Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Нижегородский государственный университет
им. Н.И. Лобачевского

ВЫСОКОПРОЗВОДИТЕЛЬНЫЕ ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА КЛАСТЕРНЫХ СИСТЕМАХ

Материалы пятого Международного
научно-практического семинара

22–25 ноября 2005 г.

Издательство Нижегородского госуниверситета
Нижний Новгород
2005

УДК 681.3.012:51
ББК 32.973.26–018.2:22
В 93

В93 **Высокопроизводительные параллельные вычисления на кластерных системах.** Материалы пятого Международного научно-практического семинара / Под ред. проф. **Р.Г. Стронгина.** Нижний Новгород: Изд-во Нижегородского госуниверситета, 2005. 253 с.

Сборник сформирован по итогам научного семинара, посвященного теоретической и практической проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа. Семинар проходил в Нижнем Новгороде 22–25 ноября 2005 года.

Вошедшие в сборник материалы семинара представляют интерес для преподавателей и научных сотрудников высших учебных заведений, а также для инженеров, аспирантов и студентов вузов.

Отв. за выпуск к.ф.-м.н., доцент **В.А. Гришагин**

ББК 32.973.26–018.2:22

Поддержка семинара

Российский фонд фундаментальных исследований

Компания Intel Technologies

Фонд содействия развитию малых форм предприятий в научно-технической сфере

Компания eLine

22–25 ноября 2005 года Вычислительный Центр РАН, Институт математического моделирования РАН, Нижегородский государственный университет им. Н.И. Лобачевского провели в Нижнем Новгороде второй Международный научно-практический семинар и Всероссийскую молодежную школу «Высокопроизводительные параллельные вычисления на кластерных системах».

Главная направленность семинара и школы состояла в обсуждении основных аспектов организации высокопроизводительных вычислений в кластерных в кластерных компьютерных системах, активизации научно-практической деятельности исследователей в этой перспективной области развития современных средств вычислительной техники, обмене опытом учебно-образовательной деятельности при подготовке специалистов в области параллельных вычислений.

Проблематика семинара нацелена на рассмотрение следующих вопросов параллельных вычислений:

- принципы построения кластерных вычислительных систем;
- методы управления параллельных вычислениями в кластерных системах;
- параллельные алгоритмы решения сложных вычислительных задач;
- программные среды и средства для разработки параллельных программ;
- прикладные программные системы параллельных вычислений;
- методы анализа и оценки эффективности параллельных программ;
- проблемы подготовки специалистов в области параллельных вычислений.

В данный сборник включены материалы сообщений, которые представлены как в виде статей, так и в виде кратких тезисов. Материалы сборника упорядочены в алфавитном порядке по фамилии первого автора.

ОРГКОМИТЕТ СЕМИНАРА

Стронгин Р.Г.	председатель оргкомитета, ректор ННГУ, профессор, д.ф.-м.н.
Гергель В.П.	заместитель председателя оргкомитета, профессор, д.т.н., ННГУ
Батищев Д.И.	профессор, д.т.н., ННГУ
Евтушенко Ю.Г.	директор Вычислительного центра РАН, чл.-корр. РАН
Нестеренко Л.В.	директор по стратегии и технологиям Нижегородской лаборатории Intel (INNL), к.ф.-м.н.
Сергеев Я.Д.	профессор, д.ф.-м.н., ННГУ, Калабрийский университет (Италия)
Четверушкин Б.Н.	директор Института математического моделирования РАН, чл.-корр. РАН
Гришагин В.А.	ученый секретарь семинара, к.ф.-м.н., ННГУ

АРХИТЕКТУРА И СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРНЫХ СИСТЕМ

А.И. Аветисян, О.И. Самоваров, Д.А. Грушин

Институт системного программирования Российской академии наук, Москва

1. Введение

В ИСП РАН ведутся исследования по параллельному программированию и высокопроизводительным кластерным технологиям, поддерживаемые Российским Фондом Фундаментальных Исследований, Министерством промышленности науки и технологии Российской Федерации, Российской Академией наук и др. В рамках этих проектов ведутся работы, связанные с проектированием и построением высокопроизводительных кластерных систем, разработкой системного программного обеспечения и средств параллельного программирования для таких систем. Кроме того, ведутся работы связанные с интеграцией кластерных ресурсов в инфраструктуру Grid.

В данной работе затрагиваются вопросы архитектуры, проектирования и др. особенности построения высокопроизводительных кластерных систем, приводится обзор системного ПО, используемого при построении в ИСП РАН кластеров, а также рассказывается о высокопроизводительных кластерах построенных в ИСП РАН.

2. Особенности архитектуры вычислительных кластерных систем

На современном этапе в тех областях, интенсивное развитие которых невозможно представить без высокопроизводительных вычислений широко используются модульные вычислительные системы, построенные на базе компонент общего назначения с использованием свободно распространяемого системного программного обеспечения с открытым исходным кодом называемые кластерами. Архитектура таких вычислительных систем представляет собой множество однородных по аппаратуре и настройкам системного ПО модулей, объединенных специальной высокопроизводительной коммуникационной средой, имеющих единый центр доступа, администрирования, мониторинга и работающих под управление специального системного программного обеспечения. Благодаря открытости и модульности архитектуры, использования распространенных на рынке компонент (процессоров,

материнских плат, оперативной памяти, дисковой памяти и т.д.), свободно распространяемого ПО такие вычислительные системы имеют рад преимуществ: высокая производительность, эффективность, простота наращиваемости мощностей, простота эксплуатации и обслуживания и т.д. однако наиболее привлекательным свойством таких вычислительных систем является оптимальное соотношения цена/производительность.

Однако для того чтобы на практике получить описанные преимущества при использовании вычислительной системы для решения задач конкретной прикладной области на конкретном программном комплексе необходимо решить ряд задач еще на этапе проектирования вычислительной системы. Так архитектура вычислительного кластера должна учитывать особенности, используемой прикладным пакетом модели вычислительной системы, параметры аппаратуры кластера должны быть согласованны с требованиями прикладного пакета к производительности узлов и характеристикам коммуникационного оборудования, используемые системные библиотеки установленные на кластере должны поддерживать оптимизации прикладного пакета и т.д. Решение данных задач требует глубокого понимания внутренней архитектуры прикладного пакета, используемых вычислительных параллельных алгоритмов, исследования современных технологий (многоядерных мультитредовых микропроцессоров, многопроцессорных платформ, высокопроизводительных сетевых технологий и т.д.) используемых при построении кластерных систем, разработки аналитических моделей позволяющих прогнозировать характеристики кластера для конкретного прикладного продукта, практического опыта построения таких систем.

Таким образом, построение высокопроизводительного кластера представляет собой комплексную задачу, которая включает в себя несколько этапов. Одним из наиболее важных является этап проектирования кластера. На этом этапе требования к кластерной вычислительной системе рассматриваться как его характеристики (производительность, эффективность, масштабируемость и т.д.) [1]. В этом случае, в соответствии с требуемыми характеристиками кластера и дополнительными требованиями заказчика (в том числе и бюджетом проекта), производится проектировочный расчет, и выбираются значения параметров аппаратной части кластера: выбор параметров вычислительного узла (разрядность, количество процессоров, объем памяти, объем кэша и др.), количество вычислительных узлов, характеристики коммуника-

ционного оборудования, выбираются управляющий узел и управляющая сеть. В общем случае при проектировании кластера его характеристики задаются для теста HPL (High Performance Linpack). Для случаев, когда кластер проектируется под определенный пакет прикладных программ, вместо теста HPL используются тесты, характеризующие соответствующий класс задач. После определения проектных параметров и с учетом дополнительных требований (например, бюджет проекта) принимаются конструкционные решения о компоновке, системе энергоснабжения и охлаждения кластера.

Следующим важным этапом при построении кластера является установка и настройка системного программного обеспечения.

3. Системное программное обеспечение кластера

Важной частью вычислительного кластера является его системного программное обеспечение, от выбора которого, правильной установки и конфигурации зависят не только его технические характеристики (производительность, эффективность масштабируемость), но и такие потребительские качества как удобство эксплуатации, надежность, безопасность. Известно, что не существует специализированных операционных систем для кластеров. Узлы кластерной вычислительной системы работают под управлением собственной независимой локальной копии операционной системы, в качестве которой, как правило, используется тот или иной дистрибутив ОС Linux. С учетом этих особенностей, для решения задачи по обеспечению возможности пользователей работать с кластером как с единой вычислительной системой коллективного пользования в настоящее время предлагается использовать специализированные системы управления кластерами. Такие системы работают совместно с базовой, установленной на узлы операционной системой и предлагают средства по инсталляции кластера, централизованного управления и мониторинга кластером, по обеспечению управления потоком заданий пользователей, выделению ресурсов кластера для их решения и т.д.

В своих работах связанных с построением высокопроизводительных кластеров мы используем системное программное обеспечение, построенное на базе операционной системы Linux RedHat и системы управления кластерами OSCAR[2]. В состав OSCAR входят средства позволяющие решать следующие задачи:

1. Инсталляция и модернизация кластера. Поддержание целостности системного окружения

Системное окружение (ОС и средства обеспечивающие работу

кластера), вычислительных узлов кластера устанавливается с использованием средств, входящих состав OSCAR. Особенностью используемых пакетов является то, что инсталляция операционного окружения осуществляется с единого, заранее настроенного по производительности и надежности работы образа ОС. Это позволяет:

- обеспечить максимально возможную однородность системного окружения вычислительных узлов;
- проводить модернизацию кластера, обеспечивая принцип однородности системного окружения;
- обеспечить максимально быстрое восстановление системного окружения вычислительных узлов в случаях сбоев.

2. Централизованное управление и администрирование

OSCAR предлагает средства, которые позволяют управлять кластером и производить его администрирование из единого центра – управляющего узла. Это обеспечивает централизованное:

- выполнение коллективных операций;
- управление параллельным окружением;
- управление бюджетами пользователей.

3. Коллективный доступ к ресурсам кластера

Коллективный доступ к ресурсам кластера в OSCAR обеспечивается использованием системы пакетной обработки заданий OpenPBS и планировщик заданий MAUI, особенностями которых являются:

- поддержка системы очередей заданий;
- возможность разделять общее вычислительное поле кластера на логические подмножества;
- возможности реализации гибких политик по выделению ресурсов в зависимости от требований задачи к ресурсам, загрузки кластера, времени суток, дня недели и т.д.;
- существующая система приоритетов позволяет реализовать различные сценарии доступа к ресурсам кластера для разных групп пользователей;
- наличие backfill алгоритмов обеспечивающих максимально равномерную загрузку ресурсов кластера.

4. Мониторинг кластера

Система управления кластером OSCAR предоставляет средства мониторинга уровня операционной системы и уровня системы пакетной обработки заданий. Первое обеспечивается системой Ganglia и позволяет получать информацию о загрузке процессоров вычислительных узлов, памяти, сетевых интерфейсов использовании локальных

дисков и т.д. В качестве средств мониторинга кластера уровня системы пакетной обработки в OSCAR предлагается система CLUMON которая работает совместно с OpenPBS и позволяет получать информацию о занятости ресурсов кластера, состоянии очередей заданий, прохождении заданий пользователей, и т.д. Обе системы имеют Web интерфейс.

Как член ассоциации Gelato [3] ИСП РАН ведет работу по интеграции собственных программных продуктов в систему управления OSCAR. В дополнение к стандартным, входящим в OSCAR пакетам на кластеры устанавливаются разработанные в ИСП РАН средства:

- активного мониторинга аппаратуры кластера;
- обеспечения доступа к ресурсам кластера через Web.

Система активного мониторинга аппаратной части кластера позволяет определять параметры аппаратуры кластера влияющие на его работоспособность: температуры процессоров, материнской платы, винчестеров и др., частоту вращения вентиляторов процессоров, корпуса, и др., наиболее критичные рабочие напряжения: напряжение ядра процессора, и пр. Администратор кластера имеет возможность выбрать интересующие его параметры аппаратуры кластера, выбрать узел или узлы кластера и передав запрос получить таблицу в которой отражаются режимы работы узлов кластера. При этом зеленый фон в таблице означает то, что параметр находится в допустимых пределах, красный цвет фона сигнализирует об аварийной ситуации, желтый говорит о том, что информация, о данном параметре не доступна.

В настоящее время, для того чтобы получить ресурсы кластера пользователь должен иметь непосредственный доступ к управляющему узлу кластера через один из протоколов удаленного доступа, например ssh. При этом взаимодействие с кластером происходит через терминальный вызов процедур командного интерпретатора и системы пакетной обработки заданий. Это не всегда удобно и безопасно. Для повышения удобства и безопасности эксплуатации кластера, необходимо обеспечить более высокий уровень абстракции взаимодействия пользователя и системы управления кластером.

Clusterweb – система разработана в ИСП РАН и обеспечивает доступ к ресурсам кластера в нотациях функционирующей системы пакетной обработки заданий, при этом внутренняя архитектура кластера для пользователя остается «черным ящиком» [4]. Поскольку система имеет Web интерфейс, нет необходимости, заботится об организации терминального доступа на управляющую машину кластера по тем или иным протоколам, решать административные вопросы, связанные с

возможностью их использования, достаточно иметь Web-броузер. Clusterweb позволяет: создавать сценарии выполнения задания пользователя, выбирать требуемое параллельное окружение, обеспечить отправку и постановку в очередь на выполнение заданий пользователя, транспортировку стандартных потоков вывода на рабочую станцию пользователя. Clusterweb реализован на основе Java и XML с использованием сервера Tomcat. ClusterWeb может использоваться с различными системами управления кластерами, которые имеют интерфейс командной строки или API.

4. Высокопроизводительные кластеры, построенные ИСП РАН

ИСП РАН совместно с компанией C.I.Technology были разработаны и построены несколько высокопроизводительных кластерных систем малой и средней мощности [5]. Построение кластеров включает в себя полный цикл работ связанных с проектированием, наладкой аппаратной части, установкой и настройкой системного программного обеспечения, тестирования, документирования, введения в эксплуатацию, обучения персонала, обеспечения дальнейшей поддержки функционирования и развития систем.

Первый кластер был установлен и введен в ИСП РАН в марте 2002 г. Система построена на базе процессоров AMD Athlon 1800+ с использованием в качестве вычислительной сети Myrinet 2000. Кластер имеет 8 двухпроцессорных вычислительных узлов (всего 16 процессоров), 8 GB оперативной памяти, 72 GB дисковой памяти на управляющем узле и 160 GB на локальных дисках вычислительных узлов. Пиковая производительность (Rpic) кластера составляет 48 Gflops, максимальная производительность по HPL(Rmax) составляет 29.5 Gflops и эффективность (Rmax/Rpic) 61,4%. Доступ к ресурсам кластера через ClusterWeb открыт по адресу: http://omega-new.ispras.ru:8080/clusterweb/, к системе мониторинга http://omega-new.ispras.ru/clumon/, к системе мониторинга аппаратуры: http://omega-new.ispras.ru/cgi-bin/hmon.cgi.

В июне 2003 построен и введен в эксплуатацию кластер Вычислительного центра им. А.А. Дородницына РАН. Кластер построен на базе процессоров Intel Xeon 2.6 с использованием в качестве вычислительной сети Myrinet 2000. Кластер состоит из 8 двухпроцессорных вычислительных узлов (всего 16 процессоров), 32 GB оперативной памяти, 144 GB дисковой памяти на управляющем узле и 288 GB дисковой памяти на локальных накопителях вычислительных узлов. Пиковая производительность (Rpic) кластера составляет 83.2 Gflops, максимальная производительность по HPL (Rmax) составляет 59.27 Gflops и эффек-

тивность (Rmax/Rpic) 71,2%.

В 2003 году был выигран государственный тендер и построен кластер для Кубанского Университета. Система построена на базе процессоров AMD Athlon 2400+ с использованием в качестве вычислительной сети Myrinet 2000. Кластер состоит из 8 двухпроцессорных вычислительных узлов (всего 16 процессоров), 16 GB оперативной памяти, 40 GB дисковой памяти на управляющем узле и 320 GB на локальных дисках вычислительных узлов. Пиковая производительность (Rpic) кластера составляет 64 Gflops, максимальная производительность по HPL(Rmax) составляет 35,54 Gflops и эффективность (Rmax/Rpic) 55%.

В марте 2004 года был выигран международный конкурс на получение гаранта МНТЦ (Международный научно-технический центр) на построение кластера для Национальной академии наук Армении. В мае 2004 года кластер построен и введен в эксплуатацию. Кластер установлен в Институте информатики и проблем автоматизации Национальной Академии Наук Армении. Система построена на базе процессоров Intel Xeon 3.06 с использованием в качестве вычислительной сети Myrinet 2000. Вычислительные узлы построены на базе серверной материнской платы SW7501CW2, важной особенностью которой является наличие высокоскоростной шины PCI–X (64-bit/133MHz), обеспечивающей эффективное согласование производительности узлов с вычислительной сетью Myrinet 2000, а также наличие интегрированного интерфейса Gigabit Ethernet. Вычислительные узлы собраны в конструкционных модулях размерностью 2U. При построении вычислительных узлов кластера было применено оригинальное техническое решение, позволяющее исключить использование «райзеров», что значительно повысило надежность. Кластер состоит из 64 двухпроцессорных вычислительных узлов (всего 128 процессоров), 64 GB оперативной памяти, 240 GB дисковой памяти на управляющем узле и 2.56 TB на локальных дисках вычислительных узлов. Пиковая производительность (Rpic) кластера составляет 783,36 Gflops, максимальная производительность по HPL(Rmax) составляет 483,6 Gflops и эффективность (Rmax/Rpic) 61,7%. В ближайшее время планируется расширить оперативную память кластера, что позволит достичь эффективности 70%.

В настоящее время ведутся работы по построению высокопроизводительного вычислительного Myrinet кластера для Тамбовского Государственного Университета, который планируется ввести в эксплуатацию до конца текущего года.

Литература

1. *Sergey Gaissaryan*, *Arutyun Avetisyan*, *Oleg Samovarov*, *Dmitry Grushin*. Comparative Analysis of High-Performance Clusters' Communication Environments Using HPL Test. // High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference on (HPCAsia'04), July 20-22, IEEE Computer Society, 2004. Omiya Sonic City, Tokyo, Japan, pp. 473-476.
2. *Arutyun Avetisyan*, *Oleg Samovarov*, *Dmitry Grushin*, *Andrey Ryzhov*. «Clusterweb – a WEB-based cluster management interface». M. Estrada, A. Gelbukh (Eds.) // Avances en la Ciencia de la Computacion, ENC'04, Colima, Mexico, pp. 489-495.
3. OSCAR. http://www.gelato.org/software/view.php?id=1_18
4. Gelato. http://www.gelato.org/
5. ИСП РАН. http://www.ispras.ru/news/armcluster.html

ОРГАНИЗАЦИЯ ОБМЕНА ДАННЫМИ НА ПАРАЛЛЕЛЬНЫХ КОМПЬЮТЕРАХ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Е.В. Адуцкевич

Институт математики НАН Беларуси, *Минск* (*Беларусь*)

Введение

Для отображения алгоритмов, заданных последовательными программами, на параллельные компьютеры с распределенной памятью требуется распределить операции и данные алгоритма между процессорами, а также установить порядок выполнения операций и обмена данными. При выполнении алгоритмов на таких компьютерах реализация коммуникаций между процессорами имеет, как правило, большие накладные расходы. Поэтому распределение операций и данных между процессорами следует производить таким образом, чтобы уменьшить количество и объем коммуникаций; кроме того, следует оптимизировать структуру коммуникаций.

Задаче уменьшения коммуникационных затрат посвящено много исследований. Самый очевидный подход – разбиение на блоки независимых вычислений [1, 2]. Заметим, что на практике алгоритм далеко не всегда допускает декомпозицию на независимые части. Другой подход направлен на минимизацию обменов данными. Этого можно добиться через построение блочных версий алгоритма, при разбиении специальным образом пространства итераций гнезд циклов [1, 3]; в этом случае не затрагивается проблема размещения массивов данных и ее нужно

решать отдельно. Предлагаются также методы поиска начального (до начала выполнения вычислений) размещения операций и данных по процессорам, при котором во время выполнения программы требуется как можно меньше обменов данными [4, 5].

Мы будем рассматривать такую схему выполнения параллельной программы, при которой размещение операций и данных по процессорам установлено и не меняется в процессе выполнения программы. Даже оптимальное начальное распределение данных не исключает необходимости в обменах данными между процессорами, в локальной памяти которых хранятся данные, и процессорами, в которых эти данные переопределяются или используются как аргументы для вычислений. Известно, что на параллельных компьютерах с распределенной памятью структурированные коммуникации, такие как бродкаст (broadcast), разброс (scatter), сборка (gather), редукция (reduction), а также трансляция (translation) данных выполняются быстрее, чем большое количество коммуникаций точка-точка (point-to-point) между процессором, в локальной памяти которого хранится данное, и каждым процессором, который использует или переопределяет это данное. Поэтому желательно выявлять возможность организации таких коммуникаций. Некоторые пути решения этой задачи намечены в работе [6].

В настоящей работе развиваются идеи, предложенные в работе [6]. Сформулированы условия, позволяющие определять возможность организации часто используемых быстрых коммуникаций – бродкаста и трансляции данных, а также описан способ установления схемы обменов данными. Проведенные исследования могут быть использованы при автоматизированном распараллеливании программ.

Постановка задачи

Будем рассматривать аффинные гнезда циклов; в этом случае выражения индексов элементов массивов и границы изменения параметров циклов являются аффинными функциями от параметров циклов и внешних переменных.

Пусть в гнезде циклов имеется K операторов S_β и используется L массивов a_l. Область изменения параметров гнезда циклов для оператора S_β будем обозначать V_β; область изменения индексов l-го массива будем обозначать W_l. Обозначим n_β – число циклов, окружающих оператор S_β, v_l – размерность l-го массива; тогда $V_\beta \subset \mathbf{Z}^{n\beta}$, $W_l \subset \mathbf{Z}^{vl}$. Вектор параметров цикла будем обозначать $J \in \mathbf{Z}^{n\beta}$, вектор внешних переменных алгоритма – $N \in \mathbf{Z}^e$, e – число внешних переменных.

Реализацию (выполнение) оператора S_β при конкретных значениях β и вектора параметров цикла J будем называть операцией и обозначать $S_\beta(J)$.

Для упрощения обозначений будем считать, что все рассматриваемые далее аффинные функции имеют вид $f: V \to G$, $V \subset \mathbf{Z}^m$, $G \subset \mathbf{Z}^g$, $f(J) = AJ + BN + C$, где $A \in \mathbf{Z}^{g \times m}$, $B \in \mathbf{Z}^{g \times e}$, $C \in \mathbf{Z}^g$. Пусть Δ – элемент множества G. Будем рассматривать множество $V^\Delta = \{J \in V \mid f(J) = \Delta\}$. Множество V^Δ будем называть невырожденным, если $\dim(\ker A) \neq 0$ и существует вектор $J_0 \in V^\Delta$ такой, что $J_0 + u_i \in V$, где u_i – любой базисный вектор пересечения $\ker A \cap \mathbf{Z}^m$. Пусть заданы две аффинные функции $f_1: V \to G_1$ и $f_2: V \to G_2$, которые имеют вид $f_1(J) = A_1J + B_1N + C_1$ и $f_2(J) = A_2J + B_2N + C_2$; Δ_1 и Δ_2 – элементы множества G_1 и G_2 соответственно. Множество $V^{\Delta 1} \cap V^{\Delta 2}$ будем называть невырожденным, если $\dim(\ker A_1 \cap \ker A_2 \neq 0$ и существует вектор $J_0 \in V^{\Delta 1} \cap \cap V^{\Delta 2}$ такой, что $J_0 + u_i \in V$ где u_i – любой базисный вектор пересечения $\ker A_1 \cap \ker A_2 \cap \mathbf{Z}^m$.

Пусть индексы элементов l-го массива, встречающегося в операторе S_β и относящегося к q-му входу элементов этого массива в оператор, выражаются аффинной функцией $\overline{F}_{l,\beta,q}: V_\beta \to W_l$; зависимость операции $S_\beta(J)$ от операции $S_\alpha(I)$ задается аффинной функцией $\overline{\Phi}_{\alpha,\beta}: V_{\alpha,\beta} \to V_\alpha$, $V_{\alpha,\beta} \subseteq V_\beta$.

Распределение операций и массивов данных в r-мерном пространстве виртуальных процессоров зададим соответственно аффинными функциями $c_\xi^{(\beta)}: V_\beta \to \mathbf{Z}, 1 \le \beta \le K$ и $d_\xi^{(l)}: W_l \to \mathbf{Z}, 1 \le l \le L, 1 \le \xi \le r$. Порядок выполнения операций процессорами (таймирование) зададим аффинными функциями $t_\xi^{(\beta)}: V_\beta \to \mathbf{Z}, 1 \le \beta \le K, 1 \le \xi \le n - r$, где $n = \max\limits_{1 \le \beta \le K} \max n_\beta$. Векторы $(c_1^{(\beta)}(J), ..., c_r^{(\beta)}(J))$, $(d_1^{(l)}(F), ..., d_r^{(l)}(F))$ и $(t_1^{(\beta)}(J), ..., c_{n-r}^{(\beta)}(J))$ будем обозначать $(\bar{c}^{(\beta)}(J), (\bar{d}^{(l)}(F)$ и $(\bar{t}^{(\beta)}(J)$ соответственно.

Функции $c_\xi^{(\beta)}$, $d_\xi^{(l)}$ и $t_\xi^{(\beta)}$ являются одним из основных инструмен-

тов исследований в методах статического распараллеливания программ [1–6].

Условия отсутствия коммуникаций между процессорами можно в общем виде выразить равенствами $c_{\xi}^{(\beta)}(J) - d_{\xi}^{(l)}(\overline{F}_{l,\beta,q}(J)) = 0, J \in V_{\beta}$. Выполнение этих условий для всех l, β, q, ξ означает, что для любого значения вектора параметров цикла J расстояние между процессором, в котором выполняется операция, и процессором, в локальной памяти которого хранится необходимый для выполнения операции элемент массива, равно нулю. Следовательно, нет необходимости в обменах данными между процессорами. Если условия выполняются не для всех l, β, q, ξ, то требуется обмен данными в процессе выполнения программы.

Рассмотрим часто встречающиеся виды коммуникаций: точка-точка, бродкаст, трансляция. Коммуникация точка-точка – это обмен данными между двумя процессорами, в одном из которых данное хранится, а в другом используется как аргумент или вычисляется. Бродкаст (одновременное распространение) – это обмен данными между группой процессоров, в одном из которых данное хранится, а в других одновременно (на одной итерации) используется как аргумент. Трансляция – это передача данного от процессора к процессору в случае, если данное используется в разных процессорах по очереди.

На параллельных компьютерах с распределенной памятью желательно вместо большого количества коммуникаций точка-точка организовывать более быстрые коммуникации, такие как бродкаст и трансляция данных. Задачей данной работы является получение условий для выявления возможности организации таких коммуникаций и установление схемы обмена данными между процессорами.

2. Организация бродкаста и трансляции данных

Обозначим через $P(z_1, \ldots, z_r)$ процессор, размещенный в точке $(z_1, \ldots, z_r)$ пространства виртуальных процессоров. Согласно функциям $d_{\xi}^{(l)}, c_{\xi}^{(\beta)}$ и $t_{\xi}^{(\beta)}$ данные $a_l(\overline{F}_{l,\beta,q}(J))$ хранятся в локальной памяти процессоров $P(\overline{d}^{(l)}(\overline{F}_{l,\beta,q}(J)))$ и используются на итерациях $\overline{t}^{(\beta)}(J)$ в процессорах $P(\overline{c}^{(\beta)}(J))$.

Будем обозначать матрицу A в аффинных функциях $\overline{F}_{l,\beta,q}, \overline{\Phi}_{\alpha,\beta}$,

$c_\xi^{(\beta)}$ и $t_\xi^{(\beta)}$ через $F_{l,\beta,q}$, $\mathbf{\Phi}_{\alpha,\beta}$, $\boldsymbol{\pi}^{(\beta,\,\xi)}$ и $\boldsymbol{\tau}^{(\beta,\,\xi)}$ соответственно. Обозначим через $\Pi^{(\beta)}$ – матрицу, строки которой составлены из векторов $\boldsymbol{\pi}^{(\beta,\,\xi)}$, $1 \le \xi \le r$, через $T^{(\beta)}$ – матрицу, строки которой составлены из векторов $\boldsymbol{\tau}^{(\beta,\,\xi)}$, $1 \le \xi \le n - r$.

Пусть F – элемент множества W_l. Обозначим $V_{l,\beta,q}^{(F)} = \{J \in V_\beta \mid \overline{F}_{l,\beta,q}(J) = F\}$ – множество итераций исходного гнезда циклов, на которых на q-м входе массива a_l в оператор S_β используется одно и то же данное $a_l(F)$. Пусть T – элемент множества значений вектора $\bar{t}^{(\beta)}(J)$. Обозначим $V_\beta^{(T)} = \{J \in V_\beta \mid \bar{t}^{(\beta)}(J) = T\}$ – множество итераций исходного гнезда циклов, на которых выполняется оператор S_β и которые реализуются в пространстве виртуальных процессоров на итерации T. Будем еще рассматривать множество $V_{l,\beta,q}^{(F)} \cap V_\beta^{(T)}$.

Обозначим через $U_{l,\beta,q}^{(\beta)}$ матрицу, столбцы которой составлены из базисных векторов пересечения $\ker F_{l,\beta,q} \cap \ker T^{(\beta)} \cap \mathbf{Z}^{n_\beta}$.

В следующем утверждении сформулированы условия, определяющие возможность организации бродкаста данных.

Утверждение 1. На итерации T можно организовать бродкаст данного $a_l(F)$ от процессора $P(\overline{d}^{(l)}(F))$ к процессорам $P(\overline{c}^{(\beta)}(J))$, $J \in V_{l,\beta,q}^{(F)} \cap V_\beta^{(T)}$, если функция $\overline{F}_{l,\beta,q}$ встречается в правой части оператора S_β, множество $P(\overline{c}^{(\beta)}(J)), J \in V_{l,\beta,q}^{(F)} \cap V_\beta^{(T)}$, является невырожденным, и выполняется одно из следующих условий:

а) массив a_l встречается только в правых частях операторов алгоритма,

б) для истинной зависимости, порожденной q-м входом массива a_l в оператор S_β и задаваемой функцией зависимостей $\overline{\mathbf{\Phi}}_{\alpha,\beta}$ выполняется условие $\mathbf{\Phi}_{\alpha,\beta} U_{l,\beta,q}^{(\beta)} = 0$.

Обозначим через $U_{l,\beta,q}$ матрицу, столбцы которой составлены из базисных векторов пересечения $\ker F_{l,\beta,q} \cap \mathbf{Z}^{n_\beta}$; через $U^{(\beta)}$ матрицу, столбцы которой составлены из базисных векторов пересечения ker

$T^{(\beta)} \cap \mathbf{Z}^{n_\beta}$; через T_i различные значения $\bar{t}^{(\beta)}(J), J \in V^{(F)}_{l,\beta,q}$.

В следующих утверждениях сформулированы условия, определяющие возможность организации трансляции данного. В утверждении 2 исследуется случай, когда данное используется процессорами как аргумент на разных итерациях. Передача данного осуществляется на нескольких итерациях. В утверждении 3 исследуется случай, когда данное используется как аргумент и переопределяется процессорами по очереди на одной итерации. Передача данного осуществляется на одной итерации, но со сдвигом по времени при выполнении программы.

Утверждение 2. На итерациях T_i можно организовать трансляцию данного $a_l(F)$ между процессорами $P(\bar{c}^{(\beta)}(J)), J \in V^{(F)}_{l,\beta,q} \cap V^{(T)}_{\beta}$, если функция $\overline{F}_{l,\beta,q}$ встречается в правой части оператора S_β, множество $V^{(F)}_{l,\beta,q}$ является невырожденным, выполняются условия $T^{(\beta)}\ U_{l,\beta,q} \neq 0$, $\Pi^{(\beta)} U_{l,\beta,q} \neq 0$, $\operatorname{rank}\begin{pmatrix} F_{l,\beta,q} \\ T^{(\beta)} \end{pmatrix} = n_\beta$ и одно из следующих условий:

а) массив a_l встречается только в правых частях операторов алгоритма,

б) для истинной зависимости, порожденной q-м входом массива a_l в оператор S_β, и задаваемой функцией зависимостей $\overline{\Phi}_{\alpha,\beta}$ выполняется условие $\Phi_{\alpha,\beta} U^{(\beta)}_{l,\beta,q} = 0$.

Утверждение 3. Пусть элементы массива a_l используются в качестве аргументов только оператором S_β, условия отсутствия коммуникаций между процессорами не выполняются для двух фиксированных наборов индексов l, β, q_1, ξ и l, β, q_2, ξ $(q_1 \neq q_2)$, функция $\overline{F}_{l,\beta,q_1}$ стоит в левой части оператора S_β, функция $\overline{F}_{l,\beta,q_2}$ – в правой части, и существует истинная зависимость, порожденная входами q_1 и q_2 массива a_l в оператор S_β. Пусть множества $V^{(T_i)}_{\beta}$ являются невырожденными, где T_i – различные значения $\bar{t}^{(\beta)}(I), I \in V^{(F)}_{l,\beta,q_1}$, и выполняется условие $\Pi^{(\beta)} U^{(\beta)} \neq 0$.

Тогда можно организовать трансляцию данного $a_l(F)$ на каждой итерации T_i между процессорами $P(\bar{c}^{(\beta)}(J)), J \in V_{\beta}^{(T_i)}$.

Замечание. При выполнении условий утверждения 2 до трансляции данного $a_l(F)$ необходимо осуществить передачу $a_l(F)$ от процессора $P(\bar{d}^{(\beta)}(F))$ к процессору $P(\bar{c}^{(\beta)}(J_0))$ такому, что $\bar{t}^{(\beta)}(J_0)) = T_0$ лексикографически наименьший вектор из векторов T_i . Трансляцию следует осуществлять согласно лексикографической упорядоченности векторов T_i . При выполнении условий утверждения 3 до начала трансляции данного $a_l(F)$ необходимо осуществить передачу $a_l(F)$ от процессора $P(\bar{d}^{(\beta)}(F))$ к процессору $P(\bar{c}^{(\beta)}(J_1))$ такому, что $\bar{c}^{(\beta)}(J_1)$ лексикографически наименьший вектор из векторов $\bar{c}^{(\beta)}(J)$; после трансляции необходимо осуществить передачу $a_l(F)$ от процессора $P(\bar{c}^{(\beta)}(J_2))$ такого, что $\bar{c}^{(\beta)}(J_2)$ лексикографически наибольший вектор из векторов $\bar{c}^{(\beta)}(J)$, к процессору $P(\bar{d}^{(\beta)}(F))$. Трансляцию следует осуществлять согласно лексикографической упорядоченности векторов $\bar{c}^{(\beta)}(J), J \in V_{\beta}^{(T_i)}$.

3. Установление схемы обмена данными

Рассмотрим задачу установления схемы обмена данными. Пусть для аффинного гнезда циклов найдены функции $c_{\xi}^{(\beta)}, d_{\xi}^{(l)}$ и $t_{\xi}^{(\beta)}$, т.е. найдено распределение операций и данных по процессорам и установлен порядок выполнения операций процессорами (см. например [1, 4, 5]). Пусть для некоторого фиксированного набора l, β, q, ξ условия отсутствия коммуникаций между процессорами не выполняются.

Утверждения 1–3 позволяют определить группы процессоров, между которыми можно организовать бродкаст или трансляцию данных. В случаях, неоговоренных утверждениями 1–3, на итерации T можно организовать коммуникацию точка-точка между процессорами $P(\bar{d}^{(l)}(\bar{F}_{l,\beta,q}(J)))$ и $P(\bar{c}^{(\beta)}(J))$ для передачи данного $a_l(\bar{F}_{l,\beta,q}(J))$, где вектор J такой, что $\bar{t}^{(\beta)}(J) = T$.

Между процессорами каждой группы, для которой установлен вид коммуникации, обмен данными следует организовать, пользуясь сле-

дующими правилами очередности передачи и приема данных.

Пусть данное передается от процессора, в локальной памяти которого оно хранится. Передачу данного следует осуществлять после того, как будут выполнены все операции, которые его переопределяют и выполняются в исходной программе до операции $S_\beta(J)$, и после того, как процессор осуществит прием результатов этих операций от других процессоров. Пусть данное передается от процессора, в котором оно переопределяется. Передачу данного следует осуществлять после выполнения операции $S_\beta(J)$. Пусть данное передается от процессора, который использует его в качестве аргумента. Передачу данного следует осуществлять в начале итерации.

Пусть данное передается процессору, в локальной памяти которого оно хранится. Прием данного следует осуществлять до того, как будут выполняться все операции, которые используют это данное в качестве аргумента и выполняются в исходной программе после операции $S_\beta(J)$, и до того, как процессор осуществит передачу аргументов этих операций другим процессорам. Пусть данное передается процессору, который использует его в качестве аргумента. Прием данного следует осуществлять до выполнения операции $S_\beta(J)$.

Литература

1. *Воеводин В.В.*, *Воеводин Вл.В.* Параллельные вычисления. Санкт-Петербург. БХВ-Петербург. 2002. 608 с.

2. *Лиходед Н.А.* Отображение аффинных гнезд циклов на независимые процессоры. Кибернетика и системный анализ. 2003. 3. С. 169–179.

