и кибернетики МГУ им. М.В.Ломоновова. С. 252-255.

- [2] Воеводин Вл.В., Жуматий С.А. Вычислительное дело и кластерные системы. — М.: Изд-во МГУ, 2007. — 150 с.
- [3] Жуматий С.А., Князев А.С. Исследование характеристик потока задач на вычислительном кластере // Научный сервис в сети Интернет: многоядерный мир. 15 лет РФФИ: Труды Всероссийской научной конференции. 2007. М.: Изд-во МГУ. С. 139.

Оценка загруженности компьютера в различных UNIX-системах[§]

С. А. Жуматий*, С. И. Соволев*

Статья посвящена исследованию методов определения загруженности компьютера, работающего под управлением ОС UNIX/Linux. Данное исследование было проведено при реализации модуля системы метакомпьютинга, отвечающего за запуск распределенных задач в моменты простого доступных компьютеров. В статье анализируются различные способы определения загруженности вычислительных ресурсов, обсуждаются их достоинства и недостатки. Описывается метод, выбранный по результатам анализа.

Многие в детстве играли в жмурки. С завязанными глазами надо было поймать кого-нибудь из играющих и определить, кто это. Для определения приходилось полагаться только на косвенные признаки — рост, элементы одежды... Как легко было ошибиться!

Очень похожая ситуация возникла при создании системы организации метакомпьютерных расчётов. Система должна отслеживать состояние распределённой вычислительной среды, состоящей из самых разных компьютеров, работающих в самых разных режимах, принадлежащим самым разным людям и организациям. На свободных компьютерах можно запускать расчёты. Если на компьютере начинает работать приложение хозяев, расчёт необходимо прекратить, чтобы не мешать штатной работе компьютера.

Как корректно определить степень загрузки компьютера при том, что никаких привилегированных прав на компьютерах у нас нет? Задача и в самом деле напоминает игру в жмурки, но в отличии

[§]Работа выполнена при поддержке гранта РФФИ №05-07-90206

*Научно-исследовательский вычислительный центр МГУ

от игры, ошибка тут стоит дороже. Из доступных инструментов — только те системные программы, которые установлены на целевом компьютере. Основной платформой было избрано семейство ОС Linux, как наиболее распространённое в вычислительной практике, для которой мы попробуем найти методы решения задачи по определению факта загруженности компьютера.

Пользователь Windows мог бы сказать: «Решение элементарно! Есть же хранитель экрана, который запускается во время простоя компьютера — вот его и надо использовать. Система сама и запустит, и завершит его когда надо. Кстати, хранители экрана есть не только для Windows...».

Верно, но есть нюансы. Хранитель экрана запускается только во время бездействия пользователя, но не обязательно бездействия процессора. Процессор может быть занят, например, пакетной обработкой фотографий в Photoshop или преобразованием видеозаписи из одного формата в другой. А уж вычислительный сервер вообще может не иметь ни мыши, ни клавиатуры и никаких хранителей экрана в принципе. Поэтому хранители экрана нашей проблемы не решают.

Не оставляет чувство, что решение такой задачи должно быть очень простым, поскольку есть множество стандартных программ, с помощью которых можно определить степень загруженности. Но не всё так просто. Важным требованием является переносимость в рамках UNIX-систем. Однако, как мы увидим далее, даже в семействе Linux проблема не решается тривиально. Решение дополнительно осложняется тем, что на вычислительном узле у нас может не быть прав администратора, а значит, набор средств будет ограничен.

Самый «правильный» и гарантированно работающий метод — использовать протокол и инфраструктуру SNMP. Но далеко не на каждой рабочей станции установлено программное обеспечение, реализующее серверную часть SNMP. Так как рабочая станция или сервер могут быть «чужими» и прав суперпользователя мы там можем не иметь, то устанавливать своё системное программное обеспечение администратор может и не разрешить.

Аналогичная ситуация сложилась с использованием программ пакета SAR. Представление данных в этом пакете фиксировано, но он не является стандартным и установлен далеко не на всех системах.

max. На каждой ОС существует независимая реализация этого пакета, поэтому даже использовать исходные тексты не всегда возможно.

Остаётся использование стандартных программ, входящих в системные пакеты, таких как ps, top, cptime.

Самым логичным решением казалось использование программы ps, выдающей загрузку процессора отдельно по каждому процессу. Программа ps вызывалась из скрипта, запускающегося раз в минуту с помощью cron. Её вывод анализировался, отдельно суммировались проценты загрузки процессора «своими» и чужими процессами, и на основании этих данных делался вывод о необходимости запуска либо снятия метакомпьютерной задачи.

В ходе экспериментов выяснилось, что ps далеко не всегда корректно выдает сведения о потребление процессорных ресурсов. Наблюдались случаи, в которых на компьютерах с запущенной счетной программой ps демонстрировала нулевой процент загрузки активными процессами, при этом другие средства мониторинга (top) указывали, что загрузка компьютера этими процессами близка к 100%. Очевидно, это ошибка или некорректность в самой программе ps. Таким образом, данные, предоставляемые программой ps, невозможно использовать для решения поставленной задачи.

Следующим вариантом реализации определения загрузки компьютера был скрипт, анализирующий выдачу команды top, доступной на большинстве UNIX-платформ. Традиционно эта программа используется в интерактивной форме для непосредственного наблюдения, однако в ней предусмотрен режим для пакетного запуска.

Мы запускали команду в следующем виде: top -b -n 1 (пакетный режим, одна итерация). Но и от этого варианта пришлось отказаться. Во-первых, в начале запуска сама программа top достаточно сильно загружает ОС, и данные о загрузке оказываются необъективными. Во-вторых, формат выдачи результатов сильно зависит от версии команды и дистрибутива. В двух дистрибутивах Linux (RedHat Linux 7.3 и SuSE Linux 10.0), установленных на узлах суперкомпьютерного комплекса НИВЦ МГУ, версии, а точнее, ветки развития команды top оказались принципиально разными.

Вот пример части экрана, получаемого на разных дистрибутивах:

RedHat 7.3

```
4:59pm up 321 days, 6:43, 18 users, load average: 0.09, 0.23, 0.19
```

```

292 processes: 264 sleeping, 1 running, 27 zombie, 0 stopped
CPU0 states: 50.0% user, 50.0% system, 50.0% nice, 0.0% idle
CPU1 states: 18.0% user, 81.0% system, 9.0% nice, 0.0% idle
Mem: 1033296K av, 1019892K used, 13404K free, 0K shrd, 51908K
buff
Swap: 530104K av, 127548K used, 402556K free 620732K cached
PID USER PRI NI SIZE RSS SHARE STAT %CPU %MEM TIME
COMMAND
22243 root 15 0 1156 1156 804 R 8.2 0.1 0:00 top
1 root 9 0 412 376 360 S 0.0 0.0 5:51 init

SuSE 10.0
top - 17:01:31 up 174 days, 5:21, 11 users, load average: 0.27, 0.27,
0.15
Tasks: 253 total, 2 running, 247 sleeping, 0 stopped, 4 zombie
Cpu0 : 0.3% us, 2.3% sy, 0.0% ni, 96.0% id, 0.0% wa, 0.0% hi, 1.3%
si
Cpu1 : 0.0% us, 0.0% sy, 0.0% ni, 100.0% id, 0.0% wa, 0.0% hi, 0.0%
si
Mem: 2050860k total, 1405736k used, 645124k free, 191172k buffers
Swap: 979176k total, 4056k used, 975120k free, 751500k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
25171 root 15 0 57644 37m 2132 S 0.3 1.9 11:18.15 cleo
23196 golovin 15 0 26644 2520 1940 S 0.3 0.1 3:05.63 ssh

```

Видно, что для человека разница небольшая, но программы разбора такой выдачи будут значительно отличаться. И что гораздо хуже — нет никакой гарантии, что не потребуется разбор и других вариантов выдачи, о которых мы пока не знаем, но которые могут встретиться на компьютерах вычислительной среды.

Аналогичная ситуация сложилась и с программой vmstat. На различных версиях ОС она выдаёт результаты в различном формате.

RedHat 7.3

```

procs memory swap io system cpu
r b w swpd free buff cache si so bi bo in cs us sy id
0 0 0 130848 14292 148368 441620 0 0 2 2 1 2 0 3 2

```

SuSE 10.0

```
procs -----memory----- swap-- io-- system--cpri--
r b swpd free buff cache si so bi bo in cs us sy id wa
0 0 134204 62872 238056 232792 0 1 1 3 5 6 11 4 80 4
```

Самой «стандартной» программой оказалась команда `uptime`, которая выдаёт значение `loadaverage` узла. Её вывод на большинстве UNIX-платформ практически единообразен и приемлем для программного анализа. Величина `loadaverage` равна среднему числу процессов, ожидающих квант процессорного времени в очереди. Например, если в данный момент на компьютере запущено 5 счётных задач, то `loadaverage` будет не меньше 5. Команда `uptime` при запуске выдает три значения `loadaverage`: за последнюю минуту, 5 минут и 15 минут.

Однако высокое значение `loadaverage`, на которое можно было бы ориентироваться, не обязательно означает высокую загрузку процессора. К примеру, задача может ожидать завершения системного вызова, такого как запись на жёсткий диск. При этом она будет находиться в очереди, и `loadaverage` увеличится на единицу, хотя реальная загрузка процессора может быть очень низкой.

Возможна ситуация, когда средняя загрузка возрастает только за счет системных задач, и в этом случае прикладное счетное приложение также снимается со счета. Обратная ситуация исключается, т.е. низкий уровень `loadaverage` гарантирует, что процессор не занят никакими задачами. Эмпирическим путем были установлены следующие пороги, по достижению которых узел считается «занятым» или «свободным» (`Ncpri` — общее число процессоров на узле, `loadaverage` — средняя загрузка узла за последнюю минуту).

- «свободен», $\text{loadaverage} < \text{Ncpri} * 0.25$ (предполагаем, что 25% процессорного времени может требоваться системным процессам);
- «занят», $\text{loadaverage} > \text{Ncpri} + 0.9$ (запуская столько процессов прикладной задачи, сколько процессоров на узле, проверяем, не появился ли еще хотя бы один активный процесс).

Так как значение `loadaverage` не «мгновенное», а усреднённое, то вероятность запуска задачи в момент кратковременногоостоя

компьютера снижается. По этой же причине определить факт возможности запуска счётной задачи удается только по прошествии какого-то времени.

Таким образом, решение поставленной задачи крайне осложнено тем, что общедоступный набор системных программ, во-первых, не всегда соответствует стандартам, и, во-вторых, не всегда выдаёт адекватную информацию.

Если ставить задачу достаточно широко, а именно определение загруженности компьютера в рамках наиболее распространённых ОС семейства UNIX, то, с нашей точки зрения, её решение практически невозможно. Это связано с тем, что не удается найти работающего одинаково на всех UNIX-платформах штатного средства, посредством которого можно было бы определять загрузку процессора, даже если у нас есть права суперпользователя.

Если ограничить задачу только ОС Linux, то решение затруднено, но возможно. Вероятное решение — использовать данные из файловой системы /proc. Эти данные заведомо актуальны и формат необходимых файлов не меняется в зависимости от дистрибутива. Препятствием может быть усиленная система безопасности, например gr-security или se-linux, но в этом случае получение необходимых нам данных будет практически невозможно.

Проблема с чтением данных из /proc состоит в том, что объём информации довольно велик — надо анализировать столько файлов, сколько сейчас существует процессов в системе. Это десятки, а иногда и сотни файлов. Их анализ в скрипте занимает значительное время, а значит, общая картина будет сильно искажаться. Можно анализировать эти данные в программе, написанной, например, на Си. Но в этом случае необходимо компилировать эту программу перед установкой в системе, так как бинарный файл может не работать на разных архитектурах, а иногда и на одной и той же архитектуре, но с разными версиями ОС. Компиляция же вспомогательного файла нежелательна, ведь на компьютере вычислительной среды может не быть компилятора.

В конечном итоге мы остановились на варианте использования программы uptime. Это решение не всегда даёт корректные результаты. Можно получить ответ «система занята» при том, что процессор простояивает. Но нельзя получить ответ «система свободна»,

если процессор реально занят счётыми задачами. Это значит, что используя такой подход, можно «прозевать» время, когда процессор простояивает, но нельзя запустить задачу, если процессор занят.

Интенсивное тестирование скрипта определения занятости серверов на узлах суперкомпьютерного комплекса НИВЦ МГУ (запуск заданий через систему метакомпьютинга на фоне работающих штатных приложений) показало пригодность описанного алгоритма для решения поставленной задачи, с небольшой оговоркой, что иногда будет возникать «иллюзия» занятости системы. При тестировании было задействовано более 100 вычислительных узлов трёх вычислительных кластеров в течении нескольких месяцев.

Тем не менее, поиски оптимального решения продолжаются. Мы приглашаем к обсуждению специалистов, сталкивающихся с подобными задачами, присоединиться к дискуссии по определению загрузки серверов можно на форуме сайта PARALLEL.RU (<http://parallel.ru/forum/>).

Размерности коммутативных матричных алгебр[§]

О. С. ЛЕВЕДЕВА[®], Е. Е. ТЫРТЫШНИКОВ[®]

Рассмотрена задача построения верхних оценок размерностей матричных коммутативных алгебр. Задача для произвольных матричных алгебр сводится к задаче для алгебр из матриц специального вида. Получено новое простое доказательство общей точной оценки $n^2/4 + 1$; построены новые оценки, зависящие от дополнительных характеристик, связанных с дефектами матриц.

1. Введение

Под матричной алгеброй будем понимать линейное подпространство $\Omega \subseteq \mathbb{C}^{n \times n}$, замкнутое относительно операции умножения. Коммутативность означает, что все матрицы Ω попарно перестановочны: $\forall A, B \in \Omega : AB = BA$. Будем рассматривать только коммутативные алгебры. При этом не требуем, чтобы в Ω содержалась единичная матрица I .

Нас будут интересовать различные оценки размерности Ω , зависящие от n и других возможных величин, характеризующих алгебру Ω . Сразу можно отметить, что нижняя оценка равна 1 (множество скалярных матриц является алгеброй). Элементарно получается верхняя оценка $\dim(\Omega) < n^2$ при $n \geq 2$ (поскольку если $\dim \Omega = n^2$, то $\Omega = \mathbb{C}^{n \times n}$, а в $\mathbb{C}^{n \times n}$ при $n \geq 2$ всегда есть пара некоммутирующих матриц). Попытаемся построить более тонкие верхние оценки.

Известны алгебры размерности $n^2/4 + 1$. Например, множество

[§]Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ №05-01-00721)

[®]Институт вычислительной математики РАН

всех матриц вида

$$\alpha I + \begin{bmatrix} 0 & A \\ 0 & 0 \end{bmatrix} \in \mathbb{C}^{n \times n}, \quad \alpha \in \mathbb{C}, \quad A \in \mathbb{C}^{n/2 \times n/2} \quad (1)$$

является коммутативной матричной алгеброй такой размерности.

В случае матриц, подчинённых некоторым условиям, для нас представляют интерес оценки вида $c n^2 + O(n)$ ($c < 1$). Довольно легко получить оценку $n^2/2 + 1$, но она оказывается сильно завышена. Точная верхняя оценка $n^2/4 + 1$ была установлена ещё Шуром. Доказательство этого факта в русскоязычной литературе опирается на теорию нормальных форм Кравчука. Эта теория представлена в [1] в виде последовательности трёх основных теорем Кравчука и различных вспомогательных утверждений. Изучая доказательства, мы заметили, что в теореме 1 из [1] частично используется теорема 3, которая, в свою очередь, опирается на теорему 1. Все утверждения остаются, конечно, в силе, а замеченная неточность может быть исправлена. В данной работе мы покажем, что, оставаясь в том же круге идей, можно дать простое прямое доказательство теоремы Шура, не опирающееся на теорию Кравчука.

Особый интерес представляют ситуации, когда кроме размеров матриц алгебры известны какие-то дополнительные её характеристики. Мы получим оценки размерностей этих алгебр через характеристики, связанные с дефектами матриц. Но сначала поясним, как от случая произвольной матричной алгебры перейти к рассмотрению более узкого класса алгебр.

При изучении коммутативных алгебр особую роль играют классы алгебр, в которых каждая матрица имеет единственное собственное значение. Если все матрицы алгебры нильпотентны (то есть все собственные значения всех матриц равны 0), то алгебра также называется *нильпотентной*. Очевидно, в нильпотентной алгебре единицы нет.

Матрицы, отличающиеся от нильпотентных сдвигом на скалярную матрицу, будем называть *квазинильпотентными*, как и алгебры, состоящие только из таких матриц. Каждой квазинильпотентной алгебре соответствует единственная нильпотентная: $\Omega = \{\alpha I + C \mid \alpha \in \mathbb{C}, C \in \Omega_{\text{nil}}\}$, при этом их размерности отличаются на единицу $\dim(\Omega) = \dim(\Omega_{\text{nil}}) + 1$.

Существует способ свести вопрос оценки размерности произвольной матричной алгебры Ω к вопросу построения оценок для набора квазинильпотентных и нильпотентных алгебр, состоящих из матриц меньших размеров. Мы будем пользоваться тем, что одинаковое преобразование подобия для всех матриц коммутативной алгебры порождает новую коммутативную алгебру той же размерности (это объясняется тем, что матрицы базисов попарно подобны). Такие алгебры будем называть подобными.

Выберем некоторый произвольный базис $\Omega: A_0, A_1, \dots, A_{m-1}$. Эти матрицы попарно коммутируют, поэтому существует такая обратимая матрица $C \in \mathbb{C}^{n \times n}$, что $C^{-1}A_i C = \text{diag}(B_1^i, B_2^i, \dots, B_p^i)$ — блочно-диагональные матрицы (для $i = 0, \dots, m-1$), и каждый диагональный блок $B_j^i \in \mathbb{C}^{n_j \times n_j}$ обладает единственным собственным значением (то есть нильпотентен либо квазинильпотентен). В силу разложимости остальных матриц из Ω по базису, пространство $\Omega' = \text{span}(C^{-1}A_0 C, \dots, C^{-1}A_{m-1} C)$ — подобная Ω коммутативная матричная алгебра — будет содержать только блочно-диагональные матрицы, и размерность при этом сохранится $\dim(\Omega) = \dim(\Omega')$.

Из коммутации матриц $C^{-1}A_0 C, \dots, C^{-1}A_{m-1} C$ следует, что блоки $B_j^1, B_j^2, \dots, B_j^m$ будут также попарно коммутировать для $j = 1, \dots, p$. Поэтому множества $\Omega_j = \text{span}(B_j^0, B_j^1, \dots, B_j^{m-1})$ будут матричными коммутативными алгебрами — нильпотентными либо квазинильпотентными. Рассмотрим прямую сумму алгебр Ω_j : $\Omega_\Sigma = \Omega_1 \oplus \dots \oplus \Omega_p = \{\text{diag}(A_1, \dots, A_p) | A_j \in \Omega_j\} \supseteq \Omega'$. Это также матричная коммутативная алгебра, и Ω' вложена либо равна ей. Получаем $\dim(\Omega) = \dim(\Omega') \leq \dim(\Omega_\Sigma) = \sum_j \dim(\Omega_j)$.

Чтобы сократить произвол в выборе блочно-диагонального представления и добиться строгого равенства: $\Omega = \Omega' \sim \Omega_1 \oplus \dots \oplus \Omega_p$, поступим следующим образом. Среди всех матриц Ω найдём матрицу с наибольшим числом различных ненулевых собственных значений: предположим это матрица, обладающая q различными собственными значениями. Тогда существует преобразование подобия, переводящее матрицы Ω в блочно-диагональные матрицы с q квазинильпотентными блоками, при этом размер каждого блока будет равен l_i — алгебраической кратности соответствующего собственного значения λ_i выбранной матрицы (этую алгебру, подобную Ω обозначим Ω'). По этому преобразованию однозначно строятся ква-

зинильпотентные алгебры $\Omega_1, \dots, \Omega_q$ (среди них, возможно, есть одна нильпотентная, в таком случае договоримся считать, что это Ω_q). Тогда будет иметь место равенство: $\Omega = \Omega' \sim \Omega_1 \oplus \dots \oplus \Omega_q$ (и, следовательно, $\dim(\Omega) = \sum_i \dim(\Omega_i)$). Докажем это.

В силу выбора алгебры Ω' , в ней будет содержаться матрица A с q различными собственными значениями, подобная выбранной на- ми матрице. С помощью этой матрицы мы построим базис алгебры Ω' из матриц $X_1^1, \dots, X_{\dim(\Omega_1)}^1, \dots, \dots, X_1^q, \dots, X_{\dim(\Omega_q)}^q$, где каждая матрица X_j^i содержит единственный ненулевой блок на главной диагонали в i -той позиции, $i = 1, \dots, q$, $j = 1, \dots, \dim(\Omega_i)$. При этом $\text{span}(X_1^1, \dots, X_{\dim(\Omega_i)}^i) = \{0\} \oplus \dots \oplus \{0\} \oplus \Omega_i \oplus \{0\} \oplus \dots \oplus \{0\}$. Из существования такого базиса непосредственно будет следовать доказываемое утверждение.

Нетрудно заметить, что если мы найдём q матриц Y_1, \dots, Y_q из Ω' , содержащих на главной диагонали ровно по одному квази- зинильпотентному блоку размера l_1, \dots, l_q соответственно, то требуемый базис можно будет выбрать из матриц любого базиса Ω' , умноженных на Y_1, \dots, Y_q . Если же один из диагональных блоков нильпотентен, то, оказывается, таких матриц Y_i можно выбрать только $q - 1$, с их помощью составляется базис алгебры $(\Omega_1 \oplus \dots \oplus \Omega_{q-1} \oplus \{0\})$, который можно дополнить до базиса Ω' матрицами с последним нильпотентным блоком.

Покажем, как избавиться от всех блоков матрицы A , кроме первого невырожденного (другими словами, как получить Y_1). Заме- тим, что матрица $\lambda_q A - A^2$ имеет ровно q собственных значений, последнее из которых 0 с алгебраической кратностью l_q . Обнулим последний блок, возведя её в степень $l_q - 1$. Полученную матрицу обозначим A_1 . Она имеет на главной диагонали $q - 1$ квазиниль- потентный блок и один нулевой (последний). При этом собственные значения, соответствующие разным блокам, могут повторяться. Те- перь избавимся от $(q - 1)$ -ого блока. Если $\lambda_1(A_1) \neq \lambda_{q-1}(A_1)$, то у матрицы $A_2 = (\lambda_{q-1}A_1 - (A_1)^2)^{l_{q-1}}$ на главной диагонали приба- вится ещё хотя бы один нулевой блок (при этом первый блок оста- нется невырожденным). Если же $\lambda_1(A_1) = \lambda_{q-1}(A_1)$, то вместо A_1 возьмём матрицу AA_1 , и для неё проделаем указанные действия. И так далее.

Не более, чем за $q - 1$ шагов получим матрицу с единственным

блоком размера l_1 в левом верхнем углу. Если среди диагональных блоков A нет нильпотентных, то все матрицы Y_1, \dots, Y_q строятся аналогично в виде многочленов от A , в противном случае строим только Y_1, \dots, Y_{q-1} .

Таким образом, произвольная матричная коммутативная алгебра подобна прямой сумме матричных коммутативных квазинильпотентных и нильпотентных алгебр, поэтому в дальнейшем ограничимся поиском оценок для них.

Покажем, как получается грубая оценка, зависящая только от n . Для этого, как и было предложено, сначала оценим размерность квазинильпотентной алгебры $\Omega_j \in \mathbb{C}^{n_j \times n_j}$ (в предположении, что $\Omega \sim \Omega' \subseteq \Omega_1 \oplus \dots \oplus \Omega_p$). Известно, что набор попарно коммутирующих матриц одновременно приводится к верхнетреугольному виду. Проделав эту операцию с матрицами базиса Ω_j , получим набор линейно независимых верхнетреугольных матриц с одинаковыми значениями на главной диагонали (это следует из квазинильпотентности). Таких матриц не более $n_j(n_j - 1)/2 + 1 = n_j^2/2 - n_j/2 + 1$. Отсюда получаем

$$\dim(\Omega') \leq \sum_{j=1}^p (n_j^2/2 - n_j/2 + 1) \leq n^2/2 - n/2 + 1.$$

Эта оценка является точной только для $n \leq 2$, поскольку при $n \geq 3$ всегда существует пара верхнетреугольных матриц, которые не будут коммутировать.

Теперь попытаемся использовать дополнительную информацию об алгебре. Для этого будем рассматривать величины, которые могут её характеризовать. Мы вводим для нильпотентных алгебр характеристики, связанные с дефектами содержащихся в них матриц. Для определения характеристик квазинильпотентных алгебр мы пользуемся тем, что любой квазинильпотентной алгебре соответствует единственная нильпотентная. Для произвольной же алгебры Ω характеристика задаётся совокупностью характеристик алгебр-слагаемых, входящих в разложение Ω в виде прямой суммы (в частности, она может равняться максимальной из характеристик для алгебр-слагаемых).

Первую используемую характеристику будем называть *общим дефектом* (или просто *дефектом*) и обозначать l_Ω или просто l .

Дефект нильпотентной алгебры Ω равен размерности её общего ядра, то есть:

$$l = \dim \bigcap_{C \in \Omega} \ker(C) = \dim\{x \in \mathbb{C}^n \mid \forall C \in \Omega : Cx = 0\} \quad (2)$$

Из нильпотентности следует $l \geq 1$ (так как у набора коммутирующих матриц всегда есть общий собственный вектор, а собственное значение у матриц нильпотентной алгебры единствено и равно нулю). Кроме того, $l < n$, если алгебра содержит хотя бы одну ненулевую матрицу.

Мы выведем оценку $\dim(\Omega) \leq l(n - l)$ (где Ω нильпотентная алгебра). Из неё, в частности, следует точная оценка в общем случае, равная $n^2/4 + 1$ (она получается максимизацией оценки по l). Мы приводим доказательство без использования нормальных форм и теорем Кравчука.

Другую характеристику алгебры будем называть *локальным дефектом* и обозначать k . Она определяется неоднозначно: мы будем говорить, что нильпотентная алгебра Ω обладает локальным дефектом k , если в ней есть хотя бы одна матрица с дефектом k . Таким образом, даже по единственной известной матрице из Ω мы сможем строить оценки размерности Ω .

Для получения оценок k мы будем пользоваться известными особенностями вида матриц, коммутирующих с жордановой формой. Выводятся оценки $\dim(\Omega) < n(k - 1/k)$ для $k > 2$, $\dim(\Omega) = n - 1$ для $k = 1$ и $\dim(\Omega) \leq 5/4n$ для $k = 2$ (для нильпотентных алгебр). Поскольку полученные оценки монотонно возрастают по k , для наилучшей оценки вводится следующая характеристика.

Минимальный дефект (k_{\min}) определяется как минимальный из локальных дефектов:

$$k_{\min} = \min_{C \in \Omega} \dim(\ker(C)) = \min_{C \in \Omega} \dim\{x \in \mathbb{C}^n \mid Cx = 0\} \quad (3)$$

С помощью k_{\min} из оценок по локальному дефекту k выбирается наилучшая оценка. Очевидно, что $l \leq k_{\min} \leq k$. Этим также можно пользоваться для улучшения оценки.

Для произвольной алгебры Ω , подобной прямой сумме квазинильпотентных и нильпотентных алгебр ($\Omega \sim \Omega_1 \oplus \dots \oplus \Omega_q$), однозначно с точностью до перестановок определяются наборы характеристик алгебр-слагаемых: $\{l_1, \dots, l_q\}$, $\{k_{\min 1}, \dots, k_{\min q}\}$, и уже по

ним строятся оценки размерности исходной алгебры Ω :

$$\dim(\Omega) = \sum_{j=1}^q \dim(\Omega_j) \quad (4)$$

2. Оценки с использованием общего дефекта

Рассмотрим произвольную коммутативную нильпотентную алгебру Ω . Поскольку все матрицы из Ω обладают единственным собственным значением 0, в общем ядре Ω

$$L = \bigcap_{C \in \Omega} \ker(C) = \{x \in \mathbb{C}^n \mid \forall C \in \Omega : Cx = 0\} \quad (5)$$

будет содержаться по меньшей мере один вектор — общий собственный вектор всех матриц из Ω (как известно, он существует у любого набора коммутативных матриц).

Как уже было сказано, общий дефект нильпотентной алгебры Ω — это размерность её общего ядра L . Если известен дефект Ω , то существует преобразование подобия, переводящее все матрицы Ω к специальному виду.

Утверждение 1. Нильпотентная алгебра Ω с дефектом l подобна алгебре, каждая матрица которой содержит ровно l нулевых столбцов в одних и тех же позициях. В частности, все матрицы Ω одновременно приводятся к виду

$$\begin{bmatrix} 0_{l \times l} & a_{l \times (n-l)} \\ 0_{(n-l) \times l} & b_{(n-l) \times (n-l)} \end{bmatrix} \quad (6)$$

Не существует алгебры, подобной Ω , все матрицы которой в одних и тех же позициях содержат более l нулевых столбцов.

Доказательство. В L построим базис a_1, \dots, a_l и дополним его до базиса \mathbb{C}^n . В новом базисе Ω примет вид (6). Если предположить, что в некотором базисе все матрицы из Ω содержат более l нулевых столбцов в одних и тех же позициях, это будет означать, что более l векторов нового базиса лежат в L . Получаем противоречие с тем, что размерность L равна l . \square

Теперь мы можем рассматривать только алгебры с матрицами в форме (6). Выберем базис A_1, \dots, A_s , где $s = \dim(\Omega)$. Для правых верхних блоков матриц базиса верно следующее утверждение:

Утверждение 2. Пусть матрицы базиса Ω имеют вид

$$\Lambda_i = \begin{bmatrix} 0_{l \times l} & a_{l \times (n-l)}^i \\ 0_{(n-l) \times l} & b_{(n-l) \times (n-l)}^i \end{bmatrix}, \quad i = 1, \dots, s. \quad (7)$$

Тогда матрица

$$\mathfrak{A} = \begin{bmatrix} a^1 \\ \vdots \\ a^s \end{bmatrix} \in \mathbb{C}^{(ls) \times (n-l)},$$

составленная из блоков a^i матриц базиса Λ_i , имеет полный столбцовый ранг: $\text{rank}(\mathfrak{A}) = n - l$.

Доказательство. Докажем это от противного. Предположим, что $\text{rank}(\mathfrak{A}) = r < n - l$. Из этого следует, что первые $n - l - r$ столбцов этой матрицы можно обнулить:

$$\exists F: F^{-1}\mathfrak{A}F = \begin{bmatrix} 0_{(n-l) \times (n-l-r)} & \tilde{\mathfrak{A}} \end{bmatrix} \quad \tilde{\mathfrak{A}} = \begin{bmatrix} \tilde{a}^1 \\ \vdots \\ \tilde{a}^s \end{bmatrix} \in \mathbb{C}^{(ls) \times r}.$$

Соответствующее преобразование подобия для матриц алгебры Ω приведёт матрицы базиса к следующему виду:

$$\begin{aligned} \tilde{\Lambda}_i &= \begin{bmatrix} I & 0 \\ 0 & F^{-1} \end{bmatrix} \Lambda_i \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix} = \\ &= \begin{bmatrix} 0_{l \times l} & 0_{l \times (n-l-r)} & \tilde{a}_{l \times r}^i \\ 0_{(n-l-r) \times l} & b_{11(n-l-r) \times (n-l-r)}^i & b_{12(n-l-r) \times r}^i \\ 0_{r \times l} & b_{21r \times (n-l-r)}^i & b_{22r \times r}^i \end{bmatrix} \end{aligned}$$

для $i = 1, \dots, s$. Все остальные матрицы алгебры будут иметь ту же структуру (поскольку они линейно выражаются через $\tilde{\Lambda}_1, \dots, \tilde{\Lambda}_s$).

После этого для любой матрицы из условия коммутации с $\tilde{\Lambda}_1, \dots, \tilde{\Lambda}_s$ получаем, что $\tilde{a}^i b_{21} = 0$ для $i = 1, \dots, s$. Это равносильно

$$\begin{bmatrix} \tilde{a}^1 \\ \vdots \\ \tilde{a}^s \end{bmatrix} b_{21} = 0$$

Матрица \mathfrak{A} имеет полный столбцовый ранг, следовательно $b_{21} = 0$.

Таким образом, матрицы алгебры имеют вид

$$\begin{bmatrix} 0 & 0 & \tilde{a}^i \\ 0 & b_{11}^i & b_{12}^i \\ 0 & 0 & b_{22}^i \end{bmatrix} \quad (8)$$

Из нильпотентности всей матрицы следует нильпотентность её диагональных блоков — b_{11}, b_{22} . Поэтому, если привести матрицы алгебры к верхнетреугольной форме, в блоке b_{11} появится нулевой первый столбец. Следовательно, во всех матрицах получившейся алгебры (подобной Ω) нулевых столбцов не менее $l+1$, и все эти столбцы стоят в одних и тех же позициях (в начале). Таким образом мы приходим к противоречию с тем, что $\dim(L) = l$. Значит, действительно, $\text{rank}(\mathfrak{A}) = n - l$, что и требовалось показать.

Теперь для доказательства основной оценки остаётся только показать связь между блоками a и b для матрицы в форме (6). \square

Утверждение 3. Если все матрицы из алгебры Ω имеют вид (6), то из того, что блок a какой-либо матрицы нулевой, следует, что её блок b также окажется нулевым.

Доказательство. Чтобы показать это, запишем условие коммутации такой матрицы со всеми матрицами базиса:

$$\begin{bmatrix} 0 & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} 0 & a^i \\ 0 & b^i \end{bmatrix} = \begin{bmatrix} 0 & a^i \\ 0 & b^i \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & b \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & bb^i \end{bmatrix} = \begin{bmatrix} 0 & a^i b \\ 0 & b^i b \end{bmatrix}.$$

Отсюда $a^i b = 0$, $i = 1, \dots, s$. Следовательно,

$$\begin{bmatrix} a^1 \\ \dots \\ a^s \end{bmatrix} b = \tilde{A}b = 0$$

Как было показано в утверждении 2, матрица \tilde{A} имеет полный столбцовый ранг, а значит $b = 0$. \square

Утверждение 3 означает, что для матрицы в форме (6) по блоку a однозначно определяется блок b . В самом деле, если

$$\begin{bmatrix} 0 & a \\ 0 & b_1 \end{bmatrix}, \begin{bmatrix} 0 & a \\ 0 & b_2 \end{bmatrix} \in \Omega,$$

то

$$\begin{bmatrix} 0 & a \\ 0 & b_1 \end{bmatrix} - \begin{bmatrix} 0 & a \\ 0 & b_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & b_1 - b_2 \end{bmatrix} \in \Omega.$$

Из утверждения 3 получаем, что $b_1 - b_2 = 0$. Значит, линейно независимых матриц в Ω не более, чем линейно независимых блоков a . Из этого уже непосредственно следует основная оценка.

Теорема 1. *Нильпотентная алгебра с дефектом l имеет размерность не более $l(n - l)$.*

Доказательство. Из утверждения 1 следует, что Ω подобна алгебре с матрицами в форме

$$\begin{bmatrix} 0 & a \\ 0 & b \end{bmatrix}. \quad (9)$$

Из утверждения 3 следует, что блок b однозначно определяется блоком a . Блок a имеет размер $l \times (n - l)$, значит различных линейно независимых блоков a не более $l(n - l)$.

Оценка, доказанная в теореме 1, не является монотонной по l , а достигает своего максимального значения $[n^2/4]$ при $l = [n/2]$. Для квазинильпотентной и произвольной алгебры отсюда следует оценка Шура

$$\dim(\Omega) \leq [n^2/4] + 1. \quad (10)$$

Как было показано в (1), существуют алгебры такой размерности, а значит, полученная оценка (10) является точной.

3. Оценки с использованием локального дефекта

Для построения оценок мы будем пользоваться тем, что если алгебра Ω вложена в некоторое линейное подпространство X , то $\dim(\Omega) \leq \dim(X)$. Если выбрать несколько матриц из Ω и задавать подпространство X как множество матриц, коммутирующих с выбранными, то, очевидно, $\Omega \subseteq X$. При выводе оценок мы будем выбирать из Ω две матрицы таким образом, чтобы максимально сузить X .

3.1. $k = 1$. Пусть в квазинильпотентной алгебре Ω есть матрица A , имеющая собственное значение λ_A геометрической кратности 1.

Перейдём к подобной алгебре Ω' , получаемой с помощью преобразования, переводящего A в жорданову форму. В Ω' есть единица, следовательно, $A' - \lambda_A I = J$ — жорданова клетка размера n — также содержится в Ω' . Найдём размерность $\tilde{\Omega}$ — множества матриц, коммутирующих с J . Очевидно, что $\Omega' \subseteq \tilde{\Omega}$, однако, можно утверждать, что $\Omega' = \tilde{\Omega}$.

Жорданова клетка J , как известно, коммутирует со всеми верхнетреугольными тёплацевыми матрицами:

$$\begin{bmatrix} b_0 & b_1 & \cdots & b_{t-1} \\ & b_0 & \cdots & b_{t-2} \\ & & \ddots & \vdots \\ 0 & & & b_0 \end{bmatrix} \quad (11)$$

Линейно независимых верхнетреугольных тёплацевых матриц всего n , и все они линейно выражаются через $I, J, J^2, \dots, J^{n-1}$ (а все эти матрицы обязательно содержатся в Ω' , поскольку алгебра по определению замкнута относительно умножения). Поэтому $\dim(\tilde{\Omega}') = n$.

Таким образом, размерность квазинильпотентной алгебры $\Omega \subseteq \mathbb{R}^{n \times n}$, содержащей матрицу геометрической кратности 1, в точности равна n .

3.2. $k = 2$. Пусть в квазинильпотентной алгебре Ω есть матрица A с собственным значением $\lambda_A = 0$ геометрической кратности 2 (единственное собственное значение A). Её жорданова форма $J = D^{-1}AD$ будет состоять из двух жордановых клеток размеров s и t : $J = \text{diag}(J_s, J_t)$. Будем считать, что $s \leq t$. Как и в случае $k = 1$, перейдём к подобной алгебре $\Omega' = \{D^{-1}CD \mid C \in \Omega\}$. Таким образом $J \in \Omega'$

Построим базис Ω' : B_0, \dots, B_{m-1} . Матрицы $I, J, J^2, \dots, J^{t-1}$ содержатся в Ω' и линейно независимы (кроме того, при $i \geq t$ $J^i = 0$). Поэтому их можно взять в качестве первых t матриц базиса: $B_i = J^i$, $i = 0, \dots, t-1$.

Очевидно, что с единичной матрицей B_0 коммутирует любая матрица соответствующего размера. Если некоторая матрица коммутирует с $B_1 = J$, то она также будет коммутировать с B_2, \dots, B_{t-1} .

Таким образом, получив размерность подпространства $\{C \in \mathbb{C}^{n \times n} | CJ - JC = 0\}$, мы будем иметь оценку для $\dim(\Omega')$. Но мы не ограничимся этим, а возьмём более узкое подпространство. Для этого некоторым оптимальным образом выберем среди матриц Ω' вторую матрицу K , не выражающуюся линейно через B_0, \dots, B_{t-1} , и будем оценивать размерность $\tilde{\Omega} = \{C \in \mathbb{C}^{n \times n} | CJ - JC = CK - KC = 0\}$. Очевидно, что $\Omega' \subseteq \tilde{\Omega}$, однако, в некоторых случаях оказывается, что $\Omega' = \tilde{\Omega}$, таким образом получаемые оценки будут точны.

Дальнейшие рассуждения зависят от того, равны ли размеры жордановых клеток.

3.2.1. Случай одинаковых жордановых блоков $s = t = n/2..$
Из условия коммутации с $B_1 = J$ следует, что матрицы B_t, \dots, B_{m-1} имеют вид

$$B_i = \begin{bmatrix} B_{i1} & B_{i2} \\ B_{i3} & B_{i4} \end{bmatrix}, \quad (12)$$

где каждый из блоков B_{ij} ($i = t, \dots, m-1$) является верхнетреугольной тёплицевой матрицей.

Отнимая от B_i ($i = t, \dots, m-1$) линейные комбинации I, J, \dots, J^{t-1} легко получить

$$B'_i = \begin{bmatrix} B'_{i1} & B'_{i2} \\ B'_{i3} & 0 \end{bmatrix} = \begin{bmatrix} B_{i1} - B_{i4} & B_{i2} \\ B_{i3} & 0 \end{bmatrix}, \quad (13)$$

при этом блоки B'_{ij} также будут верхнетреугольными тёплицевыми и, кроме того, нильпотентными (поскольку матрицы B'_i нильпотентны).

Теперь выберем матрицу K — вторую матрицу, с которой коммутируют элементы $\tilde{\Omega}$. Для этого среди блоков B'_{ij} ($i = t, \dots, m-1; j = 1, 2, 3$) найдём такой, у которого ненулевые элементы расположены ближе всего к диагонали:

$(B'_{ij})_0 = (B'_{ij})_1 = \dots = (B'_{ij})_{h-1} = 0$ для $\forall i, j$, и $\exists i_0, j_0$ такие, что $(B'_{i_0 j_0})_h \neq 0$

(под $(B'_{ij})_l$ понимаем элемент верхнетреугольной тёплицевой матрицы B'_{ij} , расположенный на l -той диагонали).

Таким образом, мы выбираем матрицу, содержащую верхнетреугольный тёплицев блок максимального ранга. Из нильпотентности и вернетреугольности блоков B'_{ij} следует, что $h \geq 1$. Если таких блоков несколько, можно взять любой. Теперь в качестве матрицы

K выбираем B_{i_0} . Как было сказано, будем оценивать размерность $\tilde{\Omega} = \{C \mid CJ - JC = CK - KC = 0\}$.