3. *Lim A.W.*, *Liao S.-W.*, *Lam M.S.* Blocking and array contraction across arbitrary nested loops using affine partitioning. Proceedings of the ACM SIGPLAN Simposium on Principles and Practice of Programming Languages, 2001.

4. *Фролов А.В.* Оптимизация размещения массивов в Фортран-программах на многопроцессорных вычислительных системах. Программирование. 1998. 3. С. 70–80.

5. *Adutskevich E.V.*, *Likhoded N.A.* Mapping affine loop nests: solving of the alignment and scheduling problems. Proc. 7th Int. Conf. on Parallel Computing Technologies (PaCT-2003). Nizhni Novgorod, Russia. Sept. 15–19, 2003. Berlin: Springer, 2003. P. 1–9.

6. *Dion M.*, *Randriamaro C.*, *Robert Y.* Compiling affine nested loops: how to optimize the residual communications after the alignment phase? J. of Parallel and Distrib. Computing. 1996. Vol. 30, 2. P. 176–187.

МОДЕЛИРОВАНИЕ ГРАВИТАЦИОННОГО ТЕРРИГЕННОГО ОСАДКА

О.Е. Амосова, Ю.А. Ткачев

Институт геологии Коми НЦ УрО РАН, Сыктывкар

Нами предпринята попытка исследования структуры обломочных горных пород и осадков компьютерным моделированием осаждения частиц и получающейся при этом их упаковки. Задача имеет большое значение в нефтяной геологии, в фациальной диагностике природных осадков и в технологии получения композиционных материалов.

Для моделирования мы использовали современные ПК (Pentium MMX 233, AMD Athlon XP 1600+ 785 904Кб ОЗУ), тем не менее, программа оказалась критической как по размеру используемой памяти, так и по времени выполнения. Максимальное число частиц в осадке достигало только 7700, тогда как для статистики необходимо, чтобы осадок состоял из нескольких десятков тысяч частиц. Один эксперимент на Pentium MMX 233 непрерывно обсчитывался около недели. На AMD Athlon XP 1600+, гораздо более мощном по производительности, моделирование заняло несколько минут, так моноразмерный осадок, состоящий из 7000 частиц был получен за 3 минуты 10 секунд, двухразмерный насыщенный, с отношением диаметров крупных и мелких частиц равным 5, состоящий из 7360 частиц, – за 6 минут. Нам удалось получить характеристики структуры лишь самых простых «осадков»: шарообразных частиц одного и двух размеров. Реальные осадки состоят из частиц различной формы (в аппроксимации – эллипсоидов с различным соотношением осей от вытянутых иглообразных до почти плоских лепешкообразных). Смоделировать такой осадок не представлялось возможным ввиду резкого увеличения времени счета (около недели на один вид осадка).

Предложенное нами компьютерное моделирование осадка заключается в следующем. Пусть задана функция распределения частиц осадка (или осадочной породы) по размерам $f(d)$ – гранулометрическая кривая. В первом приближении будем считать частицы шарообразными и осаждающимися в некой абстрактной среде (газе, жидкости, пустоте), никак не взаимодействующей с осаждающимися частицами. В этой среде нет вихрей, нет упругих аэро- или гидродинамических, электрических, магнитных сил, сил трения, действующих на частицы, а есть только сила тяжести и силы реакции опоры, возникающие при соприкосновении частиц. За этими исключениями процесс моделирования гравита-

ционного осадка соответствует физическому процессу осаждения. Моделирование осаждения производится в емкость или седиментационный «контейнер» параллелепипедальной формы. При визуализации на экране компьютера емкость простирается на всю ширину и высоту экрана и на заданную величину вглубь экрана. Над емкостью моделируется шарообразная частица диаметра d, получаемого с вероятностями, снятыми с гранулометрической кривой. Центр частицы находится на заданной высоте. Две другие координаты (горизонтальные) имеют случайное равномерное распределение над емкостью. Частица движется вертикально вниз, пока не опустится на дно, либо не коснется уже покоящихся частиц. Коснувшись одной из них, частица будет скатываться по ней, пока не коснется следующей частицы и так далее, пока по сложной траектории, скатываясь в ямки или проникая глубоко в осадок по поровым каналам, не опустится на дно, либо не займет устойчивое положение (получит три точки опоры внутри осадка). Траектория движения осаждающейся частицы представляет собой кривую в пространстве, состоящую из отдельных элементов – дуг окружностей и отрезков прямых (рис. 1). Траектория мелкой частицы в толще крупных состоит из чередования трех типов движений: *1* – свободное падение; *2* – спуск по поверхности шара; *3* – спуск по ложбине между двумя шарами.

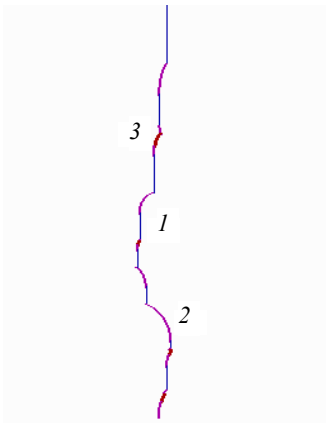


Рис. 1

По окончании осаждения одной частицы по той же схеме моделируется осаждение следующей. Диаметры частиц могут быть одинаковыми или моделироваться случайным образом с вероятностями, соответствующими заданной гранулометрической кривой. По окончании осаждения всех частиц их координаты запоминаются, записываются в файлы и в дальнейшем используются для определения емкостных характеристик полученного осадка: первичной пористости, поперечников пор, координационных чисел как для пор, так и для частиц, размера поровых каналов и т.д. Описанная модель может быть применима к осадкам бассейнов с преимущественно стоячей водой. Она может служить первой ступенью к созданию более сложных моделей, учиты-

вающих инерцию, волнения и течения. Ниже приведен разработанный нами общий алгоритм процесса седиментации.

Общий алгоритм моделирования кластического осадка

1. Задать число частиц m.

2. В качестве текущей рассмотреть первую частицу.

3. Если номер текущей частицы не больше m, тогда на п. 4. Иначе на п. 14.

4. С помощью датчика случайных чисел задать горизонтальные координаты частицы, равномерно распределенные в пределах заданной области седиментации. Диаметр частицы определить случайным числом, распределенным в соответствии с заданной гранулометрической кривой, на п. 5.

5. Частица падает вертикально вниз, пока не: а) достигнет дна – тогда на п. 13; б) коснется одной из уже осевших частиц (назовем ее первой касательной частицей), тогда – на п. 6.

6. Частица скатывается по меридиану первой касательной частицы, пока не : а) достигнет дна – тогда на п. 13; б) достигнет экватора первой касательной частицы – тогда на п. 5; в) коснется грани контейнера седиментации – тогда на п. 7; г) коснется другой частицы (второй касательной частицы), – тогда на п. 9.

7. Частица скатывается по ложбине между гранью и касательной частицей (траектория – окружность в вертикальной плоскости), пока не: а) достигнет дна – тогда на п. 13; б) достигнет экватора касательной частицы – тогда на п. 5; в) коснется другой грани контейнера седиментации – тогда на п. 13; г) коснется другой касательной частицы – тогда на п. 8.

8. Если проекция центра движущейся частицы пересекает треугольник опоры, то она остановится – тогда на п. 13, иначе скатывается либо по ложбине между первой и второй касательными частицами – тогда на п. 9, либо – по ложбине между гранью и второй касательной частицей, - тогда на п. 7, либо – по меридиану второй касательной частицы – тогда на п. 6.

9. Вычисление координат точки S смены траектории спуска по ложбине на меридиан. Определение первой и второй касательных частиц по следующему критерию. Первой касательной частицей будет та из двух частиц, чей центр расположен ниже, второй – другая – на п. 10.

10. Если ордината движущейся частицы в момент второго касания меньше ординаты точки S, тогда – на п. 6. Иначе на п. 11.

11. Частица скатывается по ложбине (траектория – окружность в наклонной плоскости) между первой и второй касательными частицами, пока не: а) достигнет дна – тогда на п. 13; б) достигнет точки смены S – тогда на п. 6; в) коснется грани контейнера седиментации – тогда на п. 13; г) коснется третьей частицы (третья касательная частица), на п. 12.

12. Если проекция центра движущейся частицы пересекает треугольник опоры, то она остановится – тогда на п. 13, иначе скатывается либо по одной из ложбин, образуемых третьей касательной частицей и одной из двух прежних касательных частиц – тогда на п. 11, либо – по меридиану третьей касательной частицы, – тогда на п.6.

13. Рассмотреть следующую частицу, на п. 3.

14. Конец.

На основе этого алгоритма разработана на языке Pascal программа моделирования гравитационного осадка с заданной гранулометрической характеристикой.

Были проведены эксперименты на моноразмерных и двухразмерных моделях, отношения диаметров d_K/d_M крупных и мелких частиц которых изменяются от 2 до 10. При этом исследовались как насыщенные, так и недосыщенные и пересыщенные в 1,5–2 раза мелкой фракцией. Насыщенный осадок – осадок, мелкая фракция которого по объему в точности заполняет поровое пространство, образуемое крупной фракцией, с учетом пористости, образуемой мелкой фракцией. Практически насыщенный осадок получается от засыпки пор моноразмерного осадка очень мелкой фракцией.

С целью ускорения работы программы предлагается распараллелить выше описанную задачу. Предлагается несколько способов распараллеливания. Первый заключается в следующем. Пусть имеется вычислительный кластер, состоящий из n процессоров. Каждый процессор имеет свой порядковый номер. Контейнер седиментации разобьем вертикальными плоскостями, параллельными его боковым граням, на n равных по объему прямоугольных призм. Пронумеруем их от 1 до n. Построим взаимно однозначное соответствие между номерами процессоров и призм. На каждом процессоре i запускаем одну и ту же программу моделирования процесса осаждения частиц в i-ой призме (i=1, .., n), т.е. каждый процессор будет осаждать свою частицу в заданной призме. Начальное положение частицы задается в пределах призмы таким образом, чтобы она целиком лежала внутри нее. Каждый раз, как вычислены координаты частиц в критический момент – момент пред-

полагаемой смены элемента траектории спуска, необходимо устраивать проверку на принадлежность частиц целиком внутренним областям соответствующих призм. Это необходимо, чтобы избежать пересечений траекторий частиц. Если условие выполнено для всех частиц, фиксируем их новое положение. Процесс повторяется. Иначе при невыполнении условия для какой-либо частицы, процессор, осуществляющий ее осаждение (пусть его номер m), ставится в состояние ожидания до тех пор, пока частица в соседней призме p не займет устойчивое положение. После этого процессор p останавливается до тех пор, пока процессор m не закончит осаждение своей частицы или его частица перейдет в другую призму. По окончании осаждения частицы каждый процессор приступает к осаждению следующей частицы в той же призме. Координаты осевших частиц записываются в массив. При таком подходе может возникнуть ситуация, при которой программа зациклится (рис. 2). Проекция контейнера седиментации, разбитого на 16 призм, на горизонтальную плоскость. Частица из 6-й призмы пересекла общую грань с соседней 7-й, из 7-й – грань 11-й, из 11-й – грань 10-й, из 10-й – грань 6-й.

Мы думаем, что при более детальной разработке алгоритма ее можно будет решить.

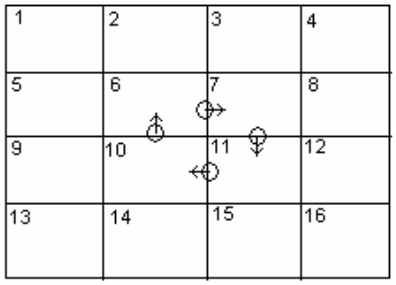


Рис. 2

Очевидно, что для минимизации времени работы программы необходимо разбить массив осевших частиц на n массивов (1, .., n) следующим образом. Если пересечение частицы с призмой k является не пустым множеством или минимальное расстояние от поверхности частицы до граней призмы меньше максимального диаметра частиц, то записываем ее в массив с номером k. Таким образом, при вычислении координат движущейся частицы в критический момент нужно будет переби-

рать на решение не все осевшие частицы, а лишь имеющие пересечение с определенной призмой.

Описанный выше способ является трудоемким в смысле его программной реализации, но очень эффективным при моделировании осадка, состоящего из очень большого числа частиц.

Второй способ. На каждом процессоре запускаем одну и ту же программу моделирования процесса осаждения частиц, т.е. каждый процессор будет осаждать свою частицу. Каждый раз, как вычислены координаты частиц в критический момент, необходимо синхронизировать работу процессоров. Т.е. процессоры должны обменяться информацией о вычисленных координатах движущихся частиц. Это необходимо, чтобы избежать критических ситуаций – пересечений траекторий частиц. Для этого необходимо определить области, в пределах которых частицы могут двигаться по заданным элементам траекторий спусков. В случае спуска по меридиану или ложбине такой областью будет сфера. При вертикальном падении областью будет круговой цилиндр. Если эти области для любых двух частиц не пересекаются, то и траектории этих частиц не пересекаются. Если критических ситуаций не возникло, фиксируем координаты частиц. Иначе (при возникновении критической ситуации, например между двумя частицами) процессор, имеющий больший порядковый номер, ставится в состояние ожидания до тех пор, пока частица, осаждаемая конкурирующим с ним процессором, не займет устойчивое положение. Все остальные процессоры продолжают вычисления. Координаты осевших частиц записываются в массив. Затем генерируются новые n частиц. Процесс продолжается. На наш взгляд, второй способ легко реализуем и требует меньшей переработки исходной программы.

Третий способ заключается в распараллеливании процедур, выполняющих вычисление координат движущейся частицы в момент предполагаемой смены одного элемента траектории спуска другим. При этом перебираются все осевшие частицы. Предлагается разбить массив осевших частиц (пусть их будет k) на n (число процессоров) массивов размера k/n. Возьмем k/n в качестве k. Вычисления поручим своему процессору. Получим n-кратное ускорение работы этих процедур.

ИНТЕГРИРОВАННАЯ СРЕДА АНАЛИЗА СТРУКТУРНОЙ И ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ И ИХ ЦЕЛЕНАПРАВЛЕННЫХ ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЙ

С.Е. Бажанов, М.М. Воронцов, В.П. Кутепов, Д.А. Шестаков

Московский Энергетический Институт (*ТУ*), *Москва*

Введение

В научной группе д.т.н. проф. В.П. Кутепова в настоящее время реализуется система функционального параллельного программирования для кластеров [1, 3, 8]. Система включает в себя высокоуровневый язык функционального параллельного программирования, инструментальную среду поддержки разработки функциональных программ на этом языке и инструментальные средства выполнения функциональных программ на кластерных системах.

В рамках этого проекта создана инструментальная среда разработки функциональных параллельных программ, которая позволяет выполнять следующие функции:

– структурный анализ функциональных программ по их схемам с целью выявления циклических, рекурсивных, взаимно рекурсивных и других определений;

– анализа вычислительной сложности параллельных программ с позиций их параллельного выполнения;

– проведение эквивалентных преобразований, позволяющих приводить схему функциональной программы в некотором смысле к оптимальной параллельной форме;

Методы и алгоритмы структурного анализа подробно рассмотрены нами в [1, 9]. В частности результаты этого анализа позволяют строить эффективные стратегии планирования процессов параллельного выполнения функциональных программ на кластерах [1].

Поэтому ниже мы подробно рассмотрим алгоритмы анализа вычислительной сложности и целенаправленных эквивалентных преобразований функциональных программ, используемые в разрабатываемой интегрированной среде, предназначающиеся для повышения эффективности процесса разработки функциональных программ и улучшения их качественных характеристик.

1. Анализ вычислительной сложности функциональных программ

Практический интерес представляет возможность получения точных оценок времени параллельного выполнения функциональных программ (ФП) при достаточно общих предположениях: известных данных о времени выполнения элементарных функций в ФП и вероятности осуществления (неосуществления) условных переходов в ней. Подобная задача оценивания временной сложности для последовательных программ решалась в различных работах [4, 5].

В [6] разработана формальная система, позволяющая оценивать время параллельного вычисления значений функций по заданным характеристикам временной сложности элементарных функций, входящих в представление функции, и вероятностям осуществления условных переходов.

Пусть задана функциональная схема (ФС) $X_i = \tau_i$, $i = 1, 2, \ldots, n$; $F = \{f_i \mid i = 1, 2, \ldots, k\}$ – множество всех входящих в термы τ_i, $i = 1, 2, \ldots, n$, базисных функций.

В [6] показано, что термы τ_i в задании ФС могут быть представлены в эквивалентной форме $\tau_i = \tau_{i1} \oplus \tau_{i2} \oplus \ldots \oplus \tau_{ik}$, $i = 1, 2, \ldots, n$, где каждый терм τ_{ij} не содержит операции ортогонального объединения $\oplus$.

Пусть $C(R)$ – функция, определяющая сложность параллельного вычисления значения любой функции R.

Для любой базисной функции f_i предполагается, что $C(f_i)$ задано и представляет собой некое действительное число. Также предполагается, что на области определения D_R интересующей функции R для каждого терма τ_{ij} задана вероятность q_{ij} того, что именно его значение будет определено при вычислении любого значения $\tau_i(d)$ для $d \in D_R$. Эта вероятность напрямую связана в общем случае с условием, определяющим выполнение этого события.

Таким образом, $\sum_{j=1}^{k_i} q_{ij} = 1$.

При этих условиях сложность параллельного вычисления значения функций X_i в задании ФС может быть определена следующим образом [6].

По заданию ФС последовательно вычисляются «приближения» фиксированной точки для X_i согласно известной формуле [6]:

$X_i^{(0)} = \phi$ (нигде не определено),

$X_i^{(k+1)} = [X_j^{(k)}/X_j \mid j = 1, 2, \ldots, n]\tau_i$, $i = 1, 2, \ldots, n$.

Очевидно, термы в правых частях приближения $X_j^{(k)}$ не содержат функциональной переменной X_i, и после их приведения к указанной выше эквивалентной форме, согласно [6], их вычислительная сложность определяется индукцией по построению:

$C(f_i) = t_i$ (время вычисления значения базисной функции f_i),

$C(\tau' \to \tau'') = \max\{C(\tau'), C(\tau'')\}$,

$C(\tau' * \tau'') = \max\{C(\tau'), C(\tau'')\}$,

$C(\tau' \bullet \tau'') = C(\tau') + C(\tau'')$.

Зная в представлении $X_i^{(k)} = \tau_{i1}^{(k)} \oplus \tau_{i2}^{(k)} \oplus \ldots \oplus \tau_{ivi}^{(k)}$ вероятности q_{ij}, $i, j = 1, 2, \ldots, v_i$, того, что при вычислении значения $X_i^{(k)}(d)$ будет определено $\tau_{ij}^{(k)}(d) = d'$ (остальные значения $\tau_{it}^{(k)}(d)$, $i_t \neq i_j$, будут не определены в силу того, что функции $\tau_{ij}^{(k)}$, $i = 1, 2, \ldots, v_i$ попарно ортогональны) можно считать, что среднее время вычисления $X_i^{(k)}(d)$ для любого d из области определения $X_{i\min}$ (минимальной фиксированной точки для X_i) равно

$$C(X_i^{(k)}) = \sum_{j-1}^{n} q_{ij} C(\tau_{ij}).$$

Более точно, эта формула есть промежуточный шаг в определении среднего времени вычисления $X_{i\min}$ при применении $X_{i\min}$ к любому элементу из области ее определения.

Среднее время параллельного вычисления значений функции $X_{i\min}$ определяется как $C_i(X_i^{(k\min)})$ для минимального значения $k = k_{\min}$ такого, что

$$|C_i(X_i^{(k)}) - C_i(X_i^{(k+1)})| \leq \varepsilon,$$

где заданная ε – точность вычисления среднего значения.

Описанная итеративная процедура вычисления среднего времени параллельного вычисления значений функций сходится [6]. Вместо нее можно применять другой, точный метод вычисления среднего значения, который сводится к решению множества систем линейных уравнений, построенных по ФС, в которых в качестве неизвестных выступают средние времена сложности $C(X_i)$ параллельных вычислений значений функций X_i в задании ФС.

Например, для ФС

$$X_1 = (f_1 * f_2) \bullet X_1 \oplus f_3 \to f_4 \bullet X_2 \oplus f_5,$$
$$X_2 = f_6 \to f_7 * f_8 \bullet X_2 \oplus f_9$$

имеем следующую систему уравнений:

$$C(X_1) = q_1 \cdot (\max\{C(f_1), C(f_2)\} + C(X_1)) +$$
$$+ q_2 \cdot \max\{C(f_3), C(f_4) + C(X_2)\} + q_3 \cdot C(f_5),$$

$$C(X_2) = q_4 \cdot \max\{C(f_6), \max\{C(f_7), C(f_8) + C(X_2)\}\} + q_5 \cdot C(f_9).$$

Здесь q_1, q_2, q_3, q_4, q_5 – вероятности для соответствующих термов в ФС; $q_1 + q_2 + q_3 = 1$; $q_4 + q_5 = 1$.

Из нее, вводя соответствующие условия – ограничения для всевозможных предположений о сложности $C(X_i)$ функциональных переменных, можно построить множество систем линейных уравнений, решая которые и проверяя выполнение введенных условий, можно получить однозначную оценку средней сложности параллельных вычислений любой определяемой в ФС функции X_i, $i = 1, 2, \ldots, n$.

Для рассматриваемого примера имеем следующие системы линейных уравнений с условиями – ограничениями:

1. $C(X_1) = q_1 \cdot (\max\{C(f_1), C(f_2)\} + C(X_1)) + q_2 \cdot C(f_3) + q_3 \cdot C(f_5)$,
   $C(X_2) = q_4 \cdot C(f_6) + q_5 \cdot C(f_9)$,
   если $C(f_4) + C(X_2) \le C(f_3)$ и $\max\{C(f_7), C(f_8) + C(X_2)\} \le C(f_6)$.
2. $C(X_1) = q_1(\max\{C(f_1), C(f_2)\} + C(X_1)) + q_2(C(f_4) + C(X_2)) + q_3 C(f_5)$,
   $C(X_2) = q_4 \max\{C(f_6), C(f_8) + C(X_2)\} + q_5 C(f_9)$,
   если $C(f_3) < C(f_4) + C(X_2)$ и $C(f_6) < \max\{C(f_7), C(f_8) + C(X_2)\}$.

Эта система уравнений сводится к двум системам линейных уравнений с рассмотрением ограничений для каждой из них:

$$C(f_7) \le C(f_8) + C(X_2) \text{ и } C(f_8) + C(X_2) < C(f_7).$$

Продолжая этот процесс, мы построим все искомые линейные системы уравнений с ограничениями, разрешая которые мы получим однозначную оценку для среднего времени вычисления $C(X_1)$ и $C(X_2)$ функций X_1 и X_2.

В интегрированной среде поддержки проектирования функциональных программ реализован описанный выше метод итерационного вычисления среднего значения параллельного вычисления значений функций.

2. Эквивалентные преобразования функциональных программ

На функциональном языке, как и на любом другом языке программирования, для реализации каждого алгоритма может быть написано

бесконечно большое число программ. Все эти программы будут функционально тождественны, однако они будут иметь различные характеристики. Так как во многих случаях основной задачей является улучшение программной реализации, то из всех имеющихся выбираются программы, имеющие оптимальные сложностные характеристики.

Кроме того, встает вопрос, как, преобразовывая программу при сохранении ее функционального значения, улучшить характеристики.

В [7] предложено исчисление преобразований для рассматриваемого языка функционального параллельного программирования. Между функциональными термами вводится отношение сильной эквивалентности. При этом не учитывается эквивалентность функций, которая проявляется при определенных (а не всевозможных) интерпретациях термов.

Исчисление преобразований включает полный и непротиворечивый набор аксиом [7]. Однако с помощью этого исчисления нельзя получить необходимый результат, так как каждое преобразование задает как прямое, так и обратное действие. Можно выделять из всех преобразований системы целенаправленных эквивалентных преобразований к необходимым формам.

Например, рассмотрим аксиому $(A \to B)\bullet C = A \to B\bullet C$ [7]. Сложность правой части равна $C_{\text{п}} = \max\{C(A),\ C(B)\} + C(C)$, сложность левой части $C_{\text{л}} = \max\{C(A),\ C(B) + C(C)\}$. Очевидно $C_{\text{п}} > C_{\text{л}}$. Таким образом, одним из возможных вариантов целенаправленных преобразований может быть преобразование к форме с минимальным временем параллельного вычисления.

Заключение

В настоящее время созданы основные блоки интегрированной программной среды, которая позволит существенно повысить эффективность и качество функциональных параллельных программ на этапе их проектирования. Разрабатывается метод графического анализа проектирования функциональных программ.

Литература

1. *Бажанов С.Е.*, *Воронцов М.М.*, *Кутепов В.П.*, *Шестаков Д.А*. Структурный анализ и планирование процессов параллельного выполнения функциональных программ. Известия РАН. Теория и системы управления, 2005, №6. С. 131–146.

2. *Бажанов С.Е.*, *Кутепов В.П.*, *Шестаков Д.А*. Разработка и реализация системы функционального параллельного программирования на вы-

числительных системах // Докл. междунар. научн. конф. «Суперкомпьютерные системы и их применение» SSA'2004. Минск: ОИПИ НАН Беларуси, 2004.

3. *Бажанов С.Е.*, *Кутепов В.П.*, *Шестаков Д.А.* Язык функционального параллельного программирования и его реализация на кластерных системах. Программирование, 2005, №5. С. 18–51.

4. *Барский А.Б.* Планирование параллельных вычислительных процессов. М.: Машиностроение, 1980.

5. *Головкин Б.А.* Расчет характеристик и планирование параллельных вычислительных процессов. М.: Радио и связь, 1983.

6. *Кутепов В.П.* Организация параллельных вычислений на системах. М.: МЭИ, 1988.

7. *Кутепов В.П.* Языки параллельных алгоритмов. Учебное пособие по курсу «Стурктуры вычислительных машин и систем». М.: МЭИ, 1978.

8. *Кутепов В.П.*, *Бажанов С.Е.* Функциональное параллельное программирование: язык, его реализация и инструментальная среда разработки программ. // Матер. IV Международ. научно-практического семинара и Всероссийской молодежной школы. Самара. 2004.

9. *Кутепов В.П.*, *Шестаков Д.А.* Анализ структурной сложности функциональных программ и его применение для планирования их параллельного выполнения на вычислительных системах. Высокопроизводительные параллельные вычисления на кластерных системах // Матер. IV Междунар. научно-практического семинара и Всероссийской молодежной школы. Самара. 2004.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ С АДАПТИВНЫМ ПОРЯДКОМ ПРОВЕРКИ ОГРАНИЧЕНИЙ

К.А. Баркалов

Нижегородский государственный университет им. Н.И Лобачевского

Данная работа продолжает развитие нового подхода к минимизации многоэкстремальных функций при невыпуклых ограничениях (см. [1]–[6]), получившего название *индексного метода* глобальной оптимизации.

Рассмотрим многомерную задачу условной глобальной оптимизации вида

$$\varphi^* = \varphi(y^*) = \min\{\varphi(y): y \in D, g_j(y) \le 0, 1 \le j \le m\},$$

$$D = \{y \in R^N: -2^{-1} \le i_i \le 2^{-1}, 1 \le I \le N\}.$$

Данная задача рассматривается в предположении, что целевая функция $\varphi(y)$ и левые части ограничений g_j, $1 \le j \le m$, являются липшицевыми функциями с соответствующими константами L_j, $1 \le j \le m + 1$.

Используя кривые типа развертки Пеано $y(x)$, однозначно отображающие отрезок [0, 1] на N-мерный гиперкуб D

$$D = \{y \in R^N: -2^{-1} \le y_i \le 2^{-1}, 1 \le I \le N\} = \{y(x): 0 \le x \le 1\},$$

исходную задачу можно редуцировать к следующей одномерной задаче:

$$\varphi(y(x^*)) = \min\{\varphi(y(x)): x \in [0, 1], g_j(y(x)) \le 0, 1 \le j \le m\}.$$

В работе [3] предложена идея одновременного использования множества кривых такого типа, что сводит многомерную задачу к нескольким связанным одномерным задачам. Каждая из одномерных задач решается на отдельном процессоре, причем процессоры обмениваются результатами итераций, что соответствует распараллеливанию решения исходной задачи. Достоинство этой схемы состоит в том, что набор одномерных задач, порождаемых одновременным применением множества разверток, более полно передает информацию о метрических свойствах исходной многомерной задачи.

В соответствии с правилами индексного метода каждая итерация, называемая *испытанием* в соответствующей точке области поиска, включает последовательную проверку выполнимости ограничений задачи в этой точке. При этом обнаружение первого нарушенного ограничения прерывает испытание и инициирует переход к точке следующей итерации. В силу того, что левые части ограничений g_j, $1 \le j \le m$, являются липшицевыми функциями, ограничение, нарушенное в некоторой точке области поиска, не будет выполняться и в некоторой окрестности этой точки. В таком случае (для сокращения числа проверок) может оказаться полезным начинать испытания в точках этой окрестности с анализа выполнимости указанного ограничения. В свете сказанного представляется целесообразным построить алгоритм глобальной оптимизации, допускающий изменение порядка проверки ограничений с учетом информации, получаемой в ходе поиска решения. В работе [5] был предложен последовательный алгоритм с переменным порядком проверки ограничений.

Новое предложение, применительно к многомерным задачам, состоит в обобщении решающих правил последовательного алгоритма из [5] на случай многих процессоров и заключается в том, что для каждой

точки итерации x_i адаптивно определяется свой порядок проверки ограничений

$$H(x_i) = \{j_{i1}, \ldots, j_{im}, j_{i,m+1}\},\ 0 \le I \le k,$$

как перестановка номеров из базовой нумерации ограничений

$$H = \{1, 2, \ldots, m, m+1\}.$$

Это позволяет начинать проверку с ограничения, для которого более вероятно нарушение в выбранной точке очередной итерации. Тем самым форсируется завершение итерации при меньших вычислительных затратах. При этом возникает потребность как в новых правилах сопоставления индексов точкам итераций (эти индексы определяют использование выражений для определения интервала, в котором будет проводиться следующая итерация), так и в специальной процедуре, формирующей порядок проверки ограничений в точке очередного испытания.

Для предложенного параллельного алгоритма с адаптивным порядком проверки ограничений проведено теоретическое исследование условий сходимости.

Работа выполнена при финансовой поддержке РФФИ (код проекта 04-01-00455).

Литература

1. *Стронгин Р.Г.* Численные методы в многоэкстремальных задачах. М.: Наука, 1978.

2. *Стронгин Р.Г.* Поиск глобального оптимума. М.: Знание, 1990.

3. *Стронгин Р.Г.* Параллельная многоэкстремальная оптимизация с использованием множества разверток // Ж. вычисл. матем. и матем. физ. 1991. Т.31. №8. С. 1173–1185.

4. *Стронгин Р.Г.*, *Баркалов К.А*. О сходимости индексного алгоритма в задачах условной оптимизации с ε-резервированными решениями // Математические вопросы кибернетики. М.: Наука, 1999. С. 273–288.

5. *Баркалов К.А.*, *Стронгин Р.Г.* Метод глобальной оптимизации с адаптивным порядком проверки ограничений // Ж. вычисл. матем. и матем. физ. 2002. Т.42. №9. С. 1338–1350.

6. *Strongin R.G.*, *Sergeyev Ya.D*. Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht, 2000.

ABOUT MARS METHOD AND PARALLEL INDEX METHOD INTEGRATION

K.A. Barkalov

University of Nizhny Novgorod, Nizhny Novgorod, Russia

V.L. Markine

Delft University of Technology, Delft, The Netherlands

Introduction

In this paper an integration of the Multipoint Approximation based on Response Surface fitting (MARS) method (developed and used in Delft University of Technology) and parallel index method (PIM) of global optimization (developed in Nizhny Novgorod State University) is discussed. A new point is that PIM is used to solve the approximation problems, which are generated by MARS method.

MARS method

Consider the optimization problem in general form:

$$F^* = \min\{F_0(y): y \in D, F_j(y) \le 1, j = 1, \ldots, M\},$$
$$D = \{y \in R^N: A_i \le y_i \le B_i, I = 1, \ldots, N\}, \quad (1)$$

where F_0 is the objective function; $F_j(y)$, $j = 1, \ldots, M$ are the constraints; $y = [y_1, \ldots, y_N]^T$ is the vector of design variables; A_i and B_i are the side limits, which define lower and upper bounds of the i-th design variable.

Components of the vector y represent various parameters of a structure, such as geometry, material, stiffness and damping properties, which can be varied to improve the design performance. Depending on the problem under consideration the objective and constraint functions can describe various structural and dynamic response quantities such as weight, reaction forces, stresses, natural frequencies, displacements, velocities, accelerations, etc. Cost, maintenance and safety requirements can be used in the formulation of the optimization problem as well. The objective function provides a basis for improvement of the design whereas the constraints impose some limitations on behaviour characteristics of a structure.

The optimization problem (1) can be solved using a conventional method of mathematical programming. However, for systems with many degrees of freedom the finite element analysis can be time consuming. As a result, the total computational effort of the optimization might become prohibitive. This difficulty has been mitigated starting from the mid-seventies

by introducing approximation concepts (see [8]).

According to the approximation concepts the original functions in (1) are replaced with approximate ones, which are computationally less time consuming. Instead of the original optimization problem (1) a succession of simpler approximated subproblems, similar to the original one and formulated using the approximation functions is to be solved. Each simplified problem then has the following form:

$$\min\{\widetilde{F}_0^k(y) : y \in D, \widetilde{F}_j^k(y) \le 1, j = 1, \mathsf{K}, M\},$$

$$D = \{y \in R^N, A_i^k \le y_i \le B_i^k, A_i^k \ge A_i, B_i^k \ge B_i, j = 1, \mathsf{K}, N\}, \qquad (2)$$

where the superscript k is the number of the iteration step, $\widetilde{F}$ is the approximation of the original function F, A_i^k and B_i^k are *move limits* defining the range of applicability of the approximations.

The solution of the problem $\boldsymbol{y}_*^{\boldsymbol{k}}$ is then chosen as starting point for the (k+1)-th step and the optimization problem (3), (4), reformulated with the new approximation functions $\widetilde{F}_j^{k+1}(y) \le 1, j = 1, \mathsf{K}, M,$ and move limits A_i^{k+1} and B_i^{k+1}, is to be solved. The process is repeated until the convergence criteria are satisfied.

More information about the approximations, the move limits strategy and the most recent developments in the MARS method can be found in [4]–[7].

Integration with parallel index method

Since the functions in (2) are chosen to be simple and computationally inexpensive, any conventional method of optimization [6] can be used to solve the problem (2). However, when the constructed approximations have multiple optima the use of a global optimization method is preferable in order to prevent premature convergence to one of the local optima.

A new point of this paper is that *parallel index method* is used to solve approximated subproblems (2). The index approach is based on a separate consideration of every constraint and does not involve penalty functions. In the index method, every iteration step performed at the corresponding point of the search domain is called a trial; it includes checking for constraints of the problem at this point. When a violation is discovered for first time, the trial is stopped, and the next iteration step is initiated.

The solution of multidimensional problems is reduced to solving

equivalent one-dimensional ones. The reduction is based on the use of mapping of a unit interval on the real axis onto a hypercube. They are realized by one-to-one continuous mappings similar to the Peano curve (also known as space filling curves) and their generalizations called *multiple scanning*. Numerical methods for approximating Peano curves (with any given accuracy) and constructive methods of inverse mappings (being multiple) are described and substantiated in [2], [3]. The last work contains also C++ programs.

In the parallel index method the functionals involved in the problem are assumed to be Lipschitzian. This assumption is typical for many other approaches (e.g., see [9]) and is natural for many applied problems, since relative variations of the functionals generally cannot exceed a certain threshold determined by the bounded energy of changes occurring in the system under study. The corresponding Lipschitz constants can be estimated by using adaptive schemes (see [1]–[3]).

The parallel index method was implemented as dynamic-link library, which can be lenked with MARS system. The results of comparing original MARS system and MARS-PIM integrated system are obtained. The comparison was carried out by using both systems to solve some real-life problems.

Acknowledgments

This work was carried out with the financial support provided by the NWO (The Netherlands Scientific Organization) grant 047.016.014 and the Russian Fund of Basic Research (RFBR) grant № 04-01-00455.

References

1. *Strongin R.G*. Numerical methods for multiextremal nonlinear programming problems with nonconvex constraints // Lecture Notes in Economics and Mathematical Systems, 1985. V. 255. P. 278–282.

2. *Strongin R.G.* Algorithms for multi-extremal mathematical programming problems employing the set of joint space-filling curves // J. of Global Optimization, 1992. №2. P. 357–378.

3. *Strongin R.G.*, *Sergeyev Ya.D*. Global optimization with non-convex constraints. Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht, 2000.

4. *Markine V.L*. Optimization of the Dynamic Behaviour of Mechanical Systems, PhD Thesis, TU Delft: Shaker Publishing BV, 1999. ISBN 90-423-0069-8.

5. *Toropov V.V*. Simulation Approach to Structural Optimization, Structural Optimization 1: 37–46. 1989.

6. *Toropov V.V*., *Markine V.L*. The Use of Simplified Numerical Models as Mid-Range Approximations, Proceedings of the 6-th AIAA/USAF/NASA/ ISSMO Symposium on Multidisciplinary Analysis and Optimization, Part 2, Bellevue WA, September 4-6, 1996: 952–958, ISBN 1-56347-218-X.

7. *Toropov V.V*., *Keulen F. van*, *Markine V.L*., *Alvarez L.F*. Multipoint Approximations Based on Response Surface Fitting: a Summary of Recent Developments. In V.V. Toropov (Ed.) Proceedings of the 1st ASMO UK/ISSMO Conference on Engineering Design Optimization, Ilkley, West Yorkshire, UK, July 8-9, 1999: 371-381, ISBN 0-86176-650-4.