Для этого среди матриц

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & 0 \end{bmatrix} \quad (14)$$

с верхнетреугольными тёпллицевыми блоками найдём все такие, которые коммутируют с B'_{i_0} (коммутация с J обеспечивается специальным видом матриц C).

Расписав блочное произведение $B'_{i_0} C = CB'_{i_0}$ и воспользовавшись тем, что верхнетреугольные тёпллицевые матрицы всегда коммутируют, получим:

$$\begin{aligned} B'_{i_0 3} C_2 &= B'_{i_0 2} C_3 \\ B'_{i_0 1} C_3 &= B'_{i_0 3} C_1 \\ B'_{i_0 2} C_1 &= B'_{i_0 1} C_2 \end{aligned} \quad (15)$$

Если $h \geq t/2$, то соотношения (15) всегда будут выполняться (поскольку каждое из произведений будет в точности равно 0). Таких линейно независимых матриц C будет не более $3(t-h)$, а значит, и не более $3/4n$. Кроме того, нужно учесть матрицы $I, J, J^2, \dots, J^{t-1}$, которых ровно $n/2$. Итого получится $\dim(\tilde{\Omega}) \leq 5/4n$.

Теперь рассмотрим случай $h < t/2$. Из соотношений (15) нельзя почерпнуть никакую информацию об элементах $c_{t-h}^1, \dots, c_{t-1}^1, c_{t-h}^2, \dots, c_{t-1}^2$ и $c_{t-h}^3, \dots, c_{t-1}^3$ (так как они участвуют с нулевыми сомножителями), и поэтому эти элементы задаются произвольно. Остальные же элементы ($c_h^1, \dots, c_{t-h-1}^1, c_h^2, \dots, c_{t-h-1}^2$ и $c_h^3, \dots, c_{t-h-1}^3$) однозначно определяются из (15) по известным $c_{i_0}^{j_0}, \dots, c_{t-h-1}^{j_0}$.

Чтобы показать это, выпишем потенциально-ненулевые блоки матриц $B'_{i_0 j}$ размеров $(t-h) \times (t-h)$:

$$B'_{i_0 j} = \begin{bmatrix} 0 & \widehat{B}_j \\ 0 & 0 \end{bmatrix} \quad (16)$$

где $j = \underbrace{1, 2, 3}$ (при этом хотя бы одна из матриц $\widehat{B}_1, \widehat{B}_2$ и \widehat{B}_3 , а именно \widehat{B}_{j_0} , гарантированно невырожденная).

$$\widehat{B}_j = \begin{bmatrix} b_h^j & b_{h+1}^j & \dots & b_{t-2}^j & b_{t-1}^j \\ 0 & b_h^j & \dots & b_{t-3}^j & b_{t-2}^j \\ \vdots & \dots & \ddots & \vdots & \vdots \\ \vdots & \dots & \dots & b_h^j & b_{h+1}^j \\ 0 & \dots & \dots & 0 & b_h^j \end{bmatrix}, j = 1, 2, 3$$

В соотношении (15) блоки \widehat{B}_j будут умножаться на блоки матриц C_j , размера $(t-h) \times (t-h)$, расположенные в правом нижнем углу:

$$C_j = \begin{bmatrix} \dots & \dots \\ 0 & \widehat{C}_j \end{bmatrix}, j = 1, 2, 3$$

При этом блоки \widehat{C}_j будут иметь вид:

$$\widehat{C}_j = \begin{bmatrix} 0 & \dots & 0 & c_h^j & c_{h+1}^j & \dots & c_{t-h-1}^j \\ 0 & \dots & 0 & 0 & c_h^j & \dots & c_{t-h-2}^j \\ \vdots & \dots & \dots & \dots & \dots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & c_h^j \\ \vdots & \dots & \dots & \dots & \dots & \dots & \vdots \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 \end{bmatrix}, j = 1, 2, 3 \quad (17)$$

Тогда (15) эквивалентно $\widehat{B}_3 \widehat{C}_2 = \widehat{B}_2 \widehat{C}_3$, $\widehat{B}_1 \widehat{C}_3 = \widehat{B}_3 \widehat{C}_1$ и $\widehat{B}_2 \widehat{C}_1 = \widehat{B}_1 \widehat{C}_2$.

Так как матрица \widehat{B}_{j_0} невырождена, по известному блоку \widehat{C}_{j_0} определяются остальные блоки \widehat{C}_j .

Итого получаем

$$((t-h-1)-(h-1)) + 3(t-1-(t-h-1)) = t+h \leq 3/4n \quad (18)$$

степеней свободы для матриц C . С учётом $I, J, J^2, \dots, J^{t-1}$ будем иметь $\dim(\tilde{\Omega}) \leq 5/4n$.

3.2.2. Случай разных жордановых блоков $s < t$, $s+t = n$.
Из условия коммутации с $B_1 = J$ следует, что матрицы B_t, \dots, B_{m-1} имеют вид

$$B_i = \begin{bmatrix} B_{i1} & B_{i2} \\ B_{i3} & B_{i4} \end{bmatrix}, \quad (19)$$

где блоки B_{i1}, B_{i4} ($i = t, \dots, m - 1$) верхнетреугольные тёплицевы (размеров $s \times s$ и $t \times t$ соответственно), а блоки B_{i1}, B_{i4} имеют вид:

$$B_{i2} = \begin{bmatrix} 0 & \widetilde{B_{i2}} \end{bmatrix}, \quad B_{i3} = \begin{bmatrix} \widetilde{B_{i3}} \\ 0 \end{bmatrix},$$

где блоки $\widetilde{B_{i1}}, \widetilde{B_{i3}}$ верхнетреугольные тёплицевы размеров $s \times s$.

Отнимая от B_i ($i = t, \dots, m - 1$) линейные комбинации I, J, \dots, J^{t-1} можно обнулить блоки B_{i4} , при этом остальные блоки сохранят указанную структуру. Получим

$$B'_i = \begin{bmatrix} B'_{i1} & B'_{i2} \\ B'_{i3} & 0 \end{bmatrix}, \quad (20)$$

$$B'_{i2} = \begin{bmatrix} 0 & \widetilde{B'_{i2}} \end{bmatrix} = \begin{bmatrix} 0 & \widetilde{B_{i2}} \end{bmatrix}, \quad B'_{i3} = \begin{bmatrix} \widetilde{B'_{i3}} \\ 0 \end{bmatrix} = \begin{bmatrix} \widetilde{B_{i3}} \\ 0 \end{bmatrix} \quad (21)$$

Условие коммутации для двух матриц B'_i, B'_j (i и j любые от t до $m - 1$):

$$B'_{i2} B'_{j3} = B'_{j2} B'_{i3} \quad (22)$$

$$B'_{i1} B'_{j2} = B'_{j1} B'_{i2} \quad (23)$$

$$B'_{i3} B'_{j1} = B'_{j3} B'_{i1} \quad (24)$$

$$B'_{i3} B'_{j2} = B'_{j3} B'_{i2} \quad (25)$$

В отличие от случая $t = s$, где блоки матриц B'_i, B'_j квадратные размера $t \times t$, мы не можем воспользоваться коммутативностью произведения блоков (поскольку блоки $B_{i1}, B_{i4}, B_{j1}, B_{j4}$ уже не квадратные). Зато, благодаря представлению (21), соотношения (23), (24), (25) эквивалентны следующим соотношениям для квадратных блоков размера $s \times s$:

$$\widetilde{B'_{i1}} \widetilde{B'_{j2}} = \widetilde{B'_{j1}} \widetilde{B'_{i2}} \quad (26)$$

$$\widetilde{B'_{i3}} \widetilde{B'_{j1}} = \widetilde{B'_{j3}} \widetilde{B'_{i1}} \quad (27)$$

$$\widetilde{B'_{i3}} \widetilde{B'_{j2}} = \widetilde{B'_{j3}} \widetilde{B'_{i2}} \quad (28)$$

(очевидно, откинув условие (22), мы не уменьшили количество матриц, удовлетворяющих условиям коммутации).

Теперь легко заметить, что полученные соотношения (26), (27), (28) обеспечивают коммутацию матриц $\widehat{B}_i, \widehat{B}_j \in \mathbb{C}^{2s \times 2s}$ с квадратными вернетреугольными тёплицевыми блоками:

$$\widehat{B}'_i = \begin{bmatrix} B'_{i1} & \widehat{B}'_{i2} \\ B'_{i3} & 0 \end{bmatrix}, \quad \widehat{B}'_j = \begin{bmatrix} B'_{j1} & \widehat{B}'_{j2} \\ B'_{j3} & 0 \end{bmatrix}$$

Мы показали, что если матрицы B'_i, B'_j коммутируют, то матрицы $\widehat{B}_i, \widehat{B}_j$ также коммутируют. Кроме того, очевидно, что матрицы B'_t, \dots, B'_{m-1} линейно независимы тогда и только тогда, когда матрицы $\widehat{B}'_t, \dots, \widehat{B}'_{m-1}$ линейно независимы.

Вопросы выбора матрицы \widehat{K} из $\widehat{B}'_t, \dots, \widehat{B}'_{m-1}$ и выяснения количества линейно независимых матриц требуемого вида, коммутирующих с \widehat{K} , сводятся к аналогичным вопросам в рассмотренном ранее случае жордановых блоков одинакового размера при $\widehat{n} = 2s$. Таких матриц (см. 18) не более $3/4\widehat{n} = 3/2s$. С учётом $I, J, J^2, \dots, J^{t-1}$ получим: $\dim(\widehat{\Omega}) \leq 3/2s + t = 3/4n - 1/4(t-s) < 5/4n$. Таким образом, оценка $\dim(\Omega) \leq 5/4n$ верна вне зависимости от соотношения между размерами жордановых блоков. Можно показать [2], что для некоторых алгебр эта оценка точна.

3.3. k одинаковых жордановых блоков. Пусть алгебра Ω содержит матрицу A , подобную жордановой форме $G = \text{diag}(\underbrace{J, \dots, J}_k) = DAD^{-1}$, где все жордановы клетки J размера $t \times t$ ($n = kt$). Тогда можно перейти к подобной алгебре $\Omega' = \{C'C = DCD^{-1}, C \in \Omega\}$, содержащей матрицу G (опять же $\dim(\Omega) = \dim(\Omega')$).

Построим базис Ω' . Поскольку матрицы I, G, \dots, G^{t-1} линейно независимы, существует базис Ω' : A_0, A_1, \dots, A_{m-1} , в котором $A_i = A^i$, $i = 0, \dots, t-1$.

В силу коммутации с G все матрицы A_l ($l = 0, \dots, m-1$) будут состоять из k^2 вернетреугольных тёплицевых блоков размера $t \times t$:

$$A_l = \begin{bmatrix} A_{11}^l & A_{12}^l & \dots & A_{1k}^l \\ A_{21}^l & A_{22}^l & \dots & A_{2k}^l \\ \dots & \dots & \dots & \dots \\ A_{k1}^l & A_{k2}^l & \dots & A_{kk}^l \end{bmatrix} \quad (29)$$

Теперь преобразуем матрицы базиса специальным образом (смысл этого преобразования разъясняется позже). Если у тёплицевых блоков матрицы $A_i = A_{11}^i, A_{22}^i, \dots, A_{kk}^i$ — совпадают первые h_i элементов в первых строках, то в силу верхнетреугольной тёплицевой структуры блоков матриц A_i первые h_i диагоналей каждого из этих блоков можно обнулить, отнимая от A_i линейные комбинации I, J, \dots, J^{t-1} (из нильпотентности матриц A_i следует, что $h_i \geq 1$). Получаем новый базис $B_0, \dots, B_{t-1}, B_t, \dots, B_{m-1}$, $B_0 = A_0 = I, B_1 = A_1 = G, \dots, B_{t-1} = A_{t-1} = G^{t-1}$ (блоки матриц нового базиса по-прежнему верхнетреугольные тёплицевы).

Теперь, как и в случае двух одинаковых жордановых блоков, определим число h — номер первой ненулевой диагонали среди блоков матриц B_i ($i = t, \dots, m-1; 1 \leq h \leq t$).

$(B_{ij}^l)|_{1s} = 0$ для $\forall l, i, j, s \leq h$ и $\exists l_0, i_0, j_0$ такие, что $(B_{i_0 j_0}^{l_0})_{1,h+1} \neq 0$. (другими словами выбираем блок наибольшего ранга r_0 среди тёплицевых блоков матриц A_t, \dots, A_{m-1} , тогда $h = t - r_0$) Матрицу B_{l_0} , которая обладает наибольшим числом блоков с ненулевой h -той диагональю, обозначим K . Отметим, что если бы мы не провели указанную махинацию с матрицами базиса, то h могло бы оказаться равным 0 (поскольку к любой матрице базиса можно прибавить I), и h не несло бы никакой полезной для нас информации (с той же целью в предыдущих параграфах мы обнуляли один из диагональных блоков).

Выясним теперь условия коммутации остальных матриц с K .

При $h \geq t/2 = n/2k$ любая матрица указанного вида будет коммутировать с K (поскольку $KB_l = B_lK = 0$). При этом среди них встретятся G^h, \dots, G^{t-1} . Получим

$$\dim(\Omega') = t - h + k^2 \frac{t}{2} \leq \frac{t}{2} + \frac{kn}{2} = \frac{n}{2} \left(k + \frac{1}{k} \right) \quad (30)$$

Далее считаем $h < t/2$.

Расписав блочные произведения $KB_l = B_lK$ и воспользовавшись коммутативностью произведения верхнетреугольных тёплицевых матриц, получим k^2 матричных уравнений вида

$$(K_{1i}B_{j1}^l + \dots + K_{ki}B_{jk}^l) - (K_{j1}B_{1i}^l + \dots + K_{jk}B_{ki}^l) = 0 \quad (31)$$

для $i, j = 1, \dots, k$.

Поскольку блоки B_{ij}^l верхнетреугольные тёплицевы, эти уравнения равносильны уравнениям для их последних столбцов. Обозначим последний столбец блока B_{ij}^l как $b_{ij}^l \in \mathbb{C}^t$ (по нему блок B_{ij}^l однозначно определяется). Тогда каждое матричное уравнение (31) эквивалентно уравнению для вектор-столбцов b_{ij}^l :

$$(K_{1i} b_{j1}^l + \dots + K_{ki} b_{jk}^l) - (K_{j1} b_{1i}^l + \dots + K_{jk} b_{ki}^l) = 0 \quad (32)$$

их также будет k^2 .

Из того, как мы задали h , следует, что не менее h последних элементов каждого из векторов b_{ij}^l нулевые. Всего получаем $k^2 h$ нулей.

Поскольку в (32) при первых h элементах b_{ij}^l стоят только нулевые сомножители (см. способ выбора K), задаются они произвольно. Итого hk^2 степеней свободы. Обрежем вектора b_{ij}^l , исключив эти первые h элементов. Получим вектора $b_{ij}^{l'} \in \mathbb{C}^{t-h}$. При этом последние $h-1$ координат векторов $b_{ij}^{l'}$ равны 0 (это следует из способа определения h).

Обозначим правые верхние потенциально-ненулевые подблоки блоков K_{ij} как $K'_{ij} \in \mathbb{C}^{(t-h) \times (t-h)}$. Они верхнетреугольные тёплицевы, один из них гарантированно невырожденный. Обозначим матрицу из преобразованных блоков в прежнем порядке K' .

Для обрезанных матриц K'_{ij} и обрезанных векторов $b_{ij}^{l'}$ соотношения (32) переписываются аналогичным образом:

$$(K'_{1i} b_{j1}^{l'} + \dots + K'_{ki} b_{jk}^{l'}) - (K'_{j1} b_{1i}^{l'} + \dots + K'_{jk} b_{ki}^{l'}) = 0 \quad (33)$$

Невырожденный блок K_{0j0} встречается среди уравнений (33) $2k$ раз, при чём можно выделить $2k-1$ уравнений, в каждом из которых он встречается в произведении с векторами, которые не фигурируют в оставшихся $2k-2$ уравнениях. Это означает, что $2k-1$ векторов $b_{ki}^{l'}$ выражаются через остальные.

Можно объяснить это иначе.

Из векторов $b_{11}^{l'}, \dots, b_{ik}^{l'}, b_{21}^{l'}, \dots, b_{kk}^{l'}$ склеим вектор $b^l \in \mathbb{C}^{k^2(t-h)}$. Для него набор уравнений (33) переписывается в виде

системы: $\mathbf{X}\mathbf{b}^l = 0$, где \mathbf{X} блочная матрица специального вида:

$$\mathbf{X} = \begin{bmatrix} -K'^{BT} + \text{diag}[K'_{11}] & \text{diag}[K'_{12}] & \dots & \text{diag}[K'_{1k}] \\ \text{diag}[K'_{21}] & -K'^{BT} + \text{diag}[K'_{22}] & \dots & \text{diag}[K'_{2k}] \\ \dots & \dots & \dots & \dots \\ \text{diag}[K'_{k1}] & \text{diag}[K'_{k2}] & \dots & -K'^{BT} + \text{diag}[K'_{kk}] \end{bmatrix}$$

K'^{BT} — блочное транспонирование;

$$\text{diag}[K'_{ij}] = \underbrace{\text{diag}(K'_{ij}, \dots, K'_{ij})}_{k} \in \mathbb{C}^{(k(t-h)) \times (k(t-h))} = \mathbb{C}^{(n-kh) \times (n-kh)}$$

— блочно-диагональная матрица.

Из способа выбора матрицы K следует, что $\text{rank}(\mathbf{X}) \geq (2k-2)(t-h) \leq 2n - 2kh + 2t - 2h$.

Тогда $(2k-1)(t-2h+1)$ элементов определяются по известным остальным $k^2(t-2h) - (2k-2)(t-2h) = (k-1)^2(t-2h+1)$ его элементам. С учётом kh^2 произвольно задаваемых элементов и I, J, \dots, J^{t-h} получаем

$$\dim(\Omega') = kh^2 + t - h + (k^2 - 2k - 2)(t - 2h) \leq n(k - 1/k) \quad (34)$$

Таким образом, из (30), (34) следует, что для алгебры, содержащей матрицу с k одинаковыми жордановыми блоками, верна оценка

$$\dim(\Omega) < \max \left[\frac{n}{2} \left(k + \frac{1}{k} \right); n \left(k - \frac{1}{k} \right) \right] \quad (35)$$

Поскольку полученная оценка является возрастающей функцией от k , то для получения наилучшей оценки нужно выбрать минимальный локальный дефект k_{\min} :

$$\dim(\Omega) < \max \left[\frac{n}{2} \left(k_{\min} + \frac{1}{k_{\min}} \right); n \left(k_{\min} - \frac{1}{k_{\min}} \right) \right] \quad (36)$$

(таким образом, мы выбираем наилучшую из локальных оценок).

Однако даже теперь полученная оценка (36) может быть завышена: во-первых, мы не учитываем, что матрицы из Ω не могут иметь меньший дефект (поскольку мы выбираем G с дефектом k_{\min} , минимальным в Ω), и, во-вторых, среди матриц, коммутирующих с G и K , мы учитываем также матрицы, не являющиеся нильпотентными (которых заведомо не может быть в Ω). Особенno значительна ошибка при больших k . Максимум оценки (36) достигается при

$k = n$ и равен $n^2 - 1$, что значительно привосходит точную верхнюю оценку $n^2/4$.

Мы не доказали, что для k блоков разных размеров оценка (36) по-прежнему будет верна, но, по аналогии со случаем $k = 2$, это кажется верным. Оставляем это утверждение в качестве гипотезы.

4. Связь оценок с l и k

Очевидно, полученные оценки и от l , и от k завышены. Например, для нильпотентной алгебры, порождённой матрицей $G = \text{diag}(J, J)$ (где J — жорданова клетка размера $n/2$), — $\Omega = \text{span}(G, G^2, \dots, G^{n/2-1})$ — общий и минимальный дефекты равны 2, и мы получаем оценки $2n - 4$ (от l) и $5/4n$ (от k_{\min}), в то время как на самом деле $\dim(\Omega) = n/2 - 1$. В данном случае оценка от l хуже, чем от k_{\min} . Однако, возможны и обратные ситуации, когда оценка от l лучше.

Поскольку $l \leq k_{\min} \leq k$ и оценка $l(n - l)$ монотонно возрастает при $l \leq n/2$, в неё можно подставлять локальные дефекты $k \leq n/2$. Если же $k > n/2$, то это уже невозможно: может оказаться, что $l = n/2$, и тогда $l(n - l) > k(n - k)$. Но в этом случае ($k > n/2$) почти всегда (а именно при $n > 2$) оценка от k будет превосходить точную верхнюю оценку $n^2/4$:

$$k > n/2 \Rightarrow n \left(k - \frac{1}{k} \right) > n \left(\frac{n}{2} - \frac{2}{n} \right) = \frac{n^2}{2} - 2 > \frac{n^2}{4} \quad (37)$$

Оценки по $k(n - k)$ и $n(k - 1/k)$ совпадут при $k = \sqrt[3]{n}$, поэтому локальную оценку можно модифицировать следующим образом:

$$\dim(\Omega) \leq f(k), \quad f(k) = \begin{cases} n, & k = 1 \\ 5/4n, & k = 2 \\ n(k - 1/k), & 2 < k \leq \sqrt[3]{n} \\ k(n - k), & \sqrt[3]{n} < k < n/2 \\ n^2/4, & n/2 \leq k \end{cases} \quad (38)$$

Эта оценка улучшается при подстановке k_{\min} . Для каждого из интервалов изменения k существуют алгебры, для которых полученная оценка будет точной.

Список литературы

- [1] Супруненко Д. А., Тышкевич Р. И. Перестановочные матрицы
— Москва: Едиториал УРСС, 2003.
- [2] K. C. O'Meara, C. Vinsonhaler. On approximately simultaneously
diagonalizable matrices // *Linear algebra and its applications*.
2006. №1. P. 39-74.

Топ50. Рейтинг наиболее производительных вычислительных систем СНГ

Д. А. Никитенко*

В последние годы сфера высокопроизводительных вычислений переживает настоящий бум. Огромными темпами растет производительность систем, расширяется круг областей применения и растет востребованность таких технологий в целом. В таких условиях даже специалистам сложно быть в курсе всех последних изменений и тенденций. Проект Топ50 [1] ставит своей целью, прежде всего, предоставить исчерпывающую информацию о наиболее мощных вычислительных системах СНГ, предоставляя возможность как проанализировать текущие тенденции, так и почерпнуть успешный опыт создания таких систем. Такая информация, безусловно, крайне полезна широкому кругу профессионалов и начинающих: разработчикам, поставщикам, исследователям, пользователям суперкомпьютеров.

1. О проекте

Начиная с момента появления первых компьютеров, исследователи постоянно сталкивались с задачами, для решения которых существующих вычислительных мощностей было недостаточно. Такие задачи возникали и возникают в самых различных областях: аэродинамика и нефтедобыча, прогноз погоды и микроэлектроника, фармацевтика и проектирование новых материалов, криптография и биоинженерия — это лишь небольшой список областей, где для успешного продвижения вперед требуются компьютеры с действительно запредельной производительностью.

*Научно-исследовательский вычислительный центр МГУ

Чтобы помочь правильно сориентироваться в мире высокопроизводительных вычислительных систем и иметь возможность оперативно отслеживать тенденции развития данной области, Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова и Межведомственный суперкомпьютерный центр РАН в мае 2004 года начали совместный проект по формированию списка 50 наиболее мощных компьютеров СНГ — Топ50 [1].

В список включаются 50 вычислительных систем, установленных на территории СНГ и показывающих к моменту выхода списка наибольшую производительность на тесте Linpack.

Рейтинг обновляется два раза в год: в начале весны и осени. Осенняя редакция рейтинга объявляется на традиционной серии Всероссийских научных конференций «Научный сервис в сети Интернет». По запросу заявителям всем попавших в список систем выдаются официальные сертификаты установленного образца.

За время своего существования, с конца 2004 года, проект заслужил репутацию качественного и объективного аналитического инструмента в области построения вычислительных систем на территории стран СНГ.

2. Предоставляемый сервис

Помимо данных о пиковой и достигнутой производительности, при подаче заявки на участие в рейтинге может указываться, в достаточной мере, исчерпывающий перечень характеристик и параметров системы: от места установки и разработчиков, и до подробного описания конфигурации отдельных узлов кластеров, типах коммуникационных сред и т.п.

Наличие такой обширной информации по системам дает возможность не только видеть конкретные решения в деталях, но и оценить, какой подход в построении вычислительных комплексов показывает себя эффективным в конкретных областях применения.

На сайте проекта можно получить как любую вышедшую редакцию рейтинга в стандартной форме, так и сформировать таблицу с отражением конкретных интересующих параметров. В сочетании с разделом «статистика», где находятся статистические данные по рейтингам, как в виде таблиц, так и графиков, пользователь получает возможность отследить все изменения и тенденции за историю

ведения рейтинга, а значит, и получает основу для того, чтобы попытаться предугадать, что будет происходить в этой сфере на следующем этапе.

3. Сопоставление с Top500

При разработке проекта Топ50, безусловно, большое внимание уделялось широко известному и, можно сказать, ставшему уже эталонным в мировом масштабе проекту Top500 [2]. Почему же тогда нужен такой рейтинг, как Топ50, если есть общепринятый Top500?

Прежде всего, это связано с масштабами. Top500 — рейтинг, в котором участвуют системы со всего мира. При нынешних темпах развития в этой области, когда каждый год появляется огромное количество новых систем, причем, все более и более мощных, очевидно, что состояние отрасли в отдельной стране не может быть в достаточной мере отражено таким списком. Просто потому, что рейтинг Top500 преследует другие цели и отражает ситуацию на мировом уровне. Но есть возможность оценить лидеров, как всемирных, так и отдельно взятых регионов. Так, например, в редакцию Top500 от ноября 2007 года попало всего лишь семь отечественных систем.

Какой производительностью должна обладать система, чтобы попасть в списки Топ50 и Top500? Нижняя граница в 7-ой редакции рейтинга Топ50 от сентября 2007г. — 253.6 GFlops, в 30-ой редакции Top500 от ноября 2007г — 5930 GFlops. В предыдущих редакциях списков — 196.1 и 4005 GFlops, соответственно.

Рейтинг Топ50 ставит своей целью отразить состояние именно отечественной составляющей области высокопроизводительных вычислений. Помочь разобраться в тенденциях и складывающейся ситуации, именно в рамках СНГ, предоставляя больше информации для размышления, и, как следствие, для дальнейшего развития работы исследователей, разработчиков, поставщиков и пользователей суперкомпьютерных технологий на территории РФ и стран СНГ — в этом основная задача проекта.

Область определения рейтингов обуславливает и темпы обновления самого содержимого списка лидирующих систем. Так, за последние четыре редакции в списке Top500 появлялось, в среднем, более 200 (а это — более 40%) новых систем, причем, в редакции от июня 2007 года — 285 (57%). В списке же Топ50 устоявшимся тем-

пом можно считать уровень в десять новых систем (или апгрейдов), то есть 20%. Однако, и об этом будет сказано далее более подробно, несмотря на такую количественную разницу, качественные тенденции в отечественной области выглядят вполне достойно.

Чем больше параметров систем в рейтинге может быть доступно пользователю, тем глубже и всестороннее могут быть сделаны выводы, тем вернее могут быть сделаны последующие решения. Именно поэтому разработчики рейтинга стараются представить всю доступную информацию о системах в максимальном объеме. И детальность информации о системах, пожалуй, выгодно отличает Топ50 от Топ500. Развитие статистических сервисов является приоритетным направлением развития проекта.

Будучи стимулируемой высокой востребованностью вычислительных мощностей, область высокопроизводительных вычислений развивается чрезвычайно интенсивными темпами. Соответственно, с появлением новых технологий, со временем меняются и подходы к исследованию, меняются и сами исследуемые параметры. Например, если еще год-два назад количество процессоров в суперкомпьютерной системе могло достаточно полно характеризовать масштаб системы, то сейчас, с появлением, активным развитием и широким внедрением многоядерности, уже число ядер в процессорах, на узлах и во всей системе в целом, рассматривается как один из основных исходных параметров системы. Все больше предпочтений отдается системам на базе многоядерных процессоров. В частности, это дает возможность сокращения числа узлов при той же пиковой производительности, а значит, уменьшения доли относительно медленных соединений между узлами. Впрочем, суперкомпьютерная система строится, исходя из определенного круга задач, которые она предназначена решать, а потому приоритеты и требования могут отличаться весьма значительно. В осенней редакции Топ50 доля систем на базе многоядерных процессоров составила 40%. Более того, все установленные в 2007 году системы в рейтинге построены на многоядерных процессорах.

4. Это интересно

Рассмотрим далее некоторые интересные факты и тенденции, которые прослеживаются при анализе последних редакций списков

Топ50 в сравнении с Топ500.

Темпы увеличения суммарной производительности систем достаточно непостоянны. Это связано с не столь частым вводом в эксплуатацию систем, занимающих лидирующие места в рейтингах. К примеру, в списке Топ500 наибольшим рост пиковой и достигнутой производительности был в редакции от ноября 2007г., и составил примерно 46% и 40% соответственно, в июньском списке — 40% и 37.8%, а в предыдущих редакциях рост был почти вдвое меньшим.

В списке Топ50 ситуация несколько иная. Принимая во внимание относительно небольшое количество как новых систем, так и их общее количество, скорее уместно говорить о среднем росте. Средний же рост пиковой и достигнутой производительности составляет 43% и 42%, соответственно. Здесь видится очень обнадеживающим, что в среднем рост эффективности значительно выше в Топ50, чем в Топ500. Это происходит, прежде всего, по той причине, что большинство новых, эффективных систем располагается в первой половине списка, т.е. они и весьма высокопроизводительны (рис. 1). А это, за счет незначительного общего количества систем, достаточно ощутимо влияет на средние показатели. Своебразным переломным моментом была весна 2007 г. Суммарная производительность списка увеличилась почти вдвое. Более того, и в 7-ой редакции рост составил около трети (рис. 2).

С ростом производительности систем вопрос их эффективного использования видится особенно важным. Будем понимать под эффективностью отношение достигнутой производительности на тесте Linpack к пиковой производительности системы. Интересно, что это соотношение достаточно постоянно не только в списке Топ500, отражающем положение вещей во всей отрасли целиком, но и в списке Топ50, несмотря на то, что количество новых систем в его новых редакциях несопоставимо меньше. Так, в последние четыре редакции (два года) в среднем по спискам эффективность в Топ500 и Топ50 колеблется от 66 до 69 процентов. Следует также заметить что новые, мощные системы, в большей своей массе, обладают эффективностью не ниже средней. Это говорит о все более рациональном подходе к их построению, о качественном росте.

Таким образом, есть основания говорить о том, что отечественная сфера высокопроизводительных вычислений развивается очень

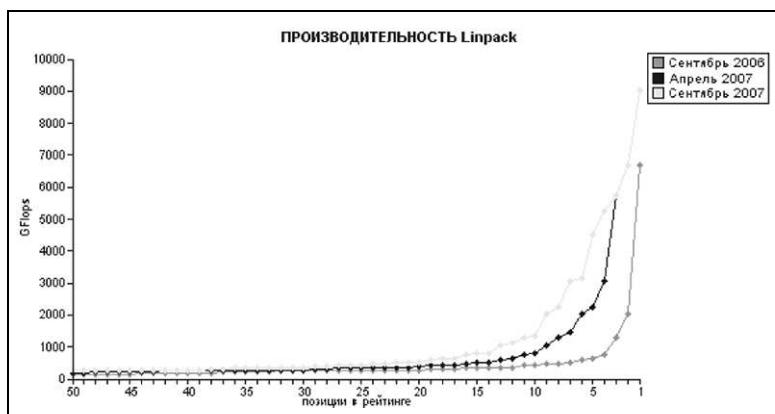


Рис. 1. Производительность на тесте Linpack

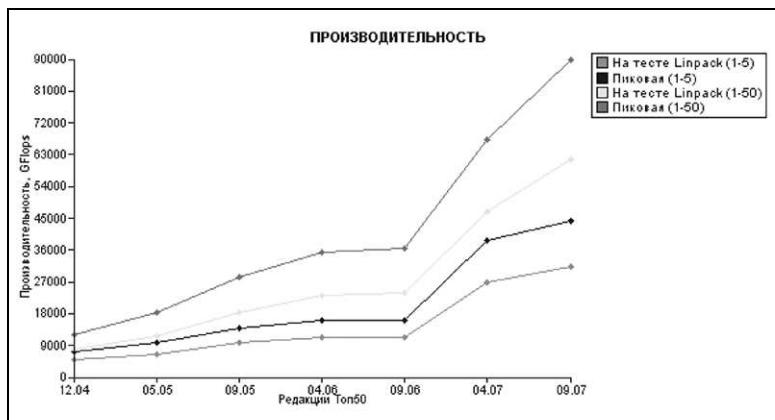


Рис. 2. Суммарная производительность систем

высокими темпами, не уступающими по качеству, даже превосходящими средние показатели по Top500.

Что касается области применения вычислительных систем, то интересны следующие тенденции. Лишь в двух редакциях Топ50 2006 года число систем, задействованных в области финансов, превысило число систем в сфере науки и образования, которое растет

на протяжении последних трех редакций (22%, 30%, 38%). Причем, в пересчете на производительность, системы сферы науки и образования абсолютно доминируют, концентрируя в себе 44%, 58% и 59% пиковой производительности от всего списка и 46%, 61% и 61%, соответственно, от достигнутой на тесте Linpack. Наравне с системами, применяемыми в области прикладных исследований, растущими численно (20%, 24% и 28%), но концентрирующими на себе достаточно постоянную часть вычислительной мощности всего списка (15% пиковой и 16-17% достигнутой), установки сферы науки и образования являются наиболее эффективными (рис. 3).

Доля же систем, применяемых в финансовой области, уменьшается, как численно, так и по совокупной мощности, достигнув нижнего порога в осенней редакции Топ50 2007 года уровня в 18% систем, но лишь 12% пиковой и 9% достигнутой производительности.

Аналогичная ситуация и с промышленностью: 16% всех систем, и всего 14% пиковой и 13% достигнутой производительности от суммарной по списку. Наглядно это продемонстрировано на диаграммах (рис. 3).

Далее, попробуем оценить изменения в характеристиках попадающих в рейтинг систем.

Прежде всего, бросается в глаза отсутствие SMP систем. Более года все системы в Топ50 являются кластерными установками. Видимо, главной причиной является относительно высокая стоимость больших систем с общей памятью.

Все больше растут масштабы самих систем. Так, среднее число процессоров и ядер в системе в осеннем списке Топ50 — 171 и 222, соответственно, при минимуме в 34 вычислительных ядра (43-е место) и максимуме — в 1312 у системы СберБанка (7-ое место).

Примечательным является соотношение оснащенности памятью к числу вычислительных ядер и процессоров. До появления многоядерных процессоров это соотношение росло в пользу увеличения объема памяти на процессор и в среднем изменялось от 1.2 до 1.8 гигабайт на процессор. С появлением же многоядерных процессоров, это отношение стало уменьшаться, составив в последней редакции 1.4 гигабайт на ядро и оставаясь порядка 1.8 гигабайт на процессор.

Ярко выраженным лидером производителей процессоров с самой первой редакции рейтинга является компания Intel. Интересно так-

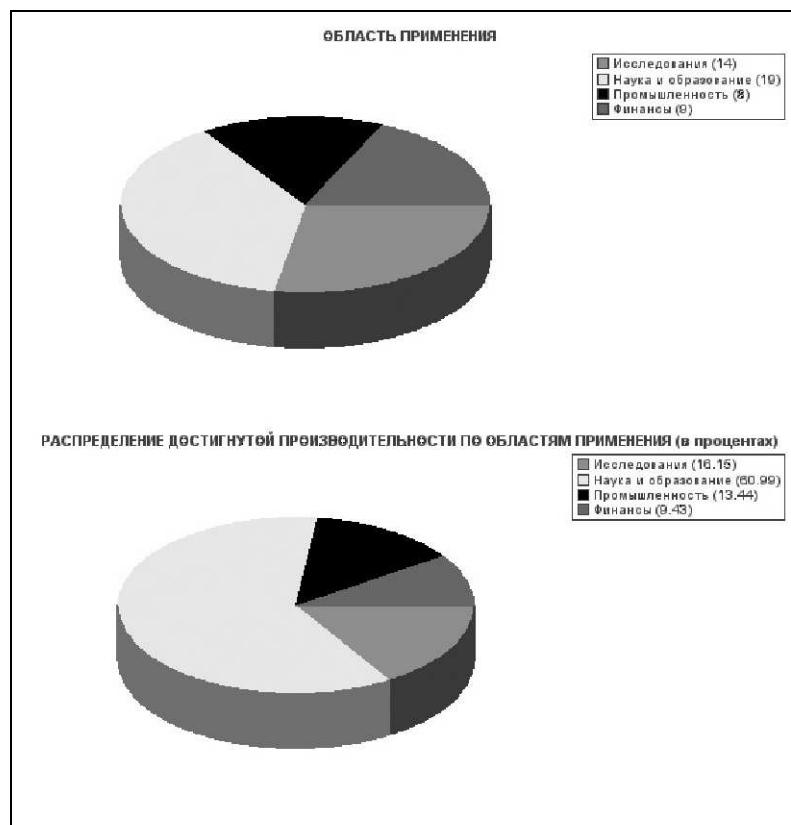


Рис. 3. Область применения систем и распределение производительности

же, что доля Intel стабильно растет и на данный момент составляет 62%. Среди них, из 31 системы 27 построены на базе Intel Xeon и лишь четыре — на процессорах семейства Itanium.

Что касается коммуникационных сред, то прослеживается явная тенденция в использовании высокоскоростных и специальных решений. Доля Gigabit Ethernet, самого бюджетного решения, падает и на данный момент уже составляет 40%, продолжая опускаться после пиковых 60% в весенней редакции 2006г. Если доля таких сред как Myrinet и SCI остается примерно одинаковой (в сумме около 10%),

то доля Infiniband продолжает расти и на данный момент составляет 40%.

Говоря о разработчиках систем, видятся интересными следующие тенденции. Во-первых, сокращается число систем собственной сборки (на данный момент — всего две системы), таким образом, все большее предпочтение отдается разработке систем специализирующимися организациями. Во-вторых, среди этих организаций все большие обороты набирают отечественные коллективы. Так, компания Т-Платформы в осеннем списке Топ50 является разработчиком 28% процентов систем, при этом являясь разработчиком и системы СКИФ Cyberia, занимающей первое место в рейтинге с пиковой производительностью 12 TFlops и достигнутой - 9 TFlops. Лишь на втором месте идет Hewlett-Packard с 26%, на третьем — IBM с 20%. В список входит уже четыре системы, созданные украинской компанией ПНВП «ЮСТАР».

5. Заключение

Что же нас ждет в ближайшем будущем? Прежде всего, следует сказать, что уже к следующему, весеннему, выходу списка Топ50 планируется запуск целого ряда мощных вычислительных систем. Вообще, начало 2008 года обещает быть насыщенным на появление новых и мощных суперкомпьютеров. В частности, в НИВЦ МГУ планируется введение в эксплуатацию новой системы пиковой производительностью 60 TFlops.

Ожидается удвоение суммарной производительности систем по списку. Таким образом, высокие темпы роста числа мощных суперкомпьютеров в Топ50 не только поддерживаются, но и явно превышают средние значения, которые уже сейчас превышают среднемировые показатели.

В заключение, хочется обратить внимание на то, что одной из главных целей создания рейтинга ставилось создание инструмента, помогающего разбираться векторах развития области высокопроизводительных вычислений. А удобство и эффективность инструмента подразумевает под собой возможность адаптации к потребностям конкретного пользователя. Группа разработчиков рейтинга старается создать максимально удобный инструмент для работы, продолжая его развитие, добавляя новые функциональные возмож-

ности, исходя из потребностей пользователей, научных групп и вычислительного сообщества в целом.

Список литературы

- [1] Топ50: список самых мощных компьютеров СНГ,
<http://supercomputers.ru>.
- [2] Top500 Supercomputing Sites, <http://www.top500.org/>.
- [3] Воеводин Вл.В. Top500: числом или уменьем// Открытые системы, №10, 2005 г. С.12–15.
(http://www.osp.ru/os/2005/10/380430/_p1.html)

Технология расчета течений со свободной границей с использованием динамических гексаэдральных сеток.

К. Д. Никитин[®]

В статье рассмотрен алгоритм моделирования течения несжимаемой жидкости в области с подвижной границей. Используются динамически изменяющиеся сгущающиеся гексаэдральные сетки со структурой восьмидерева. Свободная поверхность реализуется путем использования функции уровня и частиц.

1. Введение

Течение несжимаемой жидкости описывается уравнениями Навье-Стокса. Одним из возможных подходов к приближенному решению системы уравнений Навье-Стокса является проекционный метод [1, 2, 3]. Однако, этот метод, как и многие другие, предполагает наличие полностью фиксированной границы расчетной области. При решении задачи со свободной границей предлагается динамически изменять расчетную область. Для задания свободной границы применяются несколько подходов. Во-первых, можно описывать положение свободной поверхности жидкости параметрической функцией. Во-вторых, параметрическую функцию можно заменить полем расстояний, называемым функцией уровня [4]. Недостатком такого подхода является то, что функция уровня не в состоянии описать всю полноту движений, совершаемых поверхностью жидкости. В-третьих, положение границы области можно задавать с помощью частиц. В-четвертых, существует гибридный подход, объединяющий использование частиц и функции уровня. В этом случае частицы компенсируют недостатки функции уровня, а функция уровня помогает равномерно распределять частицы вдоль границы.

[®]Институт вычислительной математики РАН

В данной работе описывается метод решения уравнений Навье-Стокса в области со свободной границей [5], объединяющий в себе несколько эффективных подходов. В основе алгоритма лежит проекционный метод. Расчетная область динамически изменяется в соответствии с изменением функции уровня, описывающей свободную границу. Функция уровня дополняется частицами, способными отслеживать мелкие элементы поверхности [6]. При решении задачи используются сгущающиеся сетки, построенные по принципу восьмидерева [7]. Сгущение сеток происходит к свободной границе и поддерживается благодаря динамическому перестроению. Технология, объединяющая описанные выше подходы, позволяет производить качественное моделирование при невысоких вычислительных затратах.

Во втором разделе данной статьи представлен класс сеток, на которых предлагается проводить расчеты, и объясняются причины выбора такого класса. В третьем разделе рассматриваются базовые уравнения и численные методы, используемые для их приближенного решения. В разделах 4-7 делается акцент на отдельных частях алгоритма, описываются возможные трудности, возникающие в процессе моделирования, и способы их преодоления. Наконец, в последнем разделе приводятся результаты численных экспериментов.

2. Класс расчетных сеток

Для дискретизации уравнений Навье-Стокса предлагается использовать разнесенные гексаэдральные сетки построенные по принципу восьмидерева.

Искомыми функциями в задаче являются скорость, давление и скалярная функция уровня, определяющая положение свободной границы. Идея разнесенных сеток заключается в том, что рассматриваемые величины хранятся в разных частях сетки: компоненты вектора скорости — в центрах тех граней, которые ортогональны соответствующему направлению, давление — в центре ячейки, функция уровня — в вершинах сетки. Кроме того, каждой ячейке соответствует специальная метка, определяющая заполнение ячейки жидкостью. Разнесенные сетки с такими метками часто называют MAC (Marker-And-Cell) сетками.

Восьмидерево предполагает иерархическую структуру сетки, в

h_{\min}	32^{-1}	64^{-1}	128^{-1}	256^{-1}	512^{-1}
Равномерная сетка	15 208	84 720	505 768	—	—
Сгущающаяся сетка	5 180	14 268	55 924	205 740	742 836

Таблица 1. Соотношение чисел ячеек в равномерной и сгущающейся сетках для задачи с падающей каплей (см. раздел 8).

основании которой лежит куб, а более мелкие ячейки получаются делением крупных на восемь частей. Поиск элемента по номеру или по координате выполняется не более чем за $\log_8 N$ шагов, поиск соседнего — не более чем за $2 \log_8 N$ [7].

Выбор разнесенных гексаэдральных сеток построенных по принципу восьмидерева обусловлен несколькими факторами.

Во-первых, в задаче фигурируют векторное поле скоростей и скалярное поле давлений. При использовании разнесенных гексаэдральных сеток на каждую ячейку приходится одна степень свободы для давления и примерно одна степень свободы для каждой компоненты скорости. Такое распределение степеней свободы представляется наиболее экономичным.

Во-вторых, использование сгущающихся сеток более эффективно по сравнению с равномерными. В равномерной сетке зависимость числа элементов от минимального шага сетки h_{\min} — обратная кубическая, в сгущающейся к поверхности — обратная квадратичная [7]. Допустим, что сетка содержит 1 000 000 ячеек. В случае, если используются равномерные сетки, шаг $h_{\min} \approx \sqrt[3]{\frac{1}{1000000}} = 0.01$, в случае же сгущающихся сеток $h_{\min} \approx \sqrt{\frac{1}{1000000}} = 0.001$. Уменьшение шага сетки вблизи свободной границы важно, поскольку это влияет на уменьшение численной вязкости.

Покажем на примере следующей таблицы, как соотносятся числа ячеек в равномерной и сгущающейся сетке при равном минимальном шаге:

Из таблицы видно, что для сетки с миллионом ячеек, в равномерном случае $h_{\min} = 128^{-1}$, а в случае сгущающейся к поверхности сетки $h_{\min} = 512^{-1}$.

При моделировании жидкости со свободной границей наибольший интерес вызывает положение свободной поверхности, к которой и следует сгущать расчетную сетку. Однако, положение свободной границы изменяется на каждом шаге, а значит сетку необходимо уметь быстро перестраивать.

Среди динамически перестраиваемых сеток восьмидеревья имеют как ряд преимуществ, так и ряд недостатков. К преимуществам стоит отнести быстрый поиск элемента по номеру, по координате, поиск соседнего элемента в любом направлении, а также крайне легкое перестроение: за несколько операций можно сгустить или разгрубить любую часть сетки. С другой стороны, сетки, построенные на основе восьмидерева не являются конформными, что нужно учитывать при дискретизации дифференциальных операторов. Наложение условия, что стороны соседних ячеек различаются не более, чем в 2 раза, облегчает построение дискретизации.

3. Базовые уравнения и численный метод

Рассматривается система уравнений Навье-Стокса, описывающая движение несжимаемой жидкости. Она состоит из уравнений момента и несжимаемости:

$$\frac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u} \cdot \nabla) \mathbf{u} + \nabla p = \mathbf{f}, \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0, \quad (2)$$

где ν — кинематическая вязкость, t — время, $\mathbf{u} = (u, v, w)$ — поле скоростей, p — давление, а \mathbf{f} — объемные силы, действующие на жидкость (например, сила тяжести).

Стоит отметить, что граница области состоит из двух частей: неподвижной границы (стенок сосуда) и перемещающейся по инерции свободной поверхности. На неподвижной границе реализуется условие прилипания, а для свободной поверхности может быть задано соотношение между давлением жидкости и силами поверхностного натяжения.

Для описания положения свободной границы вводятся функция уровня и частицы. Поверхность определяется множеством точек, где функция уровня $\phi = 0$, а для области, заполненной жидкостью верно $\phi < 0$. Продвижение свободной поверхности описывается урав-

нением переноса функции уровня:

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0. \quad (3)$$

В ряде приложений требуется инициализировать функцию уровня как расстояние до поверхности со знаком и поддерживать это свойство на протяжении расчета. Помимо определения положения поверхности, функция уровня также несет в себе информацию о ее геометрии: единичная нормаль к поверхности определяется по формуле $\mathbf{N} = \nabla \phi / |\nabla \phi|$, а локальная кривизна есть $k = \nabla \cdot \mathbf{N}$.

Для того, чтобы отслеживать мелкие элементы, которые не могут быть описаны функцией уровня, используются специальные невесомые частицы, переносимые полем скоростей. Частицы перемещаются по закону $d\mathbf{x}_p/dt = \mathbf{u}(\mathbf{x}_p)$ и корректируют функцию уровня в случаях, когда это необходимо.

Для приближенного решения уравнений Навье-Стокса (1), (2) с подвижной границей (3) предложен ряд подходов разной степени сложности: от метода дробных шагов [5] до полностью неявных схем [8]. В данной работе будет использоваться метод дробных шагов, как наименее трудоемкий. Метод состоит из следующих шагов:

1. Обновление поля скоростей
 - (a) Решение уравнения моментов (конвективный перенос, действие диффузии и объемных сил),
 - (b) Проекция на подпространство бездивергентных скоростей;
2. Обновление положения свободной поверхности
 - (a) Продвижение функции уровня,
 - (b) Продвижение частиц,
 - (c) Взаимная корректировка частиц и функции уровня;
3. Продвижение области;
4. Перестроение сетки.

Отметим, что первый шаг представляет собой вариант проекционного метода для задачи Навье-Стокса с фиксированной границей [1, 2, 3]. В следующих разделах остановимся подробнее на некоторых шагах алгоритма.

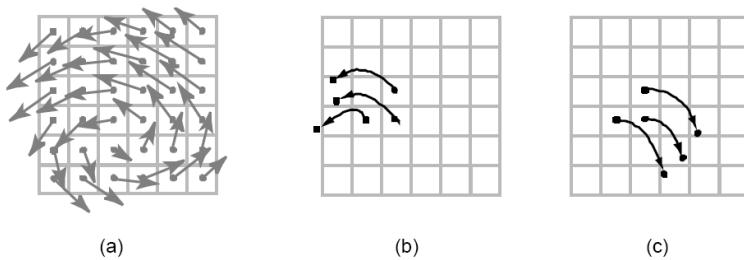


Рис. 1. Поле скоростей (а). Полулагранжев метод: вместо продвижения вперед по траектории движения в поле скоростей (б), отступаем на шаг назад (с).

4. Решение уравнения моментов

Уравнение моментов решается за два шага: сначала учитывается конвективный перенос ($u_t = -(u \cdot \nabla)u$), а затем учитывается действие диффузии и объемных сил ($u_t = v\Delta u - \nabla p + f$).

В случае диффузии и объемных сил применяется стандартная явная схема, а в случае конвективного переноса используется полулагранжев метод.

Суть полулагранжева метода заключается в следующем: для того, чтобы вычислить значение некоторой функции (скорости или функции уровня) в некоторой точке, необходимо отступить по траектории этой точки на шаг назад, тем самым найдя точку, которая переносится в рассматриваемую, и взять значение скорости в ней (рис. 1).

Полулагранжев метод выгодно отличается от явной схемы тем, что вносит меньшую численную диффузию. Этот метод позволяет использовать более крупные шаги по времени без потери устойчивости. Однако ограничение на шаг по времени все же возникает из-за других составляющих алгоритма, использующих явные схемы.

5. Проекция на пространство бездивергентных скоростей

Поле скоростей, полученное при решении уравнения момента, не является бездивергентным, т.е. не удовлетворяет условию несжима-

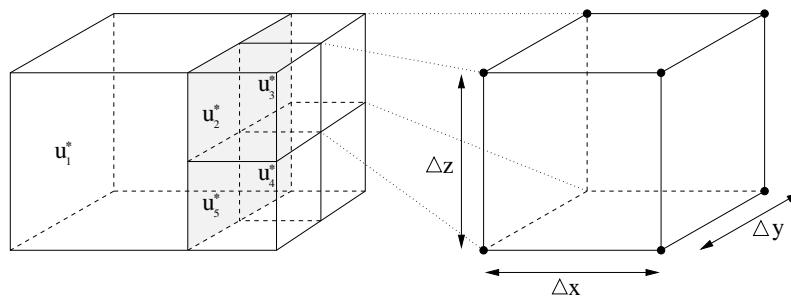


Рис. 2. Грубая ячейка, граничащая с четырьмя мелкими ячейками по одной грани.

емости. Для того, чтобы спроектировать поле скоростей на бездивергентное подпространство, вносится специальная поправка к давлению (δp). Подставляя поправку к давлению в уравнение (2) и учитывая ее вклад в поле скоростей,

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla(\delta p),$$

$$p = p^* + \delta p,$$

приходим к следующей системе уравнений:

$$-\nabla \cdot (\nabla \delta p) = -\frac{1}{\Delta t} (\nabla \cdot \mathbf{u}^*). \quad (4)$$

Краевые условия не накладываются, поскольку задача рассматривается на сеточном уровне и оператор задачи (4) есть произведение сеточных операторов дивергенции и градиента. В случае равномерной сетки можно использовать стандартные разностные аппроксимации операторов дивергенции и градиента, в случае же сгущающихся сеток рекомендуется использовать другие, более подходящие схемы. Для примера рассмотрим оператор дивергенции. Пусть имеет место следующая локальная структура сетки (рис. 2).

Запишем теорему Грина в векторной форме для большой ячейки:

$$V_{cell} \nabla \cdot \mathbf{u}^* = \sum_i (\mathbf{u}_i^* \cdot \mathbf{n}) A_i,$$

где \mathbf{n} — вектор внешней единичной нормали к границе большой ячейки, а A_i — площадь соответствующей грани ячейки.

Шаг сетки	64^{-1}	128^{-1}	256^{-1}
Ячеек	14,268	55,924	205,740
Итераций	6	7	8
Время (сек.)	0.12	0.76	2.98

Таблица 2. Среднее чисто итераций и время работы метода би-сопряженных градиентов для задачи с падающей каплей. Параметры предобусловливателя $\tau_1 = 0.01$, $\tau_2 = 0.001$. Критерий остановки — уменьшение невязки в 10^8 раз.

Для x -компоненты дивергенции имеет место представление:

$$\Delta x \Delta y \Delta z \delta u / \delta x = u_2^* A_2 + u_3^* A_3 + u_4^* A_4 + u_5^* A_5 - u_1^* A_1.$$

Следовательно,

$$\delta u^* / \delta x = ((u_2^* + u_3^* + u_4^* + u_5^*) / 4 - u_1^*) / \Delta x.$$

Выражение для y - и z -компонент получается аналогичным способом.

Уравнение (4) представляет из себя сеточную дискретизацию уравнения Пуассона. Рассмотрим оператор $L = \nabla \cdot (\nabla)$. Число обусловленности матрицы оператора L имеет квадратичную зависимость от минимального шага сетки, который в свою очередь может достигать $h = \frac{1}{1000}$, т.е. число обусловленности может быть очень большим. В зависимости от способа построения сеточного оператора градиента, матрица оператора L может получаться симметричной или несимметричной. Для решения уравнения (4) используются итерационные методы с предобусловливателем, основанным на неполной факторизации второго порядка точности [9]: в симметричном случае — метод сопряженных градиентов, в несимметричном — метод би-сопряженных градиентов [10].

В качестве иллюстрации поведения метода в таблице 2 представлены среднее число итераций и время работы метода би-сопряженных градиентов для задачи с падающей каплей (см. раздел 8).

Если минимальный шаг сетки h_{\min} уменьшится в 2 раза, то число обусловленности матрицы оператора L вырастет в 4 раза. Из

таблицы видно, что число итераций при этом растет незначительно, а время решения системы примерно пропорционально числу элементов. Эти результаты показывают эффективность использования данного метода.

6. Продвижение функции уровня

По аналогии с решением уравнения конвекции, для уравнения (3), которое продвигает функцию уровня, тоже используется полулагранжев метод.

Как было отмечено, метод функции уровня не всегда эффективен при отображении мелких деталей (капли, брызги, пики волн). Один из предлагаемых способов повышения детализации поверхности заключается в использовании частиц.

Частицы наносятся вдоль свободной поверхности и разделяются на два типа: положительные и отрицательные. Положительные частицы изначально располагаются в области, где $\phi > 0$, т.е. в слое воздушных ячеек. Отрицательные попадают в область $\phi < 0$, т.е. располагаются в жидкости. Для переноса частиц в поле скоростей используются трилинейная интерполяция для скоростей и метод Рунге-Кутта 2-го порядка:

$$k_1 = hf(x_n, y_n),$$

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right),$$

$$y_{n+1} = y_n + k_2 + O(h^2).$$

Для продвижения частиц в приграничных ячейках, не принадлежащих расчетной области, поле скоростей экстраполируется с поверхности.

При возникновении на поверхности жидкости мелких деталей, которые не могут быть отслежены функцией уровня, частицы позволяют восстановить действительное положение свободной границы. Для этого каждой частице приписываются сферическая функция уровня ϕ_p и динамически изменяющийся параметр — радиус r_p :

$$\phi_p(x) = s_p(r_p - |x - x_p|),$$

$$r_p = \begin{cases} r_{\max}, & \text{если } s_p \phi(x_p) > r_{\max}, \\ s_p \phi(x_p), & \text{если } r_{\min} \leq s_p \phi(x_p) \leq r_{\max}, \\ r_{\min}, & \text{если } s_p \phi(x_p) < r_{\min}, \end{cases}$$

где s_p — знак частицы (+1, если частица первоначально располагалась в воздухе, и -1, если частица появилась в жидкости).

Коррекция функции уровня ϕ осуществляется для всех положительных частиц, оказавшихся в области $\phi < 0$ дальше чем на свой радиус и для всех отрицательных частиц, попавших в область $\phi > 0$:

$$\phi^+ = \max(\phi_p, \phi^+),$$

$$\phi^- = \min(\phi_p, \phi^-),$$

$$\phi = \begin{cases} \phi^+, & \text{если } |\phi^+| \leq |\phi^-|, \\ \phi^-, & \text{если } |\phi^+| \geq |\phi^-|. \end{cases}$$

Поскольку в данной задаче в качестве функции уровня используется расстояние до поверхности со знаком, это свойство необходимо сохранять на протяжении всего расчета. К сожалению, после операций продвижения и коррекции функция уровня не всегда определяет расстояние, т.е. $|\nabla \phi| \neq 1$. На рис. 3 можно видеть, как отклоняется функция уровня в одном из тестовых примеров после 150 шагов.

Для поддержания свойства $|\nabla \phi| = 1$ после шагов продвижения и коррекции функции уровня проводится шаг реинициализации функции ϕ_0 , на котором решается следующее уравнение:

$$\phi_\tau + \operatorname{sgn}(\phi_0)(|\nabla \phi| - 1) = 0,$$

где sgn — сглаженная функция знака: $\operatorname{sgn}(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + (\Delta x)^2}}$.

Изначально частицы наносятся в приграничные ячейки равномерно по всей свободной поверхности, но с течением времени равномерность их распределения может нарушаться. На рис. 4 показано, что происходит с равномерно распыленным вдоль поверхности множеством частиц через некоторое время работы алгоритма. Одни частицы уходят из поверхностных ячеек в глубину, другие перераспределяются, сосредотачиваясь на некоторых участках поверхности.

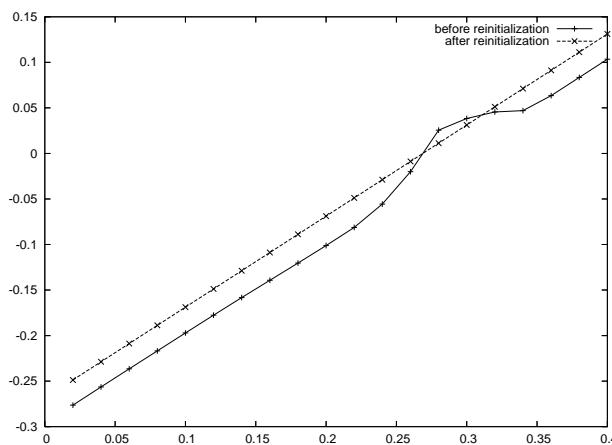


Рис. 3. Функция уровня до и после реинициализации.

Такое множество частиц становится непригодным для точного отображения мелких деталей поверхности. В связи с этим следует периодически восстанавливать равномерное распределение частиц на поверхности.

7. Продвижение области и перестроение сетки

Несмотря на то, что свободная граница определяется поверхностью $\phi = 0$, граница расчетной области проходит по граням ячеек, через которые эта поверхность проходит. Поскольку функция уровня постоянно меняется, то меняется и расчетная область, т.е. на каждом шаге необходимо определять, какие ячейки принадлежат области (заполнены жидкостью), а какие нет. Для этого используются частицы и функция уровня: после продвижения частиц для каждой поверхностной ячейки проверяется значение функции уровня и факт наличия в ней частиц. Если ячейка была заполнена жидкостью, но значение функции уровня во всех ее вершинах стало положительным, и в то же время в ней не осталось отрицательных частиц, такая ячейка помечается как пустая. Если же в воздушную ячейку попали отрицательные частицы, а значение функции уровня хотя бы в одной вершине неположительно, такая ячейка считается

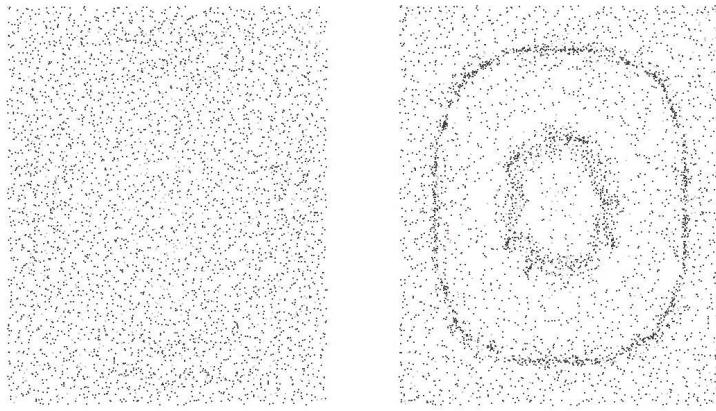


Рис. 4. Положение частиц на первом шаге алгоритма и через 150 шагов без перераспределения.

заполненной жидкостью.

Поскольку расчетная сетка сгущается к поверхности, то на каждом шаге по времени сетки необходимо перестраивать, сгущая в одних местах, разгрубляя в других. При перестроении сетки каждый раз происходит переинтерполяция с грубых ячеек на мелкие и наоборот. Осуществляется это посредством операций разгрубления и сгущения (рис. 5).

Разгрубление переводит мелкие ячейки в крупные. Значения функции уровня, попавшие в узлы большой ячейки, сохраняются, остальные отбрасываются. Для скоростей берется среднее по четырем граням, выходящим на соответствующую грань большой ячейки, а для давлений — среднее по восьми центрам мелких ячеек.

Сгущение делит грубую ячейку на восемь мелких. Значения в новых узлах, образованных на серединах ребер, получаются вычислением средних арифметических для значений на концах ребра, для центров граней — средних арифметических по четырем вершинам грани. Для скоростей и давления сначала вычисляются значения в узлах, для чего используются данные с соседних ячеек, после чего узловые скорости интерполируются в центры граней мелких ячеек,

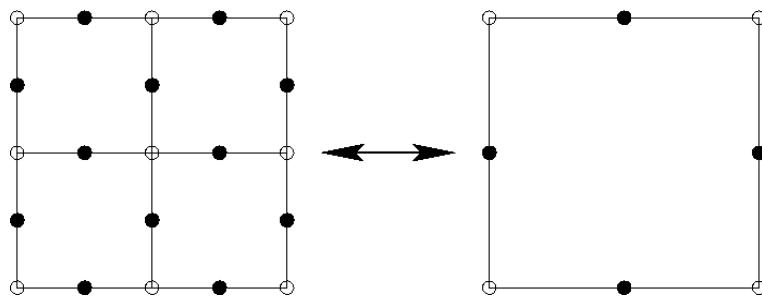


Рис. 5. Операции разгрубления и сгущения.

а давления — в центры самих ячеек.

Отметим, что для предложенных операций переинтерполяции выполняются необходимые законы сохранения: сохранение среднего давления в ячейке и средней скорости на грани.

8. Численный эксперимент

В качестве примера работы алгоритма рассматривается задача моделирования падения капли на поверхность жидкости. В этом случае единственная сила, действующая на жидкость — сила тяжести: в уравнении (1) $\mathbf{f} = (0, 0, -1)$. Коэффициент вязкости $\nu = 0.0001$. Минимальный шаг сетки $h_{min} = 32^{-1}$, шаг времени $\Delta t = 0.005$.

На рис. 6 представлено сечение объема жидкости в моменты времени $t = 5$, $t = 100$, $t = 260$ и $t = 425$. Стрелками показано поле скоростей.

На рис. 7 показан изообъем функции уровня $-0.01 \leq \phi(x) \leq 0$, показывающий положение свободной поверхности жидкости.

Список литературы

- [1] Chorin A. Numerical solution of the Navier-Stokes equations. // *Math. Comp.* 1968, V.22, p.745-762.
- [2] Яненко Н.Н. Метод дробных шагов решения многомерных задач математической физики. — Новосибирск: Наука, 1967.

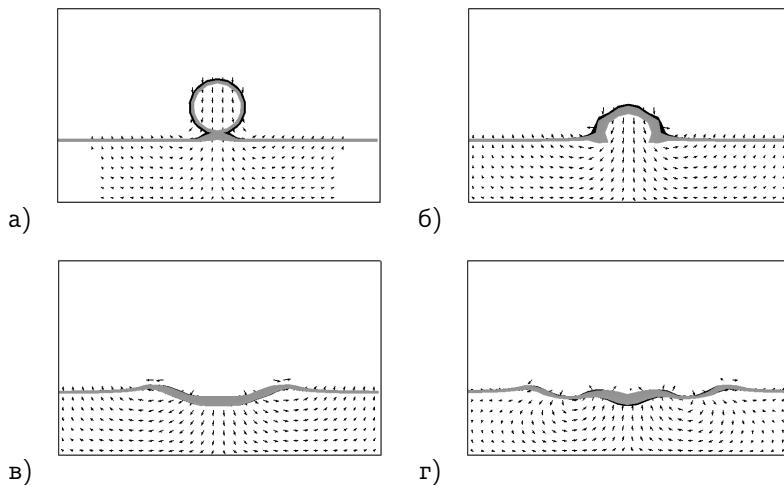


Рис. 6. Сечение изообъёма функции уровня $-0.01 \leq \phi(x) \leq 0$ и поле скоростей в момент времени а) $t = 5$, б) $t = 100$, в) $t = 260$ и г) $t = 425$.

- [3] Temam R. Sur l'approximation des équations de Navier-Stokes. // *C.R.Acad.Sci. Paris, Série A*, 1966, V.262, 219-221.
- [4] Kass M., Miller, G., Rapid, Stable Fluid Dynamics for References Computer Graphics. // *ACM SIGGRAPH 90*, 1990, 49-57.
- [5] Osher S., Fedkiw R. Level Set Methods and Dynamic Implicit Surfaces. — Springer-Verlag, 2002.
- [6] Enright D., Fedkiw R., Ferziger J., Mitchell I. A hybrid particle level set method for improved interface capturing. // *J. Comp. Phys.*, 2002, V.183, p.83-116.
- [7] Samet H. The Design and Analysis of Spatial Data Structures. — New York, Addison-Wesley, 1989.
- [8] Grob S., Reichelt V., Reusken A. A Finite Element Based Level Set Method for Two-Phase Incompressible Flows. // *Computing and Visualization in Science*, 2006.

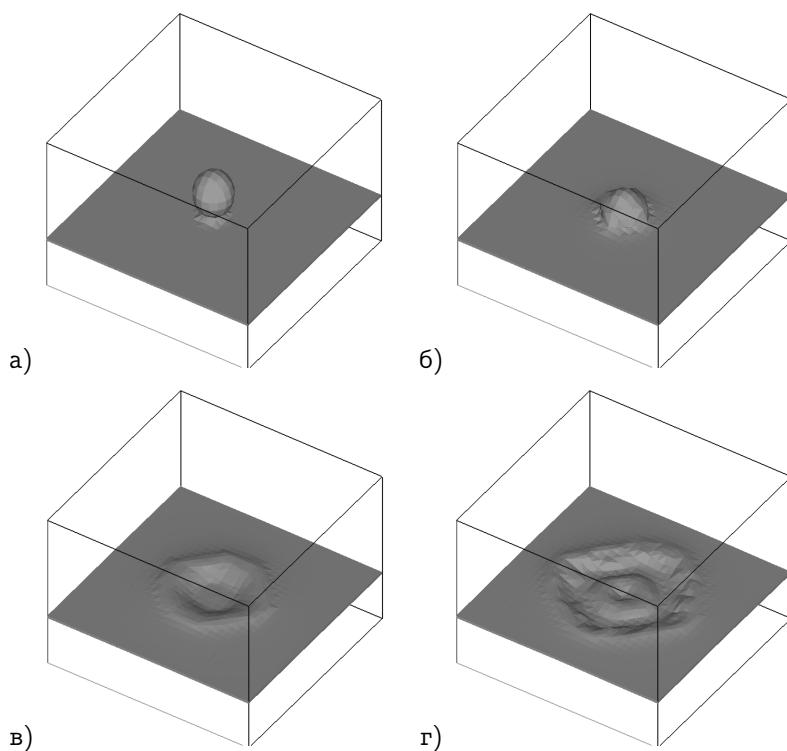


Рис. 7. Изообъем функции уровня $-0.01 \leq \phi(\mathbf{x}) \leq 0$ в момент времени а) $t = 5$, б) $t = 100$, в) $t = 260$ и г) $t = 425$.

- [9] Kaporin I. High quality preconditioning of a general symmetric positive definite matrix based on its $U^T U + U^T R + R^T U$ -decomposition. // *Numer.Linear Algebra Appl.*, 1998, V.5, p.483-509.
- [10] Saad Y. Iterative methods for sparse linear systems, Second Edition.
— Philadelphia, PA: SIAM, 2003.

Алгоритмы слепого разделения источников в пакетном режиме[§]

Д. В. Савостьянов[®]

При анализе числовых данных, изменяющихся во времени, часто есть основания считать, что наблюдаемые последовательности с достаточной точностью могут быть выражены линейными комбинации нескольких независимых компонент. Задача отыскания независимых компонент по заданной смеси называется слепым разделением сигналов. Для ее решения разработано множество методов, которые можно отнести к одному из двух классов — методы, использующие в процессе расчета весь массив данных целиком, и методы, основанные на диагонализации статистик, полученных по исходным данным. В этой статье мы показываем, что только методы второго типа допускают «пакетный» режим обработки данных, то есть могут эффективно применяться в случае, когда сигналы не могут храниться в доступной оперативной памяти во время проведения расчета.

1. Введение

1.1. История задачи. Возможно, наиболее классическим примером задачи, для решения которой требуется разделить наблюдаемые числовые последовательности на набор независимых источников, является «задача о вечеринке». На шумной вечеринке, где одновременно разговаривает множество людей, человек может концентрировать внимание на беседе с товарищем и игнорировать речь всех

[§]Работа выполнена при поддержке грантов РФФИ 05-1-00721, 06-01-08052 и Программы приоритетных фундаментальных исследований Отделения математических наук РАН «Вычислительные и информационные проблемы решения больших задач» по проекту «Матричные методы и технологии для задач со сверхбольшим числом неизвестных».

[®]Институт вычислительной математики РАН

остальных; однако услышав из дальнего конца зала свое имя, он распознает его и реагирует на призыв. Эта психофизическая способность была впервые описана в 1953 году Колином Черри [1], изучавшим проблемы, встающие перед диспетчерами крупных аэропортов, когда приходящие сообщения от нескольких пилотов смешиваются в динамике. В своей статье Черри заключил, что «способность разделять звуки и игнорировать посторонние шумы основана на характеристиках звука, таких как тон говорящего, направление, с которого приходит звук, высота звука, частота речи». Таким образом было сформулировано предположение о том, что мозг разделяет поступающую смесь звуковых сигналов, основываясь на различии в структуре исходных компонент. Механизм разделения сигналов в человеческом мозге до сих пор не изучен полностью, однако алгоритмы решения родственных задач с помощью компьютеров нашли широкое применение в области цифровой обработки сигналов, где и получили название методов (слепого) разделения сигналов. На их основе решается множество практических задач в самых различных областях:

- медицинское исследование функций мозга на основе данных магнитной энцефалографии, магнитно-резонансной томографии или спектроскопии (см., напр. [2, 3, 4, 5, 6]);
- развитие методов передачи информации через MIMO (multi-input multi-output) каналы (см., напр. [7, 8, 9]);
- анализ смесей в химии и спектрографии (см. обзор [10]);
- исследование числовых рядов в финансовой математике [11, 12], социологии, статистике, при анализе спроса;
- контроль транспортных и производственных процессов;
- мультимедийные приложения, распознавание текстов и изображений [13, 14].

Математическая постановка задачи во всех этих приложениях весьма схожа и довольно проста: предполагается, что наблюдаемые источники $y_i(t)$, $i = 1, \dots, m$, линейно выражаются через независимые компоненты $x_j(t)$, $j = 1, \dots, n$. Не ограничивая общности, можно считать, что и те, и другие имеют нулевое среднее. В

матрично-векторной записи

$$y(t) = Ax(t) + \xi(t), \quad (*)$$

векторы $y(t)$ и $x(t)$ называются соответственно векторами сигналов и источников, матрица A называется *смешивающей* и предполагается не зависящей от параметра t (дискретного или непрерывного), называемого временем. В наиболее общем случае никакой априорной информации о матрице A и характеристиках источников $x_i(t)$ не имеется — тогда говорят о методах *слепого разделения* сигналов (blind source separation, BSS), которым и посвящена наша статья. Компоненты вектора *шума* $\xi_i(t)$ полагаются независимыми и, как правило, нормально распределенными. Источники $x_i(t)$ предполагаются *независимыми*, однако при решении задачи о разделении сигналов это условие может быть по-разному использовано и отображено на алгоритм.

1.2. Классификация существующих методов решения. В простейшем случае независимость сигналов формулируется как их некоррелированность, то есть диагональность матрицы $E[x(t)x^*(t)]$, где $E[\cdot]$ — осреднение по времени. Тогда можно записать (для простоты изложения полагаем, что шумов нет)

$$\Phi_y = E[y(t)y^*(t)] = E[Ax(t)x^*(t)A^*] = A E[x(t)x^*(t)] A^* = ADA^*,$$

учитывая, что A не зависит от времени. Таким образом, A можно определить через решение задачи диагонализации матрицы ковариации. Один ответ очевиден — построить собственное разложение $\Phi_y = U\Lambda U^*$ и использовать в качестве A матрицу собственных векторов, а в качестве D — собственные значения Λ . Это решение не единственно (в качестве A может быть взята матрица вида

$$A = U\Lambda^{1/2}W\Lambda^{\dagger/2}, \quad (1)$$

где W произвольная унитарная). В самом деле, матрица U унитарна, а смешивающая матрица Λ — не обязательно. Впрочем, описанный нами сейчас метод *главных компонент* (principal component analysis, PCA), несмотря на свою простоту, широко применяется для понижения размерности при статистической обработке больших массивов данных (см. напр., [15, 16, 17]) и по сей день используется

для предобработки (так называемого «обеления») входных данных в большинстве методов слепого разделения сигналов.

Мы поняли, что одной некоррелированности источников для разделения источников недостаточно, и требуется использовать какие-то дополнительные средства описания их независимости. В зависимости от выбора этих средств, итоговый алгоритм можно отнести к одному из следующих семейств.

- Алгоритмы, в которых независимые компоненты выбираются в направлениях, максимизирующих степень их *негауссости*, минимизирующих меру *взаимной информации* или же максимизирующих *критерий правдоподобия* в линейном пространстве данных, возможно с дополнительными ограничениями. К этому семейству относятся алгоритмы *FastICA* [18], *MJICA* [22], *SNICA*. Обзоры можно найти в [20, 21].
- Алгоритмы, в которых дополнительная информация о независимости извлекается за счет одновременной диагонализации *нескольких* статистик и/или использовании кумулянтов высокого порядка [23, 25]. При этом первыми вычисляются не независимые компоненты, а смешивающая матрица. К этому семейству относится, например, метод *JADE*.

С вычислительной точки зрения различие состоит в следующем. Алгоритмы первой группы вычисляют непосредственно независимые компоненты, итерационно адаптируя *весовой вектор* w (или всю *размешивающую матрицу* W) так, чтобы функции $x_i(t) = (w_i, y(t))$ давали экстремум функционала, выбранного в качестве статистического описания независимости. Таким образом, в ходе итерационного процесса требуется хранить в памяти весь вектор данных *целиком* и на каждом шаге вычислять на его основе градиент целевой функции. Это может быть затруднительно в том случае, если объем данных слишком велик, и они не могут одновременно присутствовать в оперативной памяти (например, если речь идет о вычислениях в реальном времени на небольшом вычислительном узле). В этом случае разделение сигналов должно происходить порциями, помещающимися в оперативную память, а уменьшение размера такой порции сказывается на статистических свойствах выборки и итоговой точности разделения.

Алгоритмы второй группы работают не с самими сигналами, а только лишь с их статистиками. Центральная составляющая этих методов — одновременная диагонализация нескольких матриц статистик, причем их число может быть и достаточно большим. Алгоритмы решения этой задачи предлагались в работах [24, 26, 27, 28]; кроме того, поскольку задача одновременной диагонализации матриц эквивалентна задаче о трилинейном разложении тензора (трехмерного массива), для решения может быть использован любой алгоритм вычисления трилинейной аппроксимации тензора, со многими из которых можно познакомиться в [29]–[36]. Однако популярность этой группы методов у инженеров-вычислителей, по всей видимости, ниже, чем методов типа FastICA. Об этом говорит как сравнительно небольшое количество отчетов об успешном использовании методов одновременной диагонализации для решения задачи BSS, так и отсутствие подобных алгоритмов в доступных пакетах библиотечных программ. Вероятно, это связано с определенными трудностями при переходе с теоретико-вероятностного или статистического языка описания на язык линейной алгебры и матричной и тензорной арифметики. Однако эти методы кажутся нам очень перспективными, в частности при разделении сигналов в «пакетном» режиме, то есть в ситуации, когда невозможно разместить весь имеющийся массив данных целиком в оперативной памяти.

В этой статье мы рассмотрим вопросы реализации алгоритмов слепого разделения сигналов в пакетном режиме. В целях связности изложения мы кратко изложим схемы типовых алгоритмов из первой и второй группы, но для упрощения изложения будем пре-небречь наличием шумов в модели (*).

2. Метод FastICA

Алгоритм изложен по статье [19].

2.1. Проектирование на базис главных компонент. Как показано в первом разделе, требование некоррелированности источников позволяет определить часть информации о смешивающей матрице, записав ее в виде (1) и преобразовав модель (*) к виду

$$U\Lambda^{1/2}W\Lambda^{\dagger/2}x(t) = y(t),$$

где Λ и U — собственные значения и собственные вектора матрицы корреляции Φ_y , Λ^\dagger — псевдообратная матрица, W — неизвестная унитарная матрица. Обратив первые два сомножителя и переобозначив $x(t) := \Lambda^{\dagger/2}z(t)$ (источники определены с точностью до множителя и порядка), получаем линейную модель с *унитарной смешивающей матрицей*

$$Wx(t) = z(t), \quad \text{где } z(t) = \Lambda^{\dagger/2}U^*y(t). \quad (2)$$

В методе FastICA унитарность смешивающей матрицы строго необходима, так как она позволяет находить независимые компоненты по-одной. Предварительное «обеление» сигналов по формуле (2) часто применяется и в других алгоритмах В88, так как оно увеличивает устойчивость решаемой задачи к шумам и ошибкам округления.

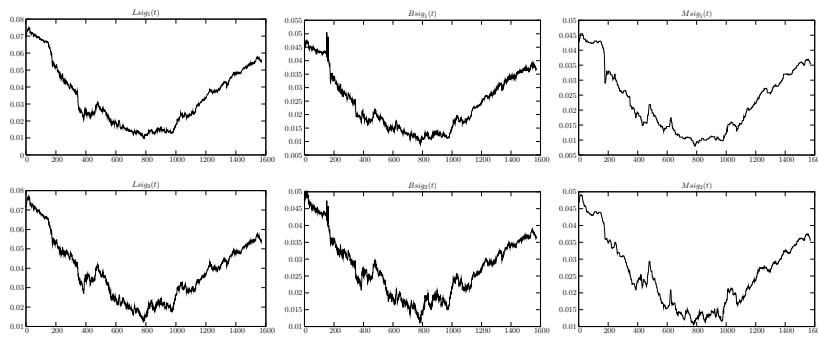
Отметим, что при вычислении псевдообратной матрицы можно считать нулевыми все диагональные элементы, меньшие некоторого порогового значения, сузив тем самым задачу с пространства, образованное линейной оболочкой сигналов, на подпространство, образованное линейной оболочкой старших компонент (сигнальное подпространство), и отфильтровав подпространство младших компонент (шумовое). Во многих приложениях, например, при анализе больших массивов схожих изображений (распознавание портретных фотографий, отпечатков пальцев) это сужение приводит к значительной экономии памяти, требуемой для хранения информации. В финансовой математике при анализе динамики курсов акций метод РСА тоже позволяет сжать массив данных, но что еще важнее — определить количество независимых факторов, действующих на рынок. На рис. 1 показан пример задачи, в которой данные о курсах 33 акций были с относительной точностью около одного процента приближены линейной комбинацией трех главных компонент.

2.2. Критерий независимости и меры негауссности. Некоррелированность источников теперь выполнена автоматически, и нам требуется некоторое дополнительное описание независимости для определения W . Отметим, что переход в базис главных компонент позволяет определять независимые компоненты по-отдельности.

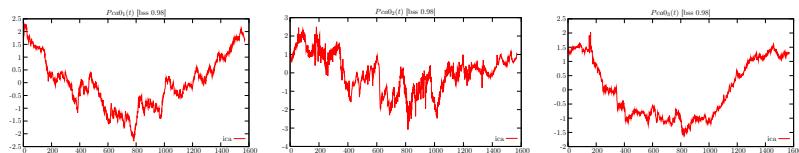
$$x(t) = W^*z(t), \quad x_i(t) = \sum_{j=1}^m w_{ji}z_j(t) = (w_i, z(t)), \quad i = 1, \dots, n,$$

Рис. 1. Применение метода главных компонент для понижения размерности при описании курсов акций

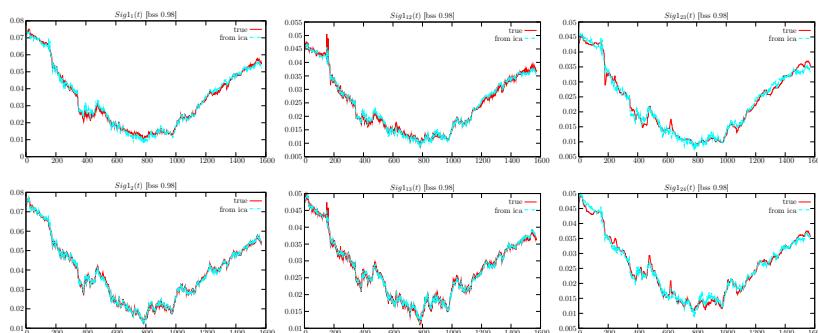
Курсы различных акций (всего 33, на рисунках 6)



Старшие главные компоненты (сигнальное пространство)



Проекции сигналов на сигнальное пространство



где весовой вектор w_i определяет i -й столбец матрицы W .