8. *Barthelemy J.-F.M*., *Haftka R.T*. Approximation Concept for Optimum Structural Design – a Review, Structural Optimization 5: 129-144. 1993.

9. *Pinter J*. Global optimization in action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications). Kluwer Academic Publishers, Dordrecht, 1996.

10. *Markine V.L*., *Barkalov K.A*., *Gergel V.P*. (2005) An Optimum Design Procedure Based On Multipoint Approximations And A Global Optimisation Method. Proceedings of the 6th World Congresses of Structural and Multidisciplinary Optimization, Rio de Janeiro, 30 May – 03 June 2005, Brazil.

ПРИНЦИПЫ ПОСТРОЕНИЯ СИСТЕМ ОПЕРАТИВНОЙ АНАЛИТИЧЕСКОЙ ОБРАБОТКИ ДАННЫХ НА ГЕТЕРОГЕННЫХ КЛАСТЕРАХ

Д.П. Бугаев

Оренбургский государственный университет

Введение

Современный уровень развития аппаратных и программных средств с некоторых пор сделал возможным повсеместное ведение баз данных оперативной информации на разных уровнях управления. В процессе своей деятельности промышленные предприятия, корпорации, ведомственные структуры, органы государственной власти и управления накопили большие объемы данных. Они хранят в себе большие потенциальные возможности по извлечению полезной аналитической информации, на основе которой можно выявлять скрытые тенденции, строить стратегию развития, находить новые решения.

В последние годы в мире оформился ряд новых концепций хранения и анализа корпоративных данных:

1) хранилища данных, или Склады данных (Data Warehouse);

2) оперативная аналитическая обработка (On-Line Analytical Processing, OLAP);

3) Интеллектуальный анализ данных – ИАД (Data Mining).

Технологии OLAP тесно связаны с технологиями построения Data Warehouse и методами интеллектуальной обработки – Data Mining. Поэтому наилучшим вариантом является комплексный подход к их внедрению.

Способы аналитической обработки данных

Для того чтобы существующие хранилища данных способствовали принятию управленческих решений, информация должна быть представлена аналитику в нужной форме, то есть он должен иметь развитые инструменты доступа к данным хранилища и их обработки.

Очень часто информационно-аналитические системы, создаваемые в расчете на непосредственное использование лицами, принимающими решения, оказываются чрезвычайно просты в применении, но жестко ограничены в функциональности. Такие статические системы называются в литературе Информационными системами руководителя (ИСР), или Executive Information Systems (EIS). Они содержат в себе предопределенные множества запросов и, будучи достаточными для повседневного обзора, неспособны ответить на все вопросы к имеющимся данным, которые могут возникнуть при принятии решений. Результатом работы такой системы, как правило, являются многостраничные отчеты, после тщательного изучения которых у аналитика появляется новая серия вопросов. Однако каждый новый запрос, непредусмотренный при проектировании такой системы, должен быть сначала формально описан, закодирован программистом и только затем выполнен. Время ожидания в таком случае может составлять часы и дни, что не всегда приемлемо. Таким образом, внешняя простота статических СППР, за которую активно борется большинство заказчиков информационно-аналитических систем, оборачивается катастрофической потерей гибкости.

Динамические СППР, напротив, ориентированы на обработку нерегламентированных запросов аналитиков к данным. Наиболее глубоко требования к таким системам рассмотрел E.F. Codd, положившей начало концепции OLAP. Работа аналитиков с этими системами заключается в интерактивной последовательности формирования запросов и изучения их результатов.

Но динамические СППР могут действовать не только в области оперативной аналитической обработки (OLAP); поддержка принятия

управленческих решений на основе накопленных данных может выполняться в трех базовых сферах:

1. *Сфера детализированных данных*. Это область действия большинства систем, нацеленных на поиск информации. В большинстве случаев реляционные СУБД отлично справляются с возникающими здесь задачами. Общепризнанным стандартом языка манипулирования реляционными данными является SQL. Информационно-поисковые системы, обеспечивающие интерфейс конечного пользователя в задачах поиска детализированной информации, могут использоваться в качестве надстроек как над отдельными базами данных транзакционных систем, так и над общим хранилищем данных.

2. *Сфера агрегированных показателей*. Комплексный взгляд на собранную в хранилище данных информацию, ее обобщение и агрегация, гиперкубическое представление и многомерный анализ являются задачами систем оперативной аналитической обработки данных (OLAP). Здесь можно или ориентироваться на специальные многомерные СУБД, или оставаться в рамках реляционных технологий. Во втором случае заранее агрегированные данные могут собираться в БД звездообразного вида, либо агрегация информации может производиться на лету в процессе сканирования детализированных таблиц реляционной БД.

3. *Сфера закономерностей*. Интеллектуальная обработка производится методами интеллектуального анализа данных (ИАД, Data Mining), главными задачами которых являются поиск функциональных и логических закономерностей в накопленной информации, построение моделей и правил, которые объясняют найденные аномалии и/или прогнозируют развитие некоторых процессов.

Преимущества использования кластеров в OLAP-системах

Для построения OLAP системы необходимы достаточно большие аппаратные ресурсы, которые позволяли бы производить сбор информации, построение необходимой структуры и предоставление актуальной информации клиентам. В полном объеме с такой задачей справится гомогенный кластер.

Основными преимуществами кластера являются:

1. Обеспечение высокого уровня готовности по сравнению с разрозненным набором компьютеров или серверов. Повышение готовности системы обеспечивает работу критических для OLAP приложений на протяжении максимально продолжительного промежутка времени. К критическим можно отнести все приложения, от которых напрямую

зависит способность OLAP системы выполнять свои функции. Как правило, использование кластера позволяет гарантировать, что в случае, если сервер или какое-либо приложение перестает нормально функционировать, другой сервер в кластере, продолжая выполнять свои задачи, возьмет на себя роль неисправного сервера (или запустит у себя копию неисправного приложения) с целью минимизации простоя пользователей из-за неисправности в системе.

2. Значительное увеличение общей производительность сети (высокая степень масштабируемости). Кластер позволяет гибко увеличивать вычислительную мощность системы, добавляя в него новые узлы и не прерывая при этом работы пользователей. Современные кластерные решения предусматривают автоматическое распределение нагрузки между узлами кластера, в результате чего одно приложение может работать на нескольких серверах и использовать их вычислительные ресурсы.

3. Уменьшение затрат на администрирование локальной сети (хорошая управляемость).

4. Обеспечение высокой доступности сетевых служб. Даже при отказе одного из серверов кластера, все обеспечиваемые кластером службы остаются доступными пользователям.

Принципы построения кластеров на базе серверов по технологии «кольцо»

В зависимости от количества узлов кластера используются различные типы соединений: кольцо, 2*D* и 3*D* торы. В небольших системах, имеющих небольшое количество узлов (до 8), эффективно использовать топологию простого кольца (см. рисунок)

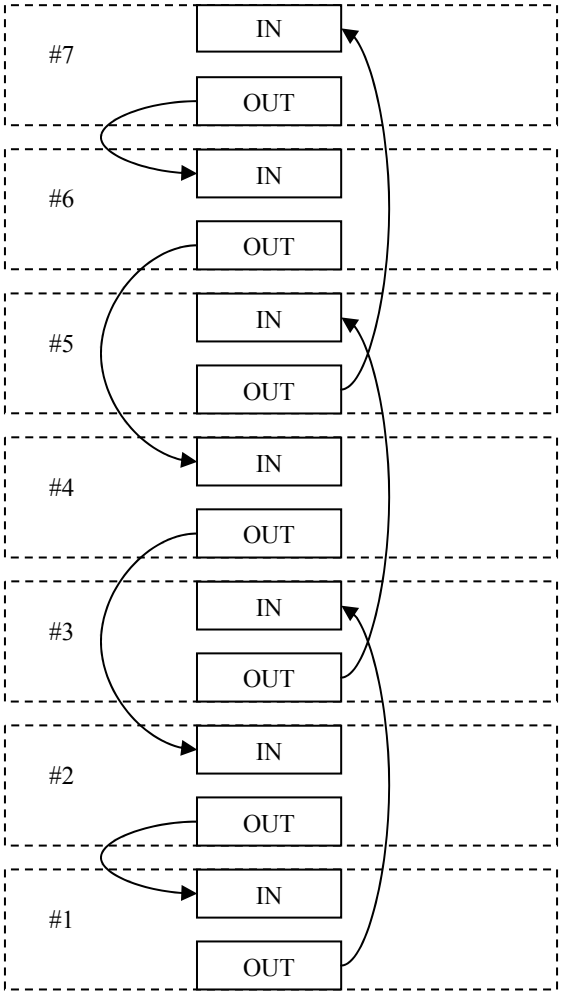
Аппаратное обеспечение для построения кластеров

Для практической реализации систем оперативной аналитической обработки данных на гетерогенных кластерах, были выбраны следующие узлы кластера:

2 сервера СУБД ORACLE, реализованных на серверах Sun SPARC и Intel Xeon,

2 Web-сервера Apache – на серверах Sun SPARC и Intel Pentium 3,

2 сервера ORACLE Express Server – на серверах Intel Pentium 3 и 1 сервер приложений для работы с OLAP и БД – на серверах Intel Pentium 3.


#7
IN
OUT
#6
IN
OUT
#5
IN
OUT
#4
IN
OUT
#3
IN
OUT
#2
IN
OUT
#1
IN
OUT

ТЕХНОЛОГИИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ ДЛЯ ИДЕНТИФИКАЦИИ ТЕХНИЧЕСКОГО СОСТОЯНИЯ ТРУБОПРОВОДОВ

А.Ю. Владова

Оренбургский государственный университет

Системы трубопроводного транспорта газа составляют основу топливоснабжения страны. Трубопроводы относятся к категории энергонапряженных объектов, отказы в которых сопряжены со значительным материальным и экологическим ущербом. С появлением промышленных образцов внутритрубных дефектоскопов стало возможным получение обширной информации по дефектам на протяжении многокилометровых участков ТП. Однако объективный анализ данных затруднен из-за большого количества длительных трудоемких вычислений, поэтому ранее разработанный автором программный комплекс моделирования и прогнозирования технического состояния трубопроводов нуждается в улучшении по критерию производительности.

Цель работы формулируется, как повышение эффективности идентификации технического состояния трубопроводов на базе технологий параллельного программирования. Для решения поставленной проблемы сформулированы следующие этапы:

– анализ структуры существующего программного комплекса (ПК) для выделения блоков, подлежащих распараллеливанию,

– выбор средств распараллеливания,

– оценка полученных результатов по разработанной иерархической структуре критериев.

Структурная схема существующего ПК [1] состоит из трех основных модулей: идентификации, прогнозирования и расчета эффективности идентификации технического состояния (ТС). Область применения – обработка и анализ баз данных, полученной внутритрубными инспекциями с помощью снарядов-дефектоскопов «Ультраскан». В первом модуле производится разбиение всей длины трубопровода на участки, затем на каждом совмещенном по инспекциям участкам, определяются параметры рельефности и рассчитывается ТС каждого парного участка для первой и второй внутритрубной инспекции. Проведенный анализ структуры ПК в рамках первой задачи позволил выявить, что процедуры выбора оптимальной модели дефектов, вычисления параметров рельефности, их нормализация, а также выбор оптимальной агрегированной модели технического состояния ТП по глубине, пло-

щади и объему дефектов подлежат распараллеливанию, как независимые вычисления (рис. 1).

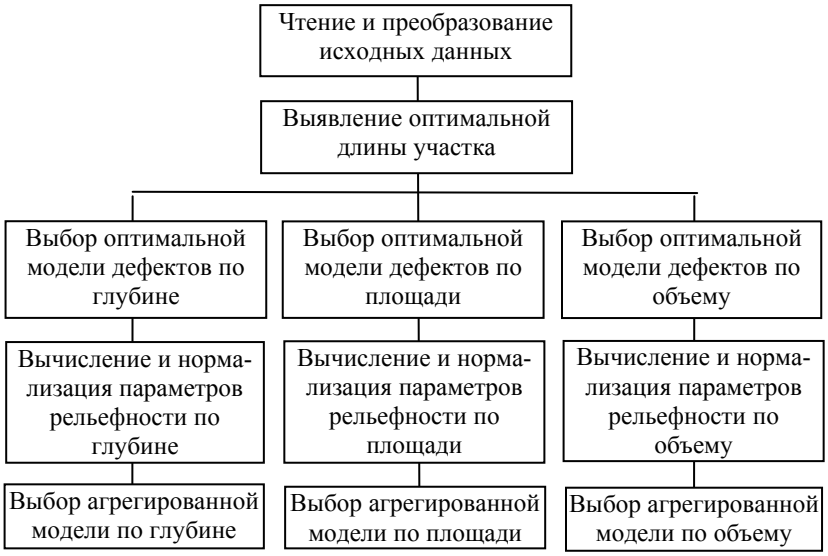


Рис. 1. Схема распараллеливания первого модуля программного комплекса

Для оценки остаточного ресурса, обеспечения надежной работы и совершенствования системы технического обслуживания и ремонта трубопроводов решается задача прогнозирования на базе дифференциальных уравнений первого и второго порядков с запаздыванием. Второй модуль ПК, решающий задачу прогнозирования и состоящий из блоков выбора модели прогнозирования, определения параметров прогнозной модели: постоянных времени и установившегося значения ТС участка, прогнозирования кинетики ТС, скорости коррозии и коррозионной устойчивости, а также анализа связи прогнозной скорости коррозии и ТС поддается распараллеливанию по критерию выбора прогнозной модели в соответствии с рис. 2.

ТП в течение всего срока службы испытывают значительные внутренние напряжения, близкие к нормативным характеристикам прочности металла, поэтому даже незначительные отклонения действительных условий от принятых в расчетах приводят систему в состояние предельного напряжения.

Работоспособность ТП, как любой сложной технической системы,

зависит от совокупности параметров, наиболее информативным из которых является комплексный показатель – эффективность функционирования.

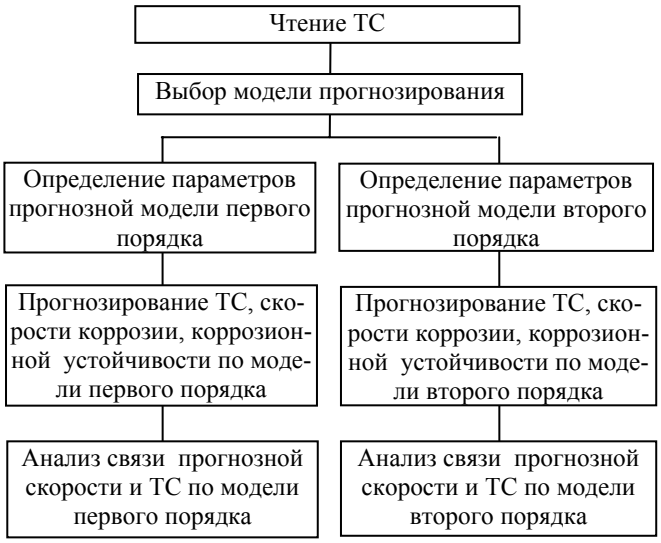


Рис. 2. Схема распараллеливания второго модуля программного комплекса

Третий модуль ПК проводит расчет эффективности идентификации по трем наиболее важным характеристикам: надежности, стоимости эксплуатации и объемной подачи газа на базе двух моделей: без учета и с учетом ТС трубопроводов. Расчеты являются независимыми процессами и распараллеливание в соответствии с рис. 3 должно повысить производительность вычислений.

Учитывая, что задача прогнозирования и оценки эффективности функционирования требуют на входе результаты вычисления ТС по нескольким инспекциям, но являются независимыми вычислениями, они также могут выполняться одновременно.

В рамках второй задачи при выборе средств распараллеливания, учитывалось, что существующий ПК написан на языке высокого уровня DELPHI и перенос на другой язык был бы неоправданно трудоемким процессом. Объем баз данных, содержащих дефекты, и большое количество возможных пересылок способствовали выбору в пользу систем с общей памятью.

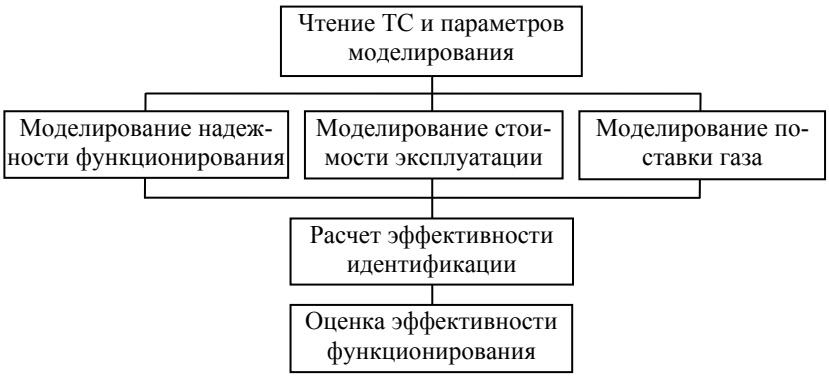


Рис. 3. Схема распараллеливания третьего модуля программного комплекса

В рамках третьей задачи предлагается оценить полученный в результате распараллеливания ПК. Анализ доступных источников оценки качества ПО, а также моделей ISO 9126, МакКола, Боема [2] из предлагаемого множества критериев таких как: надежность, уровень оптимизации, производительность, уровень распараллеливания, функциональность, переносимость, удобство использования и др. позволил выделить агрегированные компоненты первого иерархического уровня: надежность, уровень оптимизации, производительность, эффективность распараллеливания в соответствии с рис. 4.

Рис. 4. Компонентный уровень иерархической системы критериев

Интегральный критерий **эффективность кода ПК** рассчитывается по формуле:

$$C = \sum \gamma_i * X_i, \quad (1)$$

где γ_i – весовой коэффициент компоненты; X_i – нормированные значения компонент.

Нормированные значения рассчитываются как отношение фактически достигнутого уровня критерия к его потенциальному уровню. Нормировкой достигается возможность суммирования всех критериев как безразмерных и приведение их к диапазону [0, ..., 1]. Надежность программного обеспечения гораздо важнее других его характеристик, например, времени исполнения, и хотя абсолютная надежность современного программного обеспечения, по-видимому, недостижима, до сих пор не существует общепринятой меры надежности компьютерных программ [3]. В рамках нашей задачи определим надежность как способность программного кода безотказно выполнять определенные функции при заданных условиях с ожидаемой точностью. Детализируя компоненту надежности в соответствии с [4], получим, что второй иерархический уровень включает в себя следующие критерии:

– минимизация ошибок переполнения буфера как следствие неправильной организации работы с памятью. Подавляющее большинство причин некорректной работы приложений по статистике BUGTRAQ связано с такими ошибками;

– минимизация ошибок синхронизации потоков в многопоточных приложениях, т.к. обычно проблема кроется в использовании некорректных механизмов синхронизации потоков;

– погрешность вычисления, т.к. код программы, как последовательный, так и параллельный, должен выдавать верный результат с требуемой точностью. Погрешность вычисления предлагается рассчитывать как среднеквадратическое отклонение от математическое ожидания (СКО).

На количество ошибок переполнения буфера влияют по возрастающей: использование индикатора завершения строки/файла, не использование динамических переменных и структурных исключений, язык разработки, не контролирующий ошибки переполнения, отсутствие компиляторной проверки выхода за границы массива и др.

Критерий м**инимизация ошибок переполнения буфера** $Y_{\text{буф}}$ рассчитывается по формуле:

$$Y_{\text{буф}} = \sum \beta_i * X_i / X_{\text{баз}i}, \qquad (2)$$

где β_i, X_i, $X_{\text{баз}i}$ приведены в табл. 1. Весовые коэффициенты определены в соответствии с методом экспертных оценок. Обеспечение синхронизации при написании многопоточных программ необходимо для предотвращения конкурентных ситуаций, когда несколько потоков пытаются одновременно изменить одну глобальную переменную. При запуске распараллеленных программ возникают следующие проблемы:

взаимные блокировки параллельных участков; возможность зависания параллельных участков; несинхронный доступ или гонки; опасность использования сторонних процедур и библиотек.

Таблица 1

Механизмы минимизации ошибок переполнения буфера

Критерии	**Обозначение отсутствия данных в конце строки/файла**	**Использование кучи для массивов**	**Наличие структурных исключений (SEH)**	**Влияние языка разработки**
Вес β_i	1/10	2/10	3/10	4/10
X_i	Кол-во проверок индикаторов завершения	Кол-во динамичес-ких массивов	Кол-во структурных исключ-чений	Fortran C++, Delphi, Java, C#
Базовые $X_{базi}$	Кол-во строковых переменных, массивов, файлов	Общее кол-во переменных	Общее кол-во динамических переменных	Общее кол-во языков

В табл. 2 приведена оценка влияния количества критических секций, спин-блокировок и разделяемых переменных в программе и [5] на надежность кода. Под критической секцией понимается часть программы, исполнение которой может привести к возникновению эффекта гонок. Критерий Минимизация ошибок синхронизации $Y_{синх}$ рассчитывается по формуле:

$$Y_{синх} = \sum \beta_i * X_i / X_{базi.}, \quad (3)$$

где β_i, X_i, $X_{базi}$ приведены в табл. 2. Весовые коэффициенты определены в соответствии с методом экспертных оценок.

Таблица 2

Оценка влияния механизмов синхронизации

Критерии	**Влияние критических секций**	**Влияние спин-блокировки и циклического опроса**	**Влияние разделяемых переменных**
Вес β_i	1/3	2/3	
X_i	Кол-во строк кода, заключенных в критические секции	Кол-во спин-блокировок и циклических опросов	Кол-во разделяемых переменных
Базовые $X_{базi}$	Общее кол-во строк	Кол-во синхронизирующих функций	Общее кол-во переменных

Таким образом, проведен анализ схем распараллеливания и детализация компоненты надежность для ПК идентификации ТС трубопроводов.

Литература

1. *Владова А.Ю.*, *Кушнаренко В.М.*, *Кандыба Н.Е.*, *Степанов Е.П.*, *Владов Ю.Р.* Идентификация технического состояния теплоэнергетического оборудования: Монография. – Оренбург: РИК ГОУ ОГУ, 2004. – 203 с.

2. *Кулямин В.В.*, *Петренко О.Л.* Место тестирования среди методов оценки качества ПО // Тр. ин-та системного программирования, ИСП РАН. Т. 4. 2003.

3. *Романюк С.Г.* Оценка надежности программного обеспечения. НИИСИ РАН, Москва. Открытые системы, #04/1994.

4. *Касперски К.* ПК: решение проблем. – BHV, 2003. – 560 с.

5. *Хьюз К.*, *Хьюз Т.* Параллельное и распределенное программирование на C++. / Пер. с англ. – М.: Вильямс, 2004. – 672 с.

РАСПАРАЛЛЕЛИВАНИЕ УРАВНЕНИЯ КОАГУЛЯЦИИ

З.С. Гаева, А.В. Гасников[1]

ВЦ РАН, Москва

Среди всех физических процессов, влияющих на формирование частиц осадков в облаках, коагуляция играет важнейшую роль. Исследованию этого процесса посвящена обширная литература. Наибольшее распространение получило исследование коагуляционных процессов на основе уравнения Смолуховского (кинетическое уравнение пространственно-однородной коагуляции) [1, 2]:

$$\frac{\partial f(t,m)}{\partial t} = I(f,f) = -\int_0^{+\infty} K_{11}(m,m')f(t,m)f(t,m')dm' +$$

[1] Работа выполнена при финансовой поддержке РФФИ (коды проектов 03-01-00624а, 05-01-00942), РФФИофи (код проекта 05-01-08045), РГНФ (код проекта 05-02-02349а); по программе государственной поддержки ведущих научных школ (код проекта НШ – 1843.2003.1); при поддержке программы фундаментальных исследований ОМН РАН №3 «Вычислительные и информационные проблемы решения больших задач» (проект 3.13); при поддержке программы фундаментальных исследований РАН №16 «Математическое моделирование и интеллектуальные системы» (проект 4.1).

$$+\frac{1}{2}\int_0^m K_{11}(m-m',m')f(t,m-m')f(t,m')dm', \quad (1)$$

$$f(0,m)=e^{-m}, \quad t\geq 0, m\geq 0. \quad (2)$$

Считается, что указанное уравнение моделирует процессы коагуляции, протекающие, например, в облаке (коагуляция капель, агрегация снежинок), в коллоидных растворах, в космических пылевых облаках [3–5]. Функция распределения капель по массе: $f(m, t)$ определяется так, что $f(m, t)dm$ описывает среднюю концентрацию частиц системы, массы которых в момент t лежат в интервале $(m, m + dm)$. Ядро $K(m, m')$ – известная функция слияния частиц массами m и m', а ее численное значение пропорционально частоте слияний таких частиц в единице объема системы, т.е. величине, обратной среднему времени жизни частиц с указанными массами. Конкретный вид ядра получается на основе анализа эмпирического материала. Второй член в правой части уравнения коагуляции описывает рост числа частиц массы m за счет слияния частиц массами $(m - m')$ и m', а первый член – убыль частиц массы m из-за слияния этих частиц с частицами массы m'. Условие (2) задает функцию распределения капель в начальный момент времени.

Настоящую модель следует рассматривать как часть большой общей задачи, описания эволюции коагулирующих систем, которая далека от своего завершения и непосредственно связана с нуждами человечества, на пример, перераспределение осадков, осаждение облаков и туманов в окрестности аэропортов, предотвращение града [6].

Задача нахождения решения уравнения Смолуховского аналогична по своей сложности задаче нахождения решения уравнения Больцмана в газовой динамике. Поэтому зачастую проблемы кинетической теории газов и кинетической теории коагуляции являются тесно связанными.

Численно решать задачу Коши (1), (2) можно, как конечно-разностными методами, так и вариационными [7]. Решение кинетических уравнений, сопряжено с большими трудностями, обусловленными сложной структурой коэффициента коагуляции и интегрального оператора.

При использовании конечно-разностной схемы мы имеем, что коэффициент пропорциональности между временем, на котором мы следим за облаком, и временем счета без распараллеливания имеет порядок 10^9. А при распараллеливании, в N раз меньше, где N – число процессоров. Область интегрирования ~($m' \approx 10$)*(m ≈ 10)*T, шаги

~0,01*0,01*0,01, на каждом шаге ~10 операций с плавающей запятой (flop), т.е. порядка T *10^9 flop операций. Например, для того чтобы наблюдать за облаком в течение T = 100 с на одном процессоре Intel Celeron 2,4 ГГц RAM 512 Мб (под управлением Windows XP, компилятор Borland) при шаге интегрирования 0,01 требует порядка 30 минут. Это не удивительно, если учесть, что реальная производительность процессоров Intel линейки x86, нормированная на тактовую частоту 1 ГГц не превышает 300–350 Mflops/s [8]. Следовательно, только по прошествии 1000 секунд мы сможем сказать, какое состояние будет у облака через 100 секунд. Понятно, что при прогнозировании такая задержка не допустима.

Для решения задачи (1), (2) будем использовать метод Галеркина. С помощью метода Галеркина, за счет удачного выбора полной ортонормированной системы функций, можно значительно сократить объем вычислений. При решении кинетического уравнения в качестве такой системы используется система функций Дмитриева [9].

Пусть $\{\varphi_k(m) \mid 0 \le m \le \infty,\ k = 1, 2, \ldots, \infty\}$ – полная ортонормированная система функций в $L_2[0, \infty]$, полученная при ортогонализации (по Грамму–Шмидту) системы функций $\{e^{-0,5km},\ k = 1, 2, ..., \infty\}$, т.е. система функций Дмитриева. Будем искать решение в виде:

$$f(t,m) = \sum_{k=1}^{n} a_k(m)\varphi_k(t)\,. \tag{3}$$

Подставив (3) в (1) и скалярно умножив обе части полученной системы на первые n функций Дмитриева, получим систему из n обыкновенных дифференциальных уравнений относительно коэффициентов Галеркина:

$$\frac{da_k(t)}{dt} = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ijk}a_i(t)a_j(t)\,, \tag{4}$$

где

$$C_{ijk} = \int_0^\infty \varphi_k(m)\left(\int_0^\infty K_{11}(m,m')\varphi_i(m)\varphi_j(m)\,dm'\right)dm = \int_0^\infty I(\varphi_i,\varphi_j)\varphi_k(m)\,dm \tag{5}$$

с начальными данными:

$$a_k(0) = \int_0^\infty f(0,m)\varphi_k(m)\,dm\,. \tag{6}$$

Полученную задачу можно разбить на три отдельные подзадачи,

которые могут решаться независимо:

1) вычисление $n(n + 1)/2$ чисел – элементов нижней треугольной матрицы перехода G от экспоненциальной системы функций к системе функций Дмитриева [9];

2) вычисление n^3 коэффициентов C_{ijk} по формуле (5). Подчеркнем их независимость от времени, а потому их надо вычислить всего лишь один раз;

3) решение (например, с помощью простейшей, явной схемы Эйлера) задачи Коши (4), (6) для системы из n обыкновенных дифференциальных уравнений (СОДУ).

Из вышеприведенного разбиения на подзадачи становится ясно основное преимущество использования метода Галеркина по сравнению с разностной схемой. Решив сначала 1-и 2 подзадачи, мы можем решить СОДУ (4), (6), в которой коэффициент пропорциональности между временем, на котором мы следим за облаком, и временем счета без распараллеливания имеет порядок $(1/h_t)n^2 = 100\ n^2$. А при распараллеливании, в N раз меньше, где N – число процессоров.

Перейдем к распараллеливанию каждой из подзадач. Распараллеливать мы будем на MPI под Microsoft Visual C++ 6.0.

Подзадача 1) распараллеливается на n процессорах очевидным образом, поскольку есть явные формулы для элементов матрицы G [9]. Действительно, k-й процесс считает k чисел, помещает их в массив длины n, а оставшимся элементам массива он присваивает 0 значение. Затем каждый процесс вызывает MPI_Allgather [10], таким образом, каждый из n процессов знает все $n(n + 1)/2$ чисел. Следует отметить, что при использовании MPI_Allgather и некоторых других функций параллельной библиотеки, в силу их специфики, необходимо знать, как в C++ хранятся массивы. К примеру, если у нас двумерный массив, то сначала в памяти размещается его первая строка, потом вторая и т.д.

Подзадача 2) самая сложная с точки зрения объема вычислений, и, следовательно, может занимать много времени, если требуется хорошая точность. В этой связи ее имеет смысл запускать на максимально возможном количестве процессов. При фиксированном n синхронизационная накладка [10], связанная с обменом данными между процессами, растет значительно медленней с ростом числа процессов, чем время, которое экономиться за счет роста числа процессов. Это обусловлено огромным объемом вычислений, которое приходится на один процесс (следует сравнить с подзадачей 3) ниже). Таким образом, увеличив число процессов, мы увеличим точность, оставаясь в тех же вре-

менных рамках. Задача 2) считалась на n^2 процессорах.[2] Каждый (i, j)-й процесс, в декартовой топологии [10], вычисляет n коэффициентов C_{ijk} $k = 1, \ldots, n$, а затем вызывает MPI_Allgather, таким образом, каждый из n^2 процессов знает целиком трехмерный мерный массив C_{ijk}.

Подзадача 3) естественным образом распараллеливается на n процессорах. Сперва k-й процесс считает $a_k(0)$ по формуле (5). Затем процессы обмениваются $a_k(0)$ и, в результате, каждый процесс знает n-мерный вектор $\bar{a}(0)$. Заменив в (4) производную по времени конечной разностью, т.е. используя схему Эйлера, получим:

$$a_k\big((i+1)h\big) = a_k(ih) + h\sum_{i=1}^{n}\sum_{j=1}^{n} C_{ijk}\, a_i(ih) a_j(ih). \qquad (7)$$

Дальше будем действовать по индукции (база у нас уже есть). Предположим, что на шаге $i + 1$ каждый процесс знает вектор $\bar{a}(ih)$. Тогда k-й процесс сможет посчитать по формуле (7) значения $a_k((i+1)h)$, $k = 1, 2, \ldots, n$. Затем процессы обмениваются значениями $a_k((i+1)h)$ и, в результате, каждый процесс знает n-мерный вектор $\bar{a}((i+1)h)$. Заметим, что при так организованном распараллеливание отсутствует синхронизационная накладка, поскольку все процессы выполняют практически один и тот же объем вычислений. Поэтому достаточно просто синхронизировать моменты начала выполнения подзадачи 3) у всех процессов, что можно сделать с помощью MPI_Barrier [10]. Возникает естественный вопрос: почему бы не попытаться запустить 3) подзадачу на большем количестве процессов? Ответ достаточно прост: на каждом шаге каждый процесс делает n^2 арифметических операций с double и отправляет n сообщений. Поскольку при современных мощностях $n^2 < 700$ операций выполняются практически мгновенно, то временные затраты на посылку сообщений становятся сопоставимы с временем выполнения операций. Поэтому, значительно уменьшать число операций выполняемых одним процессом за счет увеличение числа процессов не имеет смысла, поскольку время при этом экономиться не будет, т.к. будет расти синхронизационная накладка.

---

[2] Как правило, $n \sim 5$–25 (число членов в ряде Галеркина), поэтому n^2 процессоров есть в нашем распоряжении. У нас был доступ к суперкомпьютеру «МВС 1000М», содержащему порядка 700 процессоров (пиковая производительность 10^{12} flops/s, реальная на порядок ниже), http://www.jscc.ru.

Теперь остановимся поподробнее на некоторых деталях реализации решения задачи. При интегрировании (5), (6) подынтегральные выражения имеют экспоненциальное убывание на бесконечности (точнее убывание вида $\sim e^{-0,5x}$), поэтому с хорошей точностью можно ограничиться интегрированием от 0 до $M_p \sim 20$–30, поскольку коэффициент при этой экспоненте, как показывает расчет, имеет, самое большое, порядок 10^3. При численном интегрировании нами использовалась формула Симпсона, которая на отрезке интегрирования $[a, b]$ дает точность:

$$| J - \tilde{J} | \leq \frac{1}{2880}(b-a)M_4 h^4, \qquad (8)$$

где M_4 – максимальное по модулю на отрезке $[a, b]$ значение 4-й производной подынтегрального выражения [11]. Так как $b - a < 30$ и подынтегральное выражение достаточно гладкое[3], то из (8) следует, что брать шаг интегрирования h меньше чем 0,01 не имеет смысла. В результате, при таком вычислении интегралов, подзадачи 1) и 2) считаются на 25 процессорах Intel Pentium IV[4] порядка 5–10 с ($h = 0,1$).

Подзадачи 1)–3) решались в одной программе. В связи с этим возникла необходимость создать новый коммуникатор [10]. Один и тот же процесс имел разные ранги в разных коммуникаторах. Также потребовалось широко распространять данные от одного (корневого) процесса всем остальным процессам в этом коммуникаторе MPI_Bcast [10]. Для вычисления интегралов были созданы две универсальные функции, которые получают в качестве одного из аргументов ссылку на функцию – подынтегральное выражение.

На рис.1, 2 представлены результаты решения задачи (1), (2) для моментов времени $T = 0,5$ и $T = 2,5$, когда $K_{11}(m, m') \equiv 1$. Для этого модельного случая известно точное решение, ему соответствует сплошная линия, а приближенному - линия с точками. Обратим внимание на то, что в ряде Галеркина было взято всего лишь $n = 5$ слагаемых, а шаг интегрирования равнялся 0,1. Из графиков можно сделать вывод о хорошей сходимости приближенного решения к точному.

---

[3] В силу экспоненциальных множителей и равномерно гладкое на $[0, +\infty]$.

[4] Этот процессор обладает наиболее высокой производительностью, нормированной на тактовую частоту, и принципиально отличается от остальных тем, что теоретически может выполнять 2 flops операции за один такт. На 25 процессорах задача запускалась на кластерах в ВЦ РАН.

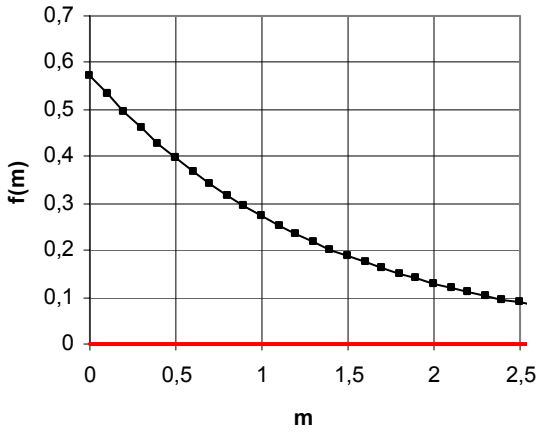


Рис. 1. $T = 0{,}5$

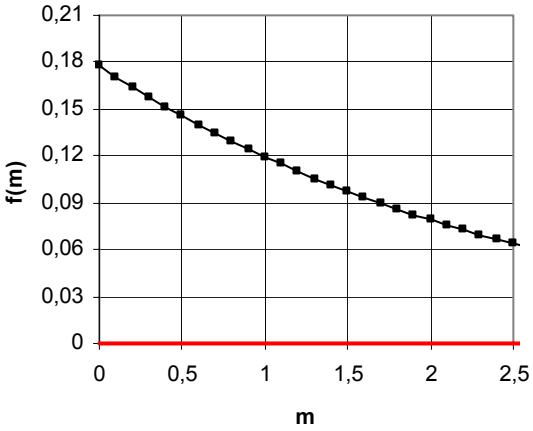


Рис. 2. $T = 2{,}5$

В заключении отметим, что использование описанного выше подхода для решения ряда систем интегро-дифференциальных уравнений позволяет существенно сократить время расчета, по сравнению с использованием распараллеленных разностных схем. Это достигается благодаря тому, что:

1) при удачном выборе базисной системы функций можно ограничиться нахождением лишь небольшого количество слагаемых в ряде

Галеркина, как правило, 10 слагаемых достаточно;

2) коэффициент пропорциональности между временем, на котором мы следим за облаком, и временем счета, при использовании метода Галеркина, оказывается на несколько порядков меньше, чем коэффициент пропорциональности соответствующий разностному методу;

3) приведенное выше распараллеливание не имеет синхронизационных накладок.