Полученная формула определяет независимую компоненту $x_i(t)$ как линейную комбинацию главных компонент, которые в свою очередь являются линейной комбинацией наблюдаемых сигналов. Будем рассматривать компоненты $x_i(t)$ как реализации во времени некоторых случайных величин, а $y_i(t)$ и $z_i(t)$ как линейную комбинацию (смесь) этих случайных величин. Классический результат теории вероятности — центральная предельная теорема — говорит, что функция распределения суммы двух случайных величин всегда ближе (строго говоря, не дальше) к нормальному распределению, чем функции распределения слагаемых. Отметим, что линейная комбинация главных компонент с произвольными весами w может быть представлена в виде

$$\xi = w^* z = w^* Wx = q^* x, \quad q = W^* w.$$

Пусть $\mathfrak{M}(\xi)$ — некоторая *мера негауссности* случайной величины ξ (пока нам не важно, как именно выбирать эту меру). Максимум

$$\mathfrak{M}(\xi) = \mathfrak{M}(q^* x) = \mathfrak{M}\left(\sum_{i=1}^n \bar{q}_i x_i\right)$$

достигается, когда в правой части равенства стоит всего лишь одно слагаемое x_i . При этом весовой вектор как-то нормирован, например $\|w\| = 1$, а в силу унитарности матрицы W это означает, что и вектор q имеет нормировку $\|q\| = \|W^* w\| = \|w\| = 1$. Таким образом, максимум меры негауссности для ξ достигается на векторе q , который имеет только один ненулевой (а следовательно единичный) элемент. Отсюда получаем

$$\arg \max_w \mathfrak{M}(w^* z) = W \arg \max_q \mathfrak{M}(q^* x) = We_i = w_i, \quad (3)$$

то есть весовой вектор w , доставляющий максимум меры негауссности линейной комбинации главных компонент, является одним из столбцов матрицы W .

Итак, *ключевой идеей* метода является поиск такой линейной комбинации главных компонент $z(t)$, для которой степень негауссности суммы максимальна. Коэффициенты этой суммы определяют столбец смешивающей матрицы W .

В качестве меры негауссности рассматривают функции вида

$$\mathfrak{M}(\xi) = \mathfrak{M}(w^* z) = E[G(|w^* z|^2)],$$

где функция $G(\cdot)$ может подбираться исходя из особенностей конкретной задачи. Если $G(\xi) = \xi^2$, то мерой негауссности является коэффициент эксцесса (kurtosis). Так называют статистику четвертого порядка, заданную формулой

$$\text{kurt}(y) = E[|y|^4] - E[y y^*] E[y y^*] - E[y y] E[y^* y^*] + E[y y^*] E[y^* y],$$

что вместе с условием независимости действительной и мнимой частей $y \in \mathbb{C}$ означает

$$\text{kurt}(y) = E[|y|^4] - 2(E[|y|^2])^2 - |E[y^2]|^2 = E[|y|^4] - 2.$$

Куртозис обращается в ноль для величин, имеющих нормальное распределение.

Эмпирически найдены и другие, часто более эффективные, выражения для функции G (ее называют функцией контраста или *контрастной функцией*). Когда мера негауссности выбрана, строят алгоритм ее максимизации при ограничениях на w .

2.3. Максимизация контрастной функции.

Максимизируем

$$\mathfrak{M}(w^* z) = E[G(|w^* z|^2)] \quad \text{при} \quad E[|w^* z|^2] = \|w\|^2 = 1.$$

Экстремум достигается в точке, где

$$\nabla E[G(|w^* z|^2)] - \beta \nabla E[|w^* z|^2] = 0.$$

Градиент вычисляется отдельно по действительным и мнимым частям w .

$$\nabla E[G(|\xi|^2)] = \begin{pmatrix} \frac{\partial}{\partial w_{1r}} \\ \frac{\partial}{\partial w_{1i}} \\ \vdots \\ \frac{\partial}{\partial w_{nr}} \\ \frac{\partial}{\partial w_{ni}} \end{pmatrix} E[G(|w^* z|^2)] = 2 \begin{pmatrix} E[\Re(z_1 \xi^*) g(|\xi|^2)] \\ E[\Im(z_1 \xi^*) g(|\xi|^2)] \\ \vdots \\ E[\Re(z_n \xi^*) g(|\xi|^2)] \\ E[\Im(z_n \xi^*) g(|\xi|^2)] \end{pmatrix},$$

где, напомним, $\xi = w^* z$. Второе слагаемое раскрывается как

$$\nabla E[|w^* z|^2] = 2 \begin{pmatrix} \Re(w_1) \\ \Im(w_1) \\ \vdots \\ \Re(w_n) \\ \Im(w_n) \end{pmatrix}$$

в силу ортогональности главных компонент $E[zz^*] = I$.

Для поиска точки экстремума мы применяем метод Ньютона. Матрица Якоби для $\nabla E[G(|w^* z|^2)]$ приближенно равна

$$\begin{aligned} \nabla^2 E[G(|w^* z|^2)] &= 2E[(\nabla^2 |w^* z|^2)g(|w^* z|^2) + \\ &\quad + 2(\nabla |w^* z|^2)(\nabla |w^* z|^2)^\top g'(|w^* z|^2)] \approx \\ &\approx 2E[g(|w^* z|^2) + |w^* z|^2 g'(|w^* z|^2)] I, \end{aligned}$$

где $g = G'$, а приближенное равенство получено разделением средних в предположении, что случайная величина и ее производная не коррелируют. Кроме того, мы воспользовались равенством $E[zz^\top] = 0$, которое следует из некоррелированности действительной и мнимой части исходных источников. Матрица Якоби для второго слагаемого равна

$$\beta \nabla^2 E[|w^* z|^2] = 2\beta I.$$

Аппроксимация всего якобиана имеет вид

$$J = 2(E[g(|w^* z|^2) + |w^* z|^2 g'(|w^* z|^2)] - \beta)I.$$

Отметим, что якобиан диагонален и аналитически обратим. Таким образом, выражение для одной итерации Ньютона

$$\begin{aligned} w^+ &= w - \frac{E[z(w^* z)^* g(|w^* z|^2)] - \beta w}{E[g(|w^* z|^2) + |w^* z|^2 g'(|w^* z|^2)] - \beta} \\ w_{\text{new}} &= w^+ / \|w^+\|. \end{aligned}$$

Домножая обе части получившегося уравнения на знаменатель, мы получаем более простое выражение

$$w^+ = E[z(w^* z)^* g(|w^* z|^2)] - E[g(|w^* z|^2) + |w^* z|^2 g'(|w^* z|^2)]w;$$

$$w_{\text{new}} = w^+ / \|w^+\|.$$

2.4. Алгоритм FastJCA.

(0) Возьмем случайный вектор $w_{\{0\}}$ единичной нормы. Положим $k = 0$.

(1) Вычислим

$$w_{\text{new}} := E[z(w^* z)^* g(|w^* z|^2)] - E[g(|w^* z|^2) + |w^* z|^2 g'(|w^* z|^2)] w;$$

Отнормируем w_{new} на единицу.

(3) Если w_{new} достаточно близок к w

завершим алгоритм;

иначе

положим $w = w_{\text{new}}$ и вернемся на шаг 1.

В качестве меры близости весовых векторов используем угол между ними.

По завершении алгоритма w дает нам один из столбцов ортогональной смешивающей матрицы W , тем самым мы определяем одну независимую компоненту $x(t)$ как скалярное произведение

$$x(t) = w^* z(t).$$

Чтобы найти другие столбцы смешивающей матрицы и соответствующие независимые компоненты, воспользуемся унитарностью W . Новые столбцы лежат в ортогональном дополнении к подпространству найденных, поэтому достаточно внедрить в алгоритм шаг ортогонализации w_{new} к пространству уже найденных столбцов W .

(2) Ортогонализуем найденный вектор к пространству, заданному столбцами W

$$w_{\text{new}} := (I - WW^*) w_{\text{new}}.$$

Отнормируем w_{new} на единицу.

После завершения алгоритма W есть смешивающая матрица, а независимые компоненты определяются как $x(t) = W^* z(t)$, где $z(t)$ — главные компоненты.

2.5. Пример. В заключении главы приведем классический пример задачи, с которой ЛСА справляется лучше, чем РСА . Рассмотрим смеси двух независимых сигналов: синусоиды и прямоугольного меандра и попытаемся их разделить. Результаты эксперимента представлены на рис. 2 — без дополнительных комментариев ясно, что главные компоненты в принципе не восстанавливают источники, тогда как независимые компоненты восстанавливают их с точностью до масштаба и порядка.

3. Одновременная диагонализация многих статистик

Изложение этой главы следует логике [25].

3.1. Почему две статистики лучше, чем одна, а больше двух — еще лучше. Использования только корреляционной статистики недостаточно, поскольку задача диагонализации одной матрицы имеет не единственное решение. Однако задача одновременной диагонализации двух матриц имеет в общем случае имеет единственное решение и, что также очень важно, разработаны стандартные алгоритмы решения этой задачи — обобщенной задачи на собственные значения. Возникает идея — использовать дополнительно к корреляционной статистике Φ_y другую статистику Q_y , диагонализуемую тем же преобразованием. Тогда задача определения смещающей матрицы A из системы уравнений

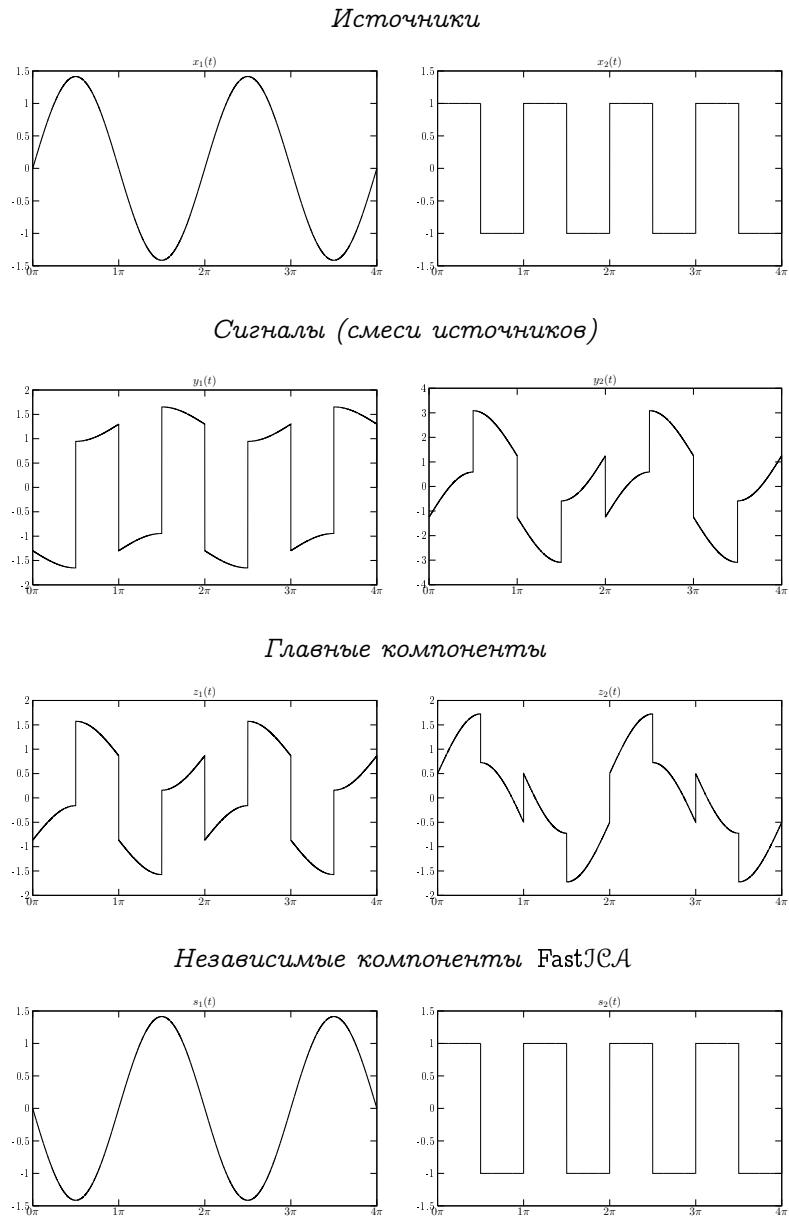
$$\Phi_y = AD\Phi A^*, \quad Q_y = AD_Q A^*$$

имеет единственное решение.

Казалось бы, проблема закрыта. Тем не менее, выбрать две различные статистики оказывается не таким уж простым делом. Если Φ и Q «похожи», то точность решения обобщенной задачи на собственные значения значительно ухудшается — новая статистика на деле не приносит новой информации. Но как выбрать кросс-статистику так, чтобы она оказалась независима от первой? Априорного ответа, подходящего для всех случаев, по всей видимости, нет. Возможный в большинстве случаев ответ — взять большое число таких статистик и среди них обязательно окажется пара независимых.

В этом случае основным вычислительным ядром метода станет задача одновременного приведения к диагональному виду всех име-

Рис. 2. Классический пример разделения синусоиды и меандра



ющихся кросс-статистик. При числе матриц более двух эта задача, вообще говоря, неразрешима. Однако в нашем случае существование решения гарантируется моделью (*). Рассмотрим задачу об одновременной диагонализации p матриц (супер-обобщенную задачу на СЗ)

$$A_k = U \Lambda_k V^\top,$$

Перепишем ее в смысле минимизации нормы отклонения

$$\|A_k - U \Lambda_k V^\top\| \rightarrow \min$$

по всем неизвестным матрицам U, V и Λ_k .

Записав эту задачу в поэлементном виде

$$a_{ij}^k = \sum_{\alpha} u_{i\alpha} \lambda_{k\alpha} v_{j\alpha},$$

видим, что она равносильна задаче представления трехмерного массива $A = [a_{ijk}]$ в виде

$$a_{ijk} = \sum_{\alpha} u_{i\alpha} v_{j\alpha} w_{k\alpha},$$

где столбцы матрицы W содержат элементы диагональных матриц Λ_k . Возникла задача *трилинейного разложения* трехмерного массива данных (тензора). Возможные подходы к ее решению описаны в [23, 24, 25] и [29]-[36].

3.2. Какие кросс-статистики можно использовать. Рекомендуемый метод выбора Q зависит от характеристик рассматриваемых сигналов.

3.2.1. Нестационарные сигналы. Если сигнал нестационарен, то его осреднение по различным отрезкам (окнам) времени приводит к различным статистикам

$$Q_y(t) = E_{(t:t+\Delta t)}[y(t)y^*(t)] = A E_{(t:t+\Delta t)}[x(t)x^*(t)]A^* = A Q_x(t)A^*.$$

Выбрав $Q_y = Q_y(t)$ для какого-то конкретного t , или взяв в качестве Q_y случайную линейную комбинацию $Q_y(t)$ для различных моментов t , мы получаем новую статистику. Вообще говоря, если она оказывается близка к исходной статистике Φ_y , то ее использование не дает нам новой информации, и задача поиска обобщенного

собственного разложения становится плохо определенной. В этом случае можно использовать набор матриц $Q_y^k = Q_y(t_k)$ для цепочки моментов t_k и свести задачу к построению одновременной диагонализации для *всех* матриц Q_y^k .

3.2.2. Цветные сигналы. Если сигнал не является белым, то возможно использование следующей кросс-корреляцией

$$Q_y(\tau) = E[y(t)y^*(t + \tau)] = A E[y(t)y^*(t + \tau)] A^* = A Q_x(\tau) A^*.$$

Если эта автокорреляция для источника не равна нулю, то эта статистика предоставляет новую информацию при выборе одного или нескольких различных сдвигов τ .

3.2.3. Не-Гауссовые сигналы. Если сигнал является стационарным и не обладает автокорреляцией, то статистики для различных t и τ не дают новой информации. В этом случае можно воспользоваться статистикой четвертого порядка, называемой *кумулянтом*

$$(C_y)_{ijkl} = E[y_i \bar{y}_j y_k \bar{y}_l] - E[y_i \bar{y}_j] E[y_k \bar{y}_l] - E[y_i y_k] E[\bar{y}_j \bar{y}_l] - E[y_i \bar{y}_l] E[\bar{y}_j y_k].$$

Кумулянт суммы независимых величин равен сумме кумулянтов. Для сигнала, имеющего нормальное распределение, кумулянт равен нулю. Поэтому при применении статистики четвертого порядка происходит автоматическая фильтрация шумов, возникающих в принимающих устройствах, или случайных погрешностей измерения, если они имеют нормальное распределение.

Статистики второго и четвертого порядка Φ_y и C_y , отвечающие наблюдаемым сигналам $y(t)$, связаны со статистиками источников следующими соотношениями^{def)}

$$\Phi_y = A \Phi_x A^*, \quad C_y = C_x \times_1 A \times_2 \bar{A} \times_3 A \times_4 \bar{A}. \quad (4)$$

Из независимости источников следует диагональность статистик Φ_x и C_x , причем второе условие означает, что тензор имеет ненулевые

^{def)}Тут символом « \times_p » обозначено умножение тензора на матрицу вдоль p -го индекса. Например для тензора $A = [a_{ijkl}]$ размера $n \times n \times n \times n$ и матрицы B размера $m \times n$ результатом операции « \times_2 » является тензор $C = [c_{ijkl}]$ размера $n \times m \times n \times n$, поиндексно определенный равенством

$$c_{ijkl} = \sum_{j'=1}^n a_{ij'kl} b_{jj'}.$$

элементы только при $i = j = k = l$, и, очевидно, предоставляет существенно больше информации для определения A . Более того, в общем случае второе уравнение системы (4) не имеет точного решения и рассматривается в смысле минимизации нормы отклонения целевого тензора от ближайшего диагонального.

$$A = \arg \min_{B, D} \|C_y - D \times_1 B \times_2 \bar{B} \times_3 B \times_4 \bar{B}\| \quad (\text{diag})$$

Решив задачу в этой формулировке, можно даже получить *сверхразрешение*, то есть выделить из n наблюдаемых смесей m исходных независимых компонент при $m > n$. Однако алгоритм решения оказывается в несколько раз сложнее, чем метод одновременной диагонализации матриц. Есть возможность пожертвовать сверхразрешением и свести задачу к более простой. Для этого рассмотрим тензор C_z размера $m \times m \times m \times m$, как семейство из m^2 матриц, объединив последние два индекса в один

$$\tilde{C} = [(c_{ij})_\alpha] = (C_z)_{ij(kl)}, \quad (kl) \leftrightarrow \alpha.$$

Заменим задачу суперсимметричной диагонализации четырехмерного тензора C более простой задачей одновременной диагонализации семейства m^2 матриц C_α . На языке тензорных операций это означает, что вместо задачи (diag) мы рассматриваем

$$A = \arg \min_{B, Q, D} \|\tilde{C} - D \times_1 B \times_2 \bar{B} \times_3 Q\| \quad (\text{diag}')$$

очевидно, накладывающую ослабленные требования на A , поскольку мы пренебрегаем структурой матрицы Q и «забываем» о ее связи с B . Новая задача имеет единственное решение, однако только при числе источников не больше числа приемников.

4. Реализация алгоритмов \mathcal{BSS} в пакетном режиме

4.1. Требования к пакетной версии алгоритмов. Под работой алгоритмов в пакетном режиме мы понимаем следующее. Пусть общая протяженность принятых сигналов T_{full} но в вычислительную систему для обработки они поступают N пакетами длины T (для простоты изложения будем полагать, что длины всех пакетов одинаковы — даже если это не так, суть метода не меняется). Эта

модель работы с данными может потребоваться по одной из следующих причин.

- Нет возможности ждать достаточно долго, пока весь сигнал длины T_{full} будет принят. Напротив, требуется получить предварительные данные о сигналах как можно быстрее (пусть и со значительной погрешностью, вызванной малой длиной накопленной информации), и затем только уточнять их по новым фрагментам полученных сигналов.
- Оперативной памяти вычислительного устройства и/или его вычислительной мощности может быть недостаточно для запуска алгоритма разделения сигналов длины T_{full} . При этом требуется принудительно расщепить рассматриваемый сигнал на несколько меньших частей, которые могут быть обработаны на имеющемся вычислительном устройстве, и работать с ними последовательно.
- В условиях реальных вычислений положение источников и/или приемников и/или состояние среды, в которой распространяется сигнал, могут меняться. Даже если эти изменения достаточно медленные, они оказывают влияние на смешивающую матрицу и точность выполнения основной линейной модели (*). В этом случае нет смысла использовать всю информацию о сигнале длины T_{full} , так как нарушение линейной модели вносит неустранимую погрешность в результат работы алгоритма. Необходимо оперировать отрезками сигнала длины T , такими что на протяжении этого времени состояние системы и следовательно элементы смешивающей матрицы A могут с удовлетворительной точностью считаться постоянными.

При реализации алгоритмов слепого разделения сигналов в пакетном режиме (будем называть их пакетными версиями алгоритмов или кратко *пакетными алгоритмами*) мы хотели бы по возможности добиться выполнения следующих условий.

- **Накопление информации.** После того, как пакетный алгоритм получил N пакетов данных длины T , точность разделения сигналов должна быть такой же, как и у исходного алгоритма, обработавшего сигнал длиной $T_{full} = NT$.

- **Сохранение устойчивости.** Если при обработке какого-то из полученных пакетов пакетный алгоритм не достигает сходимости (или не завершается корректно по любой иной причине), это не должно останавливать алгоритм в целом. Текущий пакет данных должен быть разделен с использованием размешивающей матрицы W с прошлого шага, и алгоритм должен быть продолжен.
- **Связность результата.** Поскольку источники определяются с точностью до множителя и порядка, алгоритм в пакетной версии должен позаботиться о том, чтобы разделенные сигналы в двух соседних пакетах были *сшиты*, то есть на границе раздела пакетов не наблюдалось скачка амплитуды, фазы источников или перестановки выдаваемых фрагментов источников местами.

В следующих разделах мы покажем, для каких методов можно добиться выполнения указанных требований.

4.2. Почему алгоритм FastICA не накапливает информацию. Алгоритм FastICA (и родственные ему) вычисляет размешивающую матрицу W , решая экстремальную задачу относительно меры $\mathfrak{M}(w^*z(t))$. Предположим, что передача ведется двумя пакетами по T элементов в каждом. Получив первый пакет $y^{[1]}(t)$, мы переходим в базис главных компонент $z^{[1]}(t)$ и применяя алгоритм FastICA, находим текущее приближение к смешивающей матрице $W^{(1)}$. Однако после получения второго пакета $y^{[2]}$ (и что важно, потеряв вектора $y^{[1]}(t)$ и $z^{[1]}(t)$), мы не можем максимизировать

$$\mathfrak{M}(w^*z(t)) = \mathfrak{M}(w^*z^{[1]}(t)) + \mathfrak{M}(w^*z^{[2]}(t)),$$

поскольку не можем вычислить первое слагаемое для произвольного w . Если отдельно максимизировать только второе слагаемое, точность определения матрицы $W^{(2)}$ на втором этапе пакетного алгоритма окажется не выше, чем на первом шаге — поступившая новая информация не позволяет уточнить элементы смешивающей матрицы. В этом смысле мы говорим, что не существует пакетной версии алгоритма FastICA, накапливающей информацию в ходе работы.

Тем не менее, метод FastICA может быть полезен, особенно в случае, когда используемый размер пакетов достаточно велик, а ста-

бильность матрицы A при переходе от пакета к пакету вызывает сомнения.

В следующих разделах мы покажем, что методы, основанные на диагонализации различных статистик принятых сигналов, обладают приятным свойством накопления информации, и построим соответствующие пакетные алгоритмы.

4.3. Пакетная версия метода главных компонент. В отличие от методов типа FastICA, алгоритм PCA допускает пакетную реализацию. Принципиальным является тот факт, что статистики, в отличие от меры $M(w^*z(t))$, не зависят от определяемых величин и могут аддитивно обновляться в ходе работы пакетного метода. Для вычисления статистики на «большом» отрезке времени (например на полной длине приема T_{full}) нам достаточно просуммировать с некоторыми коэффициентами статистики с «малых» фрагментов времени (например статистики по пакетам длины T). При этом коэффициенты зависят, естественно, от длин соответствующих статистик и от номера принятого пакета. Например, если уже насчитана статистика

$$\Phi_y^{(k)} := E[y(t)y^*(t)]_{[0:kT]},$$

то статистика $\Phi_y^{(k+1)}$ на следующем шаге пакетного алгоритма определится как

$$\begin{aligned} \Phi_y^{(k+1)} &:= \frac{k}{k+1}\Phi_y^{(k)} + \frac{1}{k+1}E[y(t)y^*(t)]_{[kT:(k+1)T]} = \\ &= \frac{k}{k+1}\Phi_y^{(k)} + \frac{1}{k+1}E[y^{[k+1]}(t)y^{[k+1]}(t)^*]. \end{aligned}$$

4.4. Обновление статистик для сигналов. Простой по существу метод обновления статистики, предложенный для PCA, можно обобщить, сформулировав процедуру следующим образом.

Алгоритм 1. (*Обновление статистики $\Psi_y = E[G(y)]$*).

1. Получить $y^{[1]}$. Вычислить $\Psi_y^{(1)}$.

...

k. Получить $y^{[k]}$. Вычислить статистику $E[G(y^{[k]})]$ на текущем отрезке времени. Вычислить $\Psi_y^{(k)}$ по формуле

$$\Psi_y^{(k)} = \alpha_k \Psi_y^{(k-1)} + (1 - \alpha_k) E[G(y^{[k])}] \quad (5)$$

В зависимости от схемы выбора α_k алгоритм обновления может трактоваться различным образом.

- $\alpha_k = 0$ — модель абсолютно неустойчивой среды. Данные с прошлых шагов не используются, разделение ведется по каждому новому пакету независимо от информации о предыдущих.
- $\alpha_k = \frac{k-1}{k}$ — модель устойчивой среды, уточнение размешивающей матрицы на каждом шаге. Происходит точное обновление статистик, полученная $\Psi^{(k)}$ совпадает с усреднением за период $[0 : kT]$. За счет этого на каждом шаге уменьшается статистическая погрешность, вносимая конечным размером выборки, и повышается точность вычислений смешивающей матрицы — конечно, если искомая смешивающая не меняется на протяжении k полученных пакетов.
- $\alpha_k = 1$ — модель устойчивой среды, быстрое размешивание. Новая информация не используется, статистики (а вместе с ними и размешивающая матрица) копируются с первого этапа. Предполагается, что смешивающая матрица не меняется, и размера статистики на первой выборке длины T достаточно для восстановления размешивающей матрицы с необходимой точностью. Достоинство этой модели в очень быстром выполнении алгоритма на втором и последующих этапах — по сути, происходит только лишь умножение нового пакета сигналов на размешивающую матрицу, взятую с первого этапа.

Все три схемы выбора α_k хороши в рамках своей модели и имеют свои достоинства. Безусловно, возможны и другие схемы выбора параметра α_k , отвечающие иным предположениям об скорости изменения элементов смешивающей матрицы. Как видно из рассмотренных примеров, выбор α_k оказывает огромное влияние на работу и результаты алгоритма; тем не менее, мы не можем предложить какой-то одной, «действительно наилучшей» стратегии выбора α_k . Мы полагаем, что этот вопрос может решаться следующими способами:

- аналитически — при известной степени неустойчивости смешивающей матрицы, когда хорошо известны характеристики принимаемых сигналов;

- эмпирически — при работе в режиме «лаборатории», когда мы имеем дело с записанными сигналами, не знаем степень устойчивости среды во время их получения, но можем провести серию воспроизводимых экспериментов с разными параметрами α_k , имея на них проведение достаточно большое время;
- решением оператора — когда характеристики устойчивости среды и принимаемых сигналов неясны, а принимаемый сигнал должен обрабатываться практически в режиме реального времени. В этом случае имеет смысл придерживаться стратегии накопления информации $\alpha_k = (k-1)/k$ столь долго, пока качество разделения источников выглядит удовлетворительным и не ухудшается с каждым новым пакетом данных. Если качество разделения ухудшается (разделенные источники хуже декодируются или прослушиваются), следует «очистить историю», установив $\alpha_k = 0$ и начать накопление данных заново.

4.5. Алгоритмы в пакетном режиме. Все высказанное подготовило почву для того, чтобы изложить пакетные версии *всех* алгоритмов, использующих статистики сигналов, в единообразном виде.

Алгоритм 2. (Прототип пакетного алгоритма)

Вход: пакеты сигналов $y^{[k]}$, $k = 1, \dots, N$.

Выход: пакеты источников $x^{[k]}$ и смешивающая матрица.

Параметры: статистики $\Psi_y = E[G_p(y)]$, $p = 1, \dots, n_p$, схема выбора параметров α_k .

[Описан k -тый шаг алгоритма]

- Получить $y^{[k]}$.
- Вычислить статистики $E[G_p(y^{[k]})]$ на текущем отрезке времени.
- Обновить их по формуле (5) с помощью выбранных α_k .
- Запустить алгоритм слепого разделения сигналов по статистике $\Psi_y^{(k)}$. Найти размешивающую матрицу $W^{(k-1)}$
- Разделить источники $x^{[k]} = W^{(k)}y_{(k)}$.

- «Склейте» источники $x^{[k]}$ с предыдущими пакетами $x^{[k-1]}$.

Последний этап нами пока не обсуждался, а между тем именно он обеспечивает выполнение третьего требования к пакетной версии алгоритмов — связности результата. Посвятим ему отдельный раздел.

4.6. Поддержание связности. «Разрыв» может возникать по двум причинам:

- после получения нового пакета данных активная смещающая матрица изменилась так, что источники поменялись местами;
- после получения нового пакета данных амплитуда и фаза источников поменялась так, что наблюдается скачок на границе раздела пакетов.

Для решения первой проблемы — перестановки сигналов — можно предложить два способа. Первый состоит в том, что рассматриваются две матрицы: размешивающая с прошлого шага и смещающая с текущего. Их произведение

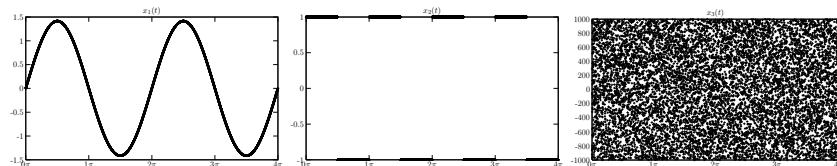
$$W^{[k-1]} A^{[k]} = P_k$$

должно быть близко к масштабированной матрице перестановки, если только смещающая матрица не меняется слишком сильно на отрезке времени порядка размера пакета. Эта матрица и указывает, как нужно упорядочить новые пакеты источников по отношению к имеющимся. Есть и другой подход, возможно, более точный, который состоит в том, чтобы рассматривать пакеты данных с некоторым небольшим наложением. Сохранив это небольшое количество элементов из прошлого пакета источников, мы можем подобрать коэффициенты масштабирования так, чтобы получить наилучшее совпадение фрагмента источников из прошлого и текущего пакетов в их области наложения. Однако этот метод в известном смысле усложняет логику программы.

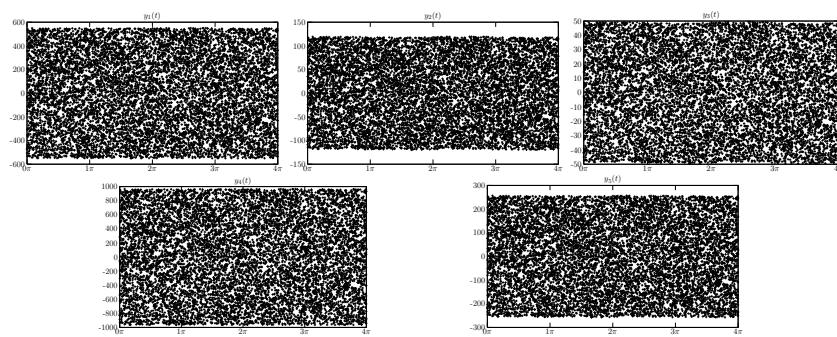
Рис. 3. Два сигнала и сильный шумовой источник

Смешиваем синусоидальный сигнал единичной нормы, прямоугольный меандр и шум порядка 10^3 .

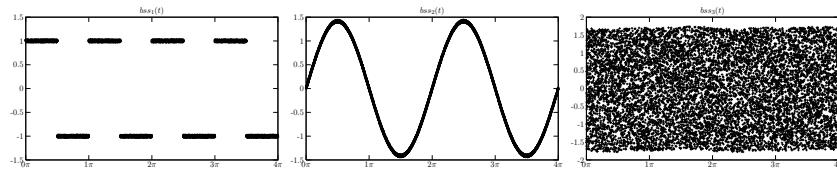
Размер выборки большой $l = 10^4$ отсчетов



Компоненты смеси выглядят, как шум



Тем не менее, метод BSS позволяет выделить источники из смеси



На выборке $l = 100$ точность вычислений ниже

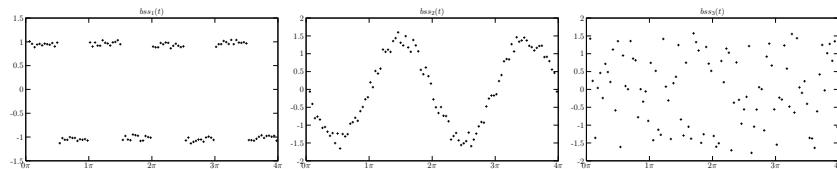
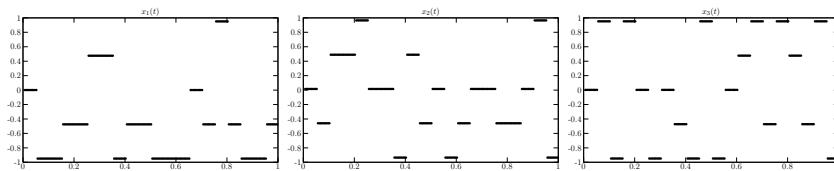
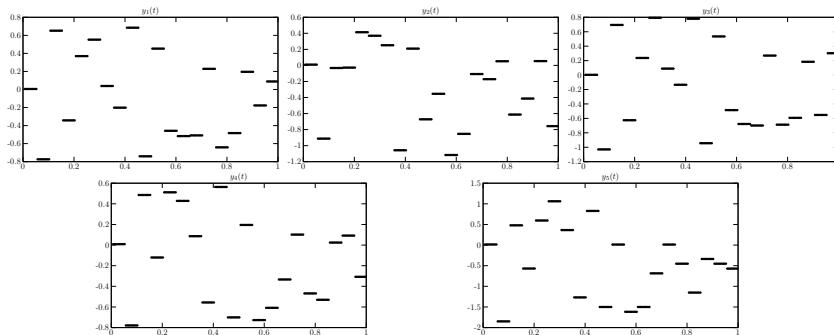


Рис. 4. Три сигнала с модуляцией типа qpsk

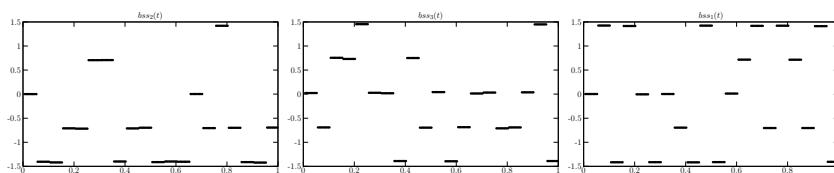
*Смешиваем три дискретных четырехпозиционных сигнала.
Размер выборки большой $l = 10^4$ отсчетов*



Компоненты смеси выглядят достаточно сложно



Тем не менее, разделение сигналов происходит успешно



5. Примеры

Рассмотрим несколько простых примеров разделения сигналов для иллюстрации возможностей метода BSS. На рисунке 3 показан эксперимент по выделению сигналов из-под сильного шумового источника. В качестве сигналов взяты традиционные синусоиды и прямоугольный меандр. Обратите внимание, что при разделении сигналов на малой выборке точность определения компонент значительно снижается.

На рисунке 4 показано разделение нескольких сигналов в модуляцией типа QPSK. Модуляция QPSK кодирует информацию, устанавливая фазу передаваемого сигнала в одну из четырех позиций. На рисунках — аналог этих сигналов в действительной арифметике, более удобный для графического отображения. Видим, что задача о разделении смеси трех кодированных сигналов может быть с успехом решена описанными методами BSS. На практике она возникает при передачи информации в MIMO каналах, например, в новых стандартах сотовой связи.

Список литературы

- [1] Cherry E. C. Some experiments on the recognition of speech, with one and with two ears // *Journal of Acoustical Society of America*. 1953. V 25(5). P. 975–979.
- [2] Hämäläinen M., Hari R., Ilmoniemi R., Knuutila J. and Lounasmaa O.V. Magnetoencephalography — theory, instrumentation, and applications to noninvasive studies of signal processing in the human brain // *Reviews of Modern Physics*. 1993. V. 65. P. 413–497.
- [3] De Lathauwer L., De Moor B., Vandewalle J. Fetal electrocardiogram extraction by source subspace separation ss *Proc. of IEEE Signal Processing / Athos Workshop on Higher-Order Statistics*, Girona, Spain. 1995. P. 134-138.
- [4] Vigário R. N. Extraction of ocular artifacts from EEG using independent component analysis // *Electroenceph. clin. Neurophysiol.* 1997. V. 103. P. 395–404.

- [5] Vigário R., Särelä J., and Oja E. Independent component analysis in wave decomposition of auditory evoked fields // *Proc. Int. Conf. on Artificial Neural Networks (ICANN'98)* 1998. P. 287–292.
- [6] Lei Xie, Jun Wu. Global optimal ICA and its application in MEG data analysis. // *Neurocomputing*. 2006. V. 69. P. 2438–2442.
- [7] B. Chen and A. P. Petropulu. Frequency domain MIMO system identification based on second and higher-order statistics // *IEEE Trans. Signal Processing*. 2001. V. 49, №8. P. 1677–1688.
- [8] De Lathauwer L., Comon P., De Moor B., Vandewalle J. ICA algorithms for 3 sources and 2 sensors // *Proc. of the IEEE Signal Processing Workshop on Higher-Order Statistics (HOS'99)*, Caesarea, Israel. 1999. P. 116–120.
- [9] De Lathauwer L., De Moor B., Vandewalle J. An algebraic approach to blind MIMO identification // *Proc. of the 2nd Int. Workshop on independent component analysis and blind source separation (ICA 2000)*, Helsinki, Finland. 2000. P. 211–214.
- [10] R. Bro. Review on Multiway Analysis in Chemistry — 2000–2005. // *Analytical Chemistry*. 2006. V. 36. P. 279.
- [11] Back A. D., Weigend, A.S. What drives stock returns? — An independent component analysis // *Proc. of the IEEE/IAFE/INFORMS Conf. on Comp. Intelligence for Financial Engineering (CIFEr)*. 1998. P. 141–156.
- [12] Moody J., Howard, Y. Term structure of interactions of foreign exchange rates // *Computational Finance — Proc. of the 6th Int'l Conf.* 1999. P. 24–35.
- [13] Draper B. A., Baek K., Bartlett M. S., Beveridge J. R. Recognizing faces with PCA and ICA // *CVIU(91)*, 2003. №1-2. P. 115–137.
- [14] Jongsun Kim, Jongmoo Choi, Yuneho Yi. Biometric Authentication — Springer Berlin, 2004.
- [15] Hotelling H. Analysis of a Complex of Statistical Variables with Principal Components // *Journal of Educational Psychology*. 1933. V. 24. P. 417–441.

- [16] Tetsuo Sato. Application of principal-component analysis on near-infrared spectroscopic data of vegetable oils for their classification // *Journal of the American Oil Chemists' Society*. 1994. V. 71, №3. P. 293–298.
- [17] Aiyyi Liu, Ying Zhang, Gehan E., Clarke R. Block principal component analysis with application to gene microarray data classification // *Statistics in Medicine*. 2002. V. 21. P. 3465–3474.
- [18] Hyvärinen A. Fast and robust fixed-point algorithm for independent component analysis // *IEEE Trans. on Neural Network*. 1999. V. 10, №3. P. 626-634.
- [19] E. Bingham and A. Hyvärinen. A fast fixed-point algorithm for independent component analysis of complex-valued signals. // *Int. J. of Neural Systems*. 2000. V. 10, №1. P. 1–8.
- [20] Comon P. Independent Component Analysis, a new concept? // *Signal Processing, Elsevier*. 1994. V. 36, №3. P. 287–314.
- [21] Hyvärinen A., Erkki Oja. Independent Component Analysis: Algorithms and Applications // *NEural Networks*. 2000. V. 13, №4-5. P. 411-430.
- [22] Stogbauer H., Kraskov A., Astakhov S. A., Grassberger P. Least Dependent Component Analysis Based on Mutual Information // *Phys. Rev.* 2004. E 70, 066123.
- [23] Cardoso J.-F., Souloumatic A. Blind beamforming for non Gaussian signals // *IEE Proc.-F*. 1993. V. 140, №6. P. 362–370.
- [24] Cardoso J.-F., Souloumatic A. Jacobi angles for simultaneous diagonalisation // *SIAM J. Mat. Anal. Appl.* 1996. V. 17 №1. P. 161–164.
- [25] Parra L., Sajda P. Blind Source Separation via Generalized Eigenvalue Decomposition // *J. of Machine Learning Research*. 2003. V. 4. P. 1261–1269.
- [26] De Lathauwer L., De Moor B., Vanderwalle J. Computation of the canonical decomposition by means of a simultaneous generalized

- Schur decomposition // *SIAM J. Matrix Anal. Appl.* 2004. V. 26. P. 295-227.
- [27] Ziehe A., Laskov P., Müller K.-R., Nolte G. A linear least-squares algorithm for joint diagonalization // *Proc. 4th Intern. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*. 2003. P. 469–474.
- [28] Ziehe A., Kawanabe M., Hamerling S., Müller K.-R. A fast algorithm for joint diagonalization with non-orthogonal transformations and its application to blind source separation // *Journal of Machine Learning Research*. 2004. V. 5. P. 801-818.
- [29] Оседецов И.В., Савостьянов Д.В. Методы разложения тензора // *Матричные методы и технологии решения больших задач Сб. под ред. Е. Е. Тыртышникова* С. 51–64.
- [30] Оседецов И.В., Савостьянов Д.В. Трехмерный аналог алгоритма крестовой аппроксимации и его эффективная реализация // *Матричные методы и технологии решения больших задач Сб. под ред. Е. Е. Тыртышникова* — ИВМ РАН, 2005. С. 65–99.
- [31] Оседецов И.В., Савостьянов Д.В. Быстрый алгоритм для одновременного приведения матриц к треугольному виду и аппроксимации тензоров // *Матричные методы и технологии решения больших задач Сб. под ред. Е. Е. Тыртышникова* — ИВМ РАН, 2005. С. 101–116.
- [32] Оседецов И.В., Савостьянов Д.В. Об одном алгоритме построения трилинейного разложения // *Матричные методы и технологии решения больших задач Сб. под ред. Е. Е. Тыртышникова* — ИВМ РАН, 2005. С. 117–130.
- [33] Оседецов И.В., Савостьянов Д.В. Минимизационные методы аппроксимации тензоров и их сравнение // *ЖВМиМФ*. 2006. Т. 46, №10. С. 1641-1650.
- [34] Oseledets I. V., Savostianov D. V., Tyrtyshnikov E. E. Tucker dimensionality reduction of three-dimensional arrays in linear time // *SIMAX*, принято к публикации.