Исследование сходимости ряда Галеркина, т.е. проверка условия устойчивости галеркинской аппроксимации интегро-дифференциального оператора [7], для (1), (2) сделана в работе [9] с использованием проблемы моментов Чебышева–Маркова–Крейна.

Мы выражаем благодарность к.ф-м.н. доценту Н.Н. Оленеву и д.ф-м.н. профессору А.А. Шананину за оказанную нам помощь при выполнении этой работы.

Литература

1. *Волощук В.М.*, *Седунов Ю.С.* Процессы коагуляции в дисперсных системах. Л.: Гидрометеоиздат, 1975.

2. *Berry E.X.* Cloud droplet growth by collection, Journal of Atmosferic Sciences. V. 24. 1967. P. 278–286.

3. *Степанов А.С.* К выводу уравнения коагуляции // Тр. ИЭМ. Вып. 23. 1971. С. 3–16.

4. *Смолуховский М.* Опыт математической теории кинетики коагуляции коллоидных растворов. В кн.: Коагуляция коллоидов. М.: ОНТИ, 1936. С. 7–36.

5. *Галкин В.А.*, Уравнение Смолуховского. М.: Физматлит, 2000. 336 с.

6. *Гаева З.С.*, *Шананин А.А.* Численный метод решения задачи управления микроструктурой градового облака // Матем. модел. 2004. Т. 16, № 12. С. 69–84.

7. *Треногин В.А.* Функциональный анализ. М.: Физматлит, 2002, гл. VII и §38.

8. http://www.computational-battery.org/Maskinvare/Flops.html, http://www.spec.org, http://www.cs.utk.edu/~rwhaley/ATLAS/x86.html.

9. *Гаева З.С.*, *Шананин А.А.* Проблемы моментов Маркова–Чебышева исследование галеркинских приближений в одной задаче агрегации кристаллов // Матем. модел. 1995. Т. 7, № 9. С. 35–54.

10. *Оленев Н.Н.* Основы параллельного программирования в системе MPI. ВЦ РАН, 2005. Сб. лабораторных работ http://www.ccas.ru/mmes/ educat/lab04k/.

11. *Косарев В.И.* 12 лекций по вычислительной математике. М.: МФТИ, 2000. 81 с.

РАЗРАБОТКА ИНТЕГРИРОВАННОЙ СРЕДЫ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ КЛАСТЕРА НИЖЕГОРОДСКОГО УНИВЕРСИТЕТА

В.П. Гергель, А.Н. Свистунов

Нижегородский государственный университет им. Н.И. Лобачевского

В работе рассматриваются проблемы создания интегрированной среды высокопроизводительных вычислений для кластера Нижегородского университета, который был предоставлен ННГУ в рамках академической программы Интел в 2001 г. [3].

Важной отличительной особенностью кластера является его неоднородность (гетерогенность). В состав кластера входят рабочие места, оснащенные процессорами Intel Pentium 4 и соединенные относительно медленной сетью (100 Мбит), 32-битные вычислительные 2- и 4-процессорные серверы, 64-битные 2-процессорные серверы, обмен данными между которыми выполняется при помощи быстрых каналов передачи данных (1000 Мбит)

Основной операционной системой, установленной на узлах кластера является ОС Windows (на рабочих станциях установлена Windows 2000 Professional, на серверах установлена Windows 2003 Advanced Server).

Дополнительная информация о структуре кластера ННГУ доступна в сети Интернет по адресу http://www.software.unn.ac.ru/cluster.htm.

Эффективное использование быстродействующего компьютерного оборудования предполагает решение двух основных проблем, возникающих при эксплуатации кластера – проблему предоставления удаленного доступа к вычислительной среде кластера и проблему эффективного управления и мониторинга вычислительных задач, выполняющихся на кластере [1].

На сегодняшний день известно довольно много различных программных систем, позволяющих решать отмеченные проблемы. Большинство подобных систем традиционно разрабатывались для ОС UNIX, однако, в последнее время появились подобные системы и для ОС семейства Windows. К числу таких систем относится, например LSF (Load Sharing Facility, http://www.platform.com), или Cluster CoNTroller (http://www.mpi-softtech.com/).

Однако использование готовых систем управления кластером затруднено рядом обстоятельств. Прежде всего, это высокая стоимость подобных систем, достигающая, для кластера подобного кластеру Ни-

жегородского университета, десятков тысяч долларов. Вторым, не менее важным обстоятельством, является закрытость подобных систем, осложняющая проведение некоторых исследовательских работ. Дело в том, что кроме задачи обеспечения функционирования кластерной системы ставилась еще и задача создания испытательного стенда для проведения экспериментов по апробации различных алгоритмов планирования задач на кластерных системах. Для кластера Нижегородского университета планирование распределения вычислительной нагрузки по узлам кластера является практически важной задачей, в силу его неоднородности.

Отмеченные факторы приводят к необходимости создания собственных средств поддержки организации высокопроизводительных вычислений на кластере.

Разрабатываемая программная система должна была решить еще одну важную задачу – задачу интеграции вычислительного кластера ННГУ с вычислительным кластером Института прикладной физики РАН (ИПФ РАН). К числу обстоятельств, затрудняющих такую интеграцию, следует отнести тот факт, что в отличие от вычислительного кластера университета, вычислительный кластер ИПФ РАН в качестве базовой операционной системы использует один из клонов UNIX[4].

При построении системы управления кластером со стороны потенциальных пользователей были определены следующие требования:

- реализация по крайней мере минимального набора операций (загрузка задачи, добавление задачи в очередь задач, получение текущего статуса очереди задач и текущей выполняемой задачи, получение результатов вычислений);
- простой и удобный способ доступа, не требующий установки на рабочих станциях пользователей какого-либо специального программного обеспечения, позволяющий получить доступ к системе из любой точки;
- собственная система авторизации пользователей не связанная напрямую с системой авторизации операционной системы;
- наличие системы очередей задач;
- сохранение задач пользователя после их выполнения и возможность их повторного запуска;
- автоматическое сохранение результатов работы;
- возможность управления несколькими вычислительными кластерами

При построении программной системы за основу была взята сло-

жившаяся архитектура системы мониторинга и управления [2], включающая, как правило, следующие составные части (см. рис.);

Архитектура системы

- компонент, взаимодействующий с пользователем (Менеджер доступа), позволяющий ставить задачи в очередь, удалять задачи из очереди, просматривать статус задач и т.д., а также ведущий базу данных пользователей и базу данных результатов;
- компонент, оперирующий с очередью заданий (Диспетчер заданий), распределяющий задания по узлам кластера и ставящий их в очередь для выполнения;
- компонент, обеспечивающий мониторинг кластера (Супервизор кластера) и непосредственное выделение ресурсов.

В настоящее время разработан и внедрен опытный вариант системы, обладающий следующими возможностями:

- поддержка минимально необходимого набора операций по управлению задачами пользователей;
- возможность доступа к системе с любого компьютера, подключенного к сети Интернет;
- отсутствие необходимости установки на компьютере пользователя специального программного обеспечения. Использование в качестве клиента web-браузера, telnet-клиента, различных специализированных программ;
- хранение очереди заданий (как ждущих своей очереди на выполнение, так и уже завершившихся, сформировавших результат) во внешней базе данных, что позволяет обеспечить устойчивость системы в случае сбоя;
- возможность замены планировщика и изменения стратегии планирования;
- ведение статистики использования задачами пользователей вычислительных ресурсов кластера;
- возможность управления несколькими вычислительными кластерами

Система активно используется широким кругом пользователей и сотрудников Центра компьютерного моделирования. Возможности, предоставляемые системой, используются при разработке и эксплуатации учебных и исследовательских программных комплексов, таких как программная система для изучения и исследования параллельных методов решения сложных вычислительных задач (ПараЛаб) и система параллельной многоэкстремальной оптимизации «Абсолют Эксперт».

Работа над системой продолжается в нескольких направлениях:

- расширение функциональности системы, производящееся в тесном контакте с конечными пользователями;
- исследования в области различных алгоритмов планирования;
- оптимизация процесса мониторинга ресурсов кластера;
- исследование подходов к созданию шлюза между кластером ННГУ и кластером ИПФ РАН;

Литература

1. *Rajkumar Buyya*. High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., 1999.

2. *Saphir W.*, *Tanner L.A.*, *Traversat B*. Job Management Requirements for NAS Parallel Systems and Clusters. NAS Technical Report NAS-95-006 February 95.

3. *Гергель В.П.*, *Стронгин Р.Г.* Высокопроизводительный вычислительный кластер Нижегородского университета. Матер. конф. Relarn. 2002. Н. Новгород.

4. *Дрейбанд М.С.* Вычислительный кластер ИПФ РАН. Матер. конф. Relarn. 2002. Н. Новгород.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ ОДНОГО АЛГОРИТМА НАХОЖДЕНИЯ ЦЕНЫ ОПЦИОНОВ БЕРМУДСКОГО ТИПА

А.С. Горбунова, Е.А. Козинов, И.Б. Мееров, А.В. Шишков
Нижегородский государственный университет им. Н.И. Лобачевского

А.Ф. Николаев
Нижегородский Центр Intel по разработке программного обеспечения

Введение

Финансовая математика – раздел прикладной математики, ориентированный на решение разнообразных задач финансовых рынков (см., например, [1]). Объекты исследования в данной области часто описываются моделями, восстановление аналитических решений для которых оказывается затруднительным. При этом существенное воздействие на процесс моделирования оказывает наличие факторов неопределенности. Как следствие, решение многих задач финансовой математики требует применения методов Монте-Карло и средств имитационного моделирования [1]. Успех применения этих методов существенно зависит от качества используемых генераторов случайных чисел и, как правило, требует больших вычислительных затрат. По этой причине разработка параллельных реализаций расчетных алгоритмов для решения задач финансовой математики оказывается особенно актуальной.

В этой работе мы рассматриваем одну из частных задач финансовой математики – вычисление цены опционов Бермудского типа. Эффективное решение этой задачи, являющееся крайне важным для практического применения, достигается с помощью параллельной реализации алгоритма Broadie–Glasserman Random Trees [2].

Постановка задачи

Рассмотрим N-мерный финансовый рынок, эволюционирующий в непрерывном времени (подробнее о модели Блэка–Шоулса и ее много-

численных обобщениях, см., например, [0, 3] и библиографию там):

$$dB_t = rB_t dt,\ B_0 > 0, \tag{1}$$

$$dS_t^i = S_t^i((\theta^i - \delta^i)dt + \sigma^i dW^i),\quad S_0^i > 0, \tag{2}$$

где процессы $B = (B_t)_{t\geq 0}$ и $S^i = (S_t^i)_{t\geq 0}$, i =1, …, N описывают поведение облигации B и i-й акции в каждый момент времени $t \geq 0$, соответственно. Поведение облигации B задается процентной ставкой r, изменение стоимости акции S^i определяется волатильностью σ^i и нормой возврата θ^i, δ^i – ставка дивиденда. Случайные возмущения в цене акции S^i описываются i-й компонентной векторного Винеровского процесса $W^i = (W_t^i)_{t\geq 0}$ $(W = (W^1, ..., W^N))$.

Предположим, что σ^i и $\mu^i\ \theta^i - \delta^i$, i = 1, …, N являются функциями времени, а ковариационная матрица $COV = (cov_{ij})_{i,j=1...N}$ отражает зависимости между ценами рисковых активов финансового рынка в каждый момент времени.

Рассмотрим одну из часто встречающихся разновидностей опциона – max-call опцион, для которого функция выплаты $h(t, S_t)$ имеет следующий вид:

$$h(t, S_t) = (\max_{i=1...N}(S_t^i) - P)^+, \tag{3}$$

где положительная константа P определяется опционным контрактом, $x^+ = \max(x, 0)$.

Решим задачу вычисления стоимости опциона Бермудского типа, который может быть предъявлен к исполнению в любой из заранее фиксированных моментов $t \in Time = \{t_0 = 0, t_1, t_2, ..., t_d = T\}$, где момент времени T задает истечение срока действия опциона.

Математический метод решения задачи

Математически поиск стоимости опциона сводится к решению стохастической оптимизационной задачи (подробнее см. [2])

$$Q = \max_{\tau \in Time} E(h(S_\tau)/B_\tau). \tag{4}$$

Пусть

$$Q(t, x) = \max\left(h(t, x), E\{Q(t+1, S_{t+1})B_t / B_{t+1}\big|S_t = x\}\right), \tag{5}$$

$$Q(t, S_T) = h(T, S_T).$$

Стоимость опциона Бермудского типа определяется как C = $Q(0, S_0)$.

Описание алгоритма

Для решения задачи был использован популярный алгоритм Broadie–Glasserman Random Trees [2], основанный на Монте-Карло моделировании. Так одиночный эксперимент состоит в генерации дерева стоимостей акций на основе модели Блэка–Шоулса, [3].

Корень дерева содержит N-мерный вектор цен акций в начальный момент времени, каждая вершина дерева хранит N-мерный вектор цен акций в соответствующий момент времени t_i из множества *Time*. Таким образом, каждый уровень дерева соответствует некоторому моменту исполнения опциона t_i. Количество уровней соответствует количеству моментов применения опциона (d), а количество ветвей дерева (K) является параметром алгоритма. Значения цен акций в узлах рассчитываются путем численного интегрирования системы стохастических дифференциальных уравнений (2) (приращения Винеровского процесса формируются с помощью генератора случайных чисел гауссова распределения). Пример дерева при $N = 1$, $K = 3$ и $d = 3$ приведен на рис. 1. В работе [2] авторы алгоритма указывают на сложность построения несмещенной оценки стоимости опциона и предлагают вычислять две оценки, первая из которых сходится к точному решению справа, а вторая – слева. Способы построения оценок подробно описаны в статье.

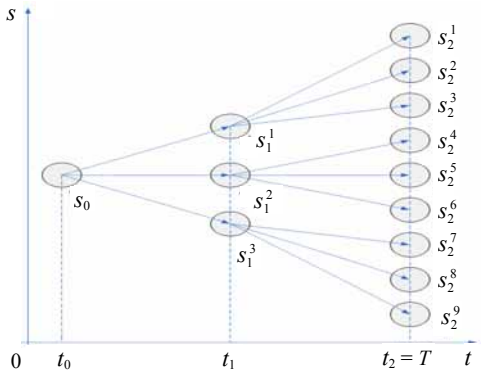


Рис. 1. Пример дерева стоимостей акций для $N = 1$, $K = 3$, $d = 3$

Повторяя указанные действия многократно, мы получаем средние значения верхней и нижней оценок стоимости Бермудского опциона, их дисперсию и определяем соответствующий доверительный интервал для цены актива.

Последовательная реализация алгоритма

Реализация описанного выше алгоритма проводилась на языке программирования C с использованием библиотек Intel® Math Kernel Library (MKL) 8.0, MPICH for Windows**, MPICH2 for Windows** и оптимизирующего компилятора Intel® C/C++ Compiler 9.0 for Windows**.

Из-за необходимости генерации и обходов дерева стоимостей акций (см. рис. 1) алгоритм является экспоненциальным по количеству моментов исполнения Бермудского опциона. «Наивная» реализация алгоритма требует хранения дерева стоимостей акций и оценок в каждом узле, что приводит к экспоненциальному росту требуемой памяти и крайне неэффективной работе алгоритма. В связи с этим мы создали оптимизированную для невырожденного случая ($d > 1$) реализацию алгоритма. Основные изменения по сравнению с «наивной» реализацией состоят в следующем:

1. Оптимизированная реализация алгоритма предполагает изменение схем обхода дерева с уменьшением объема памяти, используемой для хранения исходных и промежуточных данных,

$$\text{с } C(1 + K + K^2 + \ldots + K^{d-1})\ \textit{size of}\ (\textit{type})$$

$$\text{до } N(d + K - 1)\ \textit{size of}\ (\textit{type}) + 2((d - 2)K + 1)\ \textit{size of}\ (\textit{type}),$$

где N, d и K – параметры задачи, а *type* – тип данных для представления стоимости акций. Как следствие, затраты памяти зависят линейным образом от количества моментов исполнения опциона.

2. Эксперименты показали, что использование одинарной точности не ухудшает качество решения и уменьшает временные затраты.

Проведенные изменения позволяют получить ускорение в 2,7 раза по сравнению с «наивной» реализацией алгоритма, однако, время работы программы по-прежнему остается достаточно большим.

Параллельная реализация алгоритма

Основная идея распараллеливания алгоритма базируется на том факте, что генерация ветвей в дереве стоимостей акций происходит независимо друг от друга. Следующий за этим подсчет верхней и ниж-

ней оценок начинается с последнего уровня, далее проходит все уровни дерева до первого, используя на каждом шаге лишь результаты предыдущего шага (работает схема динамического программирования). Лишь на нулевом уровне (в корне дерева) происходит агрегация результатов, собранных на различных ветвях дерева.

Схема распараллеливания состоит в следующем:

1) головной процесс формирует первый уровень дерева;

2) построение ветвей дерева равномерно распределяется между процессами (в том числе и головным) в соответствии с рис. 2;

3) по окончании построения дерева каждый процесс пересчитывает оценки от листьев дерева к корню вплоть до первого уровня;

4) получив оценки первого уровня, процессы пересылают результат головному процессу;

5) головной процесс агрегирует полученные результаты и вычисляет верхнюю и нижнюю оценки стоимости опциона.

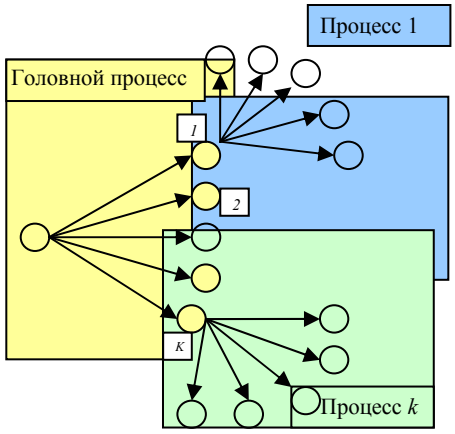


Рис. 2. Схема распараллеливания

Результаты вычислительных экспериментов

Результаты вычислительных экспериментов получены на Intel® Itanium® 2 системе (4x900MHz, 3MB L3, 2GB RAM, SCSI 2x73.4 GB). Для проведения экспериментов использовались MPICH** 1.2.5, MPICH** 2.0, Intel® MPI.

Для экспериментов были выбраны следующие значения параметров:

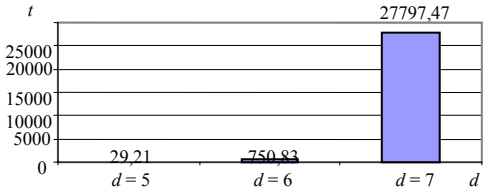
N	5
K	32
D	5, 6 и 7 в разных экспериментах
T (дней)	365
R	7%
S_0	(100; 110; 120; 130; 140)
P	100
Коэффициент корреляции	0,43
μ	(0,03; 0,035; 0,04; 0,045; 0,05)
σ	(0,34; 0,345; 0,35; 0,355; 0,36)

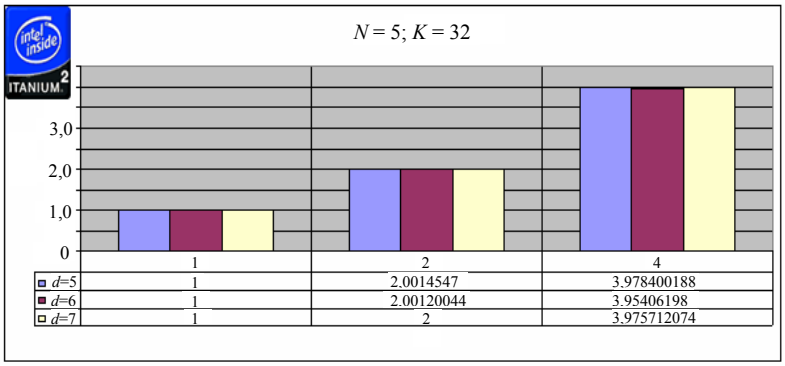
Следующие результаты демонстрирует результаты работы последовательной реализации алгоритма для 5, 6 и 7 моментов исполнения опциона и демонстрирует экспоненциальный рост временных затрат.

Описание	**Нижняя оценка**	**Верхняя оценка**	**Время, с**	**Количество выполненных тестов**
$d = 5$	39,828267	40,8709168	29,21	10
$d = 6$	39,602242	40,773254	750,83	10
$d = 7$	40,484818	41,732002	27797,47	1

Для параллельной версии алгоритма, учитывая небольшой объем пересылок, схема обеспечивает почти линейное ускорение. Данный факт подтверждается вычислительными экспериментами на различных архитектурах (IA-32, IA-64 и процессорах Intel® с EM64T технологией) под ОС Windows** и Linux**.

Приведем в качестве примера анализ ускорения вычислений для 1, 2-х и 4-х процессоров Intel® Itanium® 2.





Выводы

1. Рассмотренный в работе алгоритм допускает эффективную последовательную реализацию, связанную с использованием одинарной точности и оптимизацией работы с памятью.

2. Данный алгоритм допускает эффективное распараллеливание с почти линейным ускорением.

3. К сожалению, алгоритм является экспоненциальным по числу моментов исполнения опциона. Несмотря на эффективную параллельную реализацию, на 4-х процессорах Intel® Itanium® 2 вычисления для одного эксперимента занимают более двух часов при $d = 7$.

4. В настоящее время мы работаем над реализацией другого алгоритма, являющегося линейным по числу моментов исполнения опциона, что позволит вычислять значение опциона при $d > 7$.

Литература

1. *Ширяев А.Н.* Основы стохастической финансовой математики. В 2 т. – М.:Фазис, 1998.

2. *Boyle P.P.*, *Broadie M.*, *Glasserman P.* Monte Carlo methods for security pricing // Journal of Economic Dynamics and Control, June 1997. V. 21. P. 1267–1321.

3. *Broadie M.*, *Glasserman P.* Pricing American-style securities using simulation // Journal of Economic Dynamics and Control, June 1997. V. 21. P. 1323–1352.

4. *Black F.*, *Scholes M.* The pricing options and corporate liabilities // Journal of Political Economy, 1973. V. 3. P. 637–659.

ными знаками, либо зарегистрированными товарными знаками, права на которые принадлежат корпорации Intel или ее подразделениям на территории США и других стран.

Все фирменные наименования, товарные знаки, знаки обслуживания и иные обозначения и наименования являются объектами прав их законных правообладателей

АНАЛИЗ ЭФФЕКТИВНОСТИ РАСПАРАЛЛЕЛЕННОГО АЛГОРИТМА КОНЕЧНО-ЭЛЕМЕНТНОГО РЕШЕНИЯ ТРЕХМЕРНЫХ НЕЛИНЕЙНЫХ ЗАДАЧ ДИНАМИКИ КОНСТРУКЦИЙ НА КЛАСТЕРАХ

А.В. Гордиенко, А.В. Дудник, А.И. Кибец, Ю.И. Кибец

НИИ механики Нижегородского госуниверситета

В [1] изложен распараллеленный алгоритм конечно-элементной методики [2] решения трехмерных нелинейных задач динамики конструкций. Данный алгоритм реализован в рамках вычислительной системы (ВС) «Динамика-3» [3] и на ряде примеров исследована его эффективность.

Постановка задачи. Метод решения

В [1, 2] рассматриваются трехмерные задачи нестационарного деформирования конструкций, включающих массивные тела и оболочки. Конструкции могут быть выполнены из кусочно однородных, изотропных, упругопластических материалов и подкреплены дискретной системой криволинейных стержней, воспринимающих усилия растяжения-сжатия. Допускается, что количество армирующих стержней сравнительно невелико и их расположение в основном материале может быть нерегулярным. Отдельные конструктивные элементы могут вступать в контакт друг с другом или взаимодействовать с внешними телами.

Определяющая система уравнений сплошной среды формулируется в переменных Лагранжа. Уравнение движения выводятся из вариационного принципа Журдена. Кинематические соотношения формулируются в метрике текущего состояния. В качестве уравнений состояния для традиционных материалов используются соотношения теории течения с кинематическим и изотропным упрочнением. Бетон рассматривается как разномодульная, упругая, физически нелинейная среда, свойства которой зависят от вида напряженно-деформированного со-

стояния (НДС) и текущего уровня поврежденности материала [4]. В основу модели хрупкого разрушения разномодульных материалов положен критерий максимальных нормальных напряжений. На контактирующих поверхностях задаются условия непроникания. Гипотезы, принятые в теории тонкостенных элементов конструкций вводятся на этапе дискретизации определяющей системы уравнений. Благодаря этому можно упростить стыковку разных типов конструктивных элементов и учесть особенности их напряженно-деформированного состояния.

Решение задачи при заданных начальных и граничных условиях основано на методе конечных элементов и явной конечно-разностной схеме интегрирования по времени типа «крест» [2]. Для учета дискретного расположения подкрепляющих стержней разработан конечный элемент, описывающий взаимодействие сплошной среды и арматуры [5]. Согласно [5] напряжения в стержне заменяются статически эквивалентными силами узлов КЭ сетки основного материала, что позволяет учесть дискретность армирующей системы, не вводя дополнительных степеней свободы в конечно-элементную модель конструкции.

Метод распараллеливания

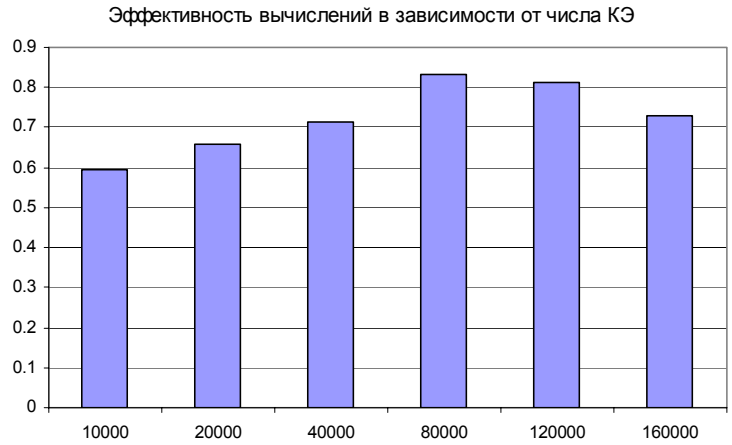
В дискретном аналоге уравнений движения матрица масс является диагональной и не возникает необходимости в ее обращении. Благодаря этому расчетная схема обладает большим параллелизмом по данным и вычислениям. В настоящей работе для ее распараллеливания был применен метод пространственной декомпозиции расчетной области (domain decomposition method) [1, 6, 7], в соответствии с которым вычисления в разных точках расчетной области распределяются в разные узлы кластера. Алгоритм решения задачи на каждом временном слое распадается на две части: последовательную и параллельную. Основной объем вычислений (определение компонент деформаций, напряжений, узловых сил, интегрирование уравнений движения и т.д.) осуществляется параллельно. В последовательной части происходит согласование рассчитанных величин, полученных на разных узлах кластера [1].

Оценка эффективности параллельных вычислений

Отладка и тестирование модернизированной версии ВС «Динамика 3» производились на кластере, включающем в себя два узла - персональные ЭВМ на базе процессоров Intel Pentium IV с тактовой частотой 2.0 ГГц. Объем оперативной памяти каждого узла составляет 512 Мб.

В качестве коммуникационной среды использовалась сеть Ethernet, обеспечивающая обмен данными со скоростью до 80 Мбит/с, и интерфейс обмена сообщениями mpich-1.2.5. Операционная система кластера – Linux ASP 7.3.

Эффективность работы распараллеленного варианта ВС «Динамика-3» оценивалась на ряде тестовых задач. В частности, рассмотрен изгиб жестко заделанной на торцах железобетонной балки [5] под действием импульса давления. Расчеты проводились на сетках, содержащих от 10000 до 160000 конечных элементов. Проведено сравнение времени решения задачи на одном и двух процессорах. На основе этих данных построена диаграмма зависимости эффективности вычислений [6], от количества конечных элементов сетки расчетной области (см. рисунок).



Согласно полученным результатам максимальная эффективность на данном кластере достигается при решении задачи на сетках, имеющих от 80000 до 120000 КЭ, и составляет более 80%. Причиной этому следует считать пропускные характеристики коммуникационной среды. При малом числе КЭ загруженность процессоров узлов небольшая и время вычислений значительно меньше времени обмена данными. С увеличением числа КЭ доля вычислений возрастает до тех пор, пока объем данных пересылаемых за единицу времени не достигает величины пропускной способности коммуникационной среды. Дальнейшее увеличение числа КЭ приводит к увеличению нагрузки на коммуникационную среду при обмене данными между узлами кластера и сниже-

нию эффективности параллельных вычислений

В процессе счета необходимость обмена данными между вычислительными узлами возникает при согласовании расчетных величин на границах смежных подобластей, принадлежащих различным узлам и при вычислении сил в контактных зонах. Доля этих последовательных частей алгоритма в общем времени решения задачи зависит от двух факторов: конкретного вида расчетной области и способа ее разбиения на подобласти. Очевидно, что увеличение участков границ смежных подобластей требует большего времени на согласование узловых сил и скоростей между ними. То же самое справедливо и для зон контакта. Поэтому для повышения эффективности вычислений на кластере необходимо еще на этапе подготовки исходных данных для решения задачи минимизировать размеры границ сшивки подобластей и зон контакта.

Еще одним моментом, на который следует обратить внимание, является то, что при образовании пластических деформаций или зон разрушения активизируется соответствующая процедура корректировки напряженно-деформированного состояния, требующая дополнительных вычислительных затрат. Это так же необходимо учитывать при разбиении расчетной области на подобласти, поскольку даже при одинаковом количественном распределении конечных элементов между вычислительными узлами, возможна их различная загруженность в процессе решения задачи.

Работа выполнена при частичном финансировании Российского фонда фундаментальных исследований (код проекта 05-08-33618-а) и ведомственной научной программы «Развитие научного потенциала высшей школы» (проект № 4661).

Литература

1. *Баженов В.Г.*, *Гордиенко А.В.*, *Кибец А.И.*, *Лаптев П.В.* Адаптация последовательной методики решения нелинейных задач динамики конструкций для многопроцессорных ЭВМ // Материалы IV Международного научно-практического семинара и Всероссийской молодежной школ «Высокопроизводительные параллельные вычисления на кластерных системах» / Под редакцией члена-корреспондента РАН В.А.Сойфера, Самара, 30сентября – 2 октября 2004 г. Самара. 2004. С. 20–25.

2. *Баженов В.Г.*, *Кибец А.И.* Численное моделирование трехмерных задач нестационарного деформирования упругопластических конструкций методом конечного элемента // Изв. РАН МТТ. 1994. № 1. С. 52–59.

3. Сертификат соответствия Госстандарта России № РОСС RU.ME20.H00338.

4. *Баженов В.Г.*, *Гордиенко А.В.*, *Дудник А.В.*, *Кибец А.И.*, *Торопов В.В.* Применение метода конечных элементов для решения трехмерных задач деформирования и разрушения кирпичной кладки при взрывном нагружении // Вестник ННГУ. Серия Механика. 2004. Вып. 1(6). С. 124–130.

5. *Дудник А.В.*, *Кибец А.И.*, *Кибец Ю.И.* Конечно-элементная методика решения трехмерной нестационарной задачи динамики дискретно армированных конструкций // Проблемы прочности и пластичности: Межвуз. сб. / Н.Новгород: Изд-во ННГУ. – 2003 – Вып.65. – С.92–96.

6. *Воеводин В.В.*, *Воеводин Вл.В.* Параллельные вычисления. БХВ-Петербург, 2002. 609 с.

7. *Копысов С.П.*, *Красноперов И.В.*, *Рычков В.Н.* Объектно-ориентированный метод декомпозиции области. Вычислительные методы и программирование. 2003. Т. 4. с. 1–18

ОЦЕНКА ТРУДОЕМКОСТИ АЛГОРИТМОВ КОЛЛЕКТИВНЫХ ОПЕРАЦИЙ MPI ДЛЯ КЛАСТЕРОВ МНОГОУРОВНЕВОЙ АРХИТЕКТУРЫ

А.В. Гришагин, А.Л. Курылев, А.В. Линев

Нижегородский государственный университет им. Н.И Лобачевского

Введение

Коллективные операции являются важным и часто используемым компонентом MPI, и в настоящее время различные исследовательские коллективы выполняют разработки, связанные с повышением их производительности. Однако в большинстве случаев при оценке стоимости операций предполагается, что расположение процессов на узлах сети не имеет значение, а число одновременно взаимодействующих пар процессов можно учесть в виде дополнительного слагаемого и в дальнейшем не учитывать его изменение. Проведенные нами исследование показывают, что в кластерах из машин с большим числом процессоров данные параметры вносят существенный вклад в результат оценки стоимости. Например, при изменении нумерации процессов в коммуникаторе время выполнения одного и того же алгоритма операции bcast может отличаться на 30%.

Кластер многоуровневой архитектуры

В качестве примера кластера с многоуровневой архитектурой рассмотрим кластер, построенный на базе POWER5-систем. Он имеет иерархическую архитектуру и содержит следующие уровни:

- сеть (network, cluster);
- узел (node);
- книга (book);
- сборка (multi-chip module);
- чип (chip);
- ядро (core);
- виртуальный процессор (virtual processor).

Каждому уровню соответствует свой способ взаимодействия процессов, выполняющихся в пределах одного объекта данного уровня (сетевое взаимодействие, общая память, общий кэш L3 или L2 и т.д.). Соответственно, время передачи сообщения существенно зависит от того, механизм какого уровня используется взаимодействующими процессами.

В настоящий момент мы ограничиваем рассмотрение двумя уровнями архитектуры:

- передача данных внутри SMP-узла через общую память;
- передача данных между SMP-узлами через сеть.

Можно принимать во внимание нижележащие уровни, но в современных операционных системах существует миграция процессов внутри SMP-узла, например, в Linux с ядрами семейства 2.6, реализованы специализированные алгоритмы, контролирующие перемещение процессов в рамках узла с учетом неоднородной архитектуры SMP-системы.

Измерение стоимости операций точка-точка

Для оценки стоимости коллективных операций мы предлагаем предварительно измерить стоимость операций точка-точка. В дальнейших рассуждениях мы рассматриваем однородный кластер (то есть кластер, состоящий из одинаковых узлов, единообразно подключенных к сети), и исходим из следующих гипотез:

- одновременная передача и прием сообщений через одно подключение к сети не изменяют время приема и передачи, так как современные сетевые компоненты функционируют в полнодуплексном режиме;
- одновременная передача и прием сообщений через общую память снижают скорость приема и передачи;
- при выполнении операции доставки сообщения время, вклад операций передачи и приема в общее время выполнения считается равным.

При принятии перечисленных гипотез для оценки времени выполнения элементарных операций на конкретном кластере необходимо выполнить следующие эксперименты:

- определение зависимости времени передачи сообщения через сеть от размера сообщения и числа передающих процессов на SMP-узле. В ходе эксперимента производится передача сообщений различного размера между парами процессов, при этом процессы каждой пары расположены на различных узлах;
- определение зависимости времени передачи сообщения через общую память от размера сообщения и числа взаимодействующих процессов на SMP-узле. В ходе эксперимента производится передача сообщений различного размера между несколькими процессами, расположенными на одном узле.

По результатам экспериментов мы можем оценить зависимость стоимости операций точка-точка от следующих параметров:

- длина сообщения;
- число процессов на SMP-узле, одновременно выполняющих прием/передачу через сетевое соединение;
- число процессов на SMP-узле, одновременно выполняющих прием/передачу через общую память.

Оценка стоимости коллективных операций

Расчет стоимости алгоритма коллективной операции выполняется для некоторого конкретного случая размещения процессов на узлах сети. Оценка выполняется следующим образом:

- определяется число шагов, требуемых алгоритму коллективной операции;
- для каждого шага алгоритма определяется:
  - какие процессы передают и принимают сообщения и размер этих сообщений,
  - какие процессы совместно используют подключение к сети для приема или передачи, и какие процессы совместно используют общую память,
  - рассчитывается время выполнения шага для каждого процесса;
- трудоемкость алгоритма принимается как сумма максимальных значений, полученных на каждом шаге.

Результаты исследований показывают, что применение предлагаемого подхода позволяет получить оценки стоимости алгоритмов кол-

лективных операций, качественно повторяющие экспериментальные данные и позволяющие проводить сравнение производительности алгоритмов.

Заключение

Проведенные исследования показывают существенную зависимость стоимости алгоритмов коллективных операций от размещения процессов на узлах сети и различия в стоимости операций точка-точка через разделяемую память и сетевое соединение. Очевидна необходимость разработки специализированных алгоритмов коллективных операций для таких кластеров. Предлагаемый подход к оценке стоимости алгоритмов коллективных операций учитывает многоуровневую архитектуру кластера и может быть использован как для реализации динамического выбора оптимального алгоритма при вызове коллективной операции, так и для оценки производительности новых алгоритмов.

Работа выполнена при поддержке IBM Faculty Awards for Innovation Program.

ПАРАЛЛЕЛЬНЫЙ МЕТОД РЕШЕНИЯ МНОГОМЕРНЫХ МНОГОЭКСТРЕМАЛЬНЫХ ЗАДАЧ С НЕВЫПУКЛЫМИ ОГРАНИЧЕНИЯМИ

В.А.Гришагин

Нижегородский государственный университет им. Н.И.Лобачевского

Я.Д. Сергеев

Калабрийский университет, Италия

Рассмотрим конечномерную задачу оптимизации (задачу нелинейного программирования)

$$f(y) \to \min, y \in Q \subseteq R^N \quad (1)$$

$$Q = \{y \in D : g_j(y) \le 0, 1 \le j \le m\} \quad (2)$$

$$D = \{y \in R^N : y_i \in [a_i, b_i], 1 \le i \le N\}, \quad (3)$$

т.е. задачу отыскания экстремальных значений целевой (минимизируемой) функции $f(y)$ в области Q, задаваемой *координатными* (3) и *функциональными* (2) *ограничениями* на выбор допустимых точек (векторов) $y = (y_1, y_2, \ldots, y_N)$, принадлежащих N-мерному евклидову пространству R^N.

Предметом рассмотрения настоящей статьи являются *многоэкстремальные* задачи оптимизации, т.е. задачи, в которых целевая функция $f(y)$ имеет в допустимой области Q несколько локальных экстремумов, а ограничения $g_j(y)$ из (2) также могут быть многоэкстремальными. Будем далее предполагать, что функции $g_j(y)$, $1 \le j \le m$ удовлетворяют условию Липшица каждое со своей константой L_j, а целевая функция $f(y)$ также является липшицевой с константой L_{m+1}.

Важнейшими факторами, определяющими сложность исследования задач данного класса, являются размерность N и структура допустимой области Q. В силу того, что глобальный экстремум является не локальной, а интегральной характеристикой минимизируемой функции, т.е. для его определения необходимо сопоставить значение в точке глобального минимума со значениями функции в остальных точках области поиска, поисковый метод многоэкстремальной оптимизации вынужден строить некоторую сетку испытаний, покрывающую допустимую область. Вследствие этого для рассматриваемого класса задач имеет место так называемое «проклятие размерности», состоящее в экспоненциальном росте вычислительных затрат при увеличении размерности. А именно: если в одномерной задаче для достижения точности решения ε требуется p вычислений функции, то в задаче с размерностью N для решения с той же точностью необходимо осуществить αp^N испытаний, где α зависит от целевой функции, допустимой области и используемого метода. В свою очередь многоэкстремальность ограничений может порождать области сложной структуры (невыпуклые, неодносвязные и даже несвязные), существенным образом влияющие на процесс поиска решения.

Для анализа задач рассматриваемого класса в данной статье предлагается подход, основанный на идеях *редукции сложности*, когда решение исходной задачи заменяется решением одной или нескольких более простых задач. Такая редукция осуществляется с помощью схемы вложенной оптимизации, позволяющей заменить решение многомерной задачи решением семейства рекурсивно вложенных одномерных подзадач, в сочетании с индексным методом учета ограничений [1, 2], обобщенным на многомерный случай [1, 3, 4].

Индексная функция

Введем обозначение $Q_0 = D$ и определим набор вложенных множеств

$$Q_j = \{y \in Q_{j-1} : g_j(y) \le 0\}, 1 \le j \le m, \tag{4}$$

для которых справедливо следующее включение:

$$D = Q_0 \supseteq Q_1 \supseteq \mathsf{K} \supseteq Q_m = Q\,, \tag{5}$$

причем все непустые множества Q_j являются замкнутыми вследствие непрерывности функций $g_j(y)$, обеспечиваемой их липшицевостью.

Введем понятие индекса $\nu(y)$ точки $y \in D$ либо как номера первого нарушенного в этой точке ограничения (в соответствии с порядком нумерации, установленным в (2)), т.е.

$$g_j(y) \le 0, 1 \le j \le \nu(y) - 1,\ g_{\nu(y)}(y) > 0\,, \tag{6}$$

либо принимающего значение $\nu(y) = m + 1$, если в точке y выполнены все ограничения.

Определим максимальное значение M среди индексов точек гиперпараллелепипеда D, т.е.

$$M = \max_{y \in D} \nu(y)\,.$$

Тогда из (4), (5) следует, что если допустимая область Q не пуста, то $M = m + 1$, а в противном случае

$$Q_{M-1} \ne \varnothing, Q_j = \varnothing, M \le j \le m.$$

Переобозначим целевую функцию как $g_{m+1}(y) = f(y)$ и введем вспомогательную задачу

$$g_M(y) \to \min, y \in Q_{M-1}\,, \tag{7}$$

которая всегда имеет решение, поскольку $g_M(y)$ непрерывна, а Q_{M-1} непусто, замкнуто и ограничено.

Сконструируем индексную функцию

$$\Phi(y) = g_{\nu(y)}(y) - \begin{cases} 0, & \nu(y) < m+1, \\ f^*, & \nu(y) = m+1, \end{cases} \tag{8}$$

где f^* – минимальное значение функции $f(y)$ в области Q. Очевидно, что индексная функция обладает свойствами

(i) $\Phi(y) > 0$, когда $\nu(y) < m + 1$, т.е. точка y является недопустимой;

(ii) $\Phi(y) = 0$, когда $\nu(y) = m + 1$ и $g_{m+1}(y) = f^*$;

(iii) $\Phi(y) > 0$, когда $\nu(y) = m + 1$ и $g_{m+1}(y) = f^*$.

Рассмотрим новую задачу

$$\Phi(y) \to \min,\ y \in D\,. \tag{9}$$

Вследствие указанных свойств индексной функции исходная задача (1)–(3) и задача (9) являются эквивалентными в том смысле, что множества их глобально-оптимальных решений совпадают, когда область Q не пуста. Таким образом, мы можем заменить отыскание глобального минимума функции $f(y)$ в сложной области Q решением задачи оптимизации индексной функции в гиперпараллелепипеде D.

Данный подход, который носит название индексного метода [1, 2], может рассматриваться как вариант метода штрафных функций, однако, в отличие от классической схемы индексный метод не требует подбора каких-либо параметров типа константы штрафа или выравнивающих коэффициентов при ограничениях. Следует также отметить, что для построения индексной функции не требуется, чтобы функции $g_j(y)$, $1 \le j \le m + 1$ существовали во всех точках области D; достаточно, чтобы каждая функция $g_j(y)$ была определена только в области Q_{j-1}. Задачи такого рода называются задачами с частично-вычислимыми ограничениями, и для их решения классический метод штрафных функций применить невозможно.

С другой стороны заметим, что, несмотря на липшицевость функций исходной задачи, функция $\Phi(y)$ является, вообще говоря, разрывной, что требует применения специальных методов оптимизации, учитывающих данное свойство.

Вторая особенность индексной функции состоит в том, что в ее формировании участвует величина f^*, которая, разумеется, неизвестна. Поэтому в численном эксперименте вместо f^* можно использовать ее оценку f_k^* по результатам вычисления целевой функции, т.е.

$$f_k^* = \min_{y^i \in Q} f(y^i)\,, \tag{10}$$

где $y^1, y^2, \ldots, y^k$ – точки области D, в которых проведены вычисления значений функции $\Phi(y)$. Это означает, что при решении задачи (9), мы используем адаптивное семейство функций

$$\Phi_k(y) = g_{\nu(y)}(y) - \begin{cases} 0, & \nu(y) < m+1, \\ f_k^*, & \nu(y) = m+1, \end{cases} \tag{11}$$

Укажем, что для функции $\Phi_k(y)$ сохраняется свойство (i), а для то-

чек $y^i \in Q$ справедливы свойства (ii)-(iii) при замене f^* на f_k^* .

Схема вложенной оптимизации

Введем обозначения

$$u_i = (y_1,...,y_i),\ v_i = (y_{i+1},...,y_N),$$

позволяющие при $1 \le i \le N-1$ записать вектор $y\,y$ в виде пары $y = (u_i, v_i)$, и примем, что $y = v_0$ при $i = 0$ и $y = u_N$ при $i = N$.

Построим семейство функций

$$\Phi^N(y) \equiv \Phi(y),$$

$$\Phi^i(u_i) = \min_{y_{i+1} \in [a_{i+1}, b_{i+1}]} \Phi(u_{i+1}),\ 0 \le i \le N-1, \qquad (12)$$

определенных на множествах

$$D_i = \{u_i \in R^i : a_s \le y_s \le b_s, 1 \le s \le i\} .$$

Тогда имеет место соотношение [3,4]

$$\min_{y \in D} \Phi(y) = \min_{y_1 \in [a_1, b_1]} \min_{y_2 \in [a_2, b_2]} \dots \min_{y_N \in [a_N, b_N]} \Phi(y). \qquad (13)$$

Как следует из (13), для решения задачи (9) достаточно решить одномерную задачу

$$\Phi^1(y_1) \to \min, \quad y_1 \in [a_1, b_1] \subseteq R^1 . \qquad (14)$$

При этом каждое вычисление функции $\Phi^1(y_1)$ в некоторой фиксированной точке $y_1 \in [a_1, b_1]$ представляет собой согласно (12) решение одномерной задачи

$$\Phi^2(y_1, y_2) \to \min, \quad y_2 \in [a_2, b_2] \subseteq R^1.$$

Эта задача является одномерной задачей минимизации по y_2, т.к. y_1 фиксировано.

В свою очередь, каждое вычисление значения функции $\Phi^2(y_1, y_2)$ при фиксированных y_1, y_2 требует решения одномерной задачи

$$\Phi^3(u_2, y_3) \to \min, ,\ y_3 \in [a_3, b_3] \subseteq R^1 ,$$

и т.д. вплоть до решения задачи

$$\Phi^N(u_{N-1}, y_N) = f(u_{N-1}, y_N) \to \min, \quad y_N \in [a_N, b_N] \subseteq R^1$$

при фиксированном u_{N-1}.

Окончательно решение задачи (9) сводится к решению семейства «вложенных» одномерных подзадач

$$\Phi^i(u_{i-1}, y_i) \to \min, \quad y_i \in [a_i, b_i] \subseteq R^1, \tag{15}$$

где фиксированный вектор $u_{i-1} \in D_{i-1}$.

Решение исходной многомерной задачи (9) через решение системы взаимосвязанных одномерных подзадач (15) называется *многошаговой схемой редукции размерности*.

Для решения одномерных подзадач (15) в следующем разделе предлагается параллельный одномерный алгоритм, обобщающий последовательный индексный метод, предложенный Д.Л.Маркиным и Р.Г.Стронгиным [1, 2].

Параллельный индексный метод одномерной оптимизации

Рассмотрим одномерную задачу (1)–(3), когда функции $g_i(y)$, $1 \le j \le m + 1$, зависят от единственной скалярной переменной y и область D является отрезком, т.е. $D = [a, b]$. В этом случае функция $\Phi(y)$ представляет собой в общем случае разрывную функцию, дуги которой между двумя соседними точками разрыва или крайними точками a, b удовлетворяют условию Липшица с константой, которая зависит от того, какая из функций $g_i(y)$ образовала данную дугу.

Опишем вычислительную схему параллельного алгоритма, предназначенного для поиска глобального минимума задачи (1)–(3).

Назовем *испытанием* операцию вычисления в некоторой точке $y \in [a, b]$ индекса $\nu(y)$ и значения $g_{\nu(y)}(y)$, и будем использовать термин *итерация* для одновременного (параллельного) выполнения нескольких испытаний (каждое на своем процессоре), обозначая числом $p(n)$ количество испытаний n-й итерации, а величиной $k(n)$ – суммарное количество испытаний, выполненных в процессе всех n итераций.

На начальной итерации поиска проведем первые испытания в $p = p(1) \ge 2$ точках $y^1, y^2, \ldots, y^k$ отрезка $[a, b]$, среди которых в обязательном порядке должны присутствовать концы отрезка a и b, причем эти испытания могут быть проведены параллельно (каждое на отдельном процессоре). По результатам испытаний будут вычислены индексы $\nu_i = \nu(y^i)$ и значения $g_{\nu_i}(y^i), 1 \le i \le p$. После этого установим номер итерации $n = 1$ и положим $k = k(1) = p$.

Общее правило получения координат новой (n + 1)-й итерации ($n \geq 1$) состоит в следующем.

Шаг 1. Координаты проведенных испытаний $y^1, y^2, \ldots, y^k$, $k = k(n)$, перенумеровываются нижним индексом в порядке возрастания, т.е.

$$a = y_0 < y_1 < \mathsf{K} < y_{\tau-1} < y_\tau = b, \tag{16}$$

где $\tau = k(n) - 1$.

Шаг 2. Каждой точке y_i множества (16) ставится в соответствие индекс $\nu_i = \nu(y_i)$ и значение z_i функции (11), т.е. $z_i = \Phi_k(y_i)$.

Шаг 3. Для функций $g_j(y)$, $1 \leq j \leq m + 1$ вычисляются величины

$$\lambda_j = \max\left\{\frac{|z_q - z_s|}{y_q - y_s} : 0 \leq s < q \leq k, \nu_s = \nu_q = j\right\}. \tag{17}$$

Если значение λ_j не может быть вычислено, оно полагается равным нулю.

Шаг 4. На базе величин (17) определяются оценки μ_j для констант Липшица L_j функций $g_j(y)$, $1 \leq j \leq m + 1$:

$$\mu_j = \begin{cases} r\lambda_j, \lambda_j > 0 \\ 1, \ \lambda_j = 0 \end{cases}, \tag{18}$$

где $r > 1$ – параметр метода.

Шаг 5. Каждому подынтервалу (y_{i-1}, y_i), $1 \leq i \leq \tau$, ставится в соответствие число

$$R(i) = \begin{cases} y_i - y_{i-1} + \dfrac{(z_i - z_{i-1})^2}{\mu_j^2 (y_i - y_{i-1})} - \dfrac{2(z_i + z_{i-1})}{\mu_j}, \nu_{i-1} = \nu_i = j \\ 2(y_i - y_{i-1}) - \dfrac{4z_i}{\mu_j}, \nu_{i-1} < \nu_i = j, \\ 2(y_i - y_{i-1}) - \dfrac{4z_{i-1}}{\mu_j}, \nu_i < \nu_{i-1} = j, \end{cases} \tag{19}$$

называемое *характеристикой* подынтервала.

Шаг 6. Характеристики $R(i)$, $1 \leq i \leq \tau$, упорядочиваются по убыванию:

$$R(t_1) \geq R(t_2) \geq \mathsf{K} \geq R(t_{\tau-1}) \geq R(t_\tau) \tag{20}$$

Шаг 7. В последовательности (20) выбираются $p = p(n + 1) \le \tau$ наибольших характеристик с номерами t_i, $1 \le i \le p$, и в интервалах, соответствующих этим характеристикам, формируются точки испытаний $y^{k+1}, \ldots, y^{k+p}$ новой $(n + 1)$-й итерации согласно выражениям

$$y^{k+s} = \begin{cases} \frac{1}{2}(y_{t_s} + y_{t_s-1}) - \frac{z_{t_s} - z_{t_s-1}}{2\mu_j}, \ \nu_{t_s-1} = \nu_{t_s} = j \ , \\ \frac{1}{2}(y_{t_s} + y_{t_s-1}), \ \nu_{t_s-1} \ne \nu_{t_s}, \end{cases} \tag{21}$$

для $1 \le s \le p$.

Шаг 8. Если

$$\min_{1 \le s \le p}(y_{t_s} - y_{t_s-1}) \le \varepsilon \tag{22}$$

где $\varepsilon > 0$ – заданная точность поиска, то поиск прекращается и в качестве решения принимается величина f_k^* из (10), а также ее координата.

Если же (22) не выполняется, осуществляется параллельная итерация метода, т.е. вычисление в точках y^{k+s}, $1 \le s \le p$, индексов этих точек $\nu(y^{k+s})$ и значений функции $\Phi_k(y^{k+s})$ – каждое испытание на своем процессоре. После этого номер итерации n увеличивается на единицу, количество испытаний $k(n + 1)$ становится равным $k(n) + p$, пересчитывается оценка (10) и происходит переход к шагу 1.

Предложенный метод относится к классу параллельных характеристических алгоритмов [5], и его сходимость к глобальному оптимуму обеспечивается условиями сходимости его последовательного прототипа [1, 2].

Заметим, что в данной работе рассматривается *синхронная* схема организации вычислений индексного метода, когда процессор, завершивший свое испытание, ожидает окончания работы остальных процессоров, участвующих в параллельной итерации метода. Но в индексной схеме время работы каждого процессора неизбежно будет различным, даже если считать, что времена вычисления значений каждой функции $g_j(y)$, $1 \le j \le m + 1$ не отличаются ни между собой, ни от точки к точке, потому что процессор завершает испытания сразу, как только обнаруживает первое нарушенное ограничение, и остальные функции задачи не вычисляет. Поэтому более эффективной представляется асинхронная реализация решающего правила индексного мето-

да, когда точки испытаний формируются динамически по запросу освободившегося процессора, как это, например, реализовано в методах [6–8].

Предположим теперь, что в схеме редукции (13) для решения одномерных задач (15) используется параллельный индексный метод с постоянным числом процессоров π_i на всех итерациях метода, решающего одномерные задачи, связанные с координатой y_i.

Тогда для решения задачи (9), а, как следствие, и задачи (1)–(3) можно использовать вплоть до

$$\Pi = \prod_{i=1}^{N} \pi_i \qquad (23)$$

параллельно работающих процессоров. При этом многошаговая схема будет генерировать до Π/π_N одномерных подзадач оптимизации, решаемых одновременно.

Если мы комбинируем многошаговую схему с одномерным параллельным алгоритмом, в котором условие остановки отлично от выполнения фиксированного одинакового числа итераций, тогда времена выполнения испытаний в параллельной итерации на i-м уровне одномерной оптимизации ($1 \le i \le N - 1$) будут неизбежно различны. Напомним, что вычисление целевой функции на всех уровнях, кроме последнего, состоит в решении новой оптимизационной подзадачи. Данный аспект также служит основанием для конструирования и исследования асинхронных алгоритмов индексной многомерной оптимизации.

Работа выполнена при поддержке РФФИ (грант № 04-01-00455).

Литература

1. *Strongin R.G.*, *Sergeyev Ya.D.* Global optimization with non-convex constraints: Sequential and parallel algorithms, Kluwer Academic Publishers, Dordrecht, Netherlands. 2000.

2. *Strongin R.G.*, *Markin D.L.* Minimization of multiextremal functions with nonconvex constraints. Cybernetics 22, 1986. P. 486–493.

3. *Carr C.R.*, *Howe C.W.* Quantitative Decision Procedures in Management and Economics. Deterministic Theory and Applications. McGraw Hill, New York, 1964.

4. *Стронгин Р.Г.* Численные методы в многоэкстремальных задачах. Информационно- статистический подход. М.: Наука, 1978.

5. *Strongin R.G.*, *Sergeyev Ya.D.*, *Grishagin V.A.* Parallel Characteristical Algorithms for Solving Problems of Global Optimization // Journal of Global Optimization, 10, 1997. P. 185–206.

6. *Sergeyev Ya.D.*, *Grishagin V.A*. Parallel asynchronous global search and the nested optimization scheme, Journal of Computational Analysis & Applications, 3(2), 2001. P. 123–145.

7. *Гришагин В.А.*, *Филатов А.А*. Параллельные рекурсивные алгоритмы многоэкстремальной оптимизации // Матер. II Международ. научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах» / Под ред. проф. Р.Г. Стронгина. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2002. С. 88–90.

8. *Гришагин В.А.*, *Сергеев Я.Д*. Эффективность распараллеливания характеристических алгоритмов глобальной оптимизации в многошаговой схеме редукции размерности. // Матер. IV Международ. научно-практического семинара и Всероссийской молодежной школы «Высокопроизводительные параллельные вычисления на кластерных системах» / Под ред. члена-корр. РАН В.А.Сойфера. Самара: Изд-во Самарского научного центра РАН, 2004. С. 70–74.

СВОБОДНЫЙ ПАРАЛЛЕЛЬНЫЙ ОТЛАДЧИК НА ОСНОВЕ GNU DDD

А.В. Гришин, А.Л. Курылев

Нижегородский государственный университет им. Н.И. Лобачевского

А.В. Коновалов, А.Г. Пегушин

ЗАО «Интел А/О», Нижний Новгород

Под отладкой в узком смысле обычно понимают деятельность, начинающуюся с обнаружения факта ошибки, и заканчивающуюся непосредственным моментом ее исправления. Общепринято, что отладка параллельных программ является значительно более ресурсоемким процессом, чем отладка последовательных. Выразительным примером здесь может служить работа [4], наглядно иллюстрирующая тот факт, что даже в очень небольших программных структурах ошибки могут скрываться десятилетиями. Попытаемся коротко проанализировать причины такой ситуации, обращая внимание на деятельность программиста в процессе отладки.

Отладка начинается с изучения симптомов ошибки (например, неправильный ответ, аварийное завершение программы и др.), но в нетривиальных случаях ошибка находится не там, где проявляется симптом. Это значит, что для успешной отладки необходимо (неоднократно) отвечать на вопрос «А как же мы сюда попали?». Удивительно, но

современные отладчики не предоставляют инструментов, позволяющих упростить эту важнейшую операцию.

На выручку приходят средства журналирования, подтверждая утверждение, что лучшее средство отладки – оператор печати. В параллельной отладке перед программистом открывается целое новое измерение: необходимо осознать, чем «сейчас» заняты другие параллельные процессы и как, в свою очередь, они пришли в это состояние. По-видимому, векторные часы адекватно моделируют интуитивное представление о «сейчас» [10], но излишне говорить, что распростран енные MPI-ориентированные продукты их не поддерживают.

Инструментальные средства отладки

Осознание сообществом системных программистов трудности параллельной отладки привело к созданию ряда инструментов, ее облегчающих. Подобного рода инструменты можно разделить на три категории:

1) средства отладки, использующие особенности интерфейса MPI. MPI является достаточно высокоуровневым механизмом межпроцессного взаимодействия (по сравнению с сокетами Беркли или общей памятью), что позволяет автоматически обнаруживать целый ряд ошибок, таких, как неправильный тип данных при широковещании MPI_Bcast или некорректная операция во время редукции MPI_Reduce. Последнее время это направление развивается весьма бурно [8], [9], [11];

2) средства отладки эффективности (Jumpshot и др., см. обзор в [1]). Обычно их рассматривают только как вспомогательные средства оптимизации, однако серьезная отладка существенно облегчается возможностью изучать общую структуру программы. Особенно в этой деятельности оказываются полезны виды отображения, производные от диаграмм Гантта. Еще одна область применения этих средств при отладке – анализ ситуаций, когда ошибки в программе маскируются средствами восстановления, присутствующими в этой программе. В результате все тесты проходят, и только по аномалии производительности можно понять, что ошибка все-таки есть [7];

3) традиционные последовательные отладчики, обобщенные для параллельной работы (Etnus TotalView, Allinea DDT, TDB проекта «Скиф»). Часто их называют параллельными отладчиками, однако в свое время отладчики на уровне исходного текста [6] были революционным явлением как раз потому, что программист видел в отладчике именно то, на чем он программировал, а не машинные команды. В этом

смысле, не смотря на стремление к естественному представлению SIMD программ в виде единого исходного текста и различных для разных процессов обрабатываемых данных и развитый аппарат групповых контрольных точек, средства описываемой группы пока не вышли на уровень «вижу работу в тех терминах, в каких программировал».

Осмысленное сравнение разнородных программных средств не может проводиться без учета человеческой психологии. Наиболее многочисленной и наиболее страдающей группой пользователей параллельных машин являются прикладные программисты (специалисты в предметных областях, использующие параллельные вычисления как еще один инструмент решения своих задач, и по возможности не желающие изучать устройство инструмента) [5]. И здесь третья категория инструментов находится вне конкуренции: интегрированные среды программирования сделали использование отладчиков прозрачным и повсеместным. Неплохи шансы и первой категории, до той поры, пока ошибки ищутся автоматически. Необходимо понимать, что с расширением круга ошибок простота использования исчезнет – если гонки (ситуации асинхронности) и легко находить автоматически (по трассе, либо сравнивая прогоны), то их присутствие во многих популярных каркасах параллельных программ делает просто показ всех гонок бессмысленным. Мощным средством поиска потенциальных ошибок является сравнительная отладка [2].

Исходя из вышеизложенного, было предложено реализовать привычные базовые возможности параллельных отладчиков в разрабатываемом свободном диалоговом отладчике. Это позволит создать сообщество пользователей, во-первых, и разработать расширяемый программный продукт с возможностью добавления инструментов высокоуровневого анализа параллельной структуры (это позволит превратить отладчик в полноценное средство параллельной отладки), во-вторых.

Свободный диалоговый отладчик

Весьма реалистичная структура диалогового параллельного отладчика изложена в [3]. Разумеется, необходимо по максимуму использовать возможности традиционных последовательных отладчиков, при этом создавая собственное ядро, управляющее параллельностью. На основе наблюдений

1) нет никакой надежды завоевать простых пользователей без развитого GUI-интерфейса, GUI не необходим системным программистам, но им и отладчик нужен в меньшей степени;

2) простые пользователи консервативны. GUI должен быть при-

вычным;

3) разработка GUI – крайне трудоемкий процесс.

Мы пошли дальше и предложили использовать готовую существующую GUI-оболочку отладчиков GNU DDD. Существенными преимуществами DDD является зрелость, обширная пользовательская база и значительное число поддерживаемых отладчиков командной строки. Наиболее существенный недостаток – недостаточно структурированная реализация (это является почти неизбежным следствием долгого существования продукта).

В качестве временного решения ядром управления параллельностью служит подсистема mpigdb из MPICH2 (поддерживается версия 1.0.1 библиотеки). Е е использование в этом качестве потребовало некоторых модификаций, которые существенно упростила прозрачная структура соответствующих исходных текстов библиотеки. Общая структура отладочной системы изображена на рис. 1.

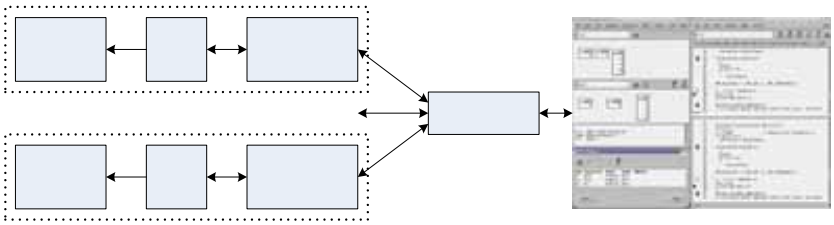


Рис. 1. Структура диалогового отладчика

Текущая версия (рис. 2) описываемой разработки предоставляет следующие возможности:

1) отслеживание местонахождения каждого процесса параллельной программы;

2) наблюдение за изменением данных на каждом процессе;

3) поддержку глобальных точек останова в параллельной программе;

4) поддержку глобального стека вызовов параллельной программы.

Описанные модификации выполнялись с DDD версии 3.3.9.

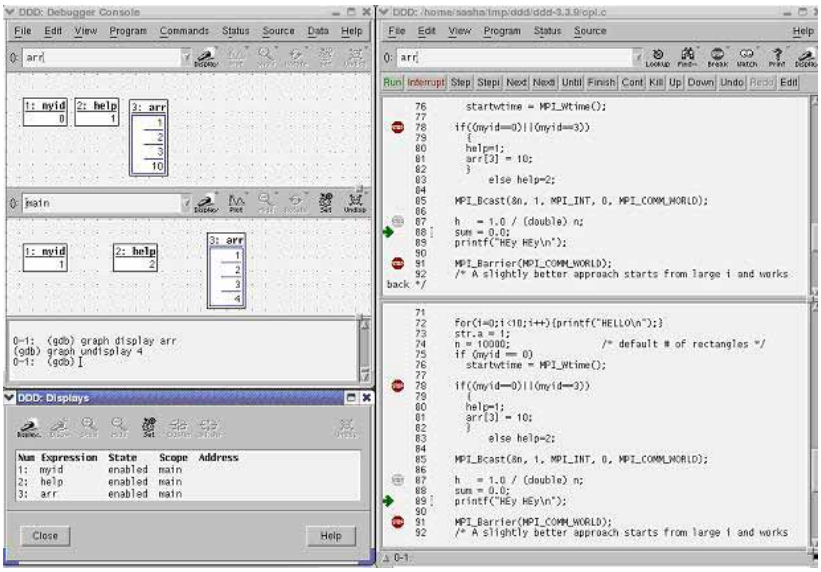


Рис 2. Внешний вид свободного диалогового отладчика

Разумеется, в настоящее время коммерческие отладчики Etnus TotalView и Allinea DDT существенно опережают по функциональности описываемую разработку. Дальнейшее развитие будет осуществляться по двум направлениям:

1) реализация традиционных средств диалоговой отладки с активным использованием преимуществ свободной лицензии проекта;

2) добавление в отладочный процесс распределенных утверждений (assert) на основе наблюдения за глобальными состояниями.

Хотелось бы выразить благодарности А.С. Игумнову (ИММ УрО РАН, г. Екатеринбург) за инициирование работы в области параллельной отладки на раннем этапе, а А.А. Ионову (ННГУ) – за участие в отладке и модификации mpigdb. Дополнительная информация о проекте, включая исходные тексты, доступна по адресу http://parallel.uran.ru/unn-itlab-mpi/ru/debugger.html.

Литература

1. *Авербух В.Л., Байдалин А.Ю.* Проектирование видов отображения для системы параллельного программирования DVM // Алгоритмы и программные средства параллельных вычислений / ИММ УрО РАН, Екатеринбург. 2001. Вып. 5.

2. *В. Ф. Алексахин, К. Н. Ефимкин, В. Н. Ильяков, В. А. Крюков, М. И. Кулешова, Ю. Л. Сазанов* Средства отладки MPI-программ в DVM-системе // Научный сервис в сети Интернет: технология распредел енных вычислений: Тр. Всерос. науч. конф. (19–24 сентября 2005, г. Новороссийск). – М.: Изд-во МГУ, 2005. С.113–115.

3. *Игумнов А.С.* Открытая платформа отладки параллельных программ // Научный сервис в сети Интернет-2003: Тр. Всерос. науч. конф. (22–27 сентября 2003, г. Новороссийск). – М.: Изд-во МГУ, 2003. С. 92–94.

4. *Карпов Ю.Г.*, *Борщев А.В.*, *Рудаков В.В.* О корректности параллельных алгоритмов // Программирование, № 4, 1996. С. 5–17.

5. *Лацис А.О.* Как построить и использовать суперкомпьютер. – М.: Бестселлер, 2003. – 274 с.

6. *Пасынков И.Г.*, *Подергина Н.В.*, *Самофалов В.В.*, *Тюрин В.Ф.*, *Ускова Т.И.* Символьный диалоговый отладчик ОС Диспак (возможности и реализация). – Свердловск, УНЦ АН СССР, 1980.

7. *Самофалов В.В.*, *Коновалов А.В.*, *Шарф С.В.* Производительность параллельного ввода-вывода // Научный сервис в сети Интернет-2003: Тр. Всероссийской научной конференции (22–27 сентября 2003, Новороссийск). – М., Изд-во МГУ, 2003. С. 212–215.

8. *Falzone C.*, *Chan A.*, *Lusk E.*, *Gropp W.* Collective Error Detection for MPI Collective Operations // Euro PVMMPI'05, 2005.

9. *Krammer B.*, *Müller M.S.*, *Resch M.M.* MPI Application Development Using the Analysis Tool MARMOT // ICCS 2004, Krakow, Poland, June 7–9, 2004. Lecture Notes in Computer Science / Springer, 2004. V. 3038. P. 464–471.

10. *Mattern F.*, *R. Schwarz* Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail // Distributed Computing. 1994. V. 7, No. 3. P. 149–174.

11. *Samofalov V.*, *Krukov V.*, *Kuhn B.*, *Zheltov S.*, *Konovalov A.*, *DeSouza J.* Automated Correctness Analysis of MPI Programs with Intel®Message Checker // Представлено к публикации в трудах конференции ParCo'2005.

ЧИСЛЕННОЕ РЕШЕНИЕ АДВЕКТИВНО-ДИФФУЗИОННЫХ УРАВНЕНИЙ НА МНОГОПРОЦЕССОРНОЙ ТЕХНИКЕ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Е.А. Данилкин

Томский Государственный Университет

Введение

Несомненно, что главной движущей силой развития параллельных компьютеров были потребности научных вычислений, а одной из са-

мых важных задач в области научных вычислений является решение систем линейных алгебраических уравнений. При решении СЛАУ высокого порядка, особое место занимают проблемы с разреженными матрицами специальной структуры, возникающими из сеточных аппроксимаций многомерных краевых задач.

Для решения таких задач невыгодно применять прямые методы вроде метода исключения Гаусса. Эти методы не учитывают структуры матрицы, вынуждают хранить матрицу целиком, что делает нерациональным использование оперативной памяти, подвержены влиянию погрешности округления. В наше время для решения этих ресурсоемких задач наиболее эффективным средством стали итерационные методы неполной факторизации [1], которые применяются совместно со спектральными или вариационными принципами ускорения процесса решения. Их главными достоинствами являются рекордная практическая экономичность, высокие теоретические оценки сходимости последовательных приближений и широкие возможности конструирования адаптивных алгоритмов для различных классов задач.

Цель данной работы заключается в исследовании возможных подходов решения систем связанных линейных дифференциальных уравнений методом конечного объема на многопроцессорной системе с распределенной памятью и выбор оптимального с точки зрения минимизации межпроцессорных обменов.

Математическая постановка задачи

Требуется решить систему двух связанных линейных дифференциальных уравнений в единичном квадрате ($(x, y) \in [0, 1]\times[1, 0]$, G – граница) с граничными условиями третьего рода

$$\begin{cases} \dfrac{\partial}{\partial x}\left(D^1 \dfrac{\partial u}{\partial x}\right)+\dfrac{\partial}{\partial y}\left(D^1 \dfrac{\partial u}{\partial y}\right)+E^1 \dfrac{\partial u}{\partial x}+B^1 v=F^1;\ \alpha^1 \dfrac{\partial u}{\partial n}\Big|_G+\beta^1 u\Big|_G=\gamma^1; \\ \dfrac{\partial}{\partial x}\left(D^2 \dfrac{\partial v}{\partial x}\right)+\dfrac{\partial}{\partial y}\left(D^2 \dfrac{\partial v}{\partial y}\right)+E^2 \dfrac{\partial v}{\partial x}+B^2 u=F^2;\ \alpha^2 \dfrac{\partial v}{\partial n}\Big|_G+\beta^2 v\Big|_G=\gamma^2; \end{cases} \quad (1)$$

где D^i, E^i, B^i, F^i достаточно раз непрерывно дифференцируемые функции. α^i, β^i, γ^i ($i = 1, 2$) константы.

Метод решения

Аппроксимация дифференциальной задачи осуществлялась на основе метода конечного объема. Основная идея этого метода заключа-

ется в разбиении расчетной области на непересекающиеся, граничащие друг с другом конечные объемы так, чтобы один узел расчетной сетки содержался только в своем конечном объеме. Разбив таким образом расчетную область, интегрируем каждое уравнение системы по каждому конечному объему. При вычислении интегралов используются интерполяционные квадратурные формулы численного интегрирования для зависимых величин, входящих в (1). В результате такого приближенного интегрирования получается дискретный аналог системы дифференциальных уравнений

$$\begin{cases} ap_{ij}u_{ij} = ae_{ij}u_{i+1\,j} + aw_{ij}u_{i-1\,j} + an_{ij}u_{ij+1} + as_{ij}u_{ij-1} + ad_{ij}v_{ij} + b_{ij};\ 0 \le i \le Nx, \\ ap'_{ij}v_{ij} = ae'_{ij}v_{i+1\,j} + aw'_{ij}v_{i-1\,j} + an'_{ij}v_{ij+1} + as'_{ij}v_{ij-1} + ad'_{ij}u_{ij} + b'_{ij};\ 0 \le j \le Ny; \end{cases} \quad (2)$$

Матрица полученной системы сеточных уравнений (2) имеет вид, показанный на рис. 1. Здесь U, V – компоненты столбца неизвестных, B – столбец свободных членов.

$$U = \{u_{ij}\}, \quad V = \{v_{ij}\}, \quad B = \begin{pmatrix} \{b_{ij}\} \\ \{b'_{ij}\} \end{pmatrix}.$$

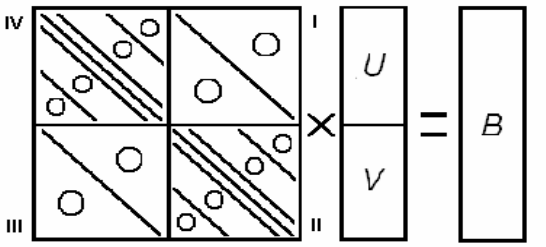


Рис. 1

Матрица системы (обозначим ее через A) имеет большое количество нулевых элементов, а ненулевые выстроены в диагонали, как показано на рис. 1. Матрица A состоит из четырех больших блоков, два из которых пятидиагональные матрицы и два содержат лишь по одной ненулевой главной диагонали.

Метод решения систем линейных алгебраических уравнений

В качестве метода решения системы сеточных уравнений рассматривается стабилизирующий метод бисопряженных градиентов [3] с предобуславливанием по Булеву [1, 2]. В настоящее время этот метод

является одним из наиболее быстродействующих по времени и количеству итераций. Он относится к семейству методов вариационно-итерационного типа и применим для решения знаконеопределенных СЛАУ с несимметричной в общем случае матрицей, для которой неизвестны спектральные свойства.