- [35] Oseledets I. V., Savostianov D. V., Tyrtyshnikov E. E.
Fast simultaneous orthogonal reduction to triangular matrices.
// SIMAX, принято к публикации.
- [36] Савостьянов Д. В. Быстрая полилинейная аппроксимация матриц и интегральные уравнения Дисс. на степ. к.ф.-м.н. — ИВМ РАН, Москва, 2006.

Об использовании мозаично-скелетных аппроксимаций при решении гиперсингулярных интегральных уравнений[§]

Д. В. Савостьянов[®], С. Л. Ставцев[®], Е. Е. Тыртышников[®]

1. Введение

При решении задач дифракции звуковых волн на поверхности сложной формы волновыми методами возникает проблема нахождения решения с увеличением частоты излучения. В статье для получения решения краевой задачи Неймана для скалярного уравнения Гельмгольца с помощью теории потенциала задача сводится к решению интегральных уравнений. Для получения численного решения интегрального уравнения требуется заполнять матрицы большого объема и решать соответствующие СЛАУ. В статье показаны результаты применения мозаично-скелетонного метода к аппроксимации таких матриц. В том числе показана эффективность предлагаемых методов, даны численные оценки роста мозаичного ранга с увеличением частоты излучения.

Применение мозаично-скелетонных методов позволило провести более детальный анализ решения интегральных уравнений. Например, получить решение, когда источник расположен близко к поверхности дифрагируемого тела.

Задача с источником, расположенным близко к поверхности, возникает при моделировании распространения звука от вентиляционного оборудования, которые обычно располагаются на крыше зданий.

[§]Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (грант РФФИ №05-01-00721а) и программы фундаментальных исследований отделения математических наук РАН «Вычислительные и информационные проблемы решения больших задач» по проекту «Матричные методы и технологии для задач со сверхбольшим числом неизвестных»

[®]Институт вычислительной математики РАН

Поэтому в статье более подробно показано решение задачи с источником, расположенным близко к поверхности. В том числе для возможности повторения сделанных преобразований показано более детальное вычисление гиперсингулярного интеграла с весовой функцией. Для применения получаемых формул также использовались мозаично-скелетонные аппроксимации.

2. Постановка задачи

Рассмотрим решение внешней краевой задачи Неймана на поверхности σ

$$\frac{\partial \varphi(M)}{\partial n_M} = -\frac{\partial \varphi_0(M)}{\partial n_M}, \quad M \in \sigma \quad (1)$$

для скалярного уравнения Гельмгольца:

$$\Delta \varphi(M) + \kappa^2 \varphi(M) = 0, \quad M \in D, \quad (2)$$

где $\kappa = \frac{2\pi\nu}{c}$ — волновое число, ν — частота источника. В (1) \vec{n}_M единичный вектор нормали к поверхности σ , направленный в область D . Область D является неограниченной.

Поверхность σ задается уравнением

$$\psi = \psi(u, v), \quad (u, v) \in \mathfrak{R}^2, \quad \psi \in C^1. \quad (3)$$

Рассмотрим случай, когда потенциал $\varphi_0(M)$ представим в виде:

$$\varphi_0(M) = \sum_{i=1}^m \frac{Q_i}{4\pi} \frac{\exp(i\kappa r_{MM_i})}{r_{MM_i}}, \quad M_i \in D, i = 1, \dots, m \quad (4)$$

Таким образом, по формуле (4) рассчитывается потенциал от системы точечных источников мощностью Q_i расположенных в точках M_i области D .

Так как область D неограничена, то зададим условия излучения на бесконечности — условия Зоммерфельда:

$$\left(\frac{\vec{r}_M}{r_M}, \nabla \varphi(M) \right) - i\kappa \varphi(M) = O\left(\frac{1}{R_{MM_0}}\right), \quad (5)$$

где R_{MM_0} — расстояние от точки M до $M_0 \in \sigma$,
 $r_M = |\vec{r}_M| = |x_M \vec{i} + y_M \vec{j} + z_M \vec{k}| = \sqrt{x_M^2 + y_M^2 + z_M^2}$,
 $\vec{i}, \vec{j}, \vec{k}$ — орты координатных осей.

3. Численный метод решения

Также как в работе [6] решение задачи (1) – (5) будем искать в виде потенциала двойного слоя, распределенного по поверхности σ

$$\varphi(M) = \int_{\sigma} g(N) G(M, N) d\sigma_N, \quad M \in D, \quad M \notin \sigma, \quad (6)$$

где $g(N)$, $N \in \sigma$ — плотность потенциала двойного слоя. Ядро интегрального оператора (6) имеет вид:

$$G(M, N) = \frac{1}{4\pi} \frac{\partial}{\partial n_N} \frac{\exp(i\kappa r_{MN})}{r_{MN}}. \quad (7)$$

Потенциал двойного слоя (6) является решением задачи (1) – (5), если его плотность $g(M)$ $M \in \sigma$ удовлетворяет интегральному уравнению

$$\int_{\sigma} g(N) \frac{\partial G(M, N)}{\partial n_M} d\sigma_N = -\frac{\partial \varphi_0(M)}{\partial n_M}, \quad M \in \sigma. \quad (8)$$

В работе [7] показано, что если поверхность σ является плоскостью $z = 0$, то решением интегрального уравнения (6) является функция

$$g(N) = 2\varphi_0(N), \quad N \in \sigma. \quad (9)$$

Из формул (4) и (9) видно, что решение интегрального уравнения (6) является осциллирующей функцией. Частота осцилляций определяется волновым числом κ . При решении трехмерных задач практики, например, дифракции звуковых волн (см. [6], [7]), дифракции Н-поляризованных электромагнитных волн (см. [3]) возникает необходимость решать интегральные уравнения для $\kappa \sim 0,1 \dots 10$.

Если поверхность σ не является плоскостью, то для решения уравнения (6) будем использовать численный метод дискретных особенностей, описанный в работах [5], [4]. Согласно этому методу поверхность σ равномерно разбивается по параметрам u, v . Для каждой ячейки σ_{ij} , $i, j = 1 \dots n$ функция $g(N)$ считается постоянной. Затем интегральное уравнение (6) записывается в расчетных точках (точках коллокации). В результате задача сводится к решению системы линейных интегральных уравнений (СЛАУ)

$$Ag = b. \quad (10)$$

Согласно методу дискретных особенностей элементы матрицы A вычисляются по формуле:

$$\begin{aligned} a_{ij}(M) = & - \int_{L_{\sigma_{ij}}} \vec{n}_M \cdot (\vec{dl} \times \nabla_N F(M, N)) + \\ & + k^2 \int_{\sigma_{ij}} F(M, N) \vec{n}_M \cdot \vec{n}_N d\sigma, \quad i, j = 1 \dots n, \end{aligned} \quad (11)$$

где $L_{\sigma_{ij}}$ — граница ячейки σ_{ij} ,

$$F(M, N) = \frac{1}{4\pi} \frac{\exp(ikr_{MN})}{r_{MN}}. \quad (12)$$

Очевидно, что точность решения интегрального уравнения определяется шагами разбиения. С другой стороны искомая функция $g(N)$ является осциллирующей и для ее расчета при больших k требуется меньший шаг разбиения. Но с уменьшением шагов разбиения увеличивается размер решаемой системы (10). Таким образом, при больших k возникает необходимость решать СЛАУ (10) больших размерностей.

Матрица A является плотной и при $n > 2 \cdot 10^4$ ее невозможно сохранить в оперативной памяти. Будем использовать метод *неполной крестовой аппроксимации* для того, чтобы приблизить исходную матрицу A матрицей малого ранга (см. [8], [9]).

Согласно этому методу исходная матрица разбивается на блоки и для заданной точности аппроксимации ε каждый блок заменяется матрицей ранга k . В результате для хранения блока $l_1 \times l_2$ требуется не $l_1 \cdot l_2$, а $k(l_1 + l_2)$ чисел. Отметим, что при использовании метода неполной крестовой аппроксимации для каждого блока необходимо вычислить только $O(k(l_1 + l_2))$ вместо $l_1 \cdot l_2$ элементов матрицы. Так как при вычислении элементов матрицы по формулам (11) интегралы берутся численно, то уменьшение количества вычисляемых элементов значительно ускоряет расчет аппроксимирующей матрицы.

4. Результаты расчетов через мозаично-скелетные аппроксимации

Исследуем эффективность этого метода с помощью численных экспериментов.

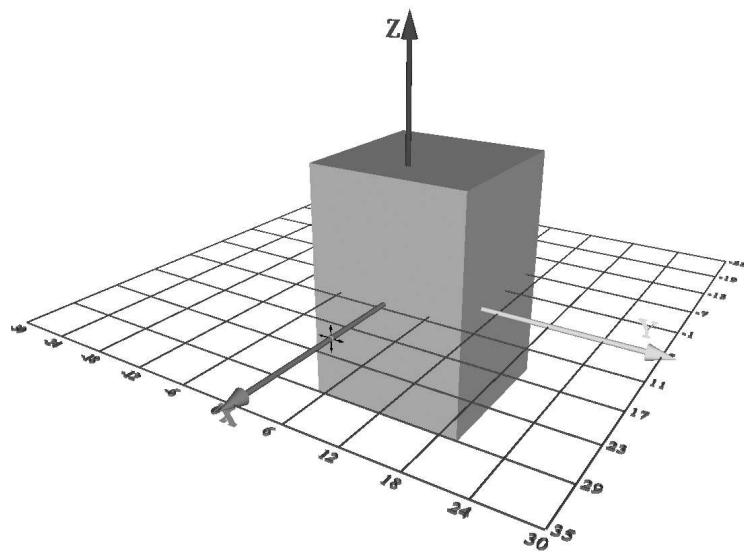


Рис. 1. Пример геометрии

Исследование проведем для решения интегрального уравнения (6), когда поверхность σ является прямоугольным параллелепипедом $20 \times 30 \times 40$. При расчете $\varphi_0(M)$ по формуле (4) положим $m = 1$; $Q_1 = 1$; $M_1(20; 0; 0)$ (см. рисунок 1).

Сравним временные затраты по заполнению матриц различной размерности прямым методом и методом, основанном на мозаично-скелетонных аппроксимациях. Результаты сравнения представлены в таблице 1.

где h — шаг сетки; n — размер матрицы; T_{ms} — время расчета матрицы с помощью мозаично-скелетонного метода; I_{re} — коэффициент сжатия действительной части матрицы; I_{im} — коэффициент сжатия мнимой части матрицы; T_{ms} — время расчета матрицы с помощью прямого метода. В расчетах волновое число k полагалось равным 0,2.

Пусть $mem(\mathfrak{A})$ — объем памяти, необходимый для хранения всех блоков саппроксимированной матрицы, $mem(A)$ — объем памяти, необходимый для хранения несаппроксимированной матрицы, тогда

Таблица 1.

h	n	T_{ms}	I_{re}	I_{im}	T_{st}
10/4	512	26сек.	100	85,7	36сек.
10/8	2048	2мин. 26сек.	44	26,2	10мин. 16сек.
10/16	8192	13мин. 25сек.	14,4	7,34	2час. 38мин. 56сек.
10/32	32768	1ч. 10 мин. 27сек.	4,63	2,19	1д. 17час. 31мин. 44сек.

коэффициент сжатия

$$I = \frac{\text{mem}(\mathcal{A})}{\text{mem}(A)} \cdot 100 \quad (13)$$

определяет эффективность метода по используемой для хранения памяти.

На коэффициент сжатия I влияют параметры решаемой задачи, в том числе волновое число κ . Исследуем зависимость мозаичного ранга $R = \frac{2n^2}{100}$ (см. [8]) от волнового числа κ для задачи, проиллюстрированной на рисунке 1, $n = 8192$. Результаты расчетов представлены в таблице 2.

В работе [2] были даны оценки зависимости мозаичного ранга R матрицы, порожденного интегральным оператором с ядром $\exp ikr/r$ от волнового числа, а именно

$$R < \text{const} (\kappa a - \log_2 \varepsilon)^\alpha \log_2 n, \quad \alpha = 2, \quad (14)$$

Таблица 2.

κ	0,1	0,2	0,3	0,4	0,5	0,6	0,7
R_{Re}	880	940	1000	1080	1170	1250	1350
R_{Im}	368	501	651	785	908	1040	1160
κ	0,8	0,9	1,0	1,1	1,2	1,3	
R_{Re}	1450	1540	1650	1740	1830	1940	
R_{Im}	1280	1410	1510	1620	1730	1820	

где a — диаметр множества, на котором строились аппроксимации (в нашей задаче — это максимальный размер параллелепипеда), ε — точность аппроксимации матрицы ($\varepsilon = 10^{-4}$).

Полагая, что для ядра (7) зависимость $R(\kappa)$ аналогична (14), рассчитаем α . Используя формулу

$$\alpha = \frac{\langle xy \rangle - \langle x \rangle \langle y \rangle}{\langle x^2 \rangle - (\langle x \rangle)^2},$$

где $x = \lg(ka - \log_2 \varepsilon)$, $y = \lg(R)$, символ $\langle \cdot \rangle$ означает среднее значение, получаем $\alpha_{Re} \approx 0,69$, $\alpha_{Im} \approx 1,32$. Таким образом, в оценке (14) значение α завышено, то есть численный эксперимент дает лучший результат, чем теоретические оценки.

С увеличением частоты, а значит и волнового числа κ увеличивается число осцилляций в рассчитываемой плотности g . Для сохранения точности вычислений будем с увеличением κ пропорционально увеличивать число точек разбиения на единицу длины (размер матрицы увеличивается пропорционально квадрату). Будем считать, что на длину волны λ приходится 2π контрольных точек. Результаты представлены в таблице 3.

В таблице 3 T — это время расчета матрицы.

Из таблицы видно, что с ростом волнового числа κ (т.е. с увеличением частоты) индекс сжатия как для действительной так и для мнимой частей матрицы уменьшается, что приводит к уменьшению степени роста времени расчета матрицы с увеличением частоты.

Мозаично-скелетонные аппроксимации позволяют более подробно исследовать задачи дифракции волн когда источник расположен

Таблица 3.

κ	0,4	0,6	0,8	1,2	1,6	2,4
h	10/4	10/6	10/8	10/12	10/16	10/24
n	512	1152	2048	4608	8192	18432
I_{Re}	100	100	75,9	49,8	37,0	24,3
I_{Im}	100	94,9	72,7	47,9	35,4	23,3
T	41с.	3мин. 13с.	7мин. 14с.	24 мин. 21с.	1ч. 10мин. 52с.	3ч. 43мин. 45с.

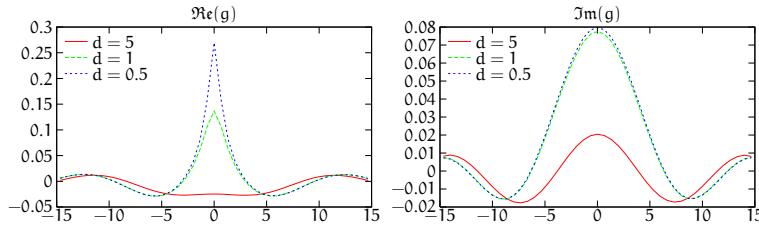


Рис. 2. Изменение функции g при приближении источника к поверхности

близко к поверхности.

В задаче, изображенной на рисунке 1, будем приближать источник к поверхности и рассмотрим как будет изменяться решение (8). Пусть d — кратчайшее расстояние от источника до поверхности. На рисунке 2 изображены изменения вдоль оси OZ действительной (а) и мнимой (б) частей решения в точках пересечения плоскости OXZ с частью поверхности σ — $x = 10$ при $d = 5\text{м}$, $d = 1\text{м}$ и $d = 0,5\text{м}$.

Видим, что с уменьшением d усиливается острота пика функции g в точке (точках) $N^* \in \sigma$ близких к источнику. Поэтому при получения решения $g(N)$ интегрального уравнения (8) требуется либо мелкая сетка, либо неравномерная сетка, сгущающаяся вблизи точек N^* . Это проиллюстрировано на рис. 2.

Однако, если решать систему относительно функции

$$h(N) = g(N) - p_Q(N), \quad N \in \sigma, \quad (15)$$

которая не содержит острых пиков, то решение можно искать на равномерной и достаточно грубой сетке.

В результате из формул (15) и (8) получаем следующее интегральное уравнение:

$$\int_{\sigma} h(N) \frac{\partial G(M, N)}{\partial n_M} d\sigma_N = - \frac{\partial \varphi_0(M)}{\partial n_M} - \\ - \int_{\sigma} p_Q(N) \frac{\partial G(M, N)}{\partial n_M} d\sigma_N, \quad M \in \sigma. \quad (16)$$

Как выбрать функцию $p_Q(N)$!? Напомним, что формула (9) является решением интегрального уравнения (8), когда поверхность

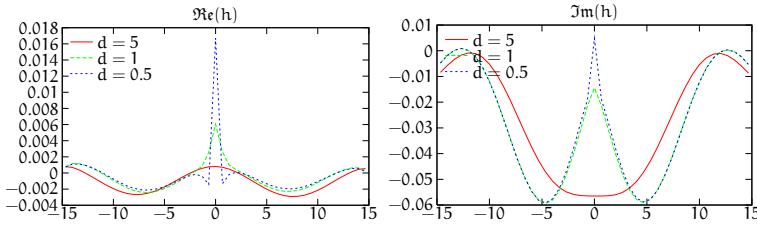


Рис. 3. Графики функции $h(N)$ в точках передней грани куба

σ — плоскость $z = 0$. Поэтому в качестве функции $p_Q(N)$ можно выбрать

$$p_Q(N) = 2\varphi_0(N), \quad N \in \sigma. \quad (17)$$

Обратим внимание на то, что острый пик существует только для действительной части решения $g(N)$. Если в функции $\varphi_0(N)$, вычисляемой по формуле (4), разложим $\exp(i\kappa r_{NM_i})$, $i = 1, \dots, m$ в степенной ряд, то очевидно, что существование пика определяется первыми членами ряда, то есть

$$\varphi_R(N) = \sum_{i=1}^m \frac{Q_i}{4\pi} \frac{1}{r_{NM_i}}, \quad (18)$$

поэтому в качестве $p_Q(M)$ можно взять величину

$$p_Q(N) = 2\varphi_R(N), \quad N \in \sigma. \quad (19)$$

На рисунке 3 приведены графики функции $h(N)$ в точках передней грани куба: на рисунке (a), если $p_Q(N)$ рассчитывается по формуле (17), на рисунке (b) — при расчете $p_Q(N)$ по (19).

На рисунке 3 приведены результаты расчета с использованием функции $g(N)$, которая расчитывалась численно, в ходе интегрального уравнения. Из-за существования пика погрешность вычислений функции $g(N)$, а значит и $h(N)$, вблизи оси OZ получилась большой.

5. Вычисление гиперсингулярного интеграла с вектором

В правой части интегрального уравнения (16) стоит гиперсингулярный интеграл

$$I(M) = \frac{1}{4\pi} \int_{\sigma} p_Q(N) \frac{\partial}{\partial n_M} \frac{\partial F(M, N)}{\partial n_N} d\sigma_N, \quad (20)$$

где $F(M, N)$, рассчитываемое по формуле (12), является фундаментальным решением уравнения Гельмгольца (2).

Представим интеграл (20) в виде

$$I(M) = \vec{n}_M \cdot \vec{\Phi}(M), \quad (21)$$

где $\vec{\Phi}(M) = \Phi_x(M)\vec{i} + \Phi_y(M)\vec{j} + \Phi_z(M)\vec{k}$, $\vec{n}_M = \cos \alpha_M \vec{i} + \cos \beta_M \vec{j} + \cos \gamma_M \vec{k}$. Здесь $\cos \alpha_M$, $\cos \beta_M$, $\cos \gamma_M$ — направляющие косинусы вектора \vec{n}_M , а также введены обозначения

$$\begin{aligned} \Phi_x(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_M \partial n_N} d\sigma_N; \\ \Phi_y(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial y_M \partial n_N} d\sigma_N; \\ \Phi_z(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial z_M \partial n_N} d\sigma_N. \end{aligned} \quad (22)$$

Пусть $\vec{n}_N = \cos \alpha \vec{i} + \cos \beta \vec{j} + \cos \gamma \vec{k}$, где $\cos \alpha$, $\cos \beta$, $\cos \gamma$ — направляющие косинусы вектора \vec{n}_N . Из определения функции $F(M, N)$ имеем $\nabla_M F(M, N) = -\nabla_N F(M, N)$. $\Phi_x(M) = -\int_{\sigma} p_Q(N) \times \cos \alpha \frac{\partial^2}{\partial x_N^2} d\sigma_N - \int_{\sigma} p_Q(N) \cos \beta \frac{\partial^2}{\partial x_N \partial y_N} d\sigma_N - \int_{\sigma} p_Q(N) \cos \gamma \frac{\partial^2}{\partial x_N \partial z_N} d\sigma_N$. Представим $\Phi_x(M)$ в следующем виде

$$\Phi_x(M) = -I_1(M) - I_2(M) - I_3(M), \quad (23)$$

где через $I_1(M)$, $I_2(M)$, $I_3(M)$ обозначим следующие интегралы

$$\begin{aligned} I_1(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N^2} \cos \alpha d\sigma_N; \\ I_2(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N \partial y_N} \cos \beta d\sigma_N; \\ I_3(M) &= \int_{\sigma} p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N \partial z_N} \cos \gamma d\sigma_N. \end{aligned} \quad (24)$$

Так как $F(M, N)$ — фундаментальное решение уравнения Гельмгольца (2), то при $N \neq M$ справедливо следующее соотношение

$$\frac{\partial^2 F(M, N)}{\partial x_N^2} = -\frac{\partial^2 F(M, N)}{\partial y_N^2} - \frac{\partial^2 F(M, N)}{\partial z_N^2} - \kappa^2 F(M, N)$$

или после умножения правой и левой частей на $p_Q(N)$, приходим к

$$\begin{aligned} p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N^2} &= -p_Q(N) \frac{\partial^2 F(M, N)}{\partial y_N^2} - \\ &- p_Q(N) \frac{\partial^2 F(M, N)}{\partial z_N^2} - \kappa^2 p_Q(N) F(M, N). \end{aligned} \quad (25)$$

Так как для дважды дифференцируемых функций $A(x, y)$ и $B(x, y)$ справедливы соотношения

$$\begin{aligned} A(x, y) \frac{\partial B(x, y)}{\partial x^2} &= \frac{\partial}{\partial x} \left(A(x, y) \frac{\partial B(x, y)}{\partial x} \right) - \frac{\partial A(x, y)}{\partial x} \frac{\partial B(x, y)}{\partial x}; \\ A(x, y) \frac{\partial B(x, y)}{\partial x \partial y} &= \frac{\partial}{\partial x} \left(A(x, y) \frac{\partial B(x, y)}{\partial y} \right) - \frac{\partial A(x, y)}{\partial x} \frac{\partial B(x, y)}{\partial y}. \end{aligned} \quad (26)$$

Применяя первое равенство (26) для $A(x, y) = p_Q(N)$ и $B(x, y) = F(M, N)$, и заменяя $\frac{\partial}{\partial y_N}$, приходим к

$$\begin{aligned} p_Q(N) \frac{\partial^2 F(M, N)}{\partial y_N^2} &= \frac{\partial}{\partial y_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) - \\ &- \frac{\partial p_Q(N)}{\partial y_N} \frac{\partial F(M, N)}{\partial y_N}, \end{aligned} \quad (27)$$

а если заменить $\frac{\partial}{\partial x}$ на $\frac{\partial}{\partial z_N}$, то

$$p_Q(N) \frac{\partial^2 F(M, N)}{\partial z_N^2} = \frac{\partial}{\partial z_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) - \frac{\partial p_Q(N)}{\partial z_N} \frac{\partial F(M, N)}{\partial z_N}. \quad (28)$$

Подставим (27) и (28) в (25) и тогда для интеграла $I_1(M)$ из (24) будет справедливо следующее соотношение

$$\begin{aligned} I_1(M) = & \int_{\sigma} \left[\frac{\partial p_Q(N)}{\partial y_N} \frac{\partial F(M, N)}{\partial y_N} + \frac{\partial p_Q(N)}{\partial z_N} \frac{\partial F(M, N)}{\partial z_N} - \right. \\ & - \kappa^2 p_Q(N) F(M, N) \left. \right] \cos \alpha d\sigma - \int_{\sigma} \left[\frac{\partial}{\partial y_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) - \right. \\ & \left. - \frac{\partial}{\partial z_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) \right] \cos \alpha d\sigma. \end{aligned}$$

Применим вторую формулу из (26) к $p_Q \frac{\partial^2 F(M, N)}{\partial x_N \partial y_N}$, полагая $A(x, y) = p_Q(N)$ и $B(x, y) = F(M, N)$, и заменяя $\frac{\partial}{\partial x}$ на $\frac{\partial}{\partial x_N}$, $\frac{\partial}{\partial y}$ на $\frac{\partial}{\partial y_N}$ приходим к

$$p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N \partial y_N} = \frac{\partial}{\partial x_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) - \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial y_N}. \quad (29)$$

Рассуждая аналогичным образом и заменяя во второй формуле (26) $\frac{\partial}{\partial x}$ на $\frac{\partial}{\partial z_N}$, $\frac{\partial}{\partial y}$ на $\frac{\partial}{\partial z_N}$ приходим к

$$p_Q(N) \frac{\partial^2 F(M, N)}{\partial x_N \partial z_N} = \frac{\partial}{\partial x_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) - \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial z_N}. \quad (30)$$

Таким образом, используя формулы (29) и (30) перепишем инте-

гравы $I_2(M)$ и $I_3(M)$:

$$I_2(M) = - \int_{\sigma} \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial y_N} \cos \beta d\sigma_N + \\ + \int_{\sigma} \cos \beta \left[- \frac{\partial}{\partial x_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) \right] d\sigma_N,$$

$$I_3(M) = - \int_{\sigma} \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial z_N} \cos \gamma d\sigma_N + \\ + \int_{\sigma} \cos \gamma \left[- \frac{\partial}{\partial x_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) \right] d\sigma_N.$$

Применяя полученные выражения для $I_1(M)$, $I_2(M)$ и $I_3(M)$, а также (23) запишем $\Phi_x(M)$ в виде

$$\Phi_x(M) = - \int_{\sigma} \left\{ \cos \alpha \left[\frac{\partial}{\partial y_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) - \right. \right. \\ \left. \left. - \frac{\partial}{\partial z_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) \right] + \left[- \frac{\partial}{\partial x_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) \right] \times \right. \\ \times \cos \beta + \cos \gamma \left[\frac{\partial}{\partial x_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) \right] \right\} d\sigma - \int_{\sigma} \left\{ \left[\frac{\partial p_Q(N)}{\partial y_N} \times \right. \right. \\ \times \frac{\partial F(M, N)}{\partial y_N} + \frac{\partial p_Q(N)}{\partial z_N} \frac{\partial F(M, N)}{\partial z_N} - \kappa^2 p_Q(N) F(M, N) \left. \right] \cos \alpha - \\ \left. \left. - \cos \beta \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial y_N} - \cos \gamma \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial z_N} \right] \right\} d\sigma. \quad (31)$$

Для дальнейших рассуждений воспользуемся формулой Стокса [1]:

$$\int_{\sigma} \left[\left(\frac{\partial R}{\partial y} - \frac{\partial Q}{\partial z} \right) \cos \alpha + \left(\frac{\partial P}{\partial y} - \frac{\partial R}{\partial x} \right) \cos \beta + \right. \\ \left. + \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) \cos \gamma \right] d\sigma = \int_{L_{\sigma}} P dx + Q dy + R dz, \quad (32)$$

где L_{σ} — граница поверхности σ . Полагая в формуле (32) $P = 0$, $Q = p_Q(N) \frac{\partial F(M, N)}{\partial z_N}$ и $R = -p_Q(N) \frac{\partial F(M, N)}{\partial y_N}$ получаем следующее

соотношение

$$\int_{\sigma} \left\{ \cos \alpha \left[\frac{\partial}{\partial y_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) - \frac{\partial}{\partial z_N} \left(p_Q(N) \frac{\partial F(M, N)}{\partial z_N} \right) + \right. \right. \\ \left. \left. + \cos \beta \left[-\frac{\partial}{\partial x_N} \left(-p_Q(N) \frac{\partial F(M, N)}{\partial y_N} \right) \right] + \cos \gamma \frac{\partial}{\partial x_N} \left[p_Q(N) \times \right. \right. \right. \\ \left. \left. \left. \times \frac{\partial F(M, N)}{\partial y_N} \right] \right\} d\sigma = \int_{L_{\sigma}} p_Q(N) \left(\frac{\partial F(M, N)}{\partial z_N} dy_N - \frac{\partial F(M, N)}{\partial y_N} dz_N \right).$$

Подставляя это соотношение в выражение (31), получаем

$$\Phi_x(M) = - \int_{L_{\sigma}} p_Q(N) \left(\frac{\partial F(M, N)}{\partial z_N} dy_N - \frac{\partial F(M, N)}{\partial y_N} dz_N \right) - \\ - \int_{\sigma} \left\{ \left[\frac{\partial p_Q(N)}{\partial y_N} \frac{\partial F(M, N)}{\partial y_N} + \frac{\partial p_Q(N)}{\partial z_N} \frac{\partial F(M, N)}{\partial z_N} - \right. \right. \\ \left. \left. - \kappa^2 p_Q(N) F(M, N) \right] \cos \alpha - \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial y_N} \cos \beta - \right. \\ \left. - \frac{\partial p_Q(N)}{\partial x_N} \frac{\partial F(M, N)}{\partial z_N} \cos \gamma \right\} d\sigma. \quad (33)$$

Рассмотрим вычисление $\Phi_x(M)$ по замкнутой поверхности. Разобьем поверхность σ на σ_1 и σ_2 , то есть $\sigma = \sigma_1 \cup \sigma_2$. При интегрировании по σ_1 $\Phi_x(M)$ содержит $\int_{L_{\sigma_1}} f_1(N) dl_{N1}$, а при интегрировании по σ_2 — $\int_{L_{\sigma_2}} f_2(N) dl_{N2}$. В силу наших построений $L_{\sigma_1} = L_{\sigma_2}$, $f_1(N) = f_2(N)$, $dl_{N1} = -dl_{N2}$. Поэтому для замкнутой поверхности σ в (33) $\int_{L_{\sigma}} p_Q(N) \left(\frac{\partial F(M, N)}{\partial z_N} dy_N - \frac{\partial F(M, N)}{\partial y_N} dz_N \right) = 0$.

Из (33) по формулам (26) имеем

$$\frac{\partial p_Q(N)}{\partial y_N} \frac{\partial F(M, N)}{\partial y_N} = \frac{\partial}{\partial y_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial y_N} \right) - \\ - F(M, N) \frac{\partial^2 p_Q(N)}{\partial y_N^2}; \quad (34)$$

$$\frac{\partial p_Q(N)}{\partial z_N} \frac{\partial F(M, N)}{\partial z_N} = \frac{\partial}{\partial z_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial z_N} \right) - \\ - F(M, N) \frac{\partial^2 p_Q(N)}{\partial z_N^2}.$$

К подынтегральной функции в (33) добавим и вычтем

$$\cos \beta \frac{\partial}{\partial x_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial y_N} \right) = F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial y_N} \times \\ \times \cos \beta + \cos \beta \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N}; \quad (35)$$

$$\cos \gamma \frac{\partial}{\partial x_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial z_N} \right) = F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial z_N} \times \\ \times \cos \gamma + \cos \gamma \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N},$$

то получим

$$\Phi_x(M) = \kappa^2 \int p_Q(N) F(M, N) \cos \alpha d\sigma - \int \left\{ \left[- \left(\frac{\partial^2 p_Q(N)}{\partial y_N^2} + \right. \right. \right. \\ \left. \left. \left. + \frac{\partial^2 p_Q(N)}{\partial z_N^2} \right) F(M, N) \right] \cos \alpha + \cos \beta \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial y_N} + \right. \right. \\ \left. \left. + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} \right] + \cos \gamma \left[F(M, N) \times \right. \right. \\ \left. \left. \times \frac{\partial^2 p_Q(N)}{\partial x_N \partial z_N} + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} \right] \right\} d\sigma - \quad (36) \\ - \int \left\{ \cos \alpha \left[\frac{\partial}{\partial y_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial y_N} \right) - \frac{\partial}{\partial z_N} \left(-F(M, N) \times \right. \right. \right. \\ \left. \left. \left. \times \frac{\partial p_Q(N)}{\partial z_N} \right) \right] + \left[- \frac{\partial}{\partial x_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial y_N} \right) \right] \cos \beta + \cos \gamma \times \\ \times \left[\frac{\partial}{\partial x_N} \left(F(M, N) \frac{\partial p_Q(N)}{\partial z_N} \right) \right] \right\} d\sigma.$$

К последнему интегралу применяем формулу Стокса (32) и применяя вышеописанные рассуждения для интеграла по контуру на замкнутой поверхности, получаем, что этот интеграл равен нулю,

тогда

$$\Phi_x(M) = \kappa^2 \int p_Q(N) F(M, N) \cos \alpha d\sigma - \int_{\sigma} \left\{ \left[- \left(\frac{\partial^2 p_Q(N)}{\partial y_N^2} + \frac{\partial^2 p_Q(N)}{\partial z_N^2} \right) F(M, N) \right] \cos \alpha + \cos \beta \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial y_N} + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} \right] + \cos \gamma \left[F(M, N) \times \left(\frac{\partial^2 p_Q(N)}{\partial x_N \partial z_N} + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} \right) \right] \right\} d\sigma. \quad (37)$$

Применим формулу (37) для функции $p_Q(M)$, рассчитываемой по формуле (17) или (19).

Рассмотрим вначале случай, когда $p_Q(M)$ рассчитывается по формуле (17). Для простоты рассуждений ограничимся случаем, когда $m = 1$. В этом случае $p_Q(M)$ удовлетворяет уравнению Гельмгольца (2), тогда

$$-\left(\frac{\partial^2 p_Q(N)}{\partial y_N^2} + \frac{\partial^2 p_Q(N)}{\partial z_N^2} \right) = \frac{\partial^2 p_Q(N)}{\partial x_N^2} + \kappa^2 p_Q(N).$$

В результате подстановки формула (37) принимает вид

$$\Phi_x(M) = \int_{\sigma} \left\{ F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N^2} \cos \alpha + \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial y_N} + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} \right] \cos \beta + \cos \gamma \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial z_N} + \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} \right] \right\} d\sigma. \quad (38)$$

В ходе аналогичных рассуждений получаем

$$\Phi_y(M) = \int_{\sigma} \left\{ F(M, N) \frac{\partial^2 p_Q(N)}{\partial y_N^2} \cos \beta + \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial y_N} + \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} - \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} \right] \cos \alpha + \cos \gamma \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial y_N \partial z_N} + \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial y_N} \right] \right\} d\sigma. \quad (39)$$

$$\begin{aligned} \Phi_z(M) = & \int_{\sigma} \left\{ F(M, N) \frac{\partial^2 p_Q(N)}{\partial z_N^2} \cos \gamma + \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial x_N \partial z_N} + \right. \right. \\ & + \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} - \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} \left. \right] \cos \alpha + \\ & + \cos \beta \left[F(M, N) \frac{\partial^2 p_Q(N)}{\partial y_N \partial z_N} + \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial y_N} - \right. \\ & \left. \left. - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial z_N} \right] \right\} d\sigma. \end{aligned} \quad (40)$$

Подставим (38)–(40) в (21), тогда в $I(M)$ будет интеграл

$$\begin{aligned} & \int_{\sigma} \left\{ \cos \alpha_M \cos \beta \left[\frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} \right] + \right. \\ & + \cos \alpha_M \cos \gamma \left[\frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} \right] + \\ & + \cos \alpha \cos \beta_M \left[\frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial x_N} - \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial y_N} \right] + \\ & + \cos \beta_M \cos \gamma \left[\frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial z_N} - \frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial y_N} \right] + \\ & + \cos \alpha \cos \gamma_M \left[\frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial x_N} - \frac{\partial F(M, N)}{\partial x_N} \frac{\partial p_Q(N)}{\partial z_N} \right] + \\ & \left. + \cos \beta \cos \gamma_M \left[\frac{\partial F(M, N)}{\partial z_N} \frac{\partial p_Q(N)}{\partial y_N} - \frac{\partial F(M, N)}{\partial y_N} \frac{\partial p_Q(N)}{\partial z_N} \right] \right\} d\sigma, \end{aligned}$$

который в силу выполнения соотношений $\nabla_N F(M, N) = -\nabla_M F(M, N)$, $\nabla_N F(M, N) = -\nabla_{M_1} p_Q(N)$ обращается в ноль, и тогда интеграл (20) равен

$$I(M) = \int_{\sigma} F(M, N) \frac{\partial}{\partial n_N} (\vec{n}_M \cdot \nabla_{M_1} p_Q(N)) d\sigma \quad (41)$$

или для m источников

$$I(M) = \sum_{i=1}^m \int_{\sigma} F(M, N) \frac{\partial}{\partial n_N} (\vec{n}_M \cdot \nabla_{M_i} p_Q(N)) d\sigma. \quad (42)$$

Теперь пусть $p_Q(M)$ рассчитывается по формуле (19). В этом случае $p_Q(M)$ удовлетворяет уравнению Лапласа, тогда

$$-\left(\frac{\partial^2 p_Q(N)}{\partial y_N^2} + \frac{\partial^2 p_Q(N)}{\partial z_N^2} \right) = \frac{\partial^2 p_Q(N)}{\partial x_N^2}.$$

В ходе рассуждений аналогичных вышеприведенным получаем формулу

$$\begin{aligned} I(M) = \kappa^2 \int_{\sigma} p_Q(N) F(M, N) \vec{n}_M \cdot \vec{n}_M d\sigma + \\ + \sum_{i=1}^m \int_{\sigma} F(M, N) \frac{\partial}{\partial n_N} \left(\vec{n}_M \cdot \nabla_{M_i} p_Q(N) \right) d\sigma. \end{aligned} \quad (43)$$

Так как точки M_i , $i = 1, 2, \dots, m$ не лежат на поверхности σ , то интегралы (42) и (43) не являются гиперсингулярными. Эти интегралы имеют устранимую особенность при $M \rightarrow N$, которая исчезает при переходе к системе отсчета на поверхности σ аналогичной полярной с полюсом в точке M .

Для численного расчета интеграла $I(M)$ применяется квадратурная формула (11). При этом для каждой ячейки постоянной считается функция $\exp(i\kappa r_{MN})$, а интеграл для оставшегося множителя вычисляется аналитически. При вычислении интеграла (17) или (19) также применяются мозаично-скелетный аппроксимации.

Список литературы

- [1] Вайникко Г. М. Лифанов И. К., Полтавский Л. Н. Численные методы в гиперсингулярных интегральных уравнениях и их приложения. М. Янус-К, 2001, 508с.
- [2] Горейнов С. А. Мозаично-скелетонные аппроксимации матриц, порожденных асимптотически гладкими и осцилляционными ядрами. // Сб.: Матричные методы и вычисления. 1999. С. 42-76.
- [3] Дмитриев В.И., Захаров Е.В. Интегральные уравнения в краевых задачах электродинамики. МГУ. 1987г. 167с.
- [4] Довгий С. А., Лифанов И. К. Методы решения интегральных уравнений. Киев, Наукова думка, 2002, 344с.
- [5] Лифанов И.К. Метод сингулярных интегральных уравнений и численный эксперимент. М. ТОО Янус. 1995, 520с.

- [6] Лифанов И.К., Ставцев С.Л. Интегральные уравнения и распространение звука в мелком море. // *Дифференциальные уравнения*. 2004. Т. 40, №9. С. 1256-1270.
- [7] Ставцев С.Л. Итерационный подход к численному решению системы интегральных уравнений для краевых задач скалярного уравнения Гельмгольца. // *Дифференциальные уравнения*. 2006. Том 42, №9. С. 1282-1290.
- [8] Tyrtyshnikov E. E. Mosaic-skeleton approximations // *Calcolo*. 1996. V. 33(1-2). P. 47-57.
- [9] Tyrtyshnikov E.E. Incomplete cross approximation in the mosaic-skeleton method // *Computing*. 2000. V. 64(4). P. 367-380.

Эффективная работа в распределенных вычислительных средах

С. И. СОВОЛЕВ*

Статья посвящена анализу факторов, влияющих на эффективность проведения расчетов в распределенных межкомпьютерных средах, построенных на основе программного комплекса X-Com/VMC. Приводятся описания типичных ситуаций, ведущих к снижению производительности распределенной среды, и рекомендации по их устранению. Обсуждаются направления развития программного комплекса X-Com/VMC, нацеленные на повышение эффективности его работы.