Практически все современные алгоритмы решения задач вида $Ax = b$ относятся к методам подпространств Крылова, основанным на минимизации следующего функционала $F(x) = \langle Ax, x\rangle - 2\langle b, x\rangle$. Решение связано с построением системы сопряженных векторов и каждое следующее приближение ищется в направлении нового полученного вектора из условия минимума функционала в этом направлении.

Bi-CGStab *алгоритм с предобуславливающей матрицей*:

x_0 – начальное приближение;

$r_0 = b - Ax_0$;

$\mathbf{r}_0$ – произвольный вектор, для которого $(\mathbf{r}_0, r_0 \neq 0)$, например, $\mathbf{r}_0 = r_0$;

$\rho_0 = \alpha = \omega_0 = 1$; $v_0 = p_0 = 0$;

for $i = 1, 2, 3, \ldots$,

$\rho_i = (\mathbf{r}_0, r_{i-1})$;

$\beta_i = (\rho_i/\rho_{i-1})(\alpha/\omega_{i-1})$;

$p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$;

находим y из $Ky = p_i$; (K – предобуславливающая матрица, $K \approx A$)

$v_i = Ay$;

$\alpha = \rho_i/(\mathbf{r}_0, v_i)$;

$s = r_{i-1} - av_i$;

находим z из $Kz = s$;

$t = Az$;

$\omega_i = (t, s)/(t, t)$;

$x_i = x_{i-1} + \alpha p_i + \omega_i s$;

если x_i достаточно точное, то остановка;

$r_i = s - \omega_i t$;

end;

Как указывается в [3] метод Bi-CGStab можно считать прямым методом, поскольку при точной арифметике он сойдется к точному решению за конечное число итераций. Это следует из свойства сопряженности векторов построенной системы.

Предобуславливатель

Для ускорения процесса сходимости и уменьшения числа итераций в данной работе используется процедура предобуславливания по методу неполной факторизации Булеева, формулы которой получаются из формул явного метода Булеева исключением итерационных членов [1, 2]. Принцип конструирования явного метода Булеева можно считать формальным обобщением знаменитого безитерационного метода прогонки для решения одномерных краевых задач, основанного на разложении трехдиагональной матрицы в произведение нижней и верхней треугольных матриц.

Существенно новым моментом для итерационных методов явилось введение приема, названного Н.И. Булеевым принципом компенсации. Он заключается в добавлении к сеточным уравнениям таких членов, которые приводят к взаимному исключению ошибок итерационных приближений, в случае, если решение является гладкой функцией.

Алгоритм предобуславливания (решение систем вида $K_y = p_i$ или $Kz = s$, $K \approx A$) состоит из двух этапов: прямого и обратного хода. На первом шаге вычисляются прогоночные коэффициенты, формулы которых имеют рекуррентный вид (где Θ – параметр компенсации $0 \le \le \Theta \le 1$):

$$P_{ij} = \frac{ae_{ij}}{g_{ij}}, Q_{ij} = \frac{an_{ij}}{g_{ij}}, T_{ij} = (b_{ij} + aw_{ij} T_{i-1\,j} + as_{ij} T_{ij-1});\ i = 1, 2, Nx - 1;$$

$$g_{ij} = ap_{ij} - aw_{ij}(P_{i-1\,j} + \Theta Q_{i-1\,j}) - as_{ij}(Q_{ij-1} + \Theta P_{ij-1});\ j = 1, 2, Ny - 1; \qquad (3)$$

А затем решение находится по формуле:

$$y_{ij} = P_{ij} y_{i+1\,j} + Q_{ij} y_{ij+1} + T_{ij};\ i = Nx - 1, ..., 0;\ j = Ny - 1, ..., 0; \qquad (4)$$

Отметим, что использующееся в настоящее время предобуславливание по методу Холесского , LU-факторизации для задач вида (1) менее эффективно, чем предобуславливание по Булееву [2].

Параллельная реализация

Распараллеливание метода Bi-CGStab для случая решения одного дифференциального уравнения осуществляется следующим образом. Используется одномерная декомпозиция по столбцам, то есть каждому процессору выделяется определенное количество столбцов, для которых он будет производить вычисления. При аппроксимации диффе-

ренциальной задачи используется шаблон «крест». Поэтому для вычисления очередного приближения в приграничных узлах необходимо пересылать значения с соседнего процессора. Алгоритм метода состоит из операций умножения матрицы на вектор, перемножения векторов и вычисления скалярных произведений, которые просто распараллеливаются. Структура матрицы без труда позволяет организовать вычисления так, что обмены требуются лишь для пересылки граничных значений и сбора-рассылки частных сумм скалярного произведения.

При предобуславливании системы линейных уравнений допустимо применять асинхронный вариант метода Булеева, когда при вычислении значений прогоночных коэффициентов в (3) и решения в (4) в каждой процессорной подобласти на границе используются значение решения с предыдущей итерации.

Для случая решения системы сеточных уравнений с матрицей, представленной на рис. 1, ситуация меняется и неудачное распределение данных по процессорам может привести к значительному возрастанию пересылок, что в свою очередь свед ет на нет ускорение параллельной программы.

На рис. 2 представлен оптимальный вариант распределений данных по процессорам, позволяющий избежать использования дополнительных пересылок. При другом способе декомпозиции потребуется больше времени на межпроцессорную пересылку данных, необходимых для проведения расчетов, вследствие размещения на одном процессоре только значений $\{u_{ij}\}$ или только $\{v_{ij}\}$.

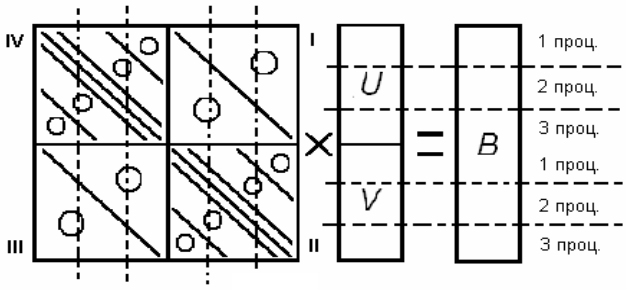


Рис. 2

ис. 2

Тестовая задача

Тестовой задачей служила система двух эллиптических уравнений с граничными условиями первого рода с известным решением:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2}+\dfrac{\partial^2 u}{\partial y^2}+\dfrac{\partial u}{\partial x}+v=1; \quad u\big|_G=2; \\ \dfrac{\partial^2 v}{\partial x^2}+\dfrac{\partial^2 v}{\partial y^2}+\dfrac{\partial v}{\partial x}+u=2; \quad v\big|_G=1; \end{cases} \tag{5}$$

Заметим, что в (5) после несложного преобразования можно перейти к независимым уравнениям:

$$\begin{cases} \dfrac{\partial^2 (u+v)}{\partial x^2}+\dfrac{\partial^2 (u+v)}{\partial y^2}+\dfrac{\partial (u+v)}{\partial x}+(u+v)=3; \quad (u+v)\big|_G=3; \\ \dfrac{\partial^2 (u-v)}{\partial x^2}+\dfrac{\partial^2 (u-v)}{\partial y^2}+\dfrac{\partial (u-v)}{\partial x}+(u-v)=-1; \quad (u-v)\big|_G=-1; \end{cases} \tag{6}$$

В качестве начального использовалось нулевое приближение. Приближенное решение соответствующих сеточных систем считалось достигнутым, если норма невязки становилась меньше $\varepsilon = 10^{-5}$.

Выводы

В таблице приведены результаты работы (число итераций и время) соответствующих последовательных алгоритмов для решения систем (5) и (6). В первой строке представлено число итераций необходимых для получения решения с заданной точностью с использованием процедуры предобуславливания во второй без е е применения.

Из таблицы следует, что предлагаемый метод совместного решения сеточных уравнений обладает большей эффективностью по сравнению с последовательным решением каждой системы сеточных уравнений на одной глобальной итерации метода.

Сетка	**100*100**	**200*200**
Bi-CGStab-P для (5)	51 (1,406 с) 409 (7,812 с)	114 (16.828с) 824 (73.671с)
Bi-CGStab-P для (6)	52 (1,370 с) 410 (7,281 с)	114 (16.234с) 863 (71.781с)
Bi-CGStab-P для (5) с последовательным решением уравнений	72 (3,710 с)	127 (31.109с)

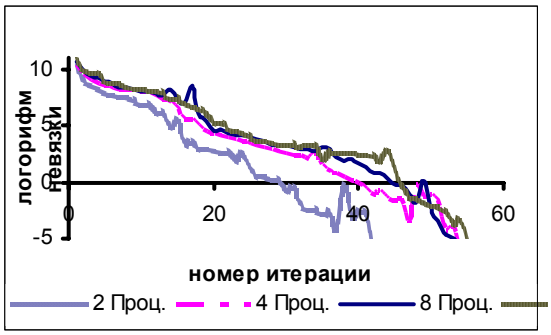


Рис. 3

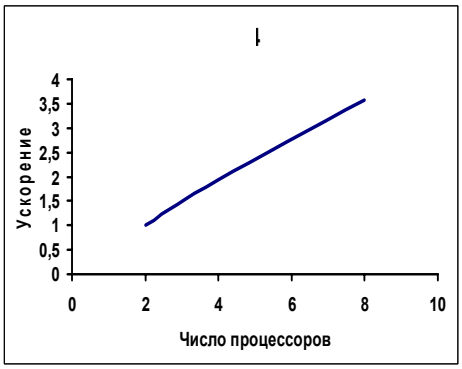


Рис. 4

Приведенные на рис. 3 графики сходимости итерационного процесса при параллельном решении системы (5) демонстрируют подобие зависимости логарифма невязки от номера итерации при различном числе процессоров. Некоторое увеличение числа глобальных итераций метода при большем числе процессоров обусловлено асинхронной природой использования предобуславливателя при параллельных вычислениях. Однако следует отметить, что это компенсируется снижением временных затрат на решение вследствие имеющего место ускорения вычислительного процесса (рис. 4).

Литература

1. *Ильин В.П.* Методы неполной факторизации для решения алгебраических систем. – М.: Наука, 1995. – 287 с.

2. *Старченко А.В*. // Вестник Томского государственного университета. 2003. №10. С. 70–80.

3. *Van der Vorst H.A*. Bi-CGSTAB: a fast and smothly converging variant of Bi-CG for the solution of nonsymmetric linear systems: Preprint 633. – Univ.Utrecht. – 1990.

ТЕХНОЛОГИЯ РАСПРЕДЕЛЕННОЙ ОБРАБОТКИ ЦВЕТНЫХ ИЗОБРАЖЕНИЙ

Д.И. Зимин, В.А. Фурсов

Самарский государственный гэрокосмический гниверситет

Постановка задачи

Задача улучшения качества цветных изображений часто должна решаться при отсутствии априорной информации о характеристиках искажений. Типичными источниками таких искажений являются дефокусировка объектива, скоростной сдвиг (смаз), и др. Они характерны, например, в случае регистрации неожиданно возникающих сюжетов и явлений, когда время на настройку регистрирующей аппаратуры крайне ограничено. В системах экологического мониторинга Земли характеристики искажающей среды также обычно изменчивы или неизвестны.

Для решения указанной задачи применяют идею «слепой» обработки сигналов [6, 2], прошедших через канал с неизвестными характеристиками на фоне шумов. В данном случае должна решаться задача слепой коррекции искажений (blind image deconvolution) при отсутствии априорной информации об импульсной реакции искажающей системы, т.е. формулируется задача восстановления двумерного, пространственно–ограниченного, неотрицательного сигнала искаженного линейным оператором размерность и параметры которого неизвестны.

В работе [2] эта задача рассматривалась применительно к обработке черно-белых полутоновых изображений. Технология построения восстанавливающих фильтров и обработки изображений строилась в виде следующих этапов:

1) выбор на искаженном изображении фрагментов, пригодных для слепой идентификации модели искажений в канале;

2) формирование из выбранных фрагментов тестовых образцов (компьютерное ретуширование изображений на фрагментах);

3) идентификация параметров искажающей системы и/или восста-

навливающего фильтра по отобранным и тестовым (ретушированным) фрагментам;

4) выбор класса и уточнение параметров восстанавливающего фильтра по критериям вычислительной сложности, устойчивости и качества обработки;

5) декомпозиция изображения;

6) обработка изображения.

При обработке цветных изображений возникают дополнительные трудности реализации указанной технологии. Дело в том, что цветные изображения обычно представляются в виде RGB-компонент. К сожалению, при таком представлении не удается организовать покомпонентную обработку, так чтобы при этом обеспечивалось высокое качество цветовоспроизведения.

В настоящей работе рассматривается технология обработки цветных изображений, в которой для преодоления указанной трудности осуществляется переход в цветовое пространство Lab. Рассматриваются также связанные с этим особенности организации распределенной обработки большеформатных цветных изображений.

Описание моделей и общей схемы обработки изображений

Рассматривается поэтапная технология, в которой предусматривается переход из пространства RGB в Lab и обратно. Каждый отсчет цветного изображения, в случае если используется модель хранения RGB, представлен в виде вектора $(r, g, b)^T$. На первом этапе для каждого отсчета цветного изображения осуществляется переход от модели RGB в модель XYZ [8] по формуле:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0{,}412453 & 0{,}357580 & 0{,}180423 \\ 0{,}212671 & 0{,}715160 & 0{,}072169 \\ 0{,}019334 & 0{,}119193 & 0{,}950227 \end{pmatrix} \begin{pmatrix} r \\ g \\ b \end{pmatrix}. \qquad (1)$$

Переход от модели XYZ в модель Lab с координатами $(L, a, b)^T$ осуществляется согласно следующим соотношениям:

$$\begin{aligned} L &= 116 \times f1(y/y_N) - 16, \\ a &= 500[f1(x/x_N) - f1(y/y_N)], \\ a &= 200[f1(y/y_N) - f1(z/z_N)], \end{aligned} \qquad (2)$$

где $x_N = 96{,}422$, $y_N = 100$, $z_N = 82{,}521$,

$$f1(t) = \begin{cases} t^{1/3} & \forall t > 0{,}008856, \\ 7{,}7867t + 0{,}137931 & \forall t \le 0{,}008856. \end{cases} \quad (1)$$

Соотношение (2) обеспечивает «растягивание» близкой к нулю области XYZ, в результате чего сглаживается неравноконтрастность XYZ.

После перехода к описанию в цветовом пространстве Lab осуществляется обработка с целью устранения искажений. Известно, что компонента L отвечает за уровень яркости, а остальные две за цветообразование, поэтому перечисленным выше типам искажений (дефокусировка, смаз, влияние турбулентностей атмосферы) преимущественно подвергается L компонента. С учетом этого технология улучшения качества цветных изображений при указанных типах искажений может быть сведена к обработке компоненты L.

Для этой компоненты осуществляется слепая параметрическая идентификация [6] восстанавливающего фильтра. Эта задача решается в классе фильтров с бесконечной импульсной характеристикой (БИХ-фильтров), для которых связь отсчетов выходного $g(n_1, n_2)$ и входного $f(n_1, n_2)$ изображений описывается соотношением [1]:

$$g(n_1, n_2) = -\sum_{(m_1, m_2) \in Q_g} a_{m_1, m_2} g(n_1 - m_1, n_2 - m_2) + \\ + \sum_{(m_1, m_2) \in Q_f} b_{m_1, m_2} f(n_1 - m_1, n_2 - m_2) + \xi(n_1, n_2), \quad (4)$$

где $a_{m_1, m_2}, b_{m_1, m_2}$ - подлежащие определению коэффициенты фильтра, $\xi(n_1, n_2)$ - дискретный шум. Затем с использованием этого фильтра обрабатывается L компонента изображения [2].

Завершающим этапом является переход из пространства Lab в пространство RGB. На первом этапе, используются соотношения, получаемые из выражения **Ошибка! Источник ссылки не найден.**:

$$x = x_N \times f2\left(\frac{a}{500} + \frac{L+16}{116}\right),$$

$$y = y_N \times f2\left(\frac{L+16}{116}\right),$$

$$z = z_N \times f2\left(\frac{L+16}{116} - \frac{b}{200}\right),$$

где

$$f2(t) = \begin{cases} t^3 & \forall t > 0{,}20689, \\ 0{,}128424t + 0{,}0177137 & \forall t \le 0{,}20689. \end{cases} \quad (6)$$

в соответствии с которыми осуществляется обратный переход в пространство XYZ. Затем осуществляется переход их пространства XYZ в пространство RGB:

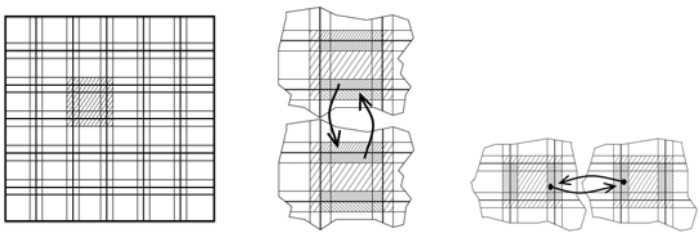
$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 3{,}240479 & -1{,}537150 & -0{,}498535 \\ -0{,}969256 & 1{,}875992 & 0{,}041556 \\ 0{,}055648 & -0{,}204043 & 1{,}057311 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \quad (7)$$

Технология распределенной обработки большеформатных изображений

Рассмотрим особенности организации распределенной обработки изображений. С вычислительной точки зрения алгоритм обработки *L*-компоненты цветного изображения в соответствии со схемой классификации, изложенной в [9] относится к классу процессов обработки, которые должны осуществляться итерационно, с обменом данными после каждой итерации. Поэтому центральным вопросом в данном случае, также как и при обработке черно-белых изображений, является декомпозиция данных.

В данном случае *L* -компонента цветного изображения для минимизации коммуникационных расходов разбивается на квадратные области [2] (рис. 1,*а*). При этом обмен данными будет происходить в последовательности, указанной на рис. 1,*б* и на рис. 1,*в*. Вследствие неоднородности многопроцессорной вычислительной системы, время обработки одинаковых фрагментов изображения в узлах будет различным. На время обработки фрагмента влияют многие параметры узла, в частности частота процессора, частота системной шины и памяти, размер КЭШа, параметры коммуникационной среды и т.д. Определение явных априорных зависимостей времени обработки изображения от его размера представляет серьезные трудности и не всегда возможно. Поэтому система строится на основе клиент-серверной архитектуры с возможностью управления процессом в ходе обработки.

Платформой для реализации программного комплекса обработки цветных изображений ImageProc выбрана Java 2 Enterprise Edition™ (J2EE) от Sun Microsystems. Данная платформа является основой серверов приложений многих компаний и хорошо документируема.



а) *б*) *в*)

Рис. 1. *а*) двумерная декомпозиция; *б*) первый этап обмена данными; *в*) второй этап обмена данными

На рис. 2 представлена структурная схема программного комплекса. Кроме основных компонентов на рисунке представлены сетевые протоколы, в соответствии с которыми происходит обмен между компонентами. При разработке пользовательского интерфейса используется идеология тонкого клиента, поэтому у пользователя нет необходимости в установке дополнительного программного обеспечения. Реализация программного комплекса выполняется с использованием свободно распространяемого программного обеспечения: Linux, JSP – контейнер Tomcat. Технология Java Server Pages (JSP) имеет ряд преимуществ по сравнению с другими технологиями создания приложений для обработки динамического WEB-содержания. Во-первых, как любая Java технология, она является кроссплатформенной. Во-вторых, JSP-технология помогает эффективно отделить представление от содержания. Использование Web Cluster в приведенной системе позволяет оперативно обрабатывать большеформатные изображения, что является важной особенностью, среди аналогичных систем.

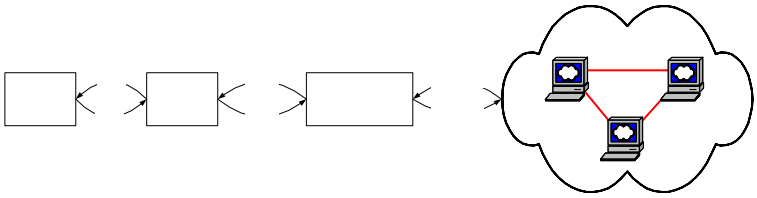


Рис. 2. Структурная схема программного комплекса

В описанной системе компоненты изображения *a* и *b* хранятся на сервере, где и осуществляется формирование выходного изображения после завершения обработки *L*-компоненты.

На рис. 3 и 4 приведены исходное искаженное и обработанное с использованием описанной технологии изображения.

Рис. 3. Искаженное изображение

Рис. 4. Восстановленное изображение

Работа выполнена при поддержке программ BRHE, INTAS и Российского фонда фундаментальных исследований (гранты №№ 03-01-00109, 04-07-90149, 04-07-96500).

Литература

1. Методы компьютерной обработки изображений / Под ред. В.А. Сойфера. М.:ква, Физматлит, 2001.

2. *Дроздов М.А.*, *Зимин Д.И.*, *Скуратов С.А.*, *Попов С.Б.*, *Фурсов В.А*. Технология определения восстанавливающих фильтров и обработки больших изображений // Компьютерная оптика. № 25, 2003.

3. *Dudgeon Dan E.*, *Mersereau Russell M.* , Multidimensional digital signal processing, Prentice-Hall, Inc., Englewood Cliffs, 1984.

4. *Pratt William K*. Digital Image processing. A wiley-interscinence publication, John Wiley and Sons, New York, 1978.

5. *Горячкин О.В*. Методы слепой обработки сигналов и их приложения в системах радиотехники и связи. М.: Радио и связь, 2003.

6. *Никоноров А.В.*, *Попов С.Б.*, *Фурсов В.А.* Идентификация моделей цветовоспроизведения // Матер. VI Международ. конф. «Распознование образов и анализ изображений: новый информационный подход», Новгород, 2002.

7. *Ford Adrian*, *Roberts Alan.* Color Space Conversions, 1998.

8. *Зимин Д.И.*, *Фурсов В.А.* Технология определения восстанавливающего фильтра и обработки цветных изображений // Компьютерная оптика. № 27, 2005.

9. *Зимин Д.И.*, *Фурсов В.А.* Итерационное планирование распределения ресурсов многопроцессорных систем // Матер. IV Международ. науч.-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах», Самара 2004.

МОДЕЛЬ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ И ЕЕ ИСПОЛЬЗОВАНИЕ ДЛЯ ОЦЕНКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ НА ИНСТРУМЕНТАЛЬНОМ КОМПЬЮТЕРЕ

В. Иванников, С. Гайсарян, А. Аветисян, В. Бабкова

Институт системного программирования РАН, Москва

1. Введение

Современные системы параллельного программирования несовершенны и трудны в использовании, а инструменты, входящие в их состав, не охватывают всего процесса разработки и опираются на технологии, разработанные десятки лет назад [1]. Кроме того, существующие средства разработки параллельных программ базируются на использовании целевой аппаратуры. Как следствие разработка каждой параллельной программы связана со значительными, и, что важнее, непредсказуемыми затратами труда прикладных программистов, производительность большинства программ составляет несколько процентов от пиковой производительности вычислительной системы. Необходимы новые технологии параллельного программирования доступные как программисту средней квалификации, так и специалистам в области других наук, умеющим программировать.

В настоящее время надежды с созданием таких технологий связаны с исследованием и разработкой адекватных моделей параллельных программ и параллельных систем и инструментальных средств, обеспечивающих разработку программ на моделях [2]. Это не только позволит перенести большую часть разработки на инструментальный компьютер, сократив время и стоимость разработки, но и существенно расширит возможности прикладного программиста по изучению и модификации разрабатываемой программы.

В настоящей работе рассматривается модель параллельной про-

граммы, которая может эффективно интерпретироваться на инструментальном компьютере, обеспечивая возможность достаточно точного предсказания времени реального выполнения параллельной программы на заданном параллельном вычислительном комплексе. Текущая версия модели разработана для параллельных программ с явным обменом сообщениями, написанным на языке Java с обращениями к библиотеке MPI, и включена в состав среды ParJava [3].

2. Модель параллельной программы

Модель параллельной Java-программы строится по ее абстрактному синтаксическому дереву. Это оказалось возможным, потому что Java – хорошо структурированный язык и граф потока управления Java-программы сохраняет структуру исходной программы. При построении модели в дереве сохраняются управляющие конструкции языка Java, а выражения и вызовы функций заменяются базовыми блоками. Определены следующие типы базовых блоков: вычислительный блок, вызов пользовательской функции, вызов внешней функции, вызов коммуникационной функции и редуцированный блок. Вычислительный базовый блок состоит из байт-кода выражений, входящих в его состав.

Одним из атрибутов вершины модели (внутренней, или базового блока) является время ее выполнения (интерпретации), оценка которого является задачей интерпретации модели. Следует отметить, что интерпретация модели позволяет оценить не только время, но и многие другие динамические атрибуты программы (объем требуемой памяти, степень локальности данных и др.).

3. Моделирование коммуникационных функций

Модель базового блока типа коммуникационная функция представляет собой описание соответствующей коммуникационной функции через базовые операции обмена. Определены следующие базовые операции обмена: `Init` (`Free`) – выделение (освобождение) ресурсов под служебные структуры данных; `Pack(n, type)` (`Unpack(n, type)`) – упаковка (распаковка) объектов, передаваемых в данной коммуникации; `Post(n)` (`Get(n)`) – отправка (прием) буфера памяти длиной n байт другому процессу; `Process(serv_msg)` – обработка служебного сообщения; `Copy(n)` – копирование буфера размерности n байт в памяти вычислительного узла; `MakeThread` – создание потока выполнения в программе;

Wait(событие) – ожидание события (под событием понимается прибытие сообщения на вычислительный узел). Например, операция `MPI_SSEND(n, type)` (блокирующая синхронная посылка), где `n` – размер передаваемого сообщения, `type` – тип пересылаемых данных, в соответствии со стандартом MPI выражается через базовые операции следующим образом:

```
{Init; Post(запрос); [Pack(n, type)];
Wait(разрешение); Post(n); Free;}
```

Операция `Pack` заключена в квадратные скобки, так как она выполняется только при использовании MPI в неоднородной сети.

Можно показать, что перечисленных операций достаточно для спецификации работы всех функций библиотеки MPI.

4. Интерпретация модели для оценки времени выполнения

Интерпретация модели параллельной Java-программы состоит в интерпретации модели метода `main()`, одного из классов, входящих в состав программы (указывается пользователем). Интерпретация модели функции (метода) состоит в интерпретации ее корня.

Интерпретация внутренней вершины зависит от типа этой вершины (последовательность, ветвление, цикл). Интерпретация вершины типа последовательность состоит в последовательной интерпретации вершин, входящих в последовательность. Интерпретация вершин типа ветвление состоит в интерпретации управляющего выражения, значение которого позволяет выбрать нужную ветвь, и последующей интерпретации выбранной ветви. Интерпретация вершины типа цикл состоит в интерпретации управляющего выражения, значение которого определяет момент выхода из цикла, и многократной интерпретации тела цикла до тех пор, пока выход из цикла не произошел. При интерпретации каждой внутренней вершины значение атрибута время вершин, подчиненных этой вершине, используются для вычисления атрибута время этой вершины.

Интерпретация базового блока типа вычислительный блок состоит в выполнении инструкций, составляющих этот блок, с целью определения значений переменных, вычисляемых в этом блоке и через управляющее выражение влияющих на работу одной из вершин типа ветвление или цикл. Время работы вычислительного блока определяется заранее, во время предварительного анализа программы.

Интерпретация базовых блоков типа вызов пользовательской функции и вызов внешней (библиотечной) функции состоит в интер-

претации выражений, определяющих значения фактических параметров соответствующего вызова, и последующей интерпретации корневого узла модели тела этой функции. В результате интерпретации вычисляется время работы корневого узла рассматриваемой функции, которое определяет атрибут время рассматриваемого базового блока.

Интерпретация базового блока типа «вызов коммуникационной функции» заключается в определении времени работы блока согласно выбранной модели коммуникаций. Поскольку длительность работы коммуникационной функции может зависеть от момента ее вызова, а также от момента вызова ответной коммуникационной функции, вводится понятие модельного времени. Параллельная программа, выполняемая на n узлах, состоит из n независимых логических процессов, каждый из которых имеет свои модельные часы. Начальные показания всех модельных часов равны 0. Показания модельных часов логического процесса обновляются после интерпретации в этом процессе каждого базового блока путем добавления значения атрибута *время* этого базового блока к текущему значению модельного времени. Интерпретация работы коммуникационных функций использует показания модельных часов различных процессов для выбора сценария выполнения коммуникации и для определения продолжительности коммуникации. Логические процессы, участвующие в коммуникации определяются интерпретатором. При интерпретации внутренних вершин показания модельных часов не меняются, так как интерпретация внутренней вершины сводится к выбору очередного базового блока, который необходимо интерпретировать.

Из 10 базовых операций, используемых для моделирования функций MPI, только операции `Post(n)` и `Get(n)` зависят от свойств коммуникационной системы. Для моделирования этих операций использована модель LogGP [4]. Согласно этой модели время выполнения операций `Post(n)` и `Get(n)` определяется по формулам:

$$Time(\texttt{Post(n)}) = b*n + d_P(n),$$

$$Time(\texttt{Get(n)}) = b*n + d_G(n),$$

где b – пропускная способность сети, а $d(n)$ – функция накладных расходов на передачу (прием), определяемая экспериментально с помощью бенчмарков.

Для вычисления длительности ожидания (операция `Wait (событие)`) достаточно иметь показания модельных часов в момент начала операции и в момент ее завершения.

5. Частичная интерпретация

При интерпретации модели параллельной программы выполняется меньший объем вычислений, чем при выполнении самой программы, однако и интерпретация может занимать значительное время. Частичная интерпретация, позволяет при определенных условиях выполнить интерпретацию части программы. Результаты интерпретации части программы могут использоваться для упрощения модели параллельной программы, что делает возможным интерпретировать программу «по частям»: определять время работы только интересующих фрагментов программы, один раз интерпретировать многократно использующиеся части программы, а также использовать результаты интерпретации частей программы при интерпретации всей программы, сокращая время одного сеанса интерпретации.

Для определения частичной интерпретации вводится операция редукции, которая определена только для вершин модели, интерпретация которых закончена и, следовательно, время ее выполнения определено.

По определению, результатом редукции базового блока является редуцированный базовый блок, время выполнения которого является константой, а байт-код исключен. Результатом редукции внутренней вершины является редуцированный базовый блок, время выполнения которого является константой. Таким образом, для применения операции редукции время выполнения редуцируемого фрагмента программы должно быть известно. Операция редукции внутренней вершины позволяет осуществлять интерпретацию отдельных поддеревьев модели программы, заменяя проинтерпретированные поддеревья редуцированными базовыми блоками. Следует отметить, что редукция отдельной вершины типа «вызов коммуникационной функции» может привести к некорректной модели программы, так как может нарушиться соответствие между посылками сообщений и их приемами в разных процессах программы. Поэтому операция редукции для вершин такого типа допустима только в составе операции редукции внутренней вершины. При этом редуцируемая вершина должна быть замкнута по коммуникациям, то есть при ее интерпретации не должно оставаться не принятых сообщений, и все операции приема должны завершаться.

Операция редукции позволяет интерпретировать программу по частям, заменяя проинтерпретированные фрагменты редуцированными блоками. В результате частичной интерпретации количество вершин модели сокращается, что позволяет сократить длительность интерпретации. При этом сохраняются оценки времени выполнения как для

программы в целом, так и для остальных единиц интерпретации, присутствующих в модели.

6. Реализация

Реализация включает средства построения модели параллельной программы и средства интерпретации модели. Из описания модели видно, что построение модели не является сложной задачей. Рассмотрим реализацию интерпретатора (рис. 1).

Интерпретатор осуществляет обход модели текущего метода согласно правилам, изложенным в п. 4 и 5. Во время интерпретации по описанию целевой вычислительной системы порождается набор логических процессов, каждый из которых строит оценку времени работы соответствующего процесса параллельной программы на целевой вычислительной системе. Максимальная оценка принимается в качестве оценки времени работы редуцируемой вершины. Если в качестве редуцируемой вершины выбрать корень одного из методов `main`, то будет интерпретирована вся программа.

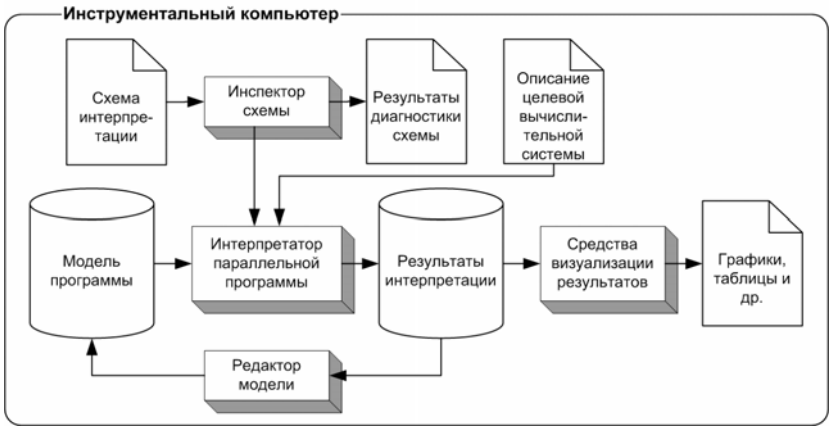


Рис. 1. Схема выполнения интерпретации
модели параллельной программы

Интерпретатор при своей работе использует различные параметры целевой системы: мощность вычислительных узлов, латентность и пропускную способность каналов передачи данных, длительность базовых операций MPI. Таким образом, располагая моделью программы и описанием целевой вычислительной системы, пользователь получает динамический профиль программы, используя только инструменталь-

ный компьютер.

Проверка точности интерпретации производилась на нескольких модельных примерах. Сравнение результатов прогноза времени выполнения с временем фактического выполнения для одного из таких примеров приведены на рис. 2. Рассматривается программа решения задачи теплопереноса в стержне с использованием явной разностной схемы. Размер матрицы увеличивался пропорционально количеству задействованных процессоров. Размер данных на одном вычислительном узле занимал приблизительно 80% памяти.

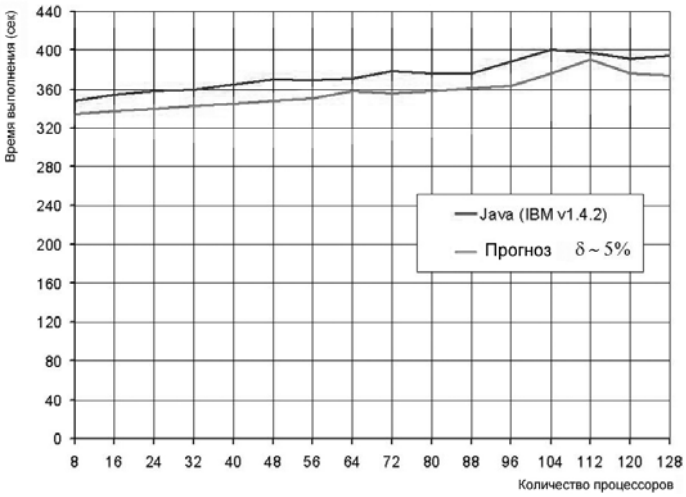


Рис. 2. Сравнение прогноза времени выполнения (серая кривая) с временем фактического выполнения (черная кривая).

Вычисления выполнялись на высокопроизводительном кластере МСЦ (PowerPC64 2.2GHz, Myrinet). Для прогнозирования времени работы программы использовалась рабочая станция с процессором Intel Pentium IV 2,6 GHz с 512 Mb оперативной памяти. Время, требуемое для получения прогноза, не превышало 60 сек. Относительная погрешность прогноза порядка 5%.

7. Заключение

Погрешность прогнозирования времени выполнения модельных программ для кластерных систем построенных на базе различных платформ и использующих различные коммуникационные подсистемы, не превышала 10%, время прогноза – нескольких минут. Это по-

зволяет говорить о применимости подхода.

В настоящее время среда ParJava используется: в Тамбовском государственном университете для разработки пакета символьно-численного решения задач линейной алгебры; в отделе систем моделирования ИСП РАН для разработки генетических алгоритмов для автономных адаптивных систем управления; в Институте физики Земли РАН для моделирования процесса зарождения торнадо. Разработка этих пакетов позволит получить более точные оценки и уточнить направление развития среды.

Дальнейшее развитие среды ParJava связано с расширением области применения моделей. Будут разработаны новые инструменты: анализаторы параллельных трасс с целью выявления узких мест, взаимоблокировок и др.; анализаторы параллельной структуры программы с целью выявления скрытого параллелизма как в циклах, так и в ациклических участках программы; преобразователи последовательных циклов в параллельные и др. Все эти инструменты будут выполняться на инструментальном компьютере, используя подходящие модели.

Литература

1. Report of the President's Information Technology Advisory Committee (PITAC) to the President on Computational Science: Ensuring America's Competitiveness. http://www.nitrd.gov/pitac/reports/.

2. Национальная целевая программа DARPA. High Productivity Computing Systems (HPCS), http://www.highproductivity.org/.

3. *Ivannikov V.*, *Gaissaryan S.*, *Avetisyan A.*, *Padaryan V.* Improving properties of a parallel program in ParJava Environment // EuroPVM/MPI, 2003, LNCS. V. 2840. P. 491–494.

4. *Culler David E.*, *Liu Lok T.*, *Martin Richard P.*, *Yoshikaw Chad*. LogP Performance Assessment of Fast Network Interfaces. // IEEE Micro, February, 1996.