1. Введение

Эффективность — одно из ключевых понятий, с которым сталкивается пользователь при работе на любой высокопроизводительной вычислительной системе. Суперкомпьютер может обладать высокой пиковой производительностью, однако реальные прикладные программы крайне редко способны полноценно задействовать все его возможности. Как правило, эффективность решения той или иной задачи определяется соответствием ее внутренней структуры архитектуре высокопроизводительной системы. В качестве примера можно отметить исследование [1], проведенное на основе данных списка Тор500 [2]. Согласно приведенным оценкам, эффективность представленных в списке вычислительных систем зависит в первую очередь от их архитектуры и слабо коррелирует с пиковой производительностью. При этом рассматривается только выполнение теста Linpack, на другом классе задач оценки уже могут быть принципиально иными.

*Научно-исследовательский вычислительный центр МГУ

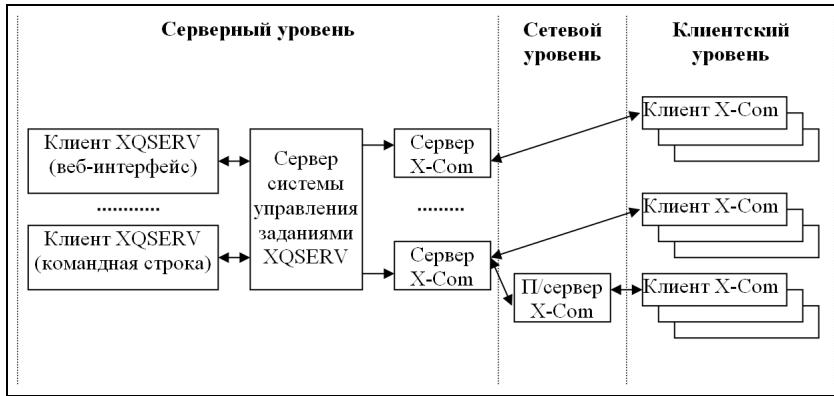


Рис. 1. Архитектура программного комплекса X-Com/VMC

Повышение эффективности выполнения конкретного приложения в конкретной вычислительной среде — отдельная и, как правило, весьма нетривиальная задача. При переходе же к распределенным вычислительным средам сложность этой задачи многократно возрастает. Приходится принимать во внимание не только характеристики всего множества вычислительных ресурсов, на которых будет выполняться приложение, но и особенности организации самой среды: ее топологию, надежность и пропускную способность каналов связи, изменчивость состава компонентов среды, накладные расходы, вносимые программными компонентами системы организации распределенных вычислений. Данная статья посвящена обсуждению различных факторов, влияющих на эффективность расчетов в распределенных средах, выявленных в ходе разработки и практического применения программного комплекса X-Com/VMC, а также методам повышения эффективности метакомпьютерных расчетов.

При работе в метакомпьютерной среде, организованной с помощью комплекса X-Com/VMC (рис. 1), любое задание пользователя последовательно проходит три уровня: серверный, сетевой и клиентский. Серверный уровень определяет политику использования доступных ресурсов, сетевой уровень отвечает за топологию среды и транспортировку всех необходимых данных между серверным уровнем и вычислительными узлами, а клиентский уровень обеспечивает

взаимодействие с прикладной программой на узлах. Рассмотрим основные факторы, влияющие на эффективность расчета на каждом из уровней.

2. Распределение заданий на серверном уровне

Механизмы серверного уровня определяют множество вычислительных ресурсов, на которых будут производиться вычисления, а также порядок выполнения заданий в метакомпьютерной среде. Распределение заданий на доступные ресурсы в X-Com/VMC производится с помощью одного из двух методов: однопоточного либо многопоточного. Однопоточный метод реализует основную идею метакомпьютерных вычислений, а именно использование максимального объема ресурсов для решения задачи. В этом случае каждое задание, поступающее на вход X-Com/VMC, последовательно запускается на всех доступных узлах. Этот метод позволяет достичь максимальной суммарной производительности подключенных ресурсов. Наибольшая эффективность при использовании однопоточного метода достигается в тех случаях, когда прикладная задача разбита на достаточно большое число вычислительных блоков-порций, при этом время обработки каждой порции невелико по сравнению с общим временем проведения расчета. При такой организации вычислений каждый узел среды вносит свой вклад в расчет, обработав какое-то количество порций.

Однако достижение максимальной суммарной производительности ресурсов вовсе не гарантирует оптимальность их использования. Предположим, что во время расчета вычислительная часть прикладной программы достаточно долго обрабатывает каждую порцию входных данных, при этом время обработки порции существенно зависит от тактовой частоты процессора, а среда объединяет узлы как с высокой, так и низкой частотой CPU. В этом случае вполне возможен вариант, при котором слабые узлы, получив свои порции в начале расчета, не закончат их обработку до момента завершения всего расчета. Подключение таких узлов для данного расчета окажется нецелесообразным; в то же время, их вполне можно было бы использовать, например, для решения относительно небольших задач либо для тестовых запусков других приложений.

Подобную функциональность реализует многопоточный метод

распределения заданий. Идея этого метода состоит в динамическом перераспределении доступных ресурсов для решения тех задач, требованиям которых они соответствуют. Требования прикладной задачи должны указываться пользователями при формировании задания. Это могут быть минимальные/максимальные значения тактовой частоты процессора, его тип, объемы оперативной и дисковой памяти, операционная система и другие. При отсутствии требований задание, как и в однопоточном методе, будет отправлено на все узлы.

Многопоточный метод позволяет разбивать всю среду на сегменты по заданным признакам, при этом каждый сегмент будет работать над своей задачей. Данный метод является более универсальным, однако при его использовании могут возникнуть проблемы уже другого характера. Запуск большой серии заданий, затребовавших для себя самые мощные ресурсы вычислительной среды, очевидно, исключит эти ресурсы из работы над другими задачами, а множество незадействованных узлов может уже не представлять интереса для практической работы. Решение подобных проблем — одно из направлений дальнейшего развития программного комплекса X-Com/VMC. Вариантом решения может быть введение в систему механизмов авторизации пользователей и задание административных ограничений на возможность резервирования определенных ресурсов для групп пользователей и их задач. Если распределенная среда используется для решения только одной прикладной задачи с различными наборами входных данных, то в качестве еще одного варианта решения можно предложить такой режим работы серверных компонентов X-Com/VMC, при котором узлам будут последовательно выдаваться порции всех запущенных в данный момент заданий.

3. Обмен данными между компонентами распределенной среды

Сетевой уровень X-Com/VMC формирует топологию распределенной среды и определяет механизмы обмена данными между серверной частью комплекса и вычислительными узлами. В качестве протокола передачи данных комплекс использует модифицированный протокол HTTP, унаследованный от транспортного уровня системы метакомпьютинга X-Com [3]. Основные причины снижения

эффективности расчетов, возникающие на сетевом уровне — неоптимальная топология среды и накладные расходы, вызываемые интенсивными обменами между серверной частью X-Com/VMC и вычислительными узлами.

Оптимальным с точки зрения эффективности является такой вариант проведения расчетов, при котором интенсивность сетевых обменов между узлами и сервером невелика, а время передачи входных и выходных данных во много раз меньше времени обработки этих данных на узлах. При передаче больших объемов данных существенное влияние начинают оказывать свойства каналов, которыми связан сервер с узлами. Низкая пропускная способность, большие задержки и невысокая надежность каналов могут свести на нет всю отдачу от подключения удаленных ресурсов. В этом случае необходимо изменить параметры прикладной задачи, а то и саму ее структуру таким образом, чтобы минимизировать объемы пересылаемых данных.

Повышение интенсивности взаимодействия узлов с сервером, которое может быть вызвано как подключением достаточно большого числа узлов, так и высокой скоростью обработки порций данных, влечет за собой рост нагрузки на серверные компоненты среды и увеличение накладных расходов. Если все узлы общаются с сервером напрямую, до какого-то момента с этой проблемой можно бороться повышением мощности компьютера, на котором работают серверные компоненты. Однако более целесообразным является введение в среду набора промежуточных серверов, позволяющих буферизировать данные между центральным сервером и частью узлов, и тем самым выровнять нагрузку на центральный сервер.

Промежуточные серверы позволяют организовать среду в виде произвольного иерархического дерева. В корне такого дерева будет располагаться центральный сервер X-Com/VMC, листьями всегда будут вычислительные узлы, а во внутренних вершинах будут находиться промежуточные серверы. Помимо буферизации данных, промежуточные серверы также позволяют подключить к расчетам компьютеры, находящиеся внутри закрытой сети, например, узлы вычислительных кластеров. Таким образом, промежуточные серверы увеличивают масштабируемость распределенных расчетов.

4. Подключение вычислительных узлов

Клиентский уровень X-Com/VMC отвечает за взаимодействие с вычислительной частью прикладной задачи непосредственно на узлах метакомпьютерной среды. Инициативу по запросу задания всегда проявляют сами узлы, что позволяет выбрать наилучший в каждом конкретном случае режим участия узлов в расчетах. Очевидно, что с точки зрения прикладной задачи оптимальным режимом для проведения расчетов является монопольный режим, когда задаче полностью предоставлены все ресурсы узла. Однако на практике узлы в таком режиме работы могут быть выделены задаче не всегда, да и время монопольного использования, скорее всего, будет ограничено — узлы вновь могут понадобиться для выполнения обычных задач. Гораздо чаще приходится применять один из методов совместной работы штатных приложений узла и заданий из распределенной среды: фоновый режим работы с пониженным приоритетом либо работа только в те моменты, когда узлы не загружены другими задачами.

Запуск задачи в фоновом режиме основывается на поддержке механизма приоритетов процессов в современных операционных системах. Программа, запущенная в фоновом режиме с пониженным приоритетом, будет получать доступ к ресурсам компьютера только тогда, когда они не требуются приложениям с более высоким приоритетом. В состав программного комплекса X-Com/VMC включен вариант клиента X-Com для ОС семейства MS Windows, реализующий подобную функциональность. Стоит отметить, что применимость данного метода во многом зависит от особенностей той или иной операционной системы и ее настроек на конкретном компьютере, учесть которые крайне сложно.

Более строгим и более «высокоуровневым» методом является запуск распределенного приложения на узле только в те моменты, когда он не занят своей штатной работой. Специальный механизм отслеживает занятость узла; если узел определяется как свободный, он подключается к расчету. Как только штатные процессы узла начинают проявлять свою активность, распределенный расчет на нем останавливается, а все процессы уничтожаются. Различные методы определения незанятости узлов описаны в статье [4].

Понятно, что эффективность использования ресурсов в таком ре-

жиме напрямую зависит от их загруженности. Практически то же самое можно сказать и о фоновом запуске. Основная разница между этими двумя методами заключается в том, что при появлении активности на узлах распределенное приложение либо приостанавливается (фоновый режим с пониженным приоритетом), либо полностью прекращается (вычисления в моменты простоя). В первом варианте приложение остается на узле, оно может оказывать влияние на другие процессы узла и мешать их работе. Во втором случае потребуется произвести перерасчет той вычислительной порции, над которой работал узел. Фоновый запуск больше ориентирован на подключение к расчетам рабочих станций, запуск в моменты простоя применяется, как правило, при подключении узлов вычислительных кластеров.

Стоит также отметить еще один способ подключения вычислительных ресурсов к распределенным расчетам — использование возможностей штатных систем управления заданиями. Возможные сложности с работой через такие системы связаны с ограничениями, которые обычно накладываются на максимальное время выполнения приложений, запущенных через них. Поэтому при использовании узлов в подобных режимах необходимо, во-первых, постоянно отслеживать и поддерживать постоянное число процессов распределенного приложения (X-Com/VMC содержит подобный механизм, работающий совместно с системой Cleo [5]), а во-вторых, по возможности самостоятельно задавать ожидаемое время обработки заданий. Задание максимально возможного времени не всегда оправдано, т.к. в этом случае система управления заданиями может попытаться пропустить сперва более короткие задачи. Наиболее целесообразно указывать время, кратное среднему времени обработки одной вычислительной порции. Тем не менее, по истечению заданного лимита времени процессы распределенного расчета могут быть прерваны системой управления заданиями, что приведет к необходимости их перерасчета. Очевидно, чем больше порций будет обработано за один прием, тем выше будет эффективность.

5. Прочие факторы

Рассмотрев причины уменьшения эффективности, возникающие на всех уровнях работы программного комплекса X-Com/VMC, стоит также упомянуть еще несколько моментов. Так, очень важным

является вопрос оптимизации самой прикладной программы. Если доступны исходные тексты программы, перед началом расчета имеет смысл провести тестирование ее выполняемого кода, полученного с помощью различных компиляторов, на тех типах программно-аппаратных платформ, которые будут задействованы в расчете. Разница во времени выполнения различных вариантов программного кода может быть весьма значительной. Так, проводя эксперименты с одной из задач электродинамики, мы обнаружили, что версия программы, полученная с помощью компилятора Intel (icc), работает практически в 4 раза быстрее, чем версия, полученная с помощью компилятора GNU (gcc).

При включении в состав распределенной среды многопроцессорных и многоядерных узлов целесообразно запускать по одному вычислительному процессу прикладной программы на каждое процессорное ядро. Однако следует принимать во внимание, что каждый новый расчетный процесс увеличивает расход оперативной памяти компьютера. Кроме того, одновременный запуск одинаковых процессов может привести к замедлению их работы по сравнению с запуском в единственном экземпляре. Причины замедления зависят как от самой прикладной программы, так и от особенностей аппаратной архитектуры вычислительного узла, и могут быть вызваны неэффективным использованием кэш-памяти процессоров, повышением нагрузки на канал процессор-память и системную шину, конфликтами при доступе к устройствам хранения данных. Перед началом реальных расчетов имеет смысл выяснить, будет ли проявляться подобный эффект на доступных узлах, в какой степени он способен повлиять на процесс вычислений в целом и какое количество вычислительных процессов на одном узле обеспечит достаточную эффективность расчета. Для поиска узких мест в прикладной программе может потребоваться ее исследование с помощью специальных отладочных средств.

Принцип централизованного управления распределенными расчетами также вносит свои ограничения. В ходе экспериментов нам удавалось смоделировать ситуацию, когда центральный сервер X-Com/VMC, взаимодействуя с достаточно большим числом вычислительных узлов, с трудомправлялся с обработкой потока входящих и исходящих данных, следствием чего являлось значитель-

ное снижение эффективности модельного расчета. Поэтому одно из дальнейших направлений развития программного комплекса X-Com/VMC — децентрализация, и в частности, распределение нагрузки центрального сервера между несколькими компьютерами. Внедрение такого механизма одновременно позволит повысить и общую надежность комплекса, поскольку в случае отказа одной из серверных машин можно выполнить реконфигурацию серверного пула и продолжить расчет.

6. Заключение

Необходимо еще раз отметить важность абсолютно всех аспектов, влияющих на эффективность расчетов в распределенной среде. Количество задач, решаемых с помощью распределенных технологий, неуклонно растет. Не всегда тривиальным оказывается выделение в задаче независимых блоков, формирование вычислительных порций, организация расчета на ста, пятистах, тысяче компьютеров. Решив эти первичные задачи, можно достигнуть значительной суммарной производительности, подключить множество удаленных узлов, провести эффектный эксперимент — но будет ли он эффективным?

Список литературы

- [1] Воеводин Вл.В. Top500: числом или уменьем// Открытые системы, №10, 2005 г. С.12–15.
(http://www.osp.ru/os/2005/10/380430/_p1.html)
- [2] Top500 Supercomputing Sites, <http://www.top500.org/>.
- [3] Система метакомпьютинга X-Com, <http://x-com.parallel.ru/>.
- [4] Жуматий С.А., Соболев С.И. Оценка загруженности компьютера в различных UNIX-системах (в настоящем сборнике).
- [5] Система управления заданиями Cleo,
<http://parcon.parallel.ru/cleo.html>

Технологический инструментарий для построения систем анализа и преобразования структуры программ

К. С. Стефанов*

В данной статье описывается предлагаемая архитектура технологического инструментария для построения систем анализа структуры программ. Инструментарий предназначен для построения специализированных систем для решения конкретных задач, возникающих при исследовании последовательных программ и их преобразования с целью исполнения на компьютерах с параллельной архитектурой.

Описываются принципы разбиения инструментария на логические уровни: внутреннее представление, зависимые и независимые от языка модули, уровень модулей целевых функций (экспертов) и уровень интерфейсов. Определяются возможные взаимодействия между модулями инструментария, расположеннымными на разных уровнях, с целью локализации зависимости от языка анализируемой программы и ОС выполнения инструментального комплекса. Рассмотрено разделение модулей на модули анализа и преобразований. Рассматриваются отдельные уровни с примерами расположенных на них модулей.

1. Введение

При создании программного обеспечения для вычислительных систем с параллельной архитектурой не только сохраняются все

*Научно-исследовательский вычислительный центр МГУ

трудности традиционного программирования, но и добавляется множество новых, связанных с организацией параллельного выполнения программы.

Ручное распараллеливание последовательных программ — занятие крайне трудоемкое. Некоторые методы автоматического распараллеливания реализуются в компиляторах для параллельных вычислительных систем. Отличительная особенность всех этих методов — они должны работать быстро, поэтому не могут быть сложными. В работе [1] проанализированы методы выявления параллелизма, заложенные в компиляторы для параллельных систем начала 90-х годов прошлого века. Оказалось, что сфера применения каждого из методов очень узка. В целом же при анализе зависимостей различных операторов компиляторам удается лишь в 15% случаев без трудностей определить характеристики этих зависимостей. Несмотря на то, что за прошедшие 10 лет появилось много новых способов распараллеливания вычислений, в общем ситуация изменилась не сильно.

Для облегчения создания параллельного программного обеспечения создаются системы автономного анализа и преобразования программ. На данный момент существует некоторое количество проектов, нацеленных на облегчение создания параллельного ПО. Это семейства SUIF, KAP, пакеты FORGE, Polaris, CAPO, система ParaWise и другие. Часть этих систем является автоматическими препроцессорами, другие имеют возможности по интерактивному просмотру результатов анализа и получению подсказок по преобразованию. Некоторые рассчитаны только на машины с общей памятью, и не имеют функциональности для работы с распределением данных. В пакете FORGE реализовано распараллеливание под компьютеры с распределенной памятью, но с довольно жесткими ограничениями на используемое распределение данных и возможную конфигурацию параллельных циклов.

При этом исходный код большинства этих систем недоступен, и использование их функций для решения других задач невозможно.

В настоящее время в НИВЦ МГУ ведутся работы по созданию технологического инструментария, который позволил бы конструировать системы анализа и преобразования программ с необходимой функциональностью на основе готовых блоков.

2. Принципы построения технологического инструментария

В основу инструментария закладываются три принципа: локализация зависимостей от конкретного языка анализируемой программы, модульность инструментария и гибкость архитектуры.

Под гибкостью мы понимаем возможность последующего построения на основе инструментария специализированных систем и конверторов, предназначенных для решения максимально широкого спектра задач, которые возникают в процессе адаптации программ под параллельные вычислительные системы: определение потенциального параллелизма программ, выделение параллельных участков кода, преобразование программ под конкретную систему параллельного программирования и т.п. Разбиение на модули осуществляется так, чтобы иметь возможность из уже написанных модулей технологического инструментария конструировать варианты новых систем для разных задач. Например, это может быть препроцессор, цель которого в полностью автоматическом режиме выявить весь имеющийся в программе параллелизм и записать его в виде комментариев специального вида. Или наоборот, это может быть система с графическим интерфейсом, показывающая пользователю структуру исследуемой программы и позволяющая ему самому провести необходимые преобразования.

Инструментарий сразу проектируется с расчетом на будущее добавление анализируемых языков процедурного типа. Именно поэтому отделяются модули, которые потребуется доработать при переходе к анализу программ на других языках.

Модульность не только облегчает построение специализированных систем с нужной функциональностью, но и позволяет использовать в качестве модулей блоки, взятые из других систем, а также внешние программы, реализовав интерфейс с ними в виде модуля-оболочки инструментария.

3. Общая структура инструментария

Все блоки системы логически разделяются на уровни (рис. 1): *внутреннее представление, зависящее от языка* (входной программы) блоки, *независимые от языка* блоки, *эксперты и интер-*

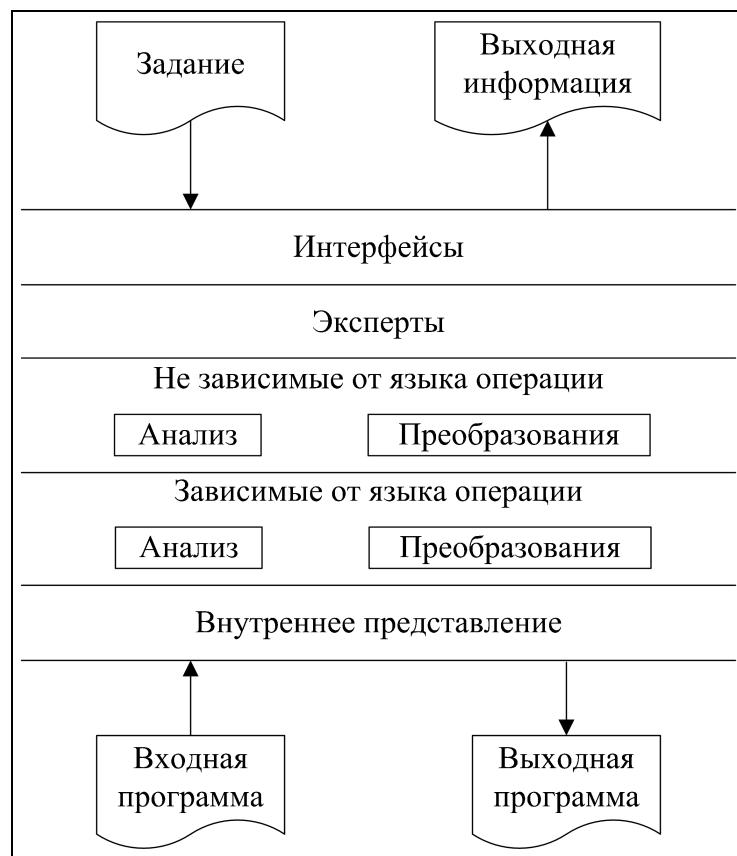


Рис. 1. Логические уровни блоков инструментального комплекса

фейсы.

Деление на уровни определяет внутреннюю структуру комплекса и ограничивает возможные взаимодействия блоков. С блоками внутреннего представления могут взаимодействовать только блоки, зависимые от языка. Интерфейс, предоставляемый зависимыми от языка блоками верхним уровням, скрывает особенности конкретных языков и детали внутреннего представления программ.

Уровни зависимых и независимых от языка операций дополнительно делятся на блоки определения свойств (анализ) и блоки пре-

образований.

Эксперт — это блок, имеющий целевую функцию, нужную пользователю системы. Например, эксперт может выявлять все потенциально параллельные циклы программы и отражать эту информацию в комментариях, или осуществлять преобразование программы под конкретную систему параллельного программирования.

Интерфейс предоставляет доступ к функциям системы внешним пользователям, в роли которых могут быть и другие программы. Все взаимодействие комплекса или созданной на его основе системы с внешним миром — с пользователем либо с другими программами — происходит через интерфейсы.

4. Уровень внутреннего представления

Уровень внутреннего представления составляют блоки, отвечающие непосредственно за работу с текстом программы (ее разбор или генерацию нового текста) и за действия со структурами данных, составляющих представление разобранной программы.

Блоки этого уровня в самой большой степени зависят от языка анализируемой программы. Именно на этом уровне осуществляется разбор текста входной программы. Структуры данных для разных входных языков могут сильно отличаться, однако интерфейс, предоставляемый блокам верхнего уровня, должен быть выражен в терминах описания языков. Поскольку мы ориентируемся прежде всего на процедурные языки, эти термины должны быть похожи для разных языков, естественно, в той степени, в какой похожи языки программирования процедурного типа.

Интерфейс блоков уровня внутреннего представления позволяет производить различные действия с объектами программы такими как операторы, выражения, переменные, программные единицы, файлы программы (если входной язык предусматривает деление программы на файлы).

Например, в Си есть один тип программной единицы — функция, в Фортране-77 типов программных единиц несколько: процедура, функция, главная программа и т.д., а в Фортране-90 добавляется еще модуль. В Си есть глобальные переменные, а в Фортране-77 связь между программными единицами по общим данным осуществляется при помощи COMMON-блоков. Интерфейс блоков внутрен-

него представления должен отражать эти и другие особенности каждого анализируемого языка.

Возможности по генерации программ должны давать возможность генерировать выходную программу из любых конструкций. При этом корректность генерируемой программы на данном уровне может проверяться лишь частично, оставляя такую проверку зависимым от языка модулям преобразований.

5. Блоки анализа и преобразований

Уровни зависимых и независимых от языка блоков содержат основные подсистемы комплекса — анализ и преобразование программ, и предоставляют функциональность, необходимую для построения экспертов.

Блоки анализа служат для определения свойств входной программы, которые затем могут быть использованы другими блоками для выполнения своих функций. Это могут быть свойства как отдельных объектов программы (является ли данный оператор оператором перехода, имеет ли данная переменная составной тип), так и свойства фрагментов программы (принадлежит ли данный фрагмент программы к линейному классу [2]).

Блоки анализа программ на выходе дают объекты-свойства, содержащие информацию о входной программе. Это могут быть элементарные объекты логического типа, описывающие наличие или отсутствие в анализируемой программе какого-либо признака, например, возможность распараллеливания конкретного цикла. Так же это могут быть сложные объекты, например, граф вызовов программы.

Программный интерфейс для работы с такими объектами должен быть определен либо в терминах анализируемых языков, если объект-свойство существует на уровне между внутренним представлением и зависимыми от языка блоками, либо в терминах, общих для анализируемых языков, если такой объект существует на уровнях выше уровня зависимых от языка блоков.

Блоки преобразований служат для выполнения базовых преобразований программы, которые могут быть использованы в качестве составных частей другими блоками преобразований или модулями-экспертами. Отнесение того или иного преобразования к уровню

зависимых или независимых от языка блоков либо же включение его как составной части в какой-нибудь эксперт определяется тем, может ли данное преобразование использовано другим экспертом. Многие преобразования широко используются при различных стратегиях распараллеливания под разные вычислительные системы. В то же время существуют и преобразования, использование которых вряд ли может выходить за пределы достаточно узкой области. Например, при преобразовании программы под систему параллельного программирования OpenMP имеет смысл определить преобразование «добавление директивы OpenMP», опирающейся на добавление комментария или директивы `#pragma`. Использование комментариев для задания директив распараллеливания в программах на Фортране применяется не только в OpenMP, а также может применяться и для записи в тексте программы каких-либо найденных ее свойств. Поэтому добавление комментария целесообразно сделать модулем преобразования.

Деление на зависимые и независимые от языка определяется возможностью определить свойство или преобразование в терминах, общих для всех анализируемых языков. Подробнее про это деление будет сказано ниже.

6. Зависимые от языка блоки

Зависимые от языка блоки — это блоки определения свойств или проведения преобразований, которым нужно знать специфику языка, на котором написана анализируемая или преобразуемая программа. Эти блоки работают через интерфейс, предоставляемый блоками уровня внутреннего представления.

Каждый такой блок должен быть повторен для каждого анализируемого входного языка. При этом интерфейс самих этих блоков, предоставляемый блокам верхних уровней, должен выражаться в терминах, общих для всех анализируемых языков программирования, а выбор блока для конкретного языка должен обеспечиваться самим инструментарием.

Зависимые от языка блоки анализа служат для определения тех свойств программы, для которых нужно знать специфику языка, на котором она написана. Например, для построения графа управления программы нужно уметь определять, в каком порядке передается

управление между операторами. Определение того, какой оператор может быть выполнен после данного, является блоком, реализация которого зависит от языка анализируемой программы. В Фортране есть управляющие операторы GOTO, IF, оператор цикла DO. В языке Си набор управляющих операторов другой: goto, if, switch, break, continue и операторы цикла for, while, do ... while. Наборы управляющих операторов различаются от языка к языку.

Зависимые от языка блоки преобразований, по аналогии с зависимыми от языка блоками анализа, определяют преобразования, специфические для данного языка. Например для добавления новой переменной нужно создать оператор, ее описывающий, и вставить в блок описаний (если входной язык требует описаний переменных). При этом информация о возможных типах переменных, а также о расположении блока описаний, структуре самих описаний и т.п. зависит от языка программирования, на котором записана преобразуемая программа.

Некоторые преобразования программы можно делать более эффективно, если не требовать сохранения корректности программы на промежуточных этапах (разумеется, полное преобразование должно сохранять корректность). Например изменение размерности массива, используемое иногда для приведения программы к линейному виду [3], может привести к тому, что операции доступа к элементам этого массива станут некорректными. Можно разработать последовательность преобразований, включающую введение нового массива и последовательную замену всех обращений к одному массиву на обращения к другому, но проще временно отключить проверку полной корректности получаемой программы после каждого шага преобразования (если такая проверка вообще проводится). Поэтому необходимо иметь возможность отключения некоторых проверок на корректность. Естественно, что ответственность за полную корректность получаемой программы лежит на программисте, реализующем преобразования в таком режиме. При этом синтаксическая корректность получаемой программы гарантируется средствами генерации программы уровня работы с внутренним представлением.

7. Независимые от языка блоки

Независимые от языка блоки анализа определяют те, обычно более «высокоуровневые», свойства, которые можно определить общим для всех языков процедурного типа образом, опираясь на информацию, поставляемую зависимыми от языка блоками. Информация, необходимая для работы данных блоков, выражается в терминах, общих для всех анализируемых языков.

Например построение графа управления процедуры может быть осуществлено опираясь на информацию, какой оператор может быть выполнен после данного. Получив информацию о переходах между операторами, граф управления в терминах отдельных операторов может быть построен без учета особенностей конкретных языков.

Независимые от языка блоки преобразования предоставляют возможности по преобразованию программы, необходимые блокам экспертов. По аналогии с независимыми от языка блоками анализа, обычно на данном уровне проводятся более «высокоуровневые» преобразования чем те, которые проводятся на уровне зависимых от языка преобразований.

Практически все составные преобразования должны находиться на данном уровне. При их проведении нужно опираться на «элементарные» преобразования, проводимые зависимыми от языка блоками преобразований.

В блоках экспертов всегда используется информация, поставляемая блоками независимого от языка уровня. Если эксперту необходима информация от блока зависимого от языка уровня или уровня внутреннего представления, то надо создать дополнительный блок-прослойку, передающий такую информацию наверх. Такая архитектура облегчит решение проблем, возникающих при добавлении новых анализируемых языков.

В некоторых случаях полезно иметь набор блоков, имеющих одинаковый интерфейс и выполняющих одну функцию разными методами. Например, одним из методов, используемых для приведения программы к линейному виду, является априорная подстановка [4]. При этом значение одной переменной заменяется на некоторое выражение. Такая замена может производиться, например, модулями, определяющими используемые переменные или индексные выражения на лету, прозрачно для вызывающих модулей. Таким образом

реализация использования подстановки сводится к изменениям в небольшом количестве модулей анализа и написании модуля уровня экспертов, определяющего правила такой подстановки.

8. Уровень модулей целевых преобразований (экспертов)

Блоки, реализующие целевую функциональность системы или конвертора, расположены на уровне экспертов. Эксперт — модуль системы, реализующий конкретную нужную пользователю функцию в автоматическом или полуавтоматическом режиме, например, распараллеливающий и преобразующий программу под конкретную систему параллельного программирования.

В рамках рассматриваемого технологического инструментария вся целевая функциональность системы оформляется модулями уровня экспертов. Эти модули могут быть как весьма простыми, например, вызывающими какой-либо блок анализа для всех операторов программы и таким образом собирающим статистику. Или же эксперт может быть сложным, реализующим множество функций по приведению программы к виду, пригодному для использования в определенной системе параллельного программирования и осуществляющий ее оптимизацию.

Различные модули уровня экспертов могут взаимодействовать с модулями своего уровня или модулями независимого от языка уровня.

В некоторых случаях эксперту может потребоваться дополнительная информация об анализируемой программе помимо содержащейся в ее тексте. Это может быть, например, информация о входных и выходных данных какой-либо подпрограммы или информация о параллельных свойствах какого-либо фрагмента программы, не поддающегося анализу. В этом случае эксперт запрашивает эту информацию у вызывающего его модуля (это может быть как другой эксперт, так и модуль уровня интерфейсов), который и сообщает ему требуемое. При вызове эксперта, которому могут понадобиться дополнительные данные, вызывающий модуль предоставляет информацию, каким образом получать требуемые сведения (или о том, что надо пользоваться значениями по умолчанию).

Таким образом может быть реализована, например, используе-

мая в системе V-Ray схема специальных комментариев [2]. Такие комментарии размещаются в тексте программы и сообщают анализирующей системе о некоторых свойствах этой программы, например, о границах изменений каких-либо переменных. В этом случае эксперт, использующий такую информацию в своей работе, запрашивает ее у вызывающего модуля. Этот модуль может быть другим экспертом, который просмотрит имеющиеся в программе комментарии и сообщит требуемую информацию (либо о ее отсутствии).

9. Уровень интерфейсов

Блоки интерфейсов предназначены для реализации различных способов взаимодействия построенной системы с внешним миром, например, графического интерфейса пользователя или интерфейса командной строки, который может использоваться как человеком, так и другими программами.

У каждой системы, созданной при помощи инструментария, должен быть один основной блок интерфейса. Этот блок должен определить и сообщить другим блокам системы:

- На каком языке написана анализируемая программа и какие файлы ей принадлежат.
- Какие эксперты будут вызываться после загрузки анализируемой программы.
- Какие еще модули интерфейсов будут использоваться.
- Куда будет записан текст сгенерированной программы и другие выходные файлы (если они есть).
- Другую информацию, необходимую вызываемым экспертам.

Как видно, основной блок интерфейса является центром общения системы с внешним миром. При этом информация для каждого из пунктов может быть получена практически независимо. Язык анализируемой программы и имена файлов могут быть получены у пользователя в окне-диалоге, могут быть параметрами командной строки или содержаться в пакетном файле описания задачи.

Другие используемые интерфейсные модули и блоки-эксперты скорее всего будут иметь различные программные интерфейсы, поэтому их использование определяется программным кодом данного модуля интерфейса. Для каких-то конкретных систем может использоваться ровно один эксперт, собирающий какую-нибудь статистику по свойствам анализируемого программного комплекса. В этом случае модуль интерфейса будет достаточно простым без использования других интерфейсов. Или наоборот, у сложной системы с графическим интерфейсом может быть предоставлен выбор по вызываемым экспертам, а для настройки каких-то экспертов могут привлекаться другие интерфейсные модули. Например, в рамках одной системы могут одновременно присутствовать возможности распараллеливания программы и под OpenMP, и под MPI, оформленные в виде разных экспертов. В этом случае основной блок интерфейса должен предоставить пользователю выбор, какой эксперт запускать, а настройки разных экспертов будут производиться соответствующими интерфейсными модулями.

Определение местоположения выходных файлов (сгенерированной или преобразованной программы и других файлов с полученной информацией) может определяться любым из способов, которым определяется расположение входных файлов. Могут быть заданы какие-то умолчания или, например, схема получения имени выходного файла из имени входного (замена суффикса и т.п.).

Как уже отмечалось выше, некоторым экспертам при работе может потребоваться дополнительная информация. Она может позволять провести более точный анализ или иногда указать эксперту про какие-то свойства программы, известные пользователю, но которые не удается получить при анализе текста программы. Если при этом эксперт вызывается напрямую из интерфейса (или через цепочку других экспертов, которым неоткуда взять эту информацию), эксперт делает запрос к вызвавшему его модулю о наличии требуемой информации, передавая сведения о каком объекте программы и что именно он хочет узнать. Интерфейс инициирует диалог с пользователем с запросом соответствующих сведений, если такая возможность предусмотрена, или сообщает, что данных у него нет. Также информация может быть получена не от пользователя а, например, из файла, описывающего задание для системы. Но в этом случае,

особенно если таких сведений может быть много, целесообразно вынести эту информацию в отдельный файл и создать еще один интерфейсный модуль, читающий данный файл и сообщающий сведения запрашивающему модулю.

При построении пользовательского интерфейса к объектам свойствам программы добавляются *объекты-представления* этих свойств. Представление используется блоком интерфейса при выдаче описания свойства пользователю, причем одному свойству может соответствовать множество представлений. Например, свойство параллельности цикла может быть представлено текстовым комментарием к заголовку этого цикла, директивой OpenMP или же разметкой вершины, соответствующей этому циклу на графе управления. В последнем случае мы имеем дело с представлением внутри другого представления.

Блоки, вырабатывающие представление по объекту, являются составными частями интерфейсов. Поскольку одно и то же представление может требоваться в разных интерфейсах, такие блоки представлений выделяются отдельными блоками для возможности повторного использования.

Некоторым представлениям может требоваться дополнительная информация. Например, одним из представлений графа зависимостей программы может служить не его проекция на ось операторов программы, а полное изображение в пространстве итераций. При этом конкретный размер изображения (границы изменений параметров циклов) зависит от значений внешних переменных. В этом случае либо должны быть предусмотрены какие-то разумные значения по умолчанию, либо надо спрашивать конкретные значения у пользователя.

В блоках уровня интерфейсов должна быть локализована зависимость от среды, в которой исполняется система. Именно блоки этого уровня знают о желательном расположении файлов, о вызовах, требуемых для открытия этих файлов. Другим блокам при этом передаются объекты языка программирования, на котором они написаны, представляющие открытый файл (например объект типа *istream* или *ostream* Си++) .

В этих же блоках локализуется зависимость от способа общения с пользователем, будь то интерфейс командной строки или графиче-

ская среда. Существует множество вариантов библиотек для построения графических интерфейсов, и каждая из них имеет некоторый ограниченный набор платформ, на который она может быть задействована. Поэтому локализация зависимости от конкретной библиотеки облегчает перенос полученной системы под другую ОС.

При построении систем анализа крайне важно тестирование корректности их работы. Если набор реализованных функций велик, то ручное тестирование становится весьма трудным, если не невозможным. При этом в процессе разработки необходимо регулярно проводить максимально полное тестирование, чтобы быть уверенным, что вносимые изменения не привели к появлению ошибки. Обычно для этого используются системы автоматического тестирования. В рамках предлагаемой архитектуры для связи с такими системами выделяются специальные модули интерфейсов. Они работают под управлением тестирующей системы, принимая от нее задания, выполняют нужные действия и передают обратно полученные результаты. На основании сравнения полученного и ожидаемого результата система тестирования может делать вывод о корректной работе тестируемой функции и формировать соответствующие отчеты.

10. Примеры блоков инструментария

На рис. 2 приведена схема инструментария с примерами блоков, расположенных на каждом уровне.

На уровне внутреннего представления располагаются блоки, отвечающие за работу со всеми объектами, составляющими программу: операторами, переменными, типами, комментариями и т.п.

На уровне зависимых от языка блоков находятся блоки, опирающиеся на специфику входного языка. Например, для определения зависимостей по данным для каждого оператора необходимо знать, какие переменные читаются, а какие изменяются в данном операторе. За это отвечает вот блок «определение используемых переменных». Для нахождения циклов в программе также имеется блок «определение явных циклов». Этот блок перечисляет все записанные явным образом, т.е. с помощью предусмотренных в языке конструкций, циклы, имеющиеся в программе.

На уровне независимых от языка блоков находятся основные блоки анализа и преобразований. Например, здесь располагаются бло-

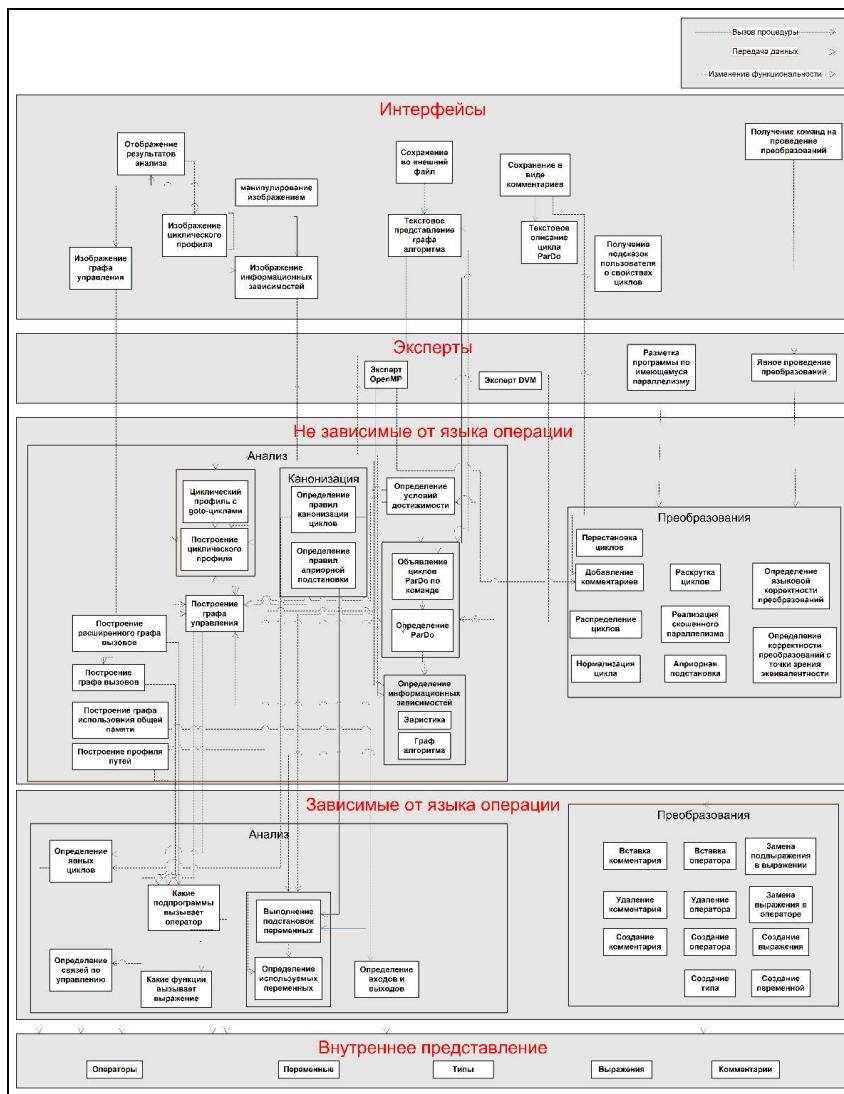


Рис. 2. Логические уровни инструментального комплекса с примерами расположенных на них блоков

ки определения информационных зависимостей. Информационную зависимость можно определять различными методами — точным, основанным на графе алгоритма, или разного рода эвристиками. На рисунке показано два блока для этой задачи. Эти блоки имеют одинаковый интерфейс и могут заменять друг друга. На них опирается блок определения свойств параллельности цикла. Здесь тоже два блока с одинаковым интерфейсом, один из которых определяет параллельность цикла на основе наличия в нем зависимостей, а второй получает от эксперта команду, какие циклы считать независимыми, после чего начинает сообщать об этом вызывающим блокам. Это может понадобиться, например, если цикл не поддается анализу, но из каких-то соображений известно, что он параллельный.

В качестве эксперта может быть модуль, реализующий преобразование последовательной программы в систему программирования OpenMP или DVM. Или же конструируемая система (и соответствующий эксперт) может размечать программу в соответствии с имеющимся в ней потенциальным параллелизмом.

На уровне интерфейсов находятся блоки получения представлений по объектам-свойствам («изображение графа управления», «изображение информационных зависимостей»). Также на этом уровне находятся блоки, обеспечивающие взаимодействие с пользователем, в частности, «получение подсказок о свойствах циклов», «получение команд на проведение преобразований».

Список литературы

- [1] Shen Z., Li Z., Yew P.-C. An Empirical Study of Fortran Programs for Parallelizing Compilers// IEEE Transactions on Parallel and Distributed Systems. 1990. Vol. 1, no. 3. Pp. 356–364.
- [2] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. 608 с.
- [3] Хмелев Д.В. Восстановление линейных индексных выражений для сведения программ к линейному классу// ЖВМ и МФ. 1998. Т. 38, № 3. С. 532–544.
- [4] Воеводин Вл.В. Теория и практика исследования параллелизма последовательных программ// Программирование. 1992. № 3. С. 38–53.

Метод скачков и аппроксимации Паде[§]

Е. Е. Тыртышников[®]

Предложены простые и полные доказательства основных фактов алгебраической теории Паде на основе «метода скачков», возникшего при изучении ведущих подматриц ганкелевой матрицы.

Алгебраическая теория аппроксимаций Паде является по сути теорией подматриц бесконечной ганкелевой матрицы. Основные факты этой теории формулируются элегантно и просто. С ними можно познакомиться, например, по книге [1]. Однако приведенные там доказательства не только сложны, но и в некоторой степениdezориентируют читателя. Цель этой заметки — дать полное, краткое и ясное изложение теории Паде на основе «метода скачков», возникшего при изучении ведущих подматриц ганкелевой матрицы [2, 6]. В качестве основного результата, помимо некоторого развития самого метода скачков, нужно рассматривать новые доказательства известных утверждений алгебраической теории Паде.

1. Ряды и матрицы

Пусть задан формальный ряд

$$f(x) = \sum_{i=0}^{\infty} a_i x^i.$$

Его аппроксимацией Паде типа (m, n) называется пара многочленов

$$u(x) = \sum_{i=0}^m u_i x^i, \quad v(x) = \sum_{i=0}^n v_i x^i$$

[§]Работа выполнена при поддержке грантов РФФИ 05-1-00721, 06-01-08052 и Программы приоритетных фундаментальных исследований Отделения математических наук РАН.

[®]Институт вычислительной математики РАН

таких, что

$$f(x)v(x) - u(x) = O(x^{m+n+1}) \quad (1)$$

при дополнительном условии

$$v(0) = 1. \quad (2)$$

Отсюда сразу же следует, что

$$f(x) - \frac{u(x)}{v(x)} = O(x^{m+n+1}).$$

Условие (1) равносильно системе линейных уравнений

$$\sum_{j=0}^n a_{i-j} v_j = 0, \quad m+1 \leq i \leq m+n,$$

или, в матричной записи,

$$\begin{bmatrix} a_{m+1} & a_m & \dots & a_{m-n+1} \\ a_{m+2} & a_{m+1} & \dots & a_{m-n} \\ \dots & \dots & \dots & \dots \\ a_{m+n} & a_{m+n-1} & \dots & a_m \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \dots \\ v_n \end{bmatrix} = 0.$$

С учетом равенства $v_0 = 1$ получаем

$$\begin{bmatrix} a_m & \dots & a_{m-n+1} \\ \dots & \dots & \dots \\ a_{m+n-1} & \dots & a_m \end{bmatrix} \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} = - \begin{bmatrix} a_{m+1} \\ \dots \\ a_{m+n} \end{bmatrix}.$$

Для наглядности возьмем $m = n = 3$, тогда

$$\begin{bmatrix} a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2 \\ a_5 & a_4 & a_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = - \begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix}.$$

В матрице этой системы каждый элемент определяется разностью строчного и столбцовго индексов — такие матрицы называются *теплицевыми*. Переставив в обратном порядке столбцы, получаем матрицу, в которой элементы определяются суммой индексов —

такие матрицы называются *ганкелевыми*. Из нашей системы после такой перестановки получается равносильная система

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 \\ a_3 & a_4 & a_5 \end{bmatrix} \begin{bmatrix} v_3 \\ v_2 \\ v_1 \end{bmatrix} = - \begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix}.$$

В общем случае это будет система следующего вида:

$$\begin{bmatrix} a_{m-n+1} & \dots & a_m \\ \dots & \dots & \dots \\ a_m & \dots & a_{m+n-1} \end{bmatrix} \begin{bmatrix} v_n \\ \dots \\ v_1 \end{bmatrix} = - \begin{bmatrix} a_{m+1} \\ \dots \\ a_{m+n} \end{bmatrix}, \quad (3)$$

или, в краткой записи,

$$A_{mn}v^n = -a^{mn}. \quad (4)$$

Ганкелева матрица A_{mn} составляется из коэффициентов формального ряда $f(x)$. При этом в ее левом нижнем углу помещается коэффициент a_m , а порядок матрицы равен n . Если $i < 0$, то считается, естественно, что $a_i = 0$. Столбец правой части a^{mn} является последним столбцом расширенной прямоугольной ганкелевой матрицы $[A_{mn}, a^{mn}]$.

Таким образом, вопрос о существовании аппроксимации Паде типа (m, n) равносителен вопросу о разрешимости линейной системы (4). Последнее означает, что

$$a^{mn} \in \text{im } A_{mn},$$

и, в силу теоремы Кронекера–Капелли, равносильно условию

$$\text{rank } A_{mn} = \text{rank}[A_{mn}, a^{mn}]. \quad (5)$$

2. Метод скачков

Рассмотрим полубесконечную ганкелеву матрицу

$$A = [a_{i+j-1}], \quad 1 \leq i, j < \infty,$$

и ее ведущие подматрицы. Пусть A_k — ведущая подматрица порядка k , а последовательность натуральных чисел

$$n_1 < n_2 < \dots$$

определяет порядки тех и только тех ведущих подматриц, которые являются невырожденными. Метод скачков, предложенный в [2], представляет собой схему перехода от некоторого компактного представления матрицы $A_{n_k}^{-1}$ к аналогичному представлению для $A_{n_{k+1}}^{-1}$. Название объясняется тем, что при этом происходит «скакок» через промежуточные вырожденные подматрицы.

С общими вопросами построения быстрых алгоритмов для ганкелевых и теплицевых матриц можно познакомиться, например, по работам [4, 5, 7]. Алгебраические свойства ганкелевых матриц, позволяющие сделать «скакок», можно найти также в книге [6]. Они тесно связаны с изученными в [3] вопросами бесконечного продолжения ганкелевой матрицы с сохранением ранга.

Метод скачков базируется на следующем наблюдении. Пусть $p = n_k$ и $q = n_{k+1}$. В силу невырожденности A_p система

$$\begin{bmatrix} a_1 & \dots & a_p & \\ \dots & \dots & \dots & \\ a_p & \dots & a_{2p-1} & \end{bmatrix} \begin{bmatrix} s_1 \\ \dots \\ s_p \end{bmatrix} = \begin{bmatrix} a_{p+1} \\ \dots \\ a_{2p} \end{bmatrix}$$

имеет единственное решение. Другими словами, усеченные до p элементов столбцы с 1-го по p -й линейно независимы, а таким же образом усеченный $p+1$ -й столбец является их линейной комбинацией с коэффициентами s_1, \dots, s_p . Вполне возможно, что те же коэффициенты позволяют получить $p+1$ -й столбец как линейную комбинацию предыдущих столбцов при усечении до $p+1$ или даже большего числа элементов.

Теорема 1. Пусть $r(p) \geq p$ — минимальный размер усечения, при котором $p+1$ -й столбец не является линейной комбинацией предыдущих столбцов. Тогда $n_{k+1} = r(n_k)$.

Доказательство. Пусть $p = n_k$ и $r = r(p)$. Тогда

$$\begin{bmatrix} a_1 & \dots & a_p & a_{p+1} \\ \dots & \dots & \dots & \dots \\ a_p & \dots & a_{2p-1} & a_{2p} \\ \dots & \dots & \dots & \dots \\ a_{r-1} & \dots & a_{r+p-2} & a_{r+p-1} \\ a_r & \dots & a_{r+p-1} & a_{r+p} \end{bmatrix} \begin{bmatrix} -s_1 \\ \dots \\ -s_p \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \dots \\ 0 \\ \dots \\ 0 \\ \gamma \end{bmatrix}, \quad \gamma \neq 0.$$

Отсюда получаем равенство

$$\begin{aligned}
 A_r & \left[\begin{array}{cc|c} -s_1 & & 1 & \\ \dots & -s_1 & 1 & \\ \dots & \dots & \dots & \\ -s_p & \dots & \dots & \\ \hline 1 & -s_p & \dots & \\ & 1 & \dots & \\ & \dots & \dots & \\ & \dots & -s_p & \\ & & 1 & \end{array} \right] = \\
 & = \left[\begin{array}{c|cccc} & a_1 & a_2 & \dots & a_p \\ & a_2 & a_3 & \dots & a_{p+1} \\ \dots & \dots & \dots & \dots & \dots \\ a_p & a_{p+1} & \dots & a_{2p-1} \\ \hline \gamma & a_{p+1} & a_{p+2} & \dots & a_{2p} \\ \dots & \dots & \dots & \dots & \dots \\ \gamma & \dots & \dots & \dots & \dots \\ \gamma & \dots & \dots & a_r & a_{r+1} \dots a_{r+p-1} \end{array} \right]. \tag{6}
 \end{aligned}$$

При $p = n_k$ матрица A_p невырожденная. Поэтому ясно, что невырожденность окаймляющей ее матрицы A_r равносильна условию $\gamma \neq 0$. \square

Следствие 1. При $n_k \leq n \leq n_{k+1}$ имеет место равенство

$$\dim \ker A_n = \min\{n - n_k, n_{k+1} - n\}.$$

Доказательство. Непосредственно из (6) вытекает, что при умножении A_n на невырожденную матрицу получается матрица, в которой имеется $\min\{n - n_k, n_{k+1} - n\}$ нулевых столбцов, а остальные столбцы линейной независимы. \square

Введем обозначение \hat{A}_n для следующего расширения матрицы A_n :

$$\hat{A}_n = \begin{bmatrix} a_1 & \dots & a_n & a_{n+1} \\ \dots & \dots & \dots & \dots \\ a_n & \dots & a_{2n-1} & a_{2n} \end{bmatrix}.$$

Следствие 2. Если $n_k \leq n < n_{k+1}$, то равенство рангов

$$\operatorname{rank} A_n = \operatorname{rank} \hat{A}_n \quad (7)$$

имеет место в том и только том случае, когда

$$n - n_k < n_{k+1} - n. \quad (8)$$

Доказательство. Согласно теореме Кронекера–Капелли, равенство (7) равносильно тому, что последний столбец расширенной матрицы \hat{A}_n является линейной комбинацией ее предыдущих столбцов. Пусть $p = n_k$ и $q = n_{k+1}$.

Запишем $p \leq n = p + i < q$. Заметим, что $p + 1$ -й столбец матрицы A_{q-1} является линейной комбинацией предыдущих столбцов с коэффициентами s_1, \dots, s_p . То же верно в отношении $p + 1$ -го столбца ее ведущих подматриц, содержащих A_{p+1} . Если

$$p + 2i < q,$$

то это верно для матрицы A_{p+2i} . Используя ганкелеву структуру матриц, отсюда легко вывести, что $p + 1 + i$ -й (последний) столбец расширенной матрицы \hat{A}_{p+i} линейно выражается через предыдущие p столбцов (с теми же коэффициентами s_1, \dots, s_p). Условие $p + 2i < q$ равносильно неравенству $n - p < q - n$.

Остается доказать, что (7) влечет за собой (8). Согласно (7) последний столбец расширенной матрицы \hat{A}_n линейно выражается через столбцы A_n . В этом случае при переходе от A_n к A_{n+1} ранг может увеличиться не более чем на 1. От противного, допустим, что

$$n - p \geq q - p.$$

Согласно следствию 1,

$$\operatorname{rank} A_n = n - \min\{n - p, q - n\} = 2n - q,$$

$$\operatorname{rank} A_{n+1} = n + 1 - \min\{n + 1 - p, q - n - 1\} = 2n - q + 2.$$

Следовательно,

$$\operatorname{rank} A_{n+1} = \operatorname{rank} A_n + 2,$$

т. е. ранг должен вырасти больше, чем на 1. \square

Следствие 3. В случае $n_k \leq n < n_{k+1}$ неравенство (8) выполняется тогда и только тогда, когда

$$\operatorname{rank} A_{n+1} - \operatorname{rank} A_n \leq 1.$$

3. Детерминантное тождество

Пусть A — матрица порядка n , а A_{ij} — ее подматрица порядка $n-1$, полученная вычеркиванием строки i и столбца j . Пусть $A_{ik;jl}$ обозначает подматрицу порядка $n-2$, полученную из A вычеркиванием пары строк с номерами i и k и пары столбцов с номерами j и l . Следующий результат известен как тождество Сильвестра.

Теорема 2. Пусть $i < k$ и $j < l$. Тогда

$$\det A \det A_{ik;jl} = \det A_{ij} \det A_{kl} - \det A_{il} \det A_{kj}.$$

Доказательство. Не ограничивая общности, можно считать, что $i = j = n-1$ и $k = l = n$. Пусть $B = A_{ik;jl}$. Тогда матрица A имеет вид

$$A = \begin{bmatrix} B & v & q \\ u & c & d \\ p & g & h \end{bmatrix}.$$

Предположим сначала, что подматрица B невырожденная. Исключая по Гауссу u и p с помощью невырожденного блока B , находим

$$\begin{bmatrix} I & 0 & 0 \\ -uB^{-1} & 1 & 0 \\ -pB^{-1} & 0 & 1 \end{bmatrix} \begin{bmatrix} B & v & q \\ u & c & d \\ p & g & h \end{bmatrix} = \begin{bmatrix} B & v & q \\ 0 & c_1 & d_1 \\ 0 & g_1 & h_1 \end{bmatrix},$$

где

$$h_1 = h - pB^{-1}q, \quad c_1 = c - uB^{-1}v, \quad g_1 = g - pB^{-1}v, \quad d_1 = d - uB^{-1}q.$$

Следовательно,

$$\det A_{ij} \det A_{kl} - \det A_{il} \det A_{kj} = (\det B)^2 (h_1 c_1 - g_1 d_1) = \det B \det A.$$

Если матрица B вырождена, то тождество справедливо при замене B на любую невырожденную матрицу B_ε . Поскольку B_ε можно выбрать сколь угодно близко к B , интересующее нас тождество получается предельным переходом. \square

4. Таблица миноров

Приступим к изучению бесконечной ганкелевой матрицы $A = [a_{i+j}]$, составленной из коэффициентов формального ряда $f(x) =$

$\sum_{i=0}^{\infty} a_i x^i$ (если $i < 0$, то $a_i = 0$). Напомним, что через A_{mn} обозначается ее ганкелева подматрица порядка n , содержащая в левом нижнем углу коэффициент a_m . Нас интересует полу бесконечная матрица $C = [c_{mn}]$, составленная из миноров матрицы A :

$$c_{mn} = \det A_{mn}, \quad 0 \leq m, n < \infty.$$

Условимся считать, что $c_{m0} = 1$.

Лемма 1.

$$c_{m,n+1} c_{m,n-1} = c_{m+1,n} c_{m-1,n} - c_{mn}^2.$$

Доказательство. Данное равенство есть не что иное, как детерминантное тождество Сильвестра (теорема 2), записанное для ганкелевой матрицы $A_{m,n+1}$ и ее подматриц, получаемых при вычеркивании первых и последних строк и столбцов и поэтому остающихся ганкелевыми. \square

Таблицу миноров C для ганкелевой матрицы A иногда называют *C-таблицей* [1]. Ее основное свойство заключается в особой структуре расположения нулей (нулевых миноров матрицы A). Будем называть *окном* любую конечную или бесконечную подматрицу, составленную из подряд идущих строк и столбцов. Окно называется *квадратным окном*, если оно соответствует конечной квадратной подматрице или бесконечной подматрице, в которой бесконечно много как строк, так и столбцов. Все примыкающие к окну элементы будем называть его *рамой* — это элементы более широкого окна на строках и столбцах, окаймляющих данное окно. Нас будут интересовать *нулевые окна* и *ненулевые рамы* — в первом случае все элементы множества суть нули, во втором все они отличны от нуля.

В дальнейшем будем считать, что $a_0 \neq 0$. Тогда $c_{0n} \neq 0$ при $n \geq 1$ (определители ганкелевых треугольных матриц с ненулевым элементом побочной диагонали). Кроме того, примем соглашение о том, что $c_{m0} = 1$ при $m \geq 0$.

Теорема 3. Любой нулевой элемент таблицы миноров C принадлежит квадратному нулевому окну с ненулевой рамой.

Доказательство. Предположим, что $c_{m,n-1} = c_{m+1,n} = 0$ или $c_{m,n+1} = c_{m+1,n} = 0$. Согласно лемме 1 находим $c_{mn} = 0$. Таким образом, матрицы вида

$$\begin{bmatrix} 0 & * \\ * & 0 \end{bmatrix}, \quad \begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix}$$

непременно оказываются нулевыми. С учетом неравенств $c_{m0} \neq 0$ и $c_{0n} \neq 0$ отсюда следует, что любой нулевой элемент матрицы C принадлежит прямоугольному нулевому окну с ненулевой рамой. Остается доказать, что любое такое окно является квадратным.

Пусть $c_{mn} \neq 0$ — элемент рамы, расположенный в ее левом верхнем углу. Предположим, что $c_{m+r,n+r} \neq 0$ — еще один элемент рамы того же окна, и докажем, что этот элемент находится в правом нижнем углу. Если это не так, то:

- (1) $c_{m+r,n+r-1} = 0$, либо
- (2) $c_{m+r-1,n+r} = 0$.

Случай (1). Заметим, что матрицы A_{mn} и $A_{m+r,n+r}$ являются невырожденными ведущими подматрицами в ганкелевой матрице $A_{m+r,n+r}$ и при этом промежуточные ведущие подматрицы $A_{m+i,n+i}$ при $0 < i < r$ являются вырожденными. Согласно методу скачков (теорема 1), $n+1$ -й столбец матрицы матрицы $A_{m+r,n+r}$ с вычеркнутой последней строкой является линейной комбинацией предыдущих столбцов.

В данном случае $c_{m+1,n} \neq 0$ и $c_{m+r+1,n+r} \neq 0$. Поэтому метод скачков можно применить к невырожденным ганкелевым матрицам $A_{m+1,n}$ и $A_{m+1+r,n+r}$ и сделать вывод о том, что $n+2$ -й столбец матрицы $A_{m+r,n+r}$ с вычеркнутой последней строкой является линейной комбинацией предыдущих столбцов, начиная со 2-го. Вычитая данные линейные комбинации из $n+1$ -го и $n+2$ -го столбцов, мы не изменим определитель матрицы $A_{m+r,n+r}$ и получим в нем два столбца с нулевыми элементами, кроме элементов последней строки. Следовательно, $c_{m+r,n+r} = \det A_{m+r,n+r} = 0$, что противоречит сделанному ранее предположению. Поэтому с необходимостью $c_{m+r,n+r-1} \neq 0$.

Случай (2). Заметим, что $c_{m+r,n+r+1} \neq 0$ (иначе в силу леммы 1 $c_{m+r,n+r} = 0$). Таким образом, матрицы $A_{m,n+1}$ и $A_{m+r,n+r+1}$ —

невырожденными ведущие подматрицы с вырожденными промежуточными подматрицами. Согласно методу скачков, $n+2$ -й столбец в $A_{m+r, n+r+1}$ с вычеркнутой последней строкой является линейной комбинацией предыдущих столбцов. Отсюда находим, что $n+2$ -й столбец матрицы $A_{m+r, n+r}$ с вычеркнутой последней строкой является линейной комбинацией предыдущих столбцов. Как и раньше, то же верно в отношении $n+1$ -го столбца той же матрицы. Опять приходим к противоречию с невырожденностью матрицы $A_{m+r, n+r}$. Следовательно, $c_{m+r-1, n+r} \neq 0$.

Таким образом, в каждом из случаев (1) и (2) мы получаем противоречие. Значит, одновременно имеем

$$c_{m+r, n+r} \neq 0, \quad c_{m+r, n+r-1} \neq 0, \quad c_{m+r-1, n+r} \neq 0.$$

Это означает, что элемент $c_{m+r, n+r}$ находится в правом нижнем углу рамы нулевого окна. Поскольку в левом верхнем углу размещается элемент c_{mn} , то окно является квадратным. Если оказалось, что $c_{m+r, n+r} \neq 0$ при всех $r > 0$, то из леммы 1 сразу же вытекает, что данное нулевое окно является бесконечным квадратным окном.

□

5. Теория Паде

Теория Паде дает необходимое и достаточное условие существования аппроксимации Паде типа (m, n) в терминах структуры нулей в таблице миноров, ассоциированной с формальным рядом $f(x)$. Как мы уже знаем, необходима и достаточна совместность системы (4). Очевидно, условие $c_{mn} = \det A_{mn} \neq 0$ является достаточным для разрешимости этой системы. Основной результат для случая $c_{mn} = 0$ формулируется следующим образом.

Теорема 4. Пусть $c_{mn} = 0$ принадлежит нулевому окну матрицы C с ненулевой рамой, имеющей в левом верхнем углу элемент $c_{kl} \neq 0$, а в правом верхнем углу элемент $c_{k+r, l+r}$. Тогда аппроксимация Паде типа (m, n) существует в том и только том случае, когда $k+l < m+n < k+l+r$.

Доказательство. Пусть c_{st} и $c_{s+p, t+p}$ принадлежат ненулевой раме данного нулевого окна и при каком-то $h > 0$ получается $m = s+h$

и $n = t + h$. Матрица A_{st} является невырожденной ведущей подматрицей в невырожденной ганкелевой матрице $A_{s+p, t+p}$, причем подматрицы $A_{s+i, t+i}$ вырождены при всех $0 < i < p$. Из метода скачков вытекает (следствие 2), что условие совместности (5) равносильно неравенству

$$(t + h) - t < (t + p) - (t + h) \Leftrightarrow h < p/2.$$

Оно выполняется в том и только случае, когда

$$m + n < k + l + r. \quad \square$$

Теорема 5. Пусть c_{kl} и $c_{k+r, l+r}$ — угловые элементы ненулевой рамы нулевого окна таблицы миноров, и предположим, что $k \leq m$, $l \leq n$. Тогда аппроксимация Паде типа (k, l) является также аппроксимацией типа (m, n) , если $k + l \leq m + n < k + l + r$.

Если c_{kl} является угловым элементом ненулевой рамы бесконечного нулевого окна, то в случае $k \leq m$, $l \leq n$ аппроксимация Паде типа (k, l) будет также аппроксимацией Паде типа (m, n) , если $k + l \leq m + n$.

Доказательство. Пусть $k \leq m$ и $l \leq n$. Достаточно заметить, что если v^l есть решение системы

$$A_{kl}v^l = -a^{kl},$$

то при условии $k + l \leq m + n < k + l + r$ выполняется также равенство

$$A_{mn} \begin{bmatrix} 0 \\ v^l \end{bmatrix} = -a^{mn}.$$

В случае бесконечного окна это верно при всех $r > 0$. \square

Список литературы

- [1] Дж. Бейкер, П. Грейвс-Моррис, *Аппроксимации Паде*, Мир, 1986.
- [2] В. В. Воеводин, Е. Е. Тыртышников, *Вычислительные процессы с теплицевыми матрицами*, Наука, 1987, 320 с.

- [3] И. С. Иохвидов, *Ганкелевы и теплицевые матрицы и формы*, Наука, 1974, 264 с.
- [4] Е. Е. Тыртышников, *Теплицевые матрицы, некоторые их аналоги и приложения*, ОВМ РАН, 1989.
- [5] Е. Е. Тыртышников, *Методы численного анализа*, Издательский центр «Академия», 2007.
- [6] G. Heinig, K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Academie-Verlag, Berlin, 1984.
- [7] E. E. Tyrtyshnikov, Euclidean Algorithm and Hankel Matrices // *Numerical Analysis and Applied Mathematics*, AIP Conference Proceedings. 2007. V. 936, Melville, New York. P. 27–30.

Об алгоритме построения конформной квази-иерархической треугольной сетки, слабо δ -аппроксимирующей заданные ломаные[§]

В. Н. Чугунов[®]

Предложен алгоритм построения конформной квази-иерархической треугольной сетки, аппроксимирующей с точностью δ набор заданных ломаных. Возможность сдвига ломаных в пределах их δ -окрестности гарантирует разрешимость задачи. Результирующая сетка имеет небольшое число треугольников и допускает реализацию многосеточного метода. Установлена конечность данного алгоритма и доказана оценка на рост числа треугольников в результирующей сетке при уменьшении параметра δ (порядка $\log_2 \delta^{-1}$). Приведены результаты работы алгоритма для конкретного заданного набора ломаных.

1. Введение

Задачи построения конформной треугольной сетки, удовлетворяющей некоторым ограничениям, часто встречаются на практике. Набор ограничений диктуется постановкой задачи. Типичными ограничениями являются отсутствие малых углов у треугольников сетки, аппроксимация границы области ребрами сетки со вторым порядком относительно длин ребер.

Известны задачи, в которых возникают дополнительные ограничения, связанные с аппроксимацией множественных внутренних

[§]Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (гранты РФФИ №№05-01-00721 и 06-01-08052) и программы фундаментальных исследований отделения математических наук РАН «Вычислительные и информационные проблемы решения больших задач» по проекту «Матричные методы и технологии для задач со сверхбольшим числом неизвестных», а также исследовательского гранта ExxonMobil Corp.

[®]Институт вычислительной математики РАН (119333, Москва, ул. Губкина, д.8) e-mail: vadim@bach.inm.ras.ru

границ. Примером может служить двумерная подзадача, возникающая при моделировании трехмерных течений в пористых средах, обладающих слоистой структурой. Для решения последней задачи удобно использовать призматические сетки с основаниями призм, лежащими на границах геологических слоев. Самые геологические слои, связанные с различными породами, имеют переменную толщину и могут сужаться вплоть до полного исчезновения на линиях вырождения, а также менять толщину скачкообразно в районе вертикальных разломов земной коры. Для успешного решения задач моделирования необходимо, чтобы призматическая сетка отражала эти особенности, а именно, линии вырождения и поверхности вертикальных разломов были аппроксимированы ребрами и гранями призм. Одним из способов построения желаемой сетки является проектирование разломов и линий вырождения на двумерную область и задание их ломаными. Имея двумерную сетку со сторонами треугольников, лежащими на ломаных, можно путем дублирования двумерной сетки для каждого геологического слоя получить трехмерную сетку, учитывающую имеющиеся особенности структуры пористой среды. Построение сетки с внутренними множественными ограничениями может быть также использовано как при решении других сложных задач, так и само по себе.

Набор условий, накладываемых на алгоритм, определяется следующими математическими требованиями, предъявляемыми к сетке. Во-первых, хотелось иметь возможность использовать получающуюся сетку при реализации многосеточных методов [2], являющихся наиболее эффективными методами решения краевых задач. Это приводит к условию, чтобы полученная сетка была заключительным звеном некоторой последовательности логически вложенных друг в друга сеток, для которой задано правило перехода от одной сетки к другой. Во-вторых, требования устойчивости аппроксимации краевых задач ограничивает класс двумерных сеток триангуляциями с треугольниками регулярной формы [7, 9]. Наряду с удовлетворением этим математическим условиям, цель алгоритма заключается в том, чтобы результирующая сетка подчинялась также набору физических требований. А именно, построенная двумерная сетка должна обладать небольшим числом треугольников в силу ограничений, накладываемых возможностями вычислительной тех-

ники при решении задач, использующих треугольную сетку. Кроме этого, так как данные о внутренних границах не могут быть точными из-за погрешностей измерений, всегда существует некоторый порог для точности аппроксимации заданных ломаных, что допускает их незначительную локальную модификацию. Вышеприведенные соображения определяют рассматриваемый класс сеток и ограничений.

Существует несколько методов генерации треугольных сеток [7, 9, 10], удовлетворяющей множественным ограничениям, а также целый ряд их программных реализаций [15, 16]. Первый алгоритм принадлежит Бейкеру, Гроссе и Рафферти [3]. Они предложили метод триангуляции области, являющейся множеством многоугольников с углами не менее 13° . В итоговой сетке углы всех треугольников не превосходят 90 градусов и наименьший угол не меньше 13° . Однако, получающаяся сетка является равномерной и, поэтому, имеет очень много сеточных элементов.

Проблема большого числа треугольников при решении рассматриваемой задачи позднее была устранена Берном, Эпштейном и Гильбертом [5]. Они создали сеточный генератор с треугольниками, имеющими ограничения на форму и размер, подчиняющиеся начальным данным. Кроме того, авторы показали, что строящаяся ими сетка имеет треугольники с отношением сторон, не превосходящим числа 5. Их алгоритм основан на идее предварительного рекурсивного разбиения области на квадраты различной формы.

Другим способом решения поставленной задачи является триангуляция Делонэ. Обзор адекватных методов для построения триангуляций Делонэ с множественными ограничениями можно найти в [1, 13]. Наиболее известные результаты представлены в работах Чью и Рапперта. Чью [8] представил триангуляцию Делонэ, в которой все треугольники имеют углы между 30 и 120 градусов, для области с наименьшим углом, образованным границами, не меньше 30 градусов. Его алгоритм производит равномерную сетку.

Рапперт [12] расширил идеи Чью, предложив алгоритм триангуляции планарного прямолинейного графа такой, что все треугольники выходной сетки имеют углы между α и $\pi - 2\alpha$ ($0 < \alpha < 20$), где α зависит от входных ограничений. Сетка является неравномерной. Алгоритм Рапперта, получая на вход некоторую триангуля-

цию Делонэ, точно аппроксимирующую ребра графа, выдает сетку с треугольниками, удовлетворяющими ограничениям на форму. При этом начальная аппроксимация сохраняется. Таким образом, данный алгоритм лишь улучшает имеющуюся триангуляцию. В статье [12] доказывается, что число треугольников в выходной сетке получается из начального числа, умножением на константу, зависящую от α .

В представляемой работе исследуется иной подход к решению задачи. В отличие от описываемых выше методов в статье представлен генератор, который получает на вход треугольную конформную сетку, точно аппроксимирующую только внешнюю границу и не учитывающую внутренние ограничения. Результирующая конформная неравномерная сетка с треугольниками, удовлетворяющими ограничениям на форму, сохраняя точную аппроксимацию внешней границы, приближено аппроксимирует внутренние границы с заданной точностью δ . При этом сама аппроксимация внутренних границ во время построения сетки может сдвигаться от границы на величину не более δ с целью безотказной работы алгоритма. Представляемый генератор обладает рядом преимуществ. Во-первых, процесс построения сетки является конструктивной процедурой. Итоговая сетка получается из начальной последовательным применением одной из двух операций: сдвига узла и разбиения треугольника на два методом бисекции. Это обстоятельство дает возможность ввести иерархию сеток для использования многосеточных методов. Во-вторых, приближенная аппроксимация внутренних границ и возможность ее сдвига позволяют применять алгоритм для разнообразных конфигураций заданных внутренних ограничений. В-третьих, число треугольников в полученной сетке будет небольшим благодаря приближенной аппроксимации внутренних границ. Мы даем на него оценку, зависящую не от начальной сетки, а от точности аппроксимации δ .

Постановки задач, рассматриваемых в этой статье, обсуждаются в разделе 2. Алгоритм решения задачи слабой δ -аппроксимации ломаных, будет описан в разделе 3. Характеристика свойств сетки, которую строит предлагаемый алгоритм, содержится в пункте 4. В разделе 5 приведены примеры работы предлагаемого алгоритма.

Автор выражает благодарность Василевскому Ю. В. за постанов-

ку задачи, консультации при ее решении и написании статьи.

2. Постановка задачи

В более общем виде задача аппроксимации ломаных может быть сформулирована следующим образом. Данна грубая начальная конформная треугольная сетка, покрывающая заданную полигональную область Ω , точно аппроксимирующую границу области Ω , и некоторое множество ломаных, задаваемых упорядоченным набором точек на плоскости. Заданная сетка будет порождать множество логически иерархических конформных сеток, получаемых последовательным многоуровневым разбиением некоторых треугольников на два подтреугольника методом бисекции [4, 11]. Конформные сетки, получаемые сдвигом узлов иерархических сеток, называются квази-иерархическими сетками. Множество квази-иерархических сеток позволяет организовать многосеточные алгоритмы на основе исключительно геометрической (топологической) информации, поэтому именно эти сетки рассматриваются ниже.

Для каждого треугольника определены две численные характеристики — качество формы и качество аппроксимации (множества ломаных). Под качеством формы Q_i треугольника i понимается величина, определяемая по формуле [6, 14]:

$$Q_i = \frac{P_0^2}{S_0} * \frac{S_i}{P_i^2} = 12\sqrt{3} * \frac{S_i}{P_i^2}, \quad (1)$$

где S_i — площадь, P_i — периметр треугольника i , P_0 , S_0 — периметр и площадь любого равностороннего треугольника. Отметим, что $Q_i \leq 1$, причем $Q_i = 1$ только для равностороннего треугольника. Для вычисления качества аппроксимации \hat{Q}_i треугольника i сначала определяется качество аппроксимации $\hat{Q}_i^{(l,j)}$ по отношению к каждому звену j каждой ломаной l : если звено j ломаной l пересекает треугольник i , то

$$\hat{Q}_i^{(l,j)} = 1 - \frac{2 * \min(S_1, S_2)}{S}, \quad (2)$$

где S — площадь треугольника, а S_1 и S_2 — площади фигур, на которые звено j ломаной l разбивает треугольник i ; в противном случае (пересечения нет) положим

$$\hat{Q}_i^{(l,j)} = 1. \quad (3)$$

Таким образом, $\hat{Q}_i^{(l,j)} = 1$ для треугольника i , не разбиваемого звеном j ломаной l . Как только качество аппроксимации $\hat{Q}_i^{(l,j)}$ определено для каждого звена j каждой ломаной l , качество аппроксимации \hat{Q}_i треугольника i берется как минимум среди найденных $\hat{Q}_i^{(l,j)}$

$$\hat{Q}_i = \min_l \min_j \hat{Q}_i^{(l,j)}. \quad (4)$$

Для всей сетки ϵ также введено качество формы сетки Q_ϵ и качество аппроксимации сетки \hat{Q}_ϵ как минимум из соответствующих качеств треугольников

$$\begin{aligned} Q_\epsilon &= \min_i Q_i, \\ \hat{Q}_\epsilon &= \min_i \hat{Q}_i. \end{aligned} \quad (5)$$

В оптимизационной задаче точной аппроксимации ломанных квази-иерархическими сетками требуется для заданного числа $0 < q_0 < 1$ построить конформную квази-иерархическую треугольную сетку ϵ_0 на Ω , удовлетворяющую условию

$$\epsilon_0 = \arg \min_{\epsilon: Q_\epsilon \geq q_0, \hat{Q}_\epsilon = 1} n_t(\epsilon), \quad (6)$$

где $n_t(\epsilon)$ — число треугольников в сетке ϵ . Другими словами, цель задачи — для заданного числа q_0 построить конформную треугольную сетку, удовлетворяющую условиям: сетка квази-иерархична; заданные ломаные аппроксимируются сторонами треугольников, т.е. ломаные лежат на ребрах треугольников; треугольники имеют хорошее качество формы; число треугольников минимально.

Однако, оптимизационная задача точной аппроксимации ломанных достаточно сложна для решения. Заменим ее менее трудной: будем считать, что заданные ломаные могут аппроксимироваться сторонами треугольников с некоторой заранее задаваемой точностью δ . Пусть $d(i)$ — диаметр треугольника i . Определим для каждого треугольника δ -качество аппроксимации \hat{Q}_i^δ

$$\hat{Q}_i^\delta = \begin{cases} \hat{Q}_i, & \text{если } d(i) > \delta, \\ 1, & \text{в противном случае,} \end{cases} \quad (7)$$

а для сетки δ -качество аппроксимации \hat{Q}_ϵ^δ

$$\hat{Q}_\epsilon^\delta = \min_i \hat{Q}_i^\delta. \quad (8)$$

То есть, ломаная может пересекать треугольник i по линии, не совпадающей с ребром, если $d(i) \leq \delta$.

Оптимизационная задача δ -аппроксимации ломанных состоит в том, чтобы для заданных чисел $0 < q_0 < 1$, $0 < \delta \ll 1$ построить конформную квази-иерархическую треугольную сетку ϵ_0 на Ω такую, что

$$\epsilon_0 = \arg \min_{\epsilon: Q_\epsilon \geq q_0, \hat{Q}_\epsilon^\delta = 1} n_t(\epsilon). \quad (9)$$

Предлагаемый ниже алгоритм предназначен для решения еще более упрощенной задачи, называемой в дальнейшем *задачей слабой δ -аппроксимации ломанных*. В этой задаче разрешается сдвигать ломаные на расстояние не больше δ . Как будет показано ниже, это обеспечивает построение приближенного решения задачи в некотором классе сеток. Фактически, мы рассматриваем построение квази-иерархической сетки ϵ_0 , удовлетворяющей условиям $Q_{\epsilon_0} \geq q_0$,

$\hat{Q}_{\epsilon_0}^\delta = 1$ и получающейся из заданной путем как можно меньшего числа действий с узлами сетки и треугольниками. Приближенность решения обусловлена заменой условия строгой минимизации числа треугольников на желание получить сетку лишь с небольшим, по-возможности, числом сеточных элементов³².

Отметим, что замена оптимизационной задачи точной аппроксимации ломанных задачей слабой δ -аппроксимации ломанных обусловлена наличием погрешности в представлении данных. Переход к упрощенной постановке, сохраняющей сущность двумерной подзадачи в рамках трехмерной задачи фильтрации, обеспечивает простоту и надежность предлагаемого алгоритма, в котором будут использоваться лишь две основные операции: сдвиг узла и разбиение треугольника на два подтреугольника [4, 11]. В качестве параметра δ можно рассматривать точность представления начальных данных. Из описания алгоритма будет видно, что при δ , стремящимся к нулю, заданные ломаные не сдвигаются, стороны треугольников почти точно их аппроксимируют, а число треугольников растет умеренно.

³²Структура алгоритма позволяет дополнительно локально измельчать построенную сетку.

3. Описание алгоритма

3.1. Обозначения и базовые операции.. Прежде, чем изложить алгоритм построения сетки, опишем предположения, при которых будет решаться задача слабой δ -аппроксимации ломаных, и введем некоторые необходимые в дальнейшем обозначения.

Данную задачу будем решать при следующих предположениях: число ломаных и число звеньев в одной ломаной не больше некоторых заданных значений; любая ломаная не имеет самопересечений; каждая ломаная лежит в замыкании Ω ; ни одна из внутренних точек любой ломаной не лежит на границе области Ω ³³.

При формулировании алгоритма нам понадобятся различные типы узлов сетки. Для этого все узы сетки мы разделим на неподвижные узлы, подвижные вдоль границы и подвижные в любом направлении. В дальнейшем нам также понадобится понятие качества формы узла, которое положим равным минимуму из качеств форм всех треугольников, содержащих данный узел и образующих суперэлемент данного узла. Также определим понятие суперэлемент треугольника как объединение суперэлементов его вершин.

Введем число ε как минимум из расстояний между двумя звеньями ломаных, не имеющих общих точек, и расстояний между звеном ломаной и отрезком границы всей области, который данное звено не пересекает. Тогда длина любого звена любой ломаной не меньше ε .

При описании процесса построения сетки мы будем использовать две вспомогательные процедуры.