ОБ ОДНОМ МЕТОДЕ ПОИСКА БАЗИСНОГО МИНОРА МАТРИЦЫ

Г.Г. Исламов, Ю.В. Коган, Д.А. Сивков, О.В. Бабич, С.А. Мельчуков, М.А. Клочков

Удмуртский государственный университет

В пособии [1] при формировании новой симплекс-таблицы были использованы некоторые формулы преобразования элементов текущей симплекс-таблицы. Важность этих формул состоит в том, что на их

основе удается сформулировать и обосновать метод поиска базисного минора матрицы, который позволяет одновременно вычислить ранг матрицы, значение базисного минора, обратную матрицу для матрицы базисного минора, координаты небазисных строк и столбцов, крайние точки и экстремальные направления многогранных множеств, а также ответить на вопросы совместности системы линейных неравенств, положительной определенности симметрических матриц, устойчивости многочленов с вещественными коэффициентами и др.

Метод заключается в построении последовательности матриц

$$A_0 = A, \ldots, A_k, A_{k+1}, \ldots, A_r, \quad (1)$$

где A_{k+1} получается пут ем простого преобразования элементов предыдущей матрицы $A_k = (a_{ij}^{(k)})_{i=1.j=1}^{m}$. В текущей матрице A_k есть запрещенные строки и запрещенные столбцы, остальные строки и столбцы этой матрицы называются разрешtнными. В матрице $A_0 = A$ все строки и столбцы разрешенные. Для перехода от A_k к следующей матрице A_{k+1} последовательности (1) в разрешенных строках и столбцах матрицы A_k возьмем произвольный ненулевой элемент $a_{\mu_k \nu_k}^{(k)}$ (называемый разрешающим элементом шага k). Если такого элемента нет, то есть нет разрешенных строк и столбцов, либо все элементы в разреш енных строках и столбцах матрицы A_k – нули, то последовательность (1) завершается матрицей $A_k : r = k$. Пусть строка μ_k и столбец ν_k – разрешенные, то формируем элементы матрицы A_{k+1} по следующему простому правилу

$$a_{ij}^{(k)} = \frac{1}{a_{\mu_k \nu_k}^{(k)}} \begin{cases} 1, & \text{если } i = \mu_k, j = \nu_k; \\ a_{ij}^{(k)}, & \text{если } i = \mu_k, j \neq \nu_k; \\ -a_{ij}^{(k)}, & \text{если } i \neq \mu_k, j = \nu_k; \\ a_{ij}^{(k)} \cdot a_{\mu_k \nu_k}^{(k)} - a_{\mu_k j}^{(k)} \cdot a_{i \nu_k}^{(k)}, & \text{если } i \neq \mu_k, j \neq \nu_k. \end{cases}$$

Список запрещенных строк и столбцов матрицы A_{k+1} получается из списка запрещ енных строк и столбцов матрицы A_k путем добавления номеров μ_k и ν_k разрешающего элемента $a_{\mu_k \nu_k}^{(k)} \neq 0$.

Теорема. *Пусть A_r - последняя матрица построенной выше последовательности* (1)*. Тогда*

1) Минор матрицы А, составленный из запрещенных строк и

столбцов матрицы A_r, базисный и равен произведению $(-1)^{p+q}\prod_{k=0}^{r-1} a^{(k)}_{\mu_k \nu_k}$, *где p и q – число инверсий соответственно в перестановке строк* $(\mu_0, \ldots, \mu_{r-1})$ *и* $(\nu_0, \ldots, \nu_{r-1})$ *столбцов разрешающих элементов;*

С точностью до перестановки строк и столбцов, определяемых перестановками строк и столбцов разрешающих элементов,

2) подматрица, составленная из запрещенных строк и столбцов матрицы A_r, является обратной матрицей к матрице указанного в заключении 1) теоремы базисного минора;

3) в базисе столбцов (строк) матрицы A_r, отвечающих найденному базисному минору, координаты тех столбцов (строк) матрицы, которые не входят в базисный минор, находятся в соответствующих столбцах (строках, взятых со знаком минус) матрицы A_r.

Следствие 1. *Все угловые миноры вещественной квадратной матрицы A порядка n положительны тогда и только тогда, когда указанный выше алгоритм для $\mu_k = \nu_k = k = 1$ дает положительную последовательности $r = n$ разрешающих элементов последовательности (1): $a_{kk}^{(k-1)} > 0, k = 1, \ldots, r$.*

Следствие 2. *Число положительных и число отрицательных собственных значений вещественной симметрической матрицы A порядка n совпадают с соответствующими числами положительных и отрицательных разрешающих элементов в последовательности (1).*

Замечание 1. *Если применить алгоритм (1) к расширенной матрице $A_0 = (A, b)$, где $b = (b_1, \ldots, b_m)^T$, предполагая, что начальный список запрещ енных столбцов содержит последний столбец b, то неотрицательность координат в разреш енных строках (если они есть) позволяет заключить о совместности системы линейных неравенств $Ax \le b$ относительно вектора $x = (x_1, \ldots, x_m)^T$ (сравнение векторов покоординатное). Если же для всех возможных последовательностей (1) указанные выше координаты содержат отрицательные числа, то система неравенств $Ax \le b$ несовместна.*

Этот результат вытекает из теоремы 1.5 монографии [2] о совместности системы линейных неравенств и утверждения 1) нашей теоремы о факторизации определителя через разрешающие элементы.

Замечание 2. *Утверждения 2) и 3) теоремы позволяют получить все крайние точки и экстремальные направления многогранного мно-*

жества $Ax = b$, $x \geq 0$ *для матрицы A ранга m, если воспользоваться алгебраической характеристикой крайней точки и экстремального направления, приведенных в монографии [3].*

Возможна реализация описанного выше метода на основе легковесных процессов (PThreads), параллельной виртуальной машины (PVM), интерфейса обмена сообщениями (MPI) и процессоров с общей памятью (OpenMP) применительно к задачам большой размерности обращения аффинного отображения, поиска образующих конуса неотрицательных решений однородной системы линейных алгебраических уравнений, установления положительной определ енности симметрических матриц и устойчивости многочленов с вещественными коэффициентами и включение его в стандартные пакеты.

Литература

1. *Исламов Г.Г.* Принципы оптимальности. Ижевск, Изд-во УдГУ, 1988. – 124 с.

2. *Черников С.Н.* Линейные неравенства. М.: Наука, 1968. – 488 с.

3. *Базара М.*, *Шетти Л.* Нелинейное программирование. Теория и алгоритмы. М.: Мир, 1982. – 583 с.

СТРАТЕГИЧЕСКОЕ НАПРАВЛЕНИЕ УДМУРТСКОГО ГОСУНИВЕРСИТЕТА

Г.Г. Исламов, Д.А. Сивков

Удмуртский государственный университет, Ижевск

Образование, наука и бизнес – это три мощных процесса в обществе, которые определяют вектор его развития. Каждый из этих процессов обладает внутренними движущими силами, которые формируются большими группами людей с принципиально различными интересами. Ускорение развития одного процесса без ускорения развития остальных процессов не имеет особого смысла, т. к. общество нуждается в синхронизированной работе этих процессов. Поэтому материальные средства, брошенные на ускорение одного процесса, не обязательно вызовут его прогресс, скорее всего они будут использованы на поддержание застоя. Главная цель образования – подготовка высококвалифицированных кадров, науки – получение принципиально новых частных и общих методов и фактов, бизнеса – поиск и использование новых форм извлечения прибыли. В образовательном процессе главный мотив – получение оригинальных знаний, умений и навыков, в

научном процессе – достижение престижа, в бизнесе – умножение денежного капитала.

Поэтому только в образовании и науке возможно скачкообразное движение вперед. Производство и бизнес могут развиваться только эволюционно, т. к. за деньги можно выполнить только тривиальную работу. От людей, занятых в сфере образования и науки, часто требуются такие усилия для поступательного движения вперед, которые невозможно оценить никакими денежными выплатами. Они настолько велики, что таких денег просто никто не даст.

В настоящее время востребованы специалисты, которые могут выдвинуть общие цели указанных трех процессов развития образования, науки и бизнеса. Они могут возникнуть только из образовательной и научной среды специалистов в области новых информационных технологий.

Удмуртский госуниверситет имеет все необходимое для того, чтобы начать подготовку таких специалистов на создаваемом факультете «Информационных технологий» по следующим направлениям: «Нанотехнологии», «Высокопроизводительные параллельные вычисления» и «Управление бизнес-процессами». Определенные успехи достигнуты в каждом из этих направлений, так в области изучения наноструктур достижения специалистов УдГУ и академических институтов ИПМ и ФТИ г. Ижевска отмечались доцентом Ю.С. Митрохиным в заметке «Нанотехнологии в XXI веке» газеты «Удмуртский университет» за 26 апреля 2005 г.

Учебно-научная лаборатория «Параллельных вычислительных систем и траспьютеров», возглавляемая профессором Г.Г. Исламовым, с 2000 г. проводит исследования в области кластерных технологий и организаций параллельных и распределенных вычислений с применением известных технологий MPI, OpenMP, DVM и др. Некоторые достижения в этой области описаны Г.Г. Исламовым в заметке «Высокопроизводительные вычисления и технологии» октябрьского номера газеты «Удмуртский университет» за 2003 г. и статьях [1-3]. Кроме того, в течение последних трех лет проводятся работы по изучению инструментария Globus Toolkit создания grid-систем научных расчетов. Полученный здесь опыт работы, а также теоретические и практические работы, проводимые в госуниверситете в области многомерного статистического анализа, создают благоприятную основу для инициирования отечественных работ по созданию собственных моделей и программных средств, аналогичных совместным разработкам IBM и SAS в

области кредитного скоринга. Корпорация IBM и компания SAS приложили значительные усилия для объединения известного семейства программных продуктов SAS Banking Intelligence Solutions и grid-технологии распределенных вычислений с целью их внедрения в банковскую сферу. Высокая стоимость предлагаемых IBM и SAS решений в области банковских технологий служит некоторым препятствием внедрения эффективных схем кредитования. Однако главным препятствием их внедрения в нашей стране служит отсутствие высококвалифицированных кадров, способных эффективно реализовывать такого рода решения на практике. Подготовку специалистов по всем указанным выше направлениям на факультете «Информационных технологий» можно начать на базе имеющихся специальностей «Прикладная математика и информатика», «Прикладная информатика» и др. в рамках соответствующих специализаций, что позволит сформировать необходимые кадры для открытия соответствующих специальностей. По направлению «Высокопроизводительные параллельные вычисления» в рамках специальности «Прикладная математика и информатика» планируется введение следующих спецкурсов: «Финансовая математика», «Защита информации», «Математическая экономика», «Системы массового обслуживания», «Вычислительные методы статистики» в разделе «Цикл общих гуманитарных и социально-экономических дисциплин», «Вычислительная геометрия», «Теория оптимального управления» и «Функциональный анализ» в разделе «Цикл математических и общих естественнонаучных дисциплин», «Оптимизация на сетях и графах», «Вычислительные методы линейной алгебры», «Численные алгоритмы нелинейной оптимизации», «Нейронные сети», «Сетевые технологии», «Grid-технологии научных расчетов», «Компьютерная графика», «Программирование в ОС Linux и Windows», «Сети Петри», «Квантовые вычисления», «Технологии распределенных вычислений», «Сетевые операционные системы», «Архитектура вычислительных систем», «Моделирование процессов молекулярной динамики и квантовой химии» в разделе «Цикл общепрофессиональных дисциплин».

Группа сотрудников и студентов Удмуртского госуниверситета под руководством профессора Г.Г. Исламова приступила к изучению кластерных решений задач управления динамическими объектами. Эти задачи приводят к численным расчетам в пространствах большой размерности. Подобные расчеты на персональном компьютере часто требуют несоизмеримых временных затрат. С другой стороны, система высокопроизводительных SMP-серверов, работающих под управлени-

ем операционной системы Linux и связанных между собой, позволяет на базе высокоскоростной сети организовать распределенные вычисления с применением известного инструментария Globus Toolkit 4 и пакетов параллельных вычислений (MPI, OpenMP, DVM, PVM) и, тем самым, получать результаты за приемлемое время. Расчеты математических моделей проводятся с широким использования специализированных пакетов SCALAPACK, PETSc, TAO, SLEPc и др. Для отладки параллельных программ на вычислительном кластере PARC имеются необходимые средства (TotalView, MPICH JumpShot, GDB, XPVM и др.).

Цель проекта – создание эффективных человеко-машинных систем реального времени управления динамическими объектами на базе высокопроизводительного кластера и инструментария Globus Toolkit 4.

Задачи проекта:

– Разработка параллельных алгоритмов расчета управления, обеспечивающего решению эволюционного уравнения необходимый векторный критерий;

– разработка параллельных алгоритмов расчета блока минимальной по рангу обратной связи с запаздыванием, обеспечивающего необходимую стабилизацию решений динамической системы.

Основные планируемые фундаментальные и прикладные результаты:

1. Эффективные критерии разрешимости эволюционных уравнений с граничными ограничениями в виде векторных неравенств, описывающих поведение динамических систем с требуемыми свойствами.

2. Прикладные программы расчета на кластере высокопроизводительных компьютеров управлений и начальных условий, обеспечивающих требуемые свойства решений эволюционных уравнений.

3. Эффективное описание блоков минимальных по рангу обратных связей с запаздыванием, гарантирующих необходимую степень стабилизации решений динамической системы.

4. Прикладные программы расчета на кластере высокопроизводительных компьютеров стабилизирующих блоков минимальной по рангу обратной связи с запаздыванием для динамических объектов.

Литература

1. *Исламов Г.Г.* От информационно-вычислительного кластера к виртуальной сети инфраструктуры образования, науки и бизнеса // Инновационные процессы в сфере образования и проблемы повышения качества подготовки специалистов / Сб. матер. международ. научно-методической

конф. 30-31 марта 2005 г. Т. 1. Ижевск: «Удмуртский университет», 2005. – С. 338–346.

2. *Исламов Г.Г.* Разработка универсальной структуры высокопроизводительного информационно-вычислительного кластера // Высокопроизводительные вычисления и технологии. Тезисы конференции. – Москва-Ижевск: Институт компьютерных исследований, 2003. – С. 88.

3. *Исламов Г.Г.* Создание научно-методического обеспечения подготовки специалистов в области высокопроизводительных кластерных технологий // Распределенные и кластерные вычисления. Избранные материалы Второй школы-семинара. Под ред. чл.-корр. РАН В.В. Шайдурова, д.ф.-м.н., проф. В.М. Садовского / Институт вычислительного моделирования СО РАН. Красноярск, 2002. – С. 54 – 69.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В ЗАДАЧАХФИЗИКО-ХИМИЧЕСКОЙ ГИДРОДИНАМИКИ: ПОДХОДЫ И ИДЕИ

В.Е. Карпов, А.И. Лобанов

Московский физико-технический институт

Физико-химические гидродинамические процессы часто встречаются в природе и технике. Их экспериментальное изучение является непростым. Во многих случаях проведение экспериментов невозможно. Проектирование новых технических установок с улучшенными экономическими и экологическими характеристиками при помощи экспериментов отнимает много времени, является дорогостоящим, а иногда даже опасным. Поэтому возникает большой интерес к компьютерному моделированию физико-химических процессов, например, к моделированию химических реакций в смеси газов.

Но проведение вычислительных экспериментов оказывается сложной задачей Применение прямого численного моделирования сильно ограничено такими компьютерными ресурсами, как память и процессорное время, и будет оставаться таковым в ближайшем будущем. Для того чтобы избежать возникающих ограничений, основные усилия исследователей в настоящее время сосредоточены в двух направлениях: упрощение математических моделей процессов и/или моделирование на параллельных вычислительных комплексах.

Используемые математические модели. Все трехмерные системы уравнений в частных производных, описывающие физико-химические гидродинамические процессы, должны включать в себя законы сохранения полной массы (уравнение непрерывности), частич-

ных масс фракций, импульса и энергии, уравнения состояния, уравнения химических реакций и правила для вычисления коэффициентов реакций. Полные системы уравнений, как правило, чрезвычайно трудны для численного решения. Поэтому обычно используются различные способы их упрощения. Основными направлениями таких упрощений являются: уменьшение пространственной размерности моделей, рассмотрение стационарных моделей, упрощение гидродинамической части моделей и моделей излучения, редукция химических реакций.

Методы дискретизации. Класс применяемых методов дискретизации, использующихся для аппроксимации систем уравнений физико-химической гидродинамики сравнительно невелик. Наиболее популярные способы аппроксимации — это метод конечных объемов на сетке в декартовых координатах [1, 2, 3] и методы конечных элементов [4, 5]. Выбор метода конечных объемов и/или метода конечных элементов позволяет получить высокую степень геометрического параллелизма на последующих стадиях решения задачи. Для численного моделирования применяются и некоторые другие схемы. В [6] пространственная дискретизация осуществляется методом конечных разностей с использованием центральных производных шестого порядка. Применение такого метода снижает схемную диссипацию, но не удаляет ее полностью. В приграничных узлах сетки используются схемы более низкого порядка.

В [7] рассматриваются методы введения искусственной сжимаемости. Такой подход приводит к уменьшению жесткости дискретных систем уравнений и делает возможным некоторое увеличение временного шага при решении системы явным методом Рунге-Кутты. Как известно, явные методы могут быть легко и эффективно распараллелены на параллельных вычислительных системах с распределенной памятью.

Численные методы. Системы уравнений, получающиеся после дискретизации, являются жесткими и обладают существенной нелинейностью. Для решения жестких систем уравнений применяются два основных класса численных методов: связанные алгоритмы и алгоритмы операторного расщепления.

Связанные методы. В [6] интегрирование по времени осуществляется с помощью явного метода Рунге-Кутты четвертого порядка. Для устойчивости проверяется выполнение условий Куранта-Фридрихса-Леви и Фурье. Дополнительный контроль точности решения обеспечивается по правилу Рунге. Численное моделирование проводилось на компьютере CRAY T3E. В качестве основного принципа распаралле-

ливания выбрано геометрическое. В [4, 5] исследовались стационарные состояния. Полученная нелинейная система решалась на параллельном комплексе методом Ньютона. Все уравнения обрабатывались одновременно, взаимосвязи между переменными включались в матрицу Якоби. Каждый процессор вычислял приращения по методу Ньютона для приписанных к нему узлов. После итерации по нелинейности полученные линейные системы решались итерационным методом с использованием программы из библиотеки AZTEC, реализующей предобусловленный метод Крылова.

Для решения уравнений, связывающих давления и скорости, и уравнений, выражающих законы сохранения, в [1] использован алгоритм SIMPLE. Для решения линеаризованных систем применялись итерационный метод с факторизацией оператора и функция пакета NAG, использующая решение СЛАУ в подпространстве Крылова. Конвективные члены аппроксимировались по схеме с центральными разностями. Для улучшения сходимости применялся многосеточный метод. Многосеточные методы были предложены Р. П. Федоренко [8]. Для декомпозиции расчетной области использовались вложенные сетки. Эта декомпозиция области вычислений была взята за основу для распараллеливания. Для балансировки загрузки процессоров использовалась статическая схема. Все процессоры считались одинаковыми по скорости работы, числа операций на каждом узле сетки равными, а подобласти выбирались так, чтобы включать примерно одинаковое количество узлов. Каждая подобласть приписывалась к одному процессору. Численное моделирование проводилось на компьютере CRAY T3E.

Аналогичный подход был рассмотрен в [2], где для решения результирующей системы уравнений использовался метод предиктор-корректор второго порядка по времени и четвертого порядка по пространству. Расчетная сетка разбивалась на подобласти, которые распределялись между процессорами. Для переносимости алгоритма использовался механизм передачи сообщений MPI. Это позволило использовать полученный код на различных параллельных платформ и оценить на них его базовую производительность. Применение явной схемы допускает простую, но достаточно эффективную балансировку загрузки процессоров. В конце работы как предиктора, так и корректора на каждом шаге процессоры обменивались граничными значениями с соседями. При расчетах требовалась только одна общая коммуникационная операция в начале каждого шага для вычисления его величи-

ны. Работа [2] — одна из немногих, где в качестве параллельной платформы использовался кластер из персональных компьютеров.

В [9] для решения гидродинамической части задачи реагирующих потоков был использован метод характеристик. Разработка численных методов решения уравнений в частных производных с использованием сеточно-характеристических методов была выполнена М.-К. М. Магомедовым и А. С. Холодовым [10]. Оригинальный путь параллельной реализации этого класса методов был разработан М. О. Васильевым [11, 12].

Операторное расщепление. Методы операторного расщепления используются в [3]. Для решения уравнений, описывающих поля давлений и скоростей, предложен SIMPLE-подобный алгоритм. Для возникающих систем алгебраических уравнений использован многосеточный подход, для уравнений сохранения массы фракций применена процедура расщепления на два шага: конвекция-диффузия и система химических реакций. Уравнения конвекции-диффузии решаются раздельно, а на химическом шаге — совместно для всех фракций и объемов с помощью модифицированного метода Ньютона для ЖС ОДУ. За основу для распараллеливания берется декомпозиция расчетной области. В качестве параллельной вычислительной системы с распределенной памятью использовался кластер Beowulf.

Другой подход к расщеплению по операторам – это использование локально-одномерных операторов. Основные идеи таких методов были разработаны Н.Н. Яненко [13], Писменом и Рэкфордом [14]. В [15] метод локально-одномерной аппроксимации операторов был объединен с быстрым преобразованием Фурье для решения уравнений Навье–Стокса, что позволило добиться хороших показателей эффективности распараллеливания как на машинах массового параллелизма с распределенной памятью, так и на кластерах из рабочих станций.

Решение жестких систем ОДУ (ЖС ОДУ). Большинство численных методов для решения задачи Коши требуют многократного вычисления вектора правой части системы уравнений. Большая размерность делает это вычисление трудоемким. Часто задачи Коши для систем ОДУ являются жесткими, для устойчивости численного решения требуются только неявные или диагонально-неявные методы. При их реализации приходится многократно вычислять матрицу Якоби и обращать ее. При использовании численных методов для решения задачи Коши необходимо управление шагом по времени. Предпочтительными оказываются методы, которые могут оценивать ошибки и использовать

оценки для управления величиной шага.

За последние годы было разработано семейство одноитерационных методов Розенброка для ЖС ОДУ и систем дифференциально–алгебраических уравнений [16–18]. Главной их особенностью является только одно вычисление и обращение матрицы Якоби на каждом шаге. В [17] приведен алгоритм построения жестко-точного А-устойчивого метода Розенброка порядка 4. Коэффициенты метода подобраны из условия минимизации вычислительной ошибки. Метод является вложенным, оценка погрешности вычислений выполняется без дополнительных затрат. Результаты сравнительных тестов [17] показывают, что построенный метод превосходит классические. В [19] метод типа Розенброка был применен для решения задач атмосферной химии.

Неявный метод решения чрезвычайно жестких систем ОДУ предложен в [20] для моделирования многофазных химических реакций в облаке с каплями, разделенными на классы по размерам. Для интегрирования уравнений, описывающих химию фаз, фазовые переходы между газом и жидкостью и перенос масс между различными классами капель, использованы формулы дифференцирования назад (ФДН) высокого порядка. Решение возникающих разреженных систем линейных уравнений проводится модификацией кода LSODE, учитывающей особенности их структуры. В [20] применяется приближенная матричная факторизация с явной генерацией матрицы Якоби. Проведенные расчеты показали, что вычислительная стоимость метода растет линейно по отношению к количеству классов капель, но он не является параллельным. По тому же пути использования приближенной матричной факторизации для уменьшения вычислительной стоимости метода типа Розенброка идут и в [21].

Другой подход к модификации ФДН методов для их параллельной реализации на машинах массового параллелизма рассматривается в [22]. Этот подход развивается для жестких систем ОДУ с низкой степенью связности. Благодаря разреженности матриц вычисление правых частей, матриц Якоби, LU разложение матриц в методе Ньютона и прямая/обратная подстановки при решении могут быть выполнены параллельно.

Использование метода минимальной невязки в приближенном неявном методе для вычисления временного шага [23] на фиксированном числе итераций позволяет свести решение нелинейной задачи к решению набора линейных. Каждая из них может быть решена явно, метод обладает высоким потенциалом для распараллеливания. Существует

большое количество параллельных реализаций методов типа Рунге–Кутта: однократно неявный метод [24, 25], диагонально-неявный метод [26], Ньютоновский параллельный итерационный солвер линейных систем для методов Рунге–Кутта [27]. Эти методы являются многостадийными, и вычисления стадий могут проводиться параллельно. В [28] исследуется возможность динамической балансировки загрузки процессоров для таких методов. Для параллельных реализаций методов типа Розенброка [29] основная идея является той же, что и для параллельных методов Рунге–Кутта. Каждая стадия обрабатывается на своем процессоре параллельно с остальными.

Заключение. Проведенный анализ работ в области задач физико-химической гидродинамики показал, что практически все проводимые исследования в настоящее время связаны с использованием численного моделирования на параллельных системах. При этом выделяются два основных подхода: геометрическое распараллеливание и использование многостадийных методов для решения жестких систем ОДУ.

Геометрическое распараллеливание применяется на этапе решения гидродинамических частей задач и легко реализуется. После решения задач в пределах своих подобластей на текущем шаге процессоры производят обмен новыми граничными значениями. Малые обмены данными делают при этом эффективным использование вычислительных комплексов с большим количеством процессоров и распределенной памятью.

Все многостадийные методы обладают достаточным потенциалом для распараллеливания. Для *s*-стадийного метода вычисления для значений стадий могут быть осуществлены параллельно на *s* процессорах. Из-за больших обменов информацией между процессорами эти методы являются особенно эффективными при использовании на машинах с разделяемой памятью с небольшим числом процессоров – по числу стадий.

С точки зрения авторов представляется целесообразным соединить геометрический параллелизм, достигаемый при конечно-разностной аппроксимации, с многостадийным параллелизмом, используемым в методах типа Розенброка, на смешанных параллельных системах – кластерах, состоящих из SMP-узлов. Геометрический параллелизм реализуется на уровне отдельных узлов кластера, в то время как параллельное решение ЖС ОДУ внутри каждой подобласти – по стадиям на процессорах узла.

Работа выполнена при поддержке РФФИ-TNO, грант № 04-07-08029 2004

Литература

1. *Robbert Verweij L. et al.* Parallel Computing for Reacting Flows Using Adaptive Grid Refinement // Contemporary Mathematics, V. 218, 1998. P. 538–546.

2. *Stone C.*, *Menon S.*. Parallel Simulations of Swirling Turbulent Flames // The Journal of Supercomputing. V. 22. 2002. P. 7–28.

3. *Còнsul R.*, *Pérez-Segarra C.D. et al.* Detailed numerical simulation of laminar flames by a parallel multiblock algorithm using loosely coupled computers // Combustion theory and modelling. V 7. 2003. P. 525–544.

4. *Salinger A.G.*, *Shadid J.N. et al*. Parallel Reacting Flow Calculations for Chemical Vapor Deposition Reactor Design // Proc. of the Int. Conf. on Comput. Eng. Sci., San Jose, Costa Rica, May 4–9, 1997.

5. *Salinger A.G.*, *Pawlowski R.P. et al*. Computational Analysis and Optimization of a Chemical Vapor Deposition Reactor with Large-Scale Computing, February 9, 2004, http://endo.sandia.gov/DAKOTA/papers/ CVD_2004.pdf.

6. *Lange M.*, *Warnatz J.* Massively Parallel Direct Numerical Simulation of Turbulent Combustion // NIC Symposium 2001, Proceedings, John von Neumann Institute for Computing, Jülich, NIC Series. V. 9. 2002. P. 419–429.

7. *Wang Yi.*, *Trouvé A.*. Artificial acoustic stiffness reduction in fully compressible, direct numerical simulation of combustion // Combustion theory and modeling. 8. 2004. P. 633–660.

8. *Федоренко Р.П.* Релаксационный метод решения разностных эллиптических уравнений. // ЖВМиМФ. 1961. Т. 1. № 5.

9. *Pakdee W.*, *Mahalingam S.* An accurate method to implement boundary conditions for reacting flows based on characteristic wave analysis // Combustion theory and modeling. V. 7. 2003. P. 705–729.

10. *Магомедов М.-К.*, *Холодов А.С*. Сеточно-характеристические численные методы – М.: Наука, 1988. – 290 с.

11. *Полуосьмак В.В*., *Васильев М.О.* Разработка алгоритмов параллельного счета для решения задач магнитной гидродинамики в применении к задаче о взрыве в верхней ионосфере / Современные проблемы фундаментальных и прикладных наук. Тр. XLVII науч. конф. МФТИ. 2004. Т. 3. С. 199–200.

12. *Vasilev M.O.*, *Repin A.Ju*. et al. Numerical researches of formation of jet stream of plasma in large-scale geophysical experiment. http://130.246.71.128/pdf/P1_070.pdf.

13. *Яненко Н.Н*. Метод дробных шагов решения многомерных задач математической физики. – Новосибирск, Наука. – 1967.

14. *Peaceman D.W.*, *Rachford H.H.* The numerical solution of parabolic and elliptic differential equations, J. SIAM, 3(1955). P. 28–41.

15. *Averbuch A.*, *Ioffe L.*, *Israeli M.*, *Vozovoi L.*. Two-dimensional parallel solver for the solution of Navier–Stokes equations with constant and variable coefficients using ADI on cells // Parallel Computing. V. 24. 1998. P. 673–699.

16. *Хайрер Э., Нерсетт С., Ваннер Г.* Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. – М.: Мир, 1990. – 512 с.

17. *Хайрер Э., Ваннер Г.* Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально – алгебраические задачи. – М.: Мир, 1999. – 685 с.

18. *Di Marzo G.A.* RODAS5(4), methodes de Rosenbrock d'ordre 5(4) adaptees aux problemes differentiels-algebriques // Memoire de diplome en Mathematiques, Universite de Geneve 1992.

19. *Sandu A.*, *Verwer J.G. et al*, Benchmarking of stiff ODE solvers for atmospheric chemistry problems II: Rosenbrock solvers. // Atmospheric Environment. V. 31, № 20. 1997. P. 3459–3472.

20. *Wolke R.*, *Knoth O.* Time-integration of multiphase chemistry in size-resolved cloud models // Applied Numerical Mathematics. V. 42. 2002. P. 473–487.

21. *Botchev M.A.*, *Verwer J.G.* A new approximate matrix factorization for implicit time integration in air pollution modeling // J. of Comput. and Appl. Math. V. 157. 2003. P. 309–327.

22. *Nordling P.*, *Sjö A.* Parallel solution of modular ODEs with application to rolling bearing dynamics // Mathematics and computers in simulation, 44 (1997). P. 495–504.

23. *Botchev M.A.*, *van der Vorst H.A.* A parallel nearly implicit time-stepping scheme Journal of Computational and Applied Mathematics, 137. 2001. P. 229–243.

24. *Voss D.A.*, *Muir P.H.*. Mono-implicit Runge-Kutta schemes for the parallel solution of initial value ODEs. // J. of Comput. and Appl. Math. V. 102. 1999. P. 235–252.

25. *Muir P.H. et al*. PMIRKDC: a parallel mono-implicit Runge–Kutta code with defect control for boundary value ODEs // Parallel Computing 29. 2003. P. 711–741.

26. *Jackson K.R.*, *Norsett S.P.* The Potential for Parallelism in Runge-Kutta Methods, *SIAM J. Numer. Anal.* 32, No. 1. 1995. P. 49–82.

27. *Petcu D.*. Experiments with an ODE Solver on a Multiprocessor System // An Int. Journal computers & mathematics with applications, 42. 2001. P. 1189–1199

28. *Ruiz J. M.M.*, *Lopera J. O.*, *Carrillo de la Plata J. A*. Component-Based Derivation of a Parallel Stiff ODE Solver Implemented in a Cluster of Com-

puters. // International Journal of Parallel Programming. V. 30, No. 2. April 2002. P. 99–148.

29. *Voss D.A.*, *Khaliq A.Q.M.*. Parallel Rosenbrock methods for chemical systems // Computers and Chemistry, 25. 2001. P. 101–107.

РАЗРАБОТКА ОДНОРАНГОВОЙ РАСПРЕДЕЛЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

С.В. Ковальчук, А.Ю. Владова

Оренбургский государственный университет

Анализ существующих распределенных систем показывает, что на данный момент самой распространенной является архитектура с постоянным выделенным сервером. Это обусловлено рядом причин:

– Статичность структуры. Выделенный сервер на все время работы остается центральным связующим звеном всей системы. Даже если структура сети представляет собой всего лишь дерево глубиной 1, корнем этого дерева на все время работы остается сервер.

– Централизованное управление. Управление системой осуществляется сервером, который контролирует подчиненные вычислительные и управляющие системы. Такой подход является наиболее легким в реализации и понимании.

– Существенное упрощение при создании программного обеспечения для таких систем. Работа с выделенным сервером позволяет избавиться от необходимости учета изменений структуры сети на стороне клиента. Кроме того, клиентская часть программы может вообще не нести в себе никаких управляющих структур, реализую лишь вычислительные функции.

– Простота выдачи задач и получения результатов. Эти операции, как и управление, системой осуществляются централизованно через выделенный сервер системы.

– Данный подход к построению архитектуры распределенной системы в большинстве случаев является приемлемым и даже оптимальным, по сравнению с другими вариантами. Однако он не лишен некоторых недостатков, которые в определенных условиях могут оказать решающее воздействие на надежность работы системы и даже удобство ее использования. К таким показателям можно отнести, например, следующие критерии:

– Наличие выделенного сервера может служить недостатком в том

случае, если вероятность выхода его из строя достаточно высока. Так как в этом случае может разрушиться вся структура сети вследствие выпадения корневого элемента построенного дерева.

– К тем же результатам может привести выход из строя канала связи с сервером. Так как, несмотря на продолжение работы сервера, он пропадает из системы.

– Стабильность связи элементов сети распределенной системы является существенным фактором, влияющим на работу всей системы, так как выход из строя даже одной линии связи может разбить сеть, связывающую вычислительные системы в лучшем случае на подсети (если используется многоуровневая древовидная структура сети). При этом продолжить корректно функционировать сможет только та подсеть, в которой остался выделенный сервер.

– В некоторых случаях наличие централизованной статичной структуры нежелательно по внешним причинам. Например, в случае, если нельзя гарантировать постоянную или даже периодическую доступность хотя бы одного компьютера в сети с целью реализации на нем серверной части системы.

Указанные условия, например, могут реализоваться в сети Интернет, в случае если нет возможности использовать какой-либо постоянно доступный сервер. В этом случае реализация распределенной системы с выделенным сервером неприемлема. Перечисленные недостатки отсутствуют у распределенных систем без выделенного сервера.

Самым распространенным решением проблем с нестабильной и низкоскоростной связью являются метакомпьютерные системы. Они успешно справляются с задачами, представляющими большие требования к вычислительным мощностям элементов сети, чем к качеству связи между ними. Примером такой системы может служить распределенная система SETI@Home, осуществляющая поиск внеземных цивилизаций.

Однако обычно в таких системах все же присутствуют статически выделенные сервера. Что так же является недостатком в условиях нестабильности связи, так как выход из строя сервера или связи с ним приводит к разрушению всей системы.

Одним из возможных решений является использование нескольких серверов в системы. Но такой подход не меняет структуры системы и, следовательно, не может полностью избавить от указанной проблемы.

Целью данной работы является попытка реализовать более производительную и стабильную архитектуру распределенных систем, не-

жели большинство ныне используемых. Одноуровневая система позволяет решать задачи распределенных вычислений в нестабильной разнородной среде передачи данных. Даже в случае динамического изменения конфигурации сети она сохраняет работоспособность с минимальными потерями данных. Реконфигурация сети может произойти даже в случае выхода из строя первоначального корня покрывающего дерева.

Для проведения исследований по данному вопросу ставится цель – повысить производительность и стабильность архитектуры распределенной системы.

Практической целью данного проекта является разработка распределенной системы на базе кафедры ПОВТАС Оренбургского Государственного Университета. Предположительно эта система существенно расширит возможности по проведению научных исследований и разработок на базе кафедры. Кроме того, система позволит проводить исследования и решать практические задачи качественно нового уровня, так как при этом появится возможность использовать сравнительно большой и разнообразный парк машин кафедры как единую метакомпьютерную систему. По сравнению со стандартной распределенной системой появится ряд дополнительных возможностей, существенно расширяющий круг функциональных возможностей системы. Так, например, благодаря принципу динамического построения дерева управления и возможности переформирования дерева появится возможность запускать задачу с любой из клиентских машин, которая вполне может выступать в роли корня управляющего дерева. Отсутствие же жестких ограничений на класс задач позволяет практически неограниченно расширять круг задач, поддающихся решению на проектируемой системе. Таким образом, проектируемая система позволит значительно расширить функциональность вычислительной сети кафедры.

Для реализации цели предлагается решить следующие задачи исследования:

1. Определить особенности функционирования системы.

2. Спроектировать одноранговую распределенную вычислительную систему (на базе имеющейся аппаратуры). Выявить ограничения, накладываемые аппаратным обеспечением и потенциальные возможности, предоставляемые им.

3. Разработать более подробный алгоритм реализации.

4. Формализовать содержание паспорта задачи и интерфейс модуля решения задачи:

4.1. Определить наиболее выгодные принципы построения оптимального покрывающего дерева.

4.2. Рассмотреть варианты разбиения различных задач на подзадачи.

4.3. Определить протокол обмена между узлами дерева.

4.4. Выявить основные параметры мониторинга системы.