Одна из них — это процедура *бисекции*, которая применима к сетке, у которой помечены некоторые треугольники. Процедура бисекции заключается в разбиении на два каждого из помеченных треугольников, а также некоторых других для сохранения конформности сетки. Важным свойством *BS* алгоритма бисекции является локальность: бисекция одного помеченного треугольника может порождать бисекцию лишь треугольников из суперэлемента помеченного треугольника [1,2]. В дальнейшем нам потребуется еще одно свойство *KBS* алгоритма бисекции, которое заключается в том, что

³³Если какая-то ломаная имеет внутреннюю точку (или отрезок) на границе, разобъем ее на две, тогда точка на границе будет граничной для обеих ломаных (в случае отрезка удаляем этот отрезок из ломаной).

применение к данной сетке любого числа уровней бисекций не ухудшит качество формы сетки более, чем в два раза [11].

Также нам будет необходима операция *проверки возможности сдвига* $S(J, R, j_0, r_0, \hat{q})$ одного узла из заданного подмножества J точек сетки в одну из точек предписанного множества R . Если все узлы в J неподвижны, процедура выдает информацию о невозможности сдвига, иначе она сначала строит подмножество \tilde{J} из всех подвижных узлов в J и моделирует сдвиг каждого элемента $j \in \tilde{J}$ в каждую из точек $r \in R$ с целью найти такую пару элементов $j_0 \in \tilde{J}$ и $r_0 \in R$, что при перемещении j_0 в r_0 треугольники не налегают друг на друга и не выворачиваются, а качество формы $q(j_0)$ узла j_0 будет равным

$$q(j_0)|_{j_0 \rightarrow r_0} = \max_{j \in \tilde{J}} \max_{r \in R} q(j)|_{j \rightarrow r},$$

для всех пар j, r , обеспечивающих конформность получаемой сетки. Если такая пара нашлась, и получаемое качество формы $q(j_0)$ узла j_0 после сдвига не ниже \hat{q} , то процедура выдает на выходе пару j_0, r_0 , в противном случае выдается информация о невозможности сдвига. Заметим, что если вершина подвижна только вдоль границы, то для нее сдвиг должен моделироваться только для тех точек R , которые находятся на границе. Отметим важное свойство SD операции сдвига. Для любого треугольника, пересекаемого некой прямой, содержащей его вершину, сдвиг ближайшей из других вершин (j) на прямую не ухудшит качество формы сдвигаемого узла $q(j)$ более чем в $\kappa(q)$ раз.

Теперь можно перейти к описанию алгоритма решения нашей задачи, который будет использовать процедуры бисекции и сдвигов узлов. Пусть начальная сетка имеет качество формы q . Будем предполагать, что порог q_0 из (9) не превосходит $q/4$, что необходимо для обеспечения возможности совершать бисекции и сдвиги.

3.2. Описание алгоритма. Рассматриваемый алгоритм построения треугольной конформной квази-иерархической сетки ϵ_0 с небольшим числом треугольников, имеющей δ -качество аппроксимации $\hat{Q}_{\epsilon_0}^\delta = 1$ и качество формы $Q_{\epsilon_0} \geq q_0$, на предварительном этапе может сдвинуть сами ломаные на величину не более δ . Это обеспечивает разрешимость задачи слабой δ -аппроксимации ломанных для заданного q_0 . Алгоритм включает в себя три этапа,

дальнейшее описание каждого из которых, при необходимости, будет заканчиваться обоснованием возможности совершения действий, составляющих основу изложенного этапа.

Этап 1 — аппроксимация общих точек соседних звеньев ломанных (т.н. точек излома), граничных точек ломаных и точек пересечения ломаных узлами сетки с помощью допустимых операций, и, если необходимо, сдвиг ломаных на величину не более δ . Этот этап состоит из нескольких шагов, проиллюстрированных на рис. 1—4.

Шаг 1-1. Строим множество Z , состоящее из точек излома, граничных точек ломаных и точек пересечения заданных ломаных (рис. 1).

Шаг 1-2. Удаляем из Z точки, которые лежат в узлах сетки, объявив соответствующие узлы сетки неподвижными. Для каждого треугольника i находим подмножество точек $Z_i \in Z$, принадлежащих этому треугольнику. Если треугольник имеет непустое множество Z_i , формируем множество L_i из трех его вершин и выполняем процедуру проверки возможности сдвига $S(L_i, Z_i, l_0, z_0, 4 * q_0)$. Если она дает положительный результат, то совершаём сдвиг l_0 в z_0 , объявляем вершину l_0 неподвижной и исключаем z_0 из Z , иначе, в случае если $d(i) > \delta$, метим данный треугольник для разбиения (рис. 2). После просмотра всех треугольников при наличии помеченных выполняем процедуру бисекции, которая дает дополнительные степени свободы. Если множество Z пусто, то переходим к этапу 2, иначе повторяем шаг 1-2 до тех пор, пока не получим сетку, в которой любой треугольник i либо не содержит точек из Z , либо содержит только одну точку и $d(i) \leq \delta$ (рис. 3).

Шаг 1-3. Для каждой точки из Z совершаем процедуру сдвига самой точки из Z на величину не более δ , что приводит к сдвигу ломаных не более, чем на δ . Для этого на треугольнике, ее содержащем, строим локальную сетку, включающую сами вершины треугольника, и из них формируем множество R_i . Определив множество L_i из трех вершин треугольника, выполняем процедуру проверки возможности сдвига $S(L_i, R_i, l_0, r_0, 2 * q_0)$, которая дает положительный результат в силу того, что сами вершины треугольника уже принадлежат локальной сетке и их качество формы не ниже $2 * q_0$, так как начальная сетка имела качество не хуже $4 * q_0$, а на шаге 1-2 подвергалась лишь бисекции и сдвигам S с качеством

$4 * q_0$. Передвигаем точку Z в эту выбранную точку локальной сетки r_0 . Если точка из Z — точка пересечения двух или более ломаных, то находим в этих ломаных звенья, содержащих эту точку и добавляем точку из Z в упорядоченные списки точек, задающих эти ломаные. Теперь в точку Z мы можем сдвинуть вершину треугольника l_0 (рис. 4) и зафиксировать ее.

На этом первый этап алгоритма завершен. В результате получаем конформную сетку с узлами, содержащими все элементы множества Z , и качеством формы, не хуже $2 * q_0$.

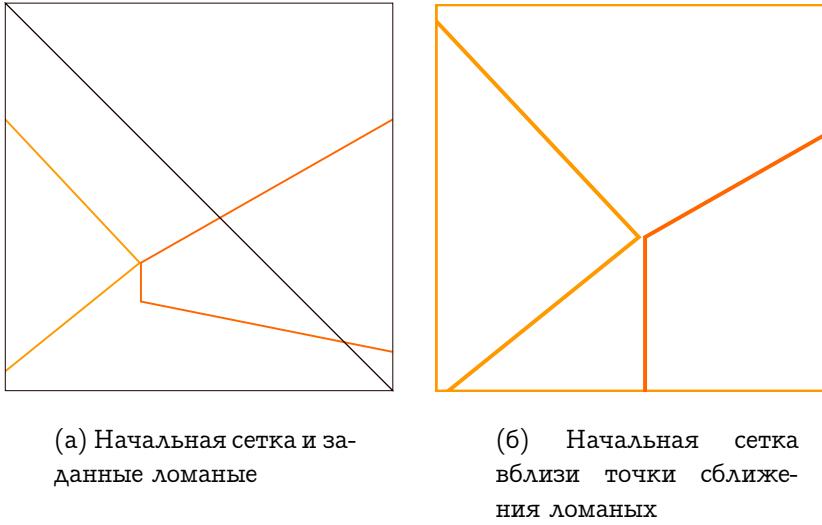
Отметим, что шаг 1-2 может производить сетку, сгущающуюся к некоторому конечному числу точек, не большему чем мощность изначального множества Z .

Достичь условия, чтобы все точки Z лежали в узлах сетки при заданном q_0 возможно лишь благодаря шагу 1-3. Если q_0 достаточно велико, то в результате работы только шагов 1-1 и 1-2 совместить все точки Z с узлами сетки не в любом случае возможно, так как большое q_0 не всегда дает возможность сдвига узла, а значит, процесс из шагов 1-1 и 1-2 без ограничения на диаметр треугольников может зациклиться. Шаг 1-3двигает сами точки Z так, чтобы для заданного q_0 можно было осуществить сдвиг. В этих случаях точка из Z сама передвигается в узел сетки, обеспечивая требуемый результат.

Если важно гарантированно сохранить взаимное расположение ломаных, необходимо выбрать δ меньше четверти ε . Действительно, перед шагом 1-3 точки, еще не аппроксимированные узлами сетки, принадлежат треугольникам с диаметром не более δ . Поэтому сдвиг точек Z может их сблизить, но лишь до расстояния $2\delta = \varepsilon/2$, что обеспечит отсутствие возможного слияния разных звеньев различных ломаных.

Из описания шага 1 видно, что величина сдвига ломаных зависит от геометрии ломаных и от δ ; кроме того, если δ устремить к нулю, то, начиная с некоторого момента, сдвига ломаных вообще не происходит. Сдвиг ломаных происходит локально, так как после каждого сдвига точки множества Z в новую точку помещается узел сетки, и рассматриваемый элемент Z исключается из множества Z . Поэтому любая точка новой ломаной будет находиться на расстоянии не более δ от своего прообраза на старой ломаной даже для

Рис. 1. Начальная сетка



суперпозиции нескольких сдвигов.

Этап 2 — приближение сторон треугольников к звеньям ломаных с использованием лишь допустимых операций.

Шаг 2-1. Если все треугольники имеют δ -качество аппроксимации равное единице, то переходим к этапу 3.

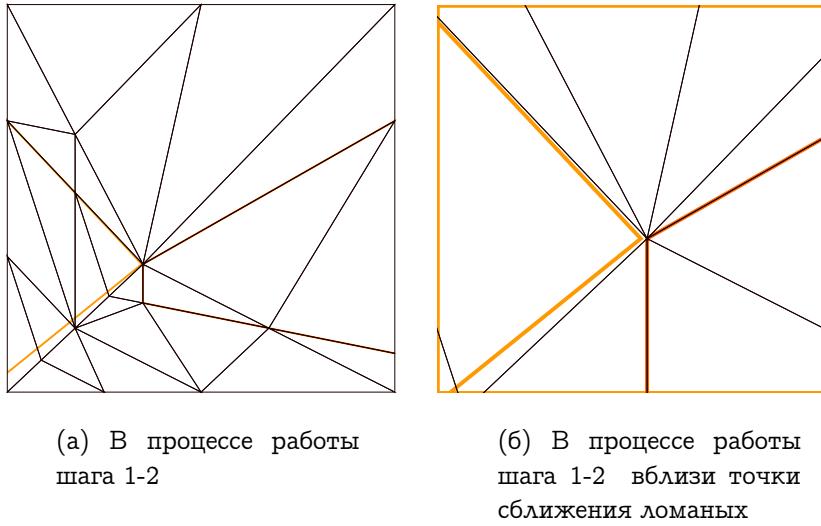
Шаг 2-2. Рассмотрим каждый треугольник i с δ -качеством аппроксимации, отличным от единицы. Если его качество аппроксимации отлично от единицы для некоторого звена j некоторой ломаной l , и хотя бы одна из его вершин, не лежащая на l , — неподвижна, то метим треугольник i для разбиения.

Шаг 2-3. Если есть помеченные треугольники, совершаем процедуру бисекции и переходим к шагу 2-1.

Шаг 2-4. Если все треугольники имеют δ -качество аппроксимации равное единице, то переходим к этапу 3.

Шаг 2-5. Для каждого треугольника i , имеющего δ -качество

Рис. 2. В процессе работы

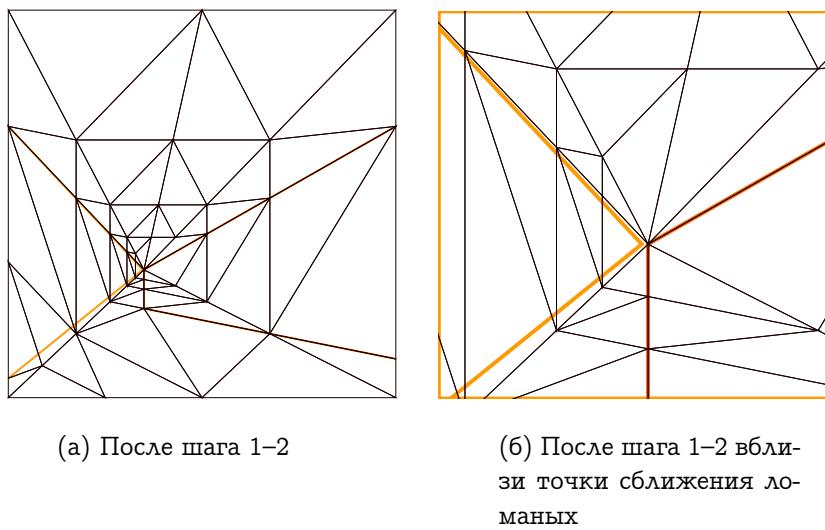


аппроксимации, отличное от единицы, находим вершину, ближайшую к одному из звеньев ломаных, которое пересекает данный треугольник не по ребру, и формируем из нее множество L_i . На соответствующем звене строим локальную сетку на той части, которая пересекает все три суперэлемента для трех вершин треугольника, образуем из нее множество R_i и вызываем процедуру проверки возможности сдвига $S(L_i, R_i, l_0, r_0, 2 * q_0)$. Если она дает отрицательный результат и $d(i) > \delta$, то метим этот треугольник для последующего разбиения. Если она дает пару l_0 и r_0 , то сносим l_0 в r_0 , фиксируем l_0 .

Шаг 2-6. Если на шаге 2-5 произошла сдвигка хотя бы одного узла, то считаем, что нет помеченных треугольников, и переходим к шагу 2-4. В противном случае совершаляем процедуру бисекции и переходим к шагу 2-4.

В результате работы второго этапа получаем конформную тре-

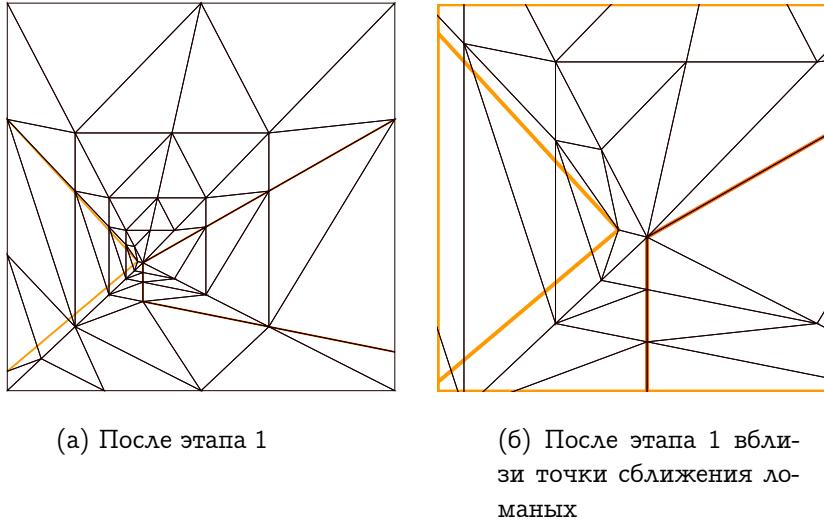
Рис. 3. После шага 1–2



угольную сетку, δ -качество аппроксимации треугольников в которой равно единице и качество формы треугольников не меньше q_0 . Про сдвиги S узлов мы требуем, чтобы качество было не хуже $2 * q_0$, так как треугольники, которые уже аппроксимировали ломаные, при выполнении алгоритма могут подвергаться бисекциям для обеспечения конформности сетки. Но по свойству KBS их качество не ухудшиться более чем в два раза, то есть их качество будет не меньше q_0 . Из описания видно, что второй этап — это повторяющиеся процедуры выполнения шагов 2-1 — 2-3 и шагов 2-4 — 2-6. В качестве примера на (рис. 5—6) приведены сетки после второго и четвертого повторений шагов 2-4 — 2-6.

Этап 3 — улучшение качества формы сетки за счет сдвиги подвижных узлов, если это возможно.

Рис. 4. После этапа 1



4. Анализ алгоритма.

В этом разделе опишем основные свойства алгоритма и сделаем некоторые важные замечания.

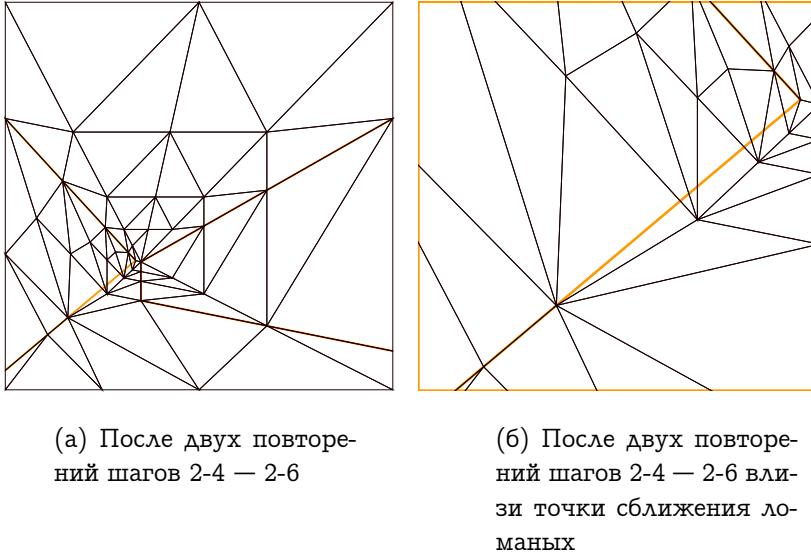
Свойство I. Представленный алгоритм строит треугольную конформную квази-иерархическую сетку ϵ_0 , которая имеет δ -качество аппроксимации $\hat{Q}_{\epsilon_0}^\delta = 1$ и качество формы $Q_{\epsilon_0} \geq q_0$, предварительно при необходимости сдвинув ломаные на величину не более δ .

Отметим, что в случае $\delta < \varepsilon/4$ расстояние между любыми двумя непересекающимися звенями остается меньше $\varepsilon/2$. Таким образом, взаимное расположение ломаных сохраняется после сдвига.

Свойство II. Алгоритм построения конформной треугольной квази-иерархической сетки, слабо аппроксимирующей ломаные с точностью δ , конечен.

Действительно, поскольку диаметр треугольника не меньше δ ,

Рис. 5. После двух повторений шагов 2-4 — 2-6

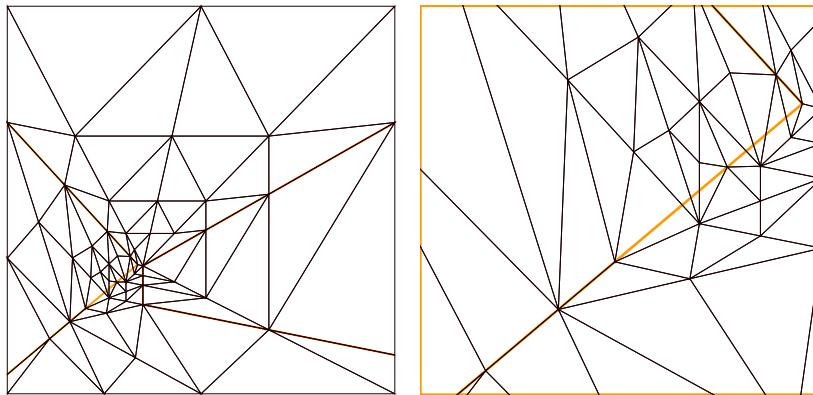


то число уровней разбиения не превышает $2 \log_2 \delta^{-1}$, т. е. конечно.

Свойство III. Пусть сетка после шага 2-3 имеет качество формы \tilde{q} . Шаги 2-5—2-6 алгоритма могут ухудшить качество формы не больше, чем в $2\kappa^2(\tilde{q})$ раз, где $\kappa(\tilde{q})$ — константа из свойства $S\mathcal{D}$ операции сдвига. Поэтому порог качества формы должен выбираться с ограничением $q_0 < \tilde{q}/2\kappa^2(\tilde{q})$. Действительно, рассмотрим отрезок звена ломаной, который еще не аппроксимирован ребрами треугольников, не содержит внутри узлов сетки, и не является частью другого большего отрезка, также не аппроксимированного ребрами треугольников. Отметим, что концы этого отрезка лежат в узлах сетки, а в силу окончания шагов 2-1 — 2-3 любой треугольник, пересекающий данный отрезок, не имеет неподвижных вершин, не лежащих на рассматриваемом отрезке.

Рассмотрим следующую последовательность действий для ап-

Рис. 6. После четырех повторений шагов 2-4 — 2-6



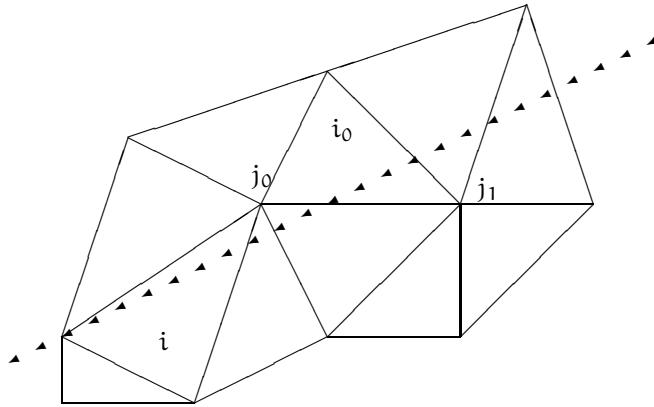
(а) После четырех повторений шагов 2-4 — 2-6

(б) После четырех повторений шагов 2-4 — 2-6 влизи точки сближения ломаных

проксимации данного отрезка. Возьмем треугольник i , пересекающийся с данным отрезком не по ребру, и содержащий один из концов отрезка (рис. 7).

При данном малом q_0 мы можем сдвинуть на ломаную ближайшую из вершин j_0 данного треугольника. По свойству SD качество формы сдвинутого узла ухудшится не более, чем в k раз (\tilde{q}/k). При этом сдвиг j_0 сразу ставит на ломаную целое ребро, изменяя качество формы только у треугольников из суперэлемента j_0 , но улучшая до единицы качество аппроксимации для треугольника i относительно звена, содержащего рассматриваемый отрезок. После сдвига j_0 в суперэлементе j_0 появляется лишь один треугольник (скажем i_0) с плохим качеством аппроксимации относительно исследуемого звена, и содержащий узел j_0 (рис. 8). Его качество формы не хуже \tilde{q}/k .

Рис. 7. Начальное расположение треугольников и звена ломаной

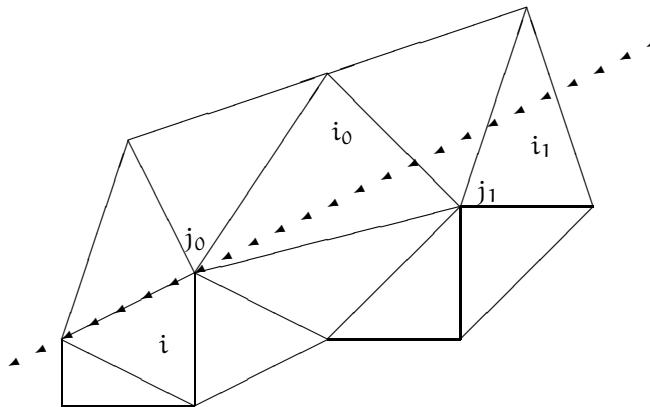


Далее, алгоритм сдвинет на ломаную узел j_1 , ближайшую вершину треугольника i_0 . Несмотря на возможное новое ухудшение качества формы треугольника i_0 до \tilde{q}/κ^2 после сдвига узла j_1 (согласно свойству SD), он начинает аппроксимировать ломаную. Поэтому далее суперэлемент j_0 не будет подвергаться сдвигам.

По аналогии с узлом j_0 после сдвига узла j_1 у нас появится треугольник i_1 с качеством аппроксимации, отличным от единицы, и содержащий вершину j_1 . Этот треугольник имеет качество формы не меньше \tilde{q}/κ (свойство SD), так как он не принадлежит суперэлементу j_0 и до сдвига j_1 имел качество формы не меньше \tilde{q} , поэтому для него можно применить операцию сдвига узла. Значит, возможен последовательный процесс аппроксимации ломаных, который не приведет к цепочке наращиваемого ухудшения качества формы каких-нибудь треугольников, а будет поддерживать качество формы аппроксимирующих треугольников не ниже \tilde{q}/κ^2 . Последующие возможные бисекции аппроксимирующих треугольников для сохранения конформности сетки могут ухудшить их качество формы не более чем в два раза (до $\tilde{q}/2\kappa^2$) по свойству KSD .

Поскольку любая последовательность действий в шагах 2-5-2-

Рис. 8. Расположение треугольников и звена ломаной после сдвига узла.



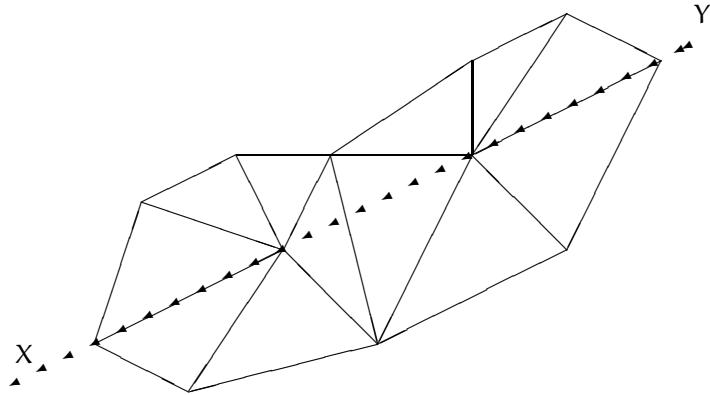
6 представляется как совокупность последовательностей, аналогичных только что рассмотренной последовательности, оценки качества формы переносятся на случай произвольного выполнения сдвигов. Это и доказывает свойство III.

Теперь перейдем к вопросу об оценке числа треугольников в зависимости от величины δ . Рассмотрим сетку, получающуюся равномерным измельчением исходной. Диаметр любого треугольника в полученной сетке меньше δ , а поэтому имеет площадь меньше $O(\delta^2)$. Следовательно, число треугольников в результирующей сетке $O(\delta^{-2})$.

Оценим число треугольников для сетки, которая имеет сгущение к линиям. Рассмотрим любое звено заданной ломаной. Пусть его длина l , а число треугольников, которое оно пересекает (включая пересечение по стороне) n . Тогда l пропорционально $n * \delta$, т. е. n пропорционально l/δ . Следовательно, число треугольников в такой сетке $O(\delta^{-1})$.

Теперь проанализируем алгоритм, описанный в предыдущем параграфе. Вопрос об оценке числа треугольников зависит от взаим-

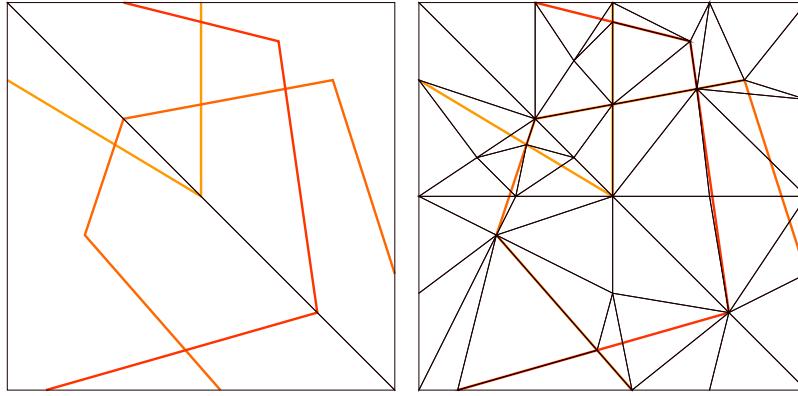
Рис. 9. Расположение треугольников, приводящее к возможному сгущению сетки



ного расположения заданных ломаных. Если $\delta \geq \varepsilon$, то алгоритм решения задачи слабой δ -аппроксимации ломаных около точек сближения непересекающихся звеньев может порождать сетку, сгущающуюся к линии. В частности, если существуют два звена, расположенные параллельно на расстоянии ε , то алгоритм произведет сетку со сгущением к звену, и поэтому имеющую число треугольников порядка $O(\delta^{-1})$. Поэтому в дальнейшем будем считать, что $\delta \ll \varepsilon$.

Свойство IV. Алгоритм, описанный в предыдущем параграфе, при $\delta \ll \varepsilon$ будет иметь число треугольников не больше $\log_2 \delta^{-1}$. Это следует непосредственно из следующей теоремы.

Теорема. Пусть дана грубая начальная конформная треугольная сетка, покрывающая заданную полигональную область Ω и точно аппроксимирующую границу области Ω . Описанный в предыдущем разделе алгоритм построения конформной квази-иерархической треугольной сетки, аппроксимирующей заданное множество ломаных с точностью $\delta \ll \varepsilon$, порождает сетку, сгущающуюся при достаточно малом q_0 к числу точек не больше $O(\log_2 \delta^{-1})$.

Рис. 10. Начальная сетка и сетка после этапа 1, $q_0 = 0.15$ 

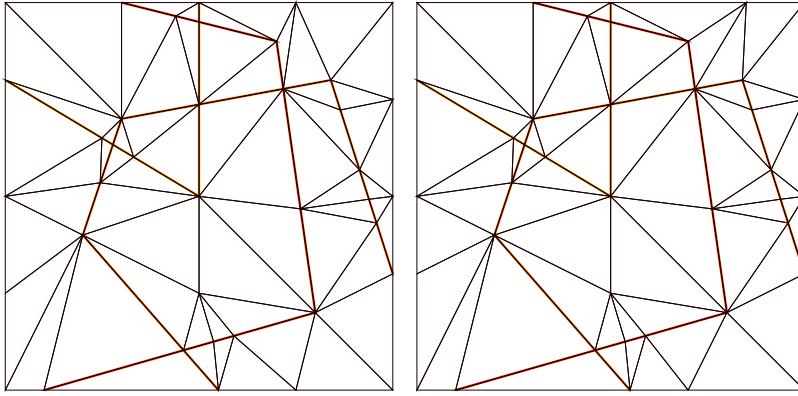
Действительно, рассмотрим сетку, сгущающуюся к одной точке. Для достижения условия, что диаметр треугольника вблизи точки сгущения стал меньше δ , необходимо совершить $O(\log_2 \delta^{-1})$ бисекций, а каждая бисекция порождает лишь ограниченное число треугольников по свойству BS . Следовательно, сетка с одной точкой сгущения имеет $O(\log_2 \delta^{-1})$ число треугольников. Если число точек сгущения $O(\log_2 \delta^{-1})$, то число треугольников в сетке имеет порядок $(\log_2^2 \delta^{-1})$.

Доказательство.

Из описания алгоритма легко видеть, что смысл первого этапа заключается в аппроксимации только лишь точек множества Z . Поэтому по его окончании мы можем (в худшем случае) получить сетку, имеющую сгущение лишь к некоторым точкам из множества Z , число которых конечно.

Так как третий этап алгоритма вообще не производит бисекций, то для доказательства теоремы достаточно показать, что именно второй этап алгоритма при достаточно малом δ производит сетку, сгущающуюся к числу точек порядка $O(\log_2 \delta^{-1})$.

Так как $\delta \ll \varepsilon$, число повторений шагов 2-1 — 2-3 не зависит от δ . Действительно, поскольку расстояние от любой неподвижной точки до любого звена, не содержащего эту точку, не меньше 0.9ε ,

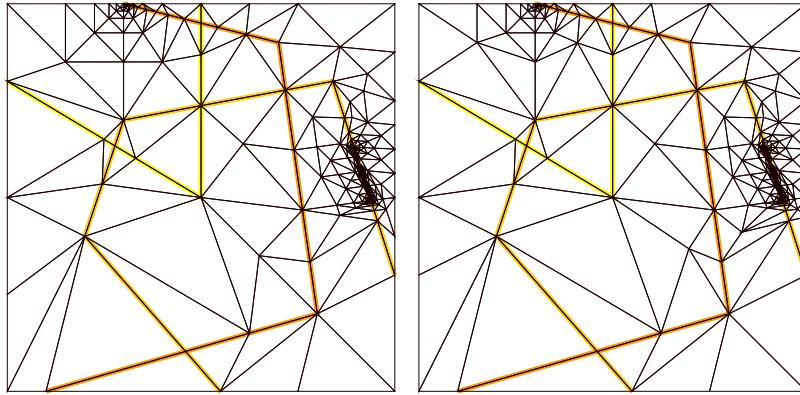
Рис. 11. Сетки после этапов 2 и 3, $q_0 = 0.15$ 

то измельчение треугольников, пересекающих звено, до диаметра 0.5ϵ обеспечит отсутствие у них неподвижных вершин. Получаем, что число повторений шагов 2-1 — 2-3 определяется лишь расположением задаваемых ломаных и не зависит от δ . Поэтому проанализируем лишь шаги 2-4 — 2-6. Пусть после шага 2-3 сетка имела качество формы \tilde{q} , а $q_0 = 0.1 * \tilde{q}$.

Рассмотрим любой отрезок ломаной, который перед шагом 2-4 не аппроксимирован ребрами треугольников, а его концы X и Y совпадают с узлами сетки. Не ограничивая общности, можно считать, что ни один из узлов сетки не лежит на этом отрезке, в противном случае, выделим из взятого отрезка подотрезок меньшей длины без внутренних сеточных узлов. Пусть число треугольников, пересекаемых отрезком XY равно m . Так как на первом этапе алгоритм производит сетку, сгущающуюся к некоторым (или никаким) из точек Z , то число m не превышает $O(\log_2 \delta^{-1})$.

Начав работу с этим отрезком, алгоритм станет сносить на него вершины. В силу малости q_0 шаги 2-4 — 2-6 будут проходить без биссектций до тех пор, пока среди m пересекающих XY треугольников не выделится m_1 пар треугольников, имеющих общее ребро, пересекающее XY , и противоположные общему ребру вершины, лежащие на XY (рис. 9). Получив множество таких пар, алгоритм

Рис. 12. Сетки после этапов 2 и 3, $q_0 = 0.21$. Сгущения порождены шагами 2.4-2.6



попытается снести на ломанную ближайшую общую вершину двух треугольников. Это не всегда возможно, так как сносимая вершина принадлежит суперэлементам сразу двух вершин, уже снесенных на XY в результате последовательных процессов аппроксимации ломаной, происходящих по разные стороны от рассматриваемой пары треугольников. Ограничение снизу на качество формы этих суперэлементов может воспрепятствовать сносу вершины, обеспечивающей аппроксимацию ломаной. В этом случае начнется процесс биссекций, который будет порождать сетку с сгущением к точке пересечения общего ребра исследуемой пары треугольников и отрезка XY . Поэтому в результате работы алгоритма мы получим сетку, сгущающуюся не более к m_1 точкам, число которых не более, чем $O(\log_2 \delta^{-1})$. \square

Отметим практическую важность данной теоремы: улучшение аппроксимации ломаных ($\delta \rightarrow 0$) требует умеренного роста числа элементов сетки, что позволяет строить сетки с небольшим числом треугольников.

5. Примеры сеток, построенных с помощью алгоритма.

Рассмотрим два примера работы нашего алгоритма, в которых иллюстрируется свойство IV. Напомним, что оценка числа треугольников в теореме является оценкой сверху: появление точек сгущения на этапах 1 и 2 зависит от взаимного расположения заданных ломаных и начальной сетки и от порога качества формы треугольников q_0 . Как правило, при умеренных значениях q_0 сгущения не происходит ни на этапе 1, ни на этапе 2 даже при малых δ , как это продемонстрировано на рис. 10–11 для $\delta = 0.008$ и $q_0 = 0.15$. Более высокий порог качества формы приводит к появлению точек сгущения, см. рис. 12. Отметим, что малость δ влияет лишь на глубину сгущения, но не на появление точек сгущения.

6. Заключение

В работе рассмотрено несколько постановок оптимизационных задач по генерации сеток, удовлетворяющих множественным ограничениям. Для одной из них предложен и проанализирован алгоритм построения квази-иерархической сетки, которая может рассматриваться как некоторое приближение к решению в соответствующем классе сеток. Характерной особенностью алгоритма является его робастность, что достигается аппроксимацией ограничений с точностью δ и возможностью локального δ -возмущения заданных ограничений. Улучшение аппроксимации ломаных ($\delta \rightarrow 0$) приводит к росту числа элементов сетки, не превышающему $\log_2^2 \delta^{-1}$.

Список литературы

- [1] Скворцов А. Триангуляция Делонэ и ее применение. — Томск, Издательство Том. ун-та. 2002.
- [2] Шайдуров В. В. Многосеточные методы конечных элементов. — Москва, Наука. 1989.
- [3] Baker B., Grosse E., Rafferty C. S. Nonobtuse triangulation of polygons // *Disc. and Comput. Geom.* 1998. V. 3. P. 147–168.
- [4] Bänsch E. Local mesh refinement in 2 and 3 dimensions // *IMPACT of Computing in Science and Engrg.* 1991. V. 3. P. 181–191.

- [5] Bern M., Eppstein D., Gilbert J. R. Provably good mesh generation // *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. 1990. IEEE. P. 231–241.
- [6] Buscaglia G., Dari E. Anisotropic mesh optimization and its application in adaptivity // *Inter. J. Numer. Meth. Engng.* 1997. V. 40. P. 4119–4136.
- [7] Carey G. Computational grids. Generation, adaptation, and solution strategies. — London. Taylor and Francis. 1997.
- [8] Chew L. P. Guaranteed-quality mesh generator for curved surfaces // *Proceedings of the Ninth Annual Symposium on Computational Geometry*. 1993. ACM. P. 274–280.
- [9] Frey P., George P.-L. Maillages: applications aux éléments finis. — Paris. Hermès Science Publications. 1999.
- [10] Liseikin V. Grid generation methods. — Springer. Berlin Heidelberg. 1999.
- [11] Rivara M. Mesh refinement processes based on the generalized bisection of simplexes // *SIAM J. Numer. Anal.* 1984. V. 21. P. 604–613.
- [12] Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generator // *Journal of Algorithms*. 1995. V. 18. N. 3. P. 548–585.
- [13] Shewchuk J. Delaunay refinement algorithms for triangular mesh generation // *Computational Geometry: theory and applications*. 2002. V. 22 (1-3). P. 21–74. См. также <http://www-2.cs.cmu.edu/~jrs/jrspapers.html#cdt>
- [14] Vassilevski Yu., Lipnikov K. Adaptive algorithm for generation of quasi-optimal meshes // *Comp. Math. Math. Phys.*. 1999. V. 39. P. 1532–1551.
- [15] <http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html>
- [16] http://www.engr.usask.ca/~macphed/finite/fe_resources/mesh.html

СОДЕРЖАНИЕ

Предисловие	3
Вл. В. Воеводин	
<i>НИВЦ МГУ: широким фронтом совместных дел</i>	5
А. В. Адинец, Н. А. Сахарных	
<i>О системе программирования вычислений общего назначения на графических процессорах</i>	25
А. С. Антонов	
<i>Влияние характеристик программно-аппаратной среды на производительность приложений</i>	51
Г. А. Бочаров, Н. А. Медведева	
<i>Численные алгоритмы анализа чувствительности и сложности описания в задачах идентификации моделей математической иммунологии</i>	65
Вад. В. Воеводин, С. И. Соболев, А. В. Фролов	
<i>Архитектура и принципы реализации коллективного банка тестов в сети Интернет</i>	87
П. А. Гаврилушкин	
<i>Структура и производительность подсистем памяти современных вычислительных платформ</i>	95
С. А. Горейнов	
<i>Об оценке сходимости метода бидиагонализации</i>	107
А. А. Данилов	
<i>Построение тетраэдральных сеток для областей с заданными поверхностными триангуляциями</i>	115
С. А. Жуматий	
<i>Комплексный подход к обслуживанию вычислительных кластеров</i>	127
С. А. Жуматий, С. И. Соболев	
<i>Оценка загруженности компьютера в различных UNIX-системах</i>	139

О. С. Лебедева, Е. Е. Тыртышников	
<i>Размерности коммутативных матричных алгебр</i>	147
Д. А. Никитенко	
<i>Ton50. Рейтинг наиболее производительных</i>	
<i>вычислительных систем СНГ</i>	169
К. Д. Никитин	
<i>Технология расчета течений со свободной границей</i>	
<i>с использованием динамических гексаэдральных сеток</i> ..	179
Д. В. Савостьянов	
<i>Алгоритмы слепого разделения источников</i>	
<i>в пакетном режиме</i>	195
Д. В. Савостьянов, С. Л. Ставцев, Е. Е. Тыртышников	
<i>Об использовании мозаично-скелетных</i>	
<i>аппроксимаций при решении гиперсингулярных</i>	
<i>интегральных уравнений</i>	225
С. И. Соболев	
<i>Эффективная работа в распределенных</i>	
<i>вычислительных средах</i>	245
К. С. Стефанов	
<i>Технологический инструментарий для построения систем</i>	
<i>анализа и преобразования структуры программ</i>	255
Е. Е. Тыртышников	
<i>Метод скачков и аппроксимации Паде</i>	271
В. Н. Чугунов	
<i>Об алгоритме построения конформной квази-иерархической</i>	
<i>треугольной сетки, слабо δ-аппроксимирующей</i>	
<i>заданные ломаные</i>	283

Научное издание

ЧИСЛЕННЫЕ МЕТОДЫ,
ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ
И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Сборник научных трудов
под ред. Вл. В. Воеводина и Е. Е. Тыртышникова

Оригинал-макет изготовлен в ИВМ РАН

Подписано в печать 28.01.2008 г.

Формат $60 \times 84 \frac{1}{16}$. Печать офсетная. Бумага офсетная №1

Усл. печ. л. 20,0. Уч.-изд. л. 21,5. Тираж 250 экз. Заказ №1.

Ордена «Знак почета» Издательство Московского университета

125009, Москва, ул. Б. Никитская, 5/7.