4.5. Разработать алгоритм сбора результатов вычисления

5. Выполнить реализацию спроектированной системы на базе одной из передовых на сегодняшний день технологий – .net.

6. Проанализировать эффективность функционирования системы по следующим параметрам: производительность, надежность, гибкость структуры сети, эффективность децентрализованного управления (задания задачи и выборки результатов)

В рамках первой задачи на основе анализа публикаций [2] в качестве особенностей функционирования определены:

– отсутствие статически выделенного сервера;

– совмещение в программе функций вычислительной и управляющей системы;

– возможность ввода задания через любой из узлов системы;

– возможность забора результатов с любого из узлов системы из числа тех, что имеют информацию о решаемой задаче;

– возможность самовосстановления с минимальными потерями данных даже при выходе из строя корневого узла построенного дерева;

– решаемая задача не задается жестко в коде программы, а загружается в виде модуля при вводе пользователем.

Рассмотрим некоторые особенности функционирования предлагаемой распределенной системы.

Предварительным этапом работы системы является построение сети, узлами которой являются вычислительные системы. Однако построение этой сети может продолжаться в процессе функционирования системы. Узлы сети постепенно получают информацию о все большем количестве доступных им соседних узлов. Таким образом, пределом в данном случае является полный граф, содержащий все вычислительные узлы системы, то есть каждый узел имеет информацию о расположении каждого узла. Однако, очевидно, что это невозможно ввиду неоднородности сети и разбиения ее на подсети, имеющие ограниченный доступ друг к другу. Информация о доступных соседних узлах используется для построения наиболее оптимального покрывающего дерева сети, по которому в последствии будет выполняться задание. Принци-

пы выбора предпочтительного варианта покрывающего дерева могут быть различными. Простейшим случаем является выборка дерева, ребра которого оптимальны по скорости передачи данных. После построения сети система готова к выполнению задания. Однако перед этим возникает необходимость описать его. Описание включает в себя две части:

1. Паспорт задачи. Включает в себя описание задачи, версию, параметры ее распределения (возможная глубина дерева, возможное количество подзадач и т.д.), ограничивающие факторы и т.п.

2. Исполняемый модуль. Включает в себя реализованные алгоритмы решения части задачи, алгоритм выделения частей задачи, алгоритм сбора полученных результатов в единое целое и т.п.

3. Для корректного функционирования узел должен располагать обеими частями описания задания. Каждый узел хранит «библиотеку» задач, решавшихся на этом узле, что избавляет от необходимости каждый раз подгружать полное описание задачи. После уяснения задачи необходимо определить конкретные параметры решения задачи. Например, диапазон перебираемых значений в случае решения соответствующей задачи. После ввода этих параметров система готова к решению поставленной задачи.

Первым этапом является построение покрывающего дерева. При этом необходимо учитывать следующие требования к целевому дереву:

- изначально выбранный принцип оптимизации дерева,
- требования, заданные в паспорте задачи,
- требования, следующие из введенных параметров задачи,
- результаты предыдущих вычислений.

Исходя из этих требований, система строит покрывающее дерево сети. При этом возможно, что ввиду ограничений будет использована не вся существующую сеть, а только ее часть. После построения дерева вычислений система приступает к раздаче подзадач, выделенных исходя из заданных параметров. При этом раздача происходит рекурсивно, то есть подзадачи разбиваются на более мелкие подзадачи, те в свою очередь на еще более мелкие и т.д. Разбиение происходит с учетом доступных мощностей. Следует учесть, что корень, производящий распределение задач на каком либо уровне берет часть вычислений на себя.

После раздачи задач система приступает непосредственно к вычислениям. В процессе вычислений производится контроль целостности построенного дерева. В случае ее нарушения управляющие струк-

туры системы производят попытки по мере сил восстановить работоспособность системы. Это заключается в динамическом перераспределении подзадач, решение которых изначально было возложено на вычислительные системы, потерявшие связь с системой. Кроме того, в процессе работы производится мониторинг сети. Мониторинг включает в себя контроль динамики работы системы.

Завершающим этапом работы системы является сбор результирующих данных. Сбор можно осуществлять как из корневого узла системы, так и из любого из узлов, имеющих информацию о решаемой задаче (аналогично с любого подобного компьютера может производиться контроль работы системы). После завершения вычислений результаты работы узла сохраняются в его «библиотеке решений». Такой подход позволит реализовать некоторое подобие алгоритмов динамического программирования и избежать повторного вычисления одинаковых подзадач.

Таким образом, можно выделить следующие особенности системы:

– однородность узлов системы. Следствием этой особенности является их взаимозаменяемость, а так же возможность легкой реструктуризации системы;

– динамическое построение дерева вычислительных систем для решения конкретной задачи;

– возможность самовосстановления с минимальными потерями в случае отказа, как отдельных узлов сети, так и связи между какими-либо узлами;

– отсутствие жестко заданной задачи. Задачи подгружаются при их постановке. Однако это не означает, что задача будет загружаться каждый раз. У каждого узла системы создается библиотека задач, которая представляет собой своеобразный кэш задач. Задачи удаляются при длительном не использовании;

– оптимизация работы системы.

Вывод

Разрабатываемая система позволяет добиться существенного прироста надежности работы по сравнению с распределенными системами иной архитектуры. При этом достигается качественно новый уровень гибкости и динамичности системы, как с точки зрения ее внутренней архитектуры, так и с точки зрения внешнего пользователя, осуществляющего управление системой. Одним из главных ее преимуществ является однородность полученной программной среды, и динамичность

нагрузки на узлы сети, что позволяет, не заботится о выделении достаточно крупных вычислительных ресурсов под управляющие структуры. При этом система предполагает автоматическое разбиение вычислительно сети на подсети, позволяющие решать независимые задачи параллельно.

Таким образом, проектируемая система предоставляет более стабильный и гибкий инструмент для решения задач, требующих ресурсы распределенной системы.

Литература

1. *Костинский А*. Метакомпьютеры.
2. *Киселев А.*, *Корнеев В.*, *Семенов Д.*, *Сахаров И.* Управление метакомпьютерными системами.
3. *Затуливетер Ю.С.* Проблемы метакомпьютинга

РАЗРАБОТКА СРЕДЫ ПАРАЛЛЕЛЬНОГО РАСПРЕДЕЛЕННОГО ПРОГРАММИРОВАНИЯ ГСПП.NET

Д.В. Котляров, В.Н. Маланин

Московский энергетический институт (*ТУ*)

Введение

Несмотря на многочисленные попытки реализовать эффективные средства разработки высокопроизводительных параллельных распределенных приложений, ни одна из широко распространенных инструментальных сред программирования не предоставляет разработчику возможностей для разработки параллельных распределенных программ, ориентированных на кластерные системы. Большинство современных программистов предпочитает выполнять программные проекты, используя интегрированные среды разработки, позволяющие минимизировать временные затраты на реализацию алгоритмов, сконцентрировав внимание на их проектировании. Поэтому, введение инструмента, обеспечивающего поддержку параллельных распределенных вычислений, в известную интегрированную среду разработки приведет к важному расширению языковых средств параллельного программирования. Важным требованием подобной интеграции является организация соответствия инструмента параллельного распределенного программирования и инфраструктуры целевой системы, как на уровне взаимодействия подсистем, так и на уровне идеологии построения про-

граммных компонентов.

В докладе описывается проект по внедрению средств граф-схемного потокового параллельного программирования (ГСППП) [1,2] в среду Microsoft Visual Studio 2005 (MS VS2005).

Основа внедряемой системы – это язык граф-схемного потокового параллельного программирования (ЯГСППП), предложенный проф.В.П. Кутеповым как «мягкое» развитие структурных (блочно-схемных) форм описания последовательных программ. ЯГСППП ориентирован на крупноблочное (модульное) потоковое программирование задач, он также может эффективно применяться для программного моделирования распределенных систем, систем массового обслуживания и др., информационные связи между компонентами которых структурированы и управляются потоками данных, передаваемых по этим связям.

Необходимо отметить следующие принципиального характера особенности ЯГСППП:

– модульный принцип построения параллельной программы, причем с возможностью программирования модулей на последовательных языках программирования;

– наглядное граф-схемное описание структуры параллельной программы в виде двух компонентов: граф-схемы и интерпретации, однозначно сопоставляющей каждому модулю подпрограмму на соответствующем языке программирования;

– интерпретация связей между модулями как связей по данным (а не по управлению);

– экспликация параллелизма как информационной независимости различных модулей граф-схемы.

Язык позволяет эффективно и единообразно представлять в программах три вида параллелизма:

– параллелизм информационно-независимых фрагментов;

– потоковый параллелизм, обязанный своим происхождением конвейерному принципу обработки данных;

– параллелизм множества данных, реализуемый в ЯГСППП через механизм тегирования, когда одна и та же программа или ее фрагмент применяются к различным данным;

Среда параллельного программирования, основанная на ЯГСППП, является сложной распределенной системой и содержит три основных элемента:

– подсистему проектирования ГСППП, в функции которой входит

взаимодействие с пользователем на этапе разработки ГСПП;

– интерпретатор или систему выполнения ГСППП, обеспечивающих поддержку операционной семантики ЯГСППП;

– подсистему планирования параллельных распределенных вычислений и управления загруженностью кластера;

В 2005 году на кафедре прикладной математики МЭИ(ТУ) был реализован прототип системы ГСППП для кластеров.

Как нам представляется, переформулирование ЯГСППП в терминах и конструкциях языков объектно-ориентированного программирования (ООП) и реализация этой новой версии в среде .NET позволит существенно расширить возможности этой среды при проектировании параллельных распределенных программ.

Инструментальная среда параллельного программирования нового поколения

Рассмотрим особенности MS VS2005, которые повлияли на выбор ее в качестве целевой системы для реализации новой версии ЯГСППП. Отметим изменения, которые необходимо ввести в ЯГСППП и общую архитектуру описанной выше среды ГСППП для того, чтобы упростить этот переход и сделать его естественным.

Основным направлением совершенствования Visual Studio остается создание средств для реализации преимуществ платформы Microsoft .NET Framework.

Кратко опишем архитектуру этой платформы и те преимущества, которые она предоставляет для реализации среды ГСППП.

.NET Framework является надстройкой над ОС Windows и включает в себя:

– постоянно расширяющийся набор языков программирования (.NET–совместимых ЯП), соответствующих спецификации Common Language Specification (CLS), определяющей минимальные требования, предъявляемые к языку платформы. В частности, CLS содержит требования к унификации типов данных (Common Type System - CTS). Так любая сущность должна являться объектом класса, порожденного от класса System.Object;

– объектно-ориентированную среду выполнения Common Language Runtime (CLR);

– обширную библиотеку классов Framework Class Library, возможности которой может использовать программа, написанная на любом NET-совместимом ЯП.

Многоязычность .NET и возможность прямого взаимодействия программ и программных компонентов, написанных на различных .NET-совместимых ЯП, позволяют легко реализовать одну из базовых особенностей ЯГСППП – возможности выбирать наиболее удобный последовательный язык при программировании подпрограмм модулей. Кроме того, в реализации подпрограмм программист сможет использовать все элементы библиотеки FCL, включающие базовые средства работы со строками, математическими операциями, взаимодействия с ОС, в том числе организацию параллельных участков подпрограмм (multithreading).

Однако наличие CTS и общая объектно-ориентированная направленность платформы, требует расширения ЯГСППП и введение в него элементов ООП. Рассмотрим элементы граф-схемной программы в терминах ООП.

Напомним, что ГСППП представляется в виде пары <ГС,I>, где ГС – граф-схема, I – интерпретация. Граф-схема или просто схема позволяет визуально представлять строящуюся из модулей программу решения задачи; интерпретация сопоставляет каждому модулю множество подпрограмм, а связям между модулями – типы данных, передаваемых между подпрограммами модулей в процессе выполнения ГСППП.

В новой интерпретации граф-схема представляет собой графическое описание взаимодействия набора объектов, описанных в форме интерфейсов. Таким образом, каждый модуль или подсхема есть интерфейс, предоставляющий некоторые услуги. Определение реализации интерфейсов происходит при интерпретации схемы, когда модулю сопоставляется класс, реализующий соответствующие услуги. Элементы схемы взаимодействуют при помощи сообщений, использующих заданные коммуникационные линии – информационные связи (ИС). В качестве сообщений могут использоваться объекты классов. Расширим требование строгой типизации ИС таким образом, чтобы ИС, которой при интерпретации был сопоставлен некоторый класс С, могла бы быть использована для передачи экземпляра класса-наследника класса С. Таким образом, учитывая механизм полиморфизма, объекты, передаваемые по связям ГС, становятся «активными», способными принимать воздействия из модуля, в который они переданы, определять свою реакцию на эти воздействия и осуществлять эти реакции, т.е. становятся агентами. Такое нововведение в ЯГСППП с нашей точки зрения позволяет значительно расширить описательные средства языка в области построения программных моделей сложных систем, в частности,

систем массового обслуживания.

MS VS2005 является одной из наиболее развитых в плане сопровождения процесса решения задач программирования. Она обеспечивает весь жизненный цикл программного продукта, начиная от этапа сбора информации и заканчивая тестированием и развертыванием, и является самодостаточным «центром» программирования в том смысле, что программисту не нужно обращаться к другим средам программирования в процессе создания решения. С точки зрения интеграции средства параллельного программирования, MS VS2005 является идеальной оболочкой для взаимодействия встраиваемого средства с пользователем на всех этапах построения и выполнения параллельной распределенной программы, представляя унифицированный интерфейс пользователя, а также доступ к средствам повышения эффективности разработки, таким как подсветка синтаксиса, IntelliSense, автоматическая генерация кодов, отображение структуры проекта [3]. Поэтому, оставляя неизменными принципы организации параллельных распределенных вычислений и управления загруженностью кластера, изложенные в [1], а также общую архитектуру системы выполнения ГСППП (модифицировав в соответствии с внесенными в язык изменениями), мы вводим в MS VS2005 средства взаимодействия с человеком для каждой из подсистем, в том числе системы проектирования ГСППП. При этом физически сервер планирования и сервер системы выполнения представляет совершенно другие приложения или службы на локальном (с запущенной VS) или удаленном компьютере. Объединение интерфейсных функций в одном центре играет очень важную роль: это позволяет связать проектирование и выполнение ГСППП в единый технологический процесс.

Заключение

Очень важным фактором успеха проекта является рациональный подход к командной разработке сложных программных продуктов. В нашей работе мы используем модель ведения проектов MSF [4], что позволяет говорить о сбалансированном подходе к проектированию и реализации среды ГСППП, а также о возможности дальнейшего ее расширения. Уже сейчас мы учитываем варианты введения в систему поддержки GRID-вычислений, работы на удаленном кластере через веб-интерфейс, а также усложнению иерархической структуры целевого кластера, закладывая в проект среды ГСППП большой потенциал для развития.

Литература

1. *Котляров Д.В*., *Кутепов В.П*., *Осипов М.А*. Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах. М: Из-во РАН, Теория и системы управления, 2005, №1.

2. *Кутепов В.П.*, *Котляров Д.В.* Управление загруженностью кластерных вычислительных систем / матер. Четвертого международ. научно-практического семинара и Всероссийской молодежной школы, Изд-во СНЦ РАН, 2004.

3. *Young Marc*, *Johnson Brian*, *Skibo Craiq*. Inside Microsoft Visual Studio .Net 2003. MS Press, 2003.

4. Учебный курс. Анализ требований и определение архитектуры решений на основе Microsoft.Net. М: Из-во Русская редакция, 2004.

РАСПРЕДЕЛЕННАЯ КОМПЬЮТЕРНАЯ СРЕДА ДЛЯ ПОДДЕРЖКИ СИСТЕМНОГО ИНЖИНИРИНГА КОСМИЧЕСКИХ СИСТЕМ

Р.К. Кудерметов

Запорожский национальный технический университет, Украина

Анализ публикаций по технологиям создания сложных систем показывает, что средства компьютерной поддержки их разработки еще недостаточно исследованы. В [1] проведен анализ состояния таких компьютерных средств и технологий, введено понятие информационно-моделирующей среды, которая обеспечивает за счет использования распределенных, параллельных вычислений, а также возможностей сети Internet, интеграцию модельного и информационного сопровождения разработок сложных систем. В настоящей работе предпринята попытка сформулировать основные требования к функциональным характеристикам распределенной компьютерной среды информационной и модельной поддержки на жизненном цикле космических систем (КС).

Согласно определению Европейского сообщества по космической стандартизации [2] КС состоит из космического сегмента, сегмента пусковых услуг и наземного сегмента. В процессе создания КС выделяют вид деятельности – системный инжиниринг (СИ), который обеспечивает интеграцию и контроль процессов проектирования, верификацию КС, соответствие характеристик КС требованиям заказчика. Процессы СИ сопровождают весь жизненный цикл КС и могут носить итеративный характер.

Упрощено СИ включает пять функций:

- функция интеграции и управления, которая координирует совместный вклад всех остальных функций и дисциплин в ходе всех этапов проекта с целью оптимизации полного описания и реализации системы;
- функция разработки и подтверждения требований к системе, обеспечивающая их полноту, непротиворечивость и соответствие требованиям заказчика;
- функция анализа, состоящая из двух подфункций, связанных между собой, но имеющих различную природу:
- определения, документирования, моделирования и оптимизации функционального представления системы (функциональный анализ);
- аналитической поддержки требований, процесса проектирования и подтверждения характеристик системы;
- функция конструкции и конфигурации, которая генерирует физическую архитектуру продукта и определяет ее в виде комплекта документации, являющегося исходными данными для процесса изготовления;
- функция подтверждения характеристик, которая представляет собой итеративный процесс сравнения результатов различных функций друг с другом, с целью их сближения с удовлетворительным результатом с точки зрения требований, функциональной архитектуры и физической конфигурации, а также определяет и выполняет процессы, в соответствии с которыми окончательный продукт проектирования проверяется на предмет соответствия заданным требованиям.

Важное место в СИ занимает моделирование, которое, при более подробном рассмотрении каждой из выше перечисленных функций СИ, является их неотъемлемой дисциплиной.

При анализе роли и подходов к организации и применению моделирования в разработке сложных систем на практике оказывается, что наиболее часто моделирование выступает как отдельный вид работы для отдельных этапов и целей СИ, как по информационному, так и по аппаратно-программному обеспечению. Потенциал моделирования более полно может быть достигнут при интеграции возможностей и результатов всех видов моделирования [3] для различных функций и дисциплин СИ. Более того, потенциал самого СИ может быть более полно реализован при интеграции моделирования с остальными дисциплинами функций СИ.

Исходя из такого подхода к рассмотрению СИ, возникает идея

объединения компьютерных аппаратных, программных и информационных средств в единую среду – распределенную компьютерную среду поддержки СИ космических систем (РКСКС). РКСКС, как инструмент обработки информации и данных, создаст предпосылки и условия организации накопления и целостности данных, обмена этими данными между функциями и дисциплинами СИ, а также между различными этапами разработки КС.

Построение распределенной вычислительной системы как интегрированной программно-аппаратной среды, определяемое такими целями, имеет много общего с задачами построения распределенных и параллельных вычислительных систем – решение задач доступа и распределения ресурсов среды, надежности и информационной безопасности, обеспечение когерентности и т.д. Но есть и собственные задачи, в частности, разработка интерфейсов для обмена данными между функциями, дисциплинами и этапами СИ; определение правил управления данными и их накопления; возможная реконфигурация архитектуры РКСКС в соответствие с задачами СИ и специальными видами моделирования (например, полунатурное моделирование, моделирование в реальном времени) и испытаний. Эти задачи связаны как с пространственным и временны*м* распределением процессов СИ, так и с разнородностью аппаратных и программных инструментальных средств СИ.

Литература

1. *Аноприенко А.Я.*, *Святный В.А*. Высокопроизводительные информационно-моделирующие среды для исследования, разработки и сопровождения сложных динамических систем // Наукові праці Донецького державного технічного університету. Сер. Проблеми моделювання та автоматизації проектування динамічних систем. Вип. 29. – Донецьк: ДонНТУ. – 2001, С. 346–367.

2. European Cooperation for Space Standadization. ECSS-E00A. http://www.estec.esa.nl/ecss.

3. *Murphy C.A.* , *Perera T.* . The definition of simulation and its role within an aerospace company, Simulation Practice and Theory 9 (2002) P. 273–291.

ДВУХЪЯДЕРНЫЕ ПРОЦЕССОРЫ КАК ПЛАТФОРМА ДЛЯ СОЗДАНИЯ БУДУЩИХ HPC-КЛАСТЕРОВ. ТЕСТИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ

М.Б. Кузьминский

Институт органической химии РАН, г. Москва

Введение

Основным направлением повышения производительности микропроцессоров (МП) в ближайшее время становится переход к многоядерной архитектуре. Потенциально интересные для высокопроизводительных (HPC) кластеров МП – как x86-совместимые Intel Xeon/Paxville, AMD Opteron, так и более дорогие IBM Power5, Intel Itanium 2 (Montecito) являются двухъядерными, и типичные МП в кластерах ближайшего будущего станут именно многоядерными МП.

Можно отметить 2 основные особенности применения современных двухъядерных МП, важные для HPC-кластеров: уменьшение пропускной способности (ПС) оперативной памяти (ОП) в расчете на ядро МП из-за разделения ядрами «общей шины» ОП [1] и резкий рост (в пределе – удвоение) процессорной производительности узла, что вызывает проблему адекватности ПС межсоединения. Уменьшение ПС в расчете на ядро вкупе с ростом числа ядер в узле ставят вопрос об эффективности распараллеливания внутри узла, в т.ч. и путем распараллеливания в модели общего поля памяти (OpenMP).

Как показано в [1], микроархитектура двухъядерных МП Opteron и некоторые их технические характеристики могут давать им преимущества, в т.ч. при использовании в HPC-кластерах, поэтому в работе проведено тестирование производительности этих МП на ряде индустриальных тестов и задачах вычислительной химии.

Тестирование производительности

Нами использован сервер Supermicro AS-102A-8 с двумя двухъядерными МП Opteron 275 и набором микросхем - AMD8131/AMD8111, с ОП DDR333 (4 модуля DIMM емкостью по 1 Гбайт, по 2 модуля на каждый разъем МП). Результаты некоторых тестов сопоставлены с данными для двухпроцессорных серверов с Opteron 242 (также с ОП DDR333), используемых в Центре компьютерного обеспечения химических исследований РАН (ЦКОХИ).

Сервер работал с SuSE Linux 9.0 для x86-64 (ядро 2.4.21-SMP), используемой также в кластере на базе Opteron 242. Применялись средст-

ва LAM MPI 7.0 и фортран-компиляторы Pathscale 2.1, Intel ifort-8.1.023, pgf77/pgf90 6.0-4. В тестах Linpack (n=1000) сопоставлены 64-разрядные библиотеки - AMD acml 2.6.0, Intel MKL 7.2, Atlas 3.7.8 и Kazushiga Goto 0.94. Для измерения времени выполнения тестов Linpack применялся наиболее точный и стабильный таймер с использованием RDTSC [2], доработанный в ЦКОХИ для поддержания архитектуры x86-64.

В качестве приложений использованы квантовохимические комплексы программ Gamess-US [3] и Gaussian-03 Rev. C02 [4], применяющие средства распараллеливания Linda/OpenMP и DDI для Gaussian и Gamess соответственно. Тестировалась стандартная двоичная версия от Gaussian, Inc., транслированная с pgf77-5.1. Gamess-US был скомпилирован нами в 32-разрядном варианте с использованием ifort-9.0, с ключами -O3 -xW.

На сервере с Opteron 242 (Табл.1) ПС ОП (и масштабирование при переходе к 2 МП) в MPI-версии тестов STREAM 5.4 обычно несколько ниже по сравнению со стандартной OpenMP-версией. Распараллеливание OpenMP-версии в серверах с одноядерными Opteron близко к идеальному (2.0), что является следствием интеграции в МП контроллера ОП.

Таблица 1

Сопоставление разных процессоров Opteron на тестах STREAM

Компилятор, ключи	Цикл	Opteron 242				Opteron 275	
		MPI		OpenMP		OpenMP	
		1ЦП	2ЦП	1ЦП	2ЦП	1ЦП	2ЦП
pathf90, как в табл.2	copy	2961	5700	3257	6339	3747	4825
	scale	2877	5509	3173	6185	3705	4819
	add	3052	5775	3362	6685	3529	4706
	triad	3165	5962	3371	6694	3517	4704
				(1)		Opteron242 (2)	
pathf90, -O3 -static	copy	3413	6324	3565	6481	2990	4200
	scale	3396	6382	3524	6352	2707	4192
	add	3461	6485	3689	7190	3146	4577
	triad	3408	6381	3696	7254	3134	4640

Примечания.

(1) Наилучший замер с ключами оптимизации, как в табл. 2; сопоставление с верхней половиной табл.1 позволяет оценить разброс в оценке ПС при измерениях.

(2) Данные при включенной опции BIOS «Node memory interleaving»; ключи оптимизации - как в табл.2

Таблица 2

Результаты тестов STREAM (Мбайт/с) на МП Opteron 275

Компилятор	Цикл	Число ядер		
		1	2	4
pgf77(1)	copy	3850	4895	4505
	scale	2339	3303	3134
	add	2462	3522	3264
	triad	2476	3520	3304
pathf90(2)	copy	3347	4825	4525
	scale	3705	4819	4509
	add	3529	4706	4336
	triad	3517	4704	4336
ifort(3)	copy	1667	3178	2961
	scale	1674	3163	2882
	add	1779	3366	3112
	triad	1796	3403	3160

Примечания. Ключи оптимизации:
(1) -O2 -Mvect=sse-Mnontemporal -Munsafe_par_align -mp;
(2) pathf90 -O3 -CG:use_prefetchnta -LNO:prefetch_ahead=4 -mp; (3) -O3 -xW -openmp .

В серверах с двухъядерными Opteron мы обнаружили проблему разделения ядрами ПС ОП (см. табл.2). Так, масштабирование ПС при переходе к применению двух ядер для pathf90 составило около 30%. Оно различно для разных компиляторов, но при переходе к 4 ядрам для всех компиляторов ПС на 4 нитях оказалась меньше, чем на двух нитях. Эта проблема позволяет объяснить плохое масштабирование некоторых приложений с числом ядер. Данные результаты, как и другие, приведенные ниже, могут быть улучшены при переходе к применению ядер от 2.6.12 и старше, имеющих эффективную поддержку NUMA-режима.

На тестах Linpack (n=100) наилучшие результаты обеспечивает компилятор ifort (Табл.3). Кстати, для МП Xeon Nocona/3.2 ГГц с использованием ifort нами была достигнута производительность 1584 MFLOPS (наивысший результат по сравнению с официальной таблицей Linpack).

Таблица 3

Производительность на тестах Linpack (*n* = 100), MFLOPS

Компилятор и ключи	Производительность	
	Opteron 242	Opteron 275
pathf90 -Ofast	788 (1)	1138
ifort -xW -O3 -ipo	989	1385 (2)
pgf77 -fastsse -tp k8-64	882	1278

Примечания. Все приведенные результаты относятся к размерности матрицы, содержащей коэффициенты перед неизвестными, равной 200x200.
(1) Свыше 870 MFLOPS при использовании ключей -O3 -ipa -IPA:linear=ON, однако в этом режиме генерируется некорректный выполняемый код. При уровне оптимизации -O2 на Opteron 242 достигается 808 MFLOPS.
(2) 1404 MFLOPS с ключом -fast и последующей коррекцией кода для обеспечения возможности выполнения на Opteron (правилами теста Linpack это запрещено).

На тестах Linpack(n=1000), Табл.4, лучшие результаты по производительности и масштабируемости для Opteron 242 дает acml. Для MKL и acml ускорение на двух МП Opteron 242 близко к 1.9, на двух ядрах Opteron 275 ускорение меньше (1.3 раза); при переходе к 4 ядрам производительность возрастает лишь на 20% для Atlas и на треть - для MKL. Это ухудшение обусловлено, вероятно, проблемой ПС ОП.

Таблица 4

Производительность на тестах Linpack (n = 1000), MFLOPS

Компилятор, библиотека	Opteron 242		Opteron 275		
	1 ядро	2 ядра	1 ядро	2 ядра	4 ядра
ifort, MKL	1808	3377	2447	4630	6147
pathf90, Atlas	2167(1)	3342	3043	4020	4818
pgf77, acml	2325	4370	3225	н/д	н/д
Пиковое значение	3200	6400	4400	8800	17600

Примечание (1): с библиотекой goto - 3155 MFLOPS

В табл. 5 приведены результаты для Opteron 275 с Gaussian, а в табл. 6 – для Gamess-US. В качестве базовой использована молекула тринитротриаминобензола из test178 к Gaussian-03 (300 базисных функций), кроме test397 (808 базисных функций). Test178 при переходе к двум ядрам Opteron 275 в OpenMP распараллеливается удовлетвори-

тельно. При работе с Linda, где обмены данными между параллельными процессами больше, очевидно, возникла проблема ПС ОП. На 4 ядрах была использована гибридная схема распараллеливания: 2 нити в OpenMP при двух Linda-процессах. В целом ускорение в OpenMP оказывается удовлетворительным, и обычно выше, чем в Linda. Данные test397/Linda указывают на проблему ПС ОП: в кластерах даже с Gigabit Ethernet удвоение числа МП приводит к ускорению в 1,.8–1,9 раза, а на 4 ядрах Opteron 275 оно сильно хуже.

Таблица 5

Ускорение при распараллеливании тестов Gaussian-03 на Opteron 275

Тест	Время, с	Ускорение при распараллеливании			
		OpenMP		Linda	
	1 ядро	2 ядра	4 ядра	2 ядра	4 ядра
test178 (1)	203	1,87	3,28	0,86	1,22 (2)
test178mp2 (3)	5814	1,86	2,74	1,85	2,88
test178cis (4)	5655	1,96	3,41	н/д	3,13
test397 (5)	13935	1,92	3,58	1,85	2,95

Примечания.
(1) Стандартный test178(RHF) в режиме nosymm;
(2) 2 Linda-процесса, каждый из которых распараллелен в OpenMP;
(3) MP2=FullDirect/6-31G** nosymm Density=Current с молекулой из test178;
(4) CIS=Direct/6-31G(2d,p) nosymm с молекулой из test178;
(5) Стандартный test397 (метод DFT) в режиме NoFMM

Таблица 6

Тесты Opteron 275 с Gamess-US

	Test 178	Test 178mp2
Время для 1 ядра, с	452	1277
Ускорение на 2 ядрах	1,97	1,71
Ускорение на 4 ядрах	3,31	3,31

В тестах Gamess-US использованы те же молекулы и методы расчета, что и в Gaussian. Применение DDI с настройкой на работу через сокеты в Gamess-US для RHF и MP2 дает вполне удовлетворительные результаты, близкие к полученным в OpenMP для Gaussian.

Для исследования возможностей распараллеливания задач квантовой химии, в которых лимитирует ПС межсоединения, в модели обмена сообщениями MPI, «не знающей» о наличии общей ОП, испытыва-

лась создаваемая в рамках проекта РФФИ 04-07-90220 программа для замены диагонализации фокиана (в полуэмпирических расчетах сверхбольших молекул) прямым построением матрицы плотности. В рамках параметризации AM/1 с использованием матричных полиномов в схеме «очистки» матрицы плотности [5] в ЦКОХИ был проведен тестовый расчет молекулы, содержащей 2600 атомов, и на 3 ядрах Opteron 275 было получено ускорение 2,75.

Таким образом, имеется большое количество задач вычислительной химии, в которых применение двухъядерных Opteron достаточно эффективно.

Автор выражает благодарность РФФИ за поддержку работы (проект 04-07-90220), компании «Т-платформы» – за возможность доступа к серверам на базе Xeon Nocona, и компании Niagara Computers – за предоставление на тестирование сервера AS-102A-8.

Литература

1. *Кузьминский М.Б.* // Открытые системы, №10 (2005). С. 16.

2. *Кузьминский М.Б. и др.* // Высокопроизводительные параллельные вычисления на кластерных системах / Сб. матер. II международ. научно-практического семинара, ННГУ, Н.Новгород, 2002. С. 169.

3. *Schmidt M.W.*, *Baldridge K.K.*, *Boatz J.A. et al.* // J. Comp. Chem. V. 14, (1993). P. 1347.

4. Gaussian 03 Revision C.02, M.J.Frish, G.W.Trucks, H.B.Shlegel e.a., Gaussian, Inc., Wallingford CT, 2004.

5. *Кузьминский М.Б. и др.* // Высокопроизводительные параллельные вычисления на кластерных системах / Сб. матер. IV международ. научно-практического семинара, СНЦ РАН, Самара, 2004. С. 141.

ПРАКТИКА ИСПОЛЬЗОВАНИЯ В КЛАСТЕРАХ АППАРАТНОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ INFINIBAND ОТ MELLANOX. РАСПАРАЛЛЕЛИВАНИЕ В ЗАДАЧАХ ВЫЧИСЛИТЕЛЬНОЙ ХИМИИ

М.Б. Кузьминский, А.М. Чернецов, О.Ю. Шамаева

Институт органической химии РАН, Москва
Московский энергетический институт

Введение

Использование современной технологии Infiniband для связи узлов высокопроизводительных вычислительных кластеров является эффек-

тивным во многих случаях. Перспективность Infiniband связана не только с высокими техническими характеристиками, в первую очередь, производительности (отметим, например, рекордную пропускную способность нового поколения адаптеров DDR Infiniband фирмы Mellanox, Inc – в 2 раза выше, чем в недавно анонсированных адаптерах Myrinet 10G). Она обусловлена открытостью архитектуры Infiniband и потенциальной широтой ее применения в различных областях, в т.ч. в системах хранения и при работе с базами данных [1].

Актуальность тестирования производительности при использовании програмно-аппаратных средств Infiniband от Mellanox связана, во-первых, с широким применением продукции именно этой фирмы (аппаратные решения Mellanox используются и некоторыми другими производителями Infiniband) и, во-вторых, широким применением в программном обеспечении Mellanox для ОС Linux разработок проекта OpenIB, т.е. «универсальных» решений. В данной работе проведено, в частности, тестирование производительности аппаратно-программных средств Infiniband 4x от Mellanox на ряде стандартных индустриальных тестов и при работе с приложениями квантовой химии.

Практика применения Infiniband и тестирование производительности

Применение Infiniband для распараллеливания вычислительных задач в кластерах может реализовываться как путем использования средств распараллеливания MPI, работающих с Infiniband, так и путем использования средств распараллеливания, работающих поверх TCP/IP, например, Linda [2] или DDI [3]. Поэтому представляют интерес как измерения производительности на тестах MPI, так и тестирование производительности TCP/IP при работе поверх Infiniband (IPoIB). Последнее актуально не только для распараллеливания, но и для других приложений TCP/IP.

В тестах было использован установленный в Центре компьютерного обеспечения химических исследований РАН (ЦКОХИ, в Институте органической химии РАН) двухузловой кластер без коммутатора, в двухпроцессорных узлах которого применялись микропроцессоры Opteron 242/1,6 ГГц с материнскими платами Tyan S2880 и двухпортовые адаптеры HCA Mellanox MTLP23108 со 128 Мбайт собственной памяти (MHXL-CF128-T) для шин PCI-X/133. В узлах использовалась ОС SuSE Linux 9.0 с SMP-ядрами 2.4.21 для x86-64.

В качестве программного обеспечения Infiniband применялся свободно доступный стек IBGD/IBHPC от Mellanox [1], причем за время

более чем годичной эксплуатации были испытаны три разные версии - 0.5.0, 1.6.1 и 1.8.0, содержащие, в частности, различные версии MPI Национального центра суперкомпьютерных приложений США (NCSA) и Государственного университета штата Огайо (OSU, США). Приводимые ниже результаты измерений при работе с MPI относятся к OSU MPI (mvapich-0.9.4).

Версия 0.5.0 при инсталляции потребовала изменений в исходном тексте HCA-драйвера, а при работе в используемой конфигурации аппаратных и программных средств было обнаружено много ошибок. Версия 1.6.1 является, по-видимому, первой достаточно стабильной. Основным достоинством последней версии 1.8.0, по нашему мнению, являются усовершенствования при работе с протоколом SDP (см. ниже).

Некоторые тесты были проведены с использованием кластера на базе двухпроцессорных узлов с Intel Xeon Nocona/3.2 ГГц и HCA от Mellanox, аналогичных используемым в ЦКОХИ, но для шин PCI-Express. В качестве коммутатора применялся Mellanox MTS2400. Кластер работал с ОС SuSE SLES9 и IBGD-1.6.0.

Использованной во всех тестах технологии Infiniband 4x отвечает сигнальная пропускная способность (ПС) 10 Гбит/с (пиковая ПС данных равна 8 Гбит/с).

Для измерения производительности стека протоколов TCP/IP применялись средства netperf версии 2.3 (см., например, [4]). В кластере ЦКОХИ нами найдено, что достигаемые показатели производительности очень далеки от аппаратных возможностей. Так, максимально достигнутая нами ПС в TCP_STREAM равна 1633 Мбит/с при практически полной загрузке процессора, в то время как в тех же условиях на Gigabit Ethernet (со встроенным адаптером Broadcom) ПС равна 939 Мбит/с. В тесте TCP_RR с однобайтовыми сообщениями, характеризующем задержки, ускорение Infiniband по отношению к Gigabit Ethernet также найдено близким (15985 против 10031 пакетов, т.е. в те же 1,7–1,6 раза). Однако нагрузка на процессор при этом гораздо ниже, не более 20%. ПС, полученная при работе с UDP_STREAM также имеет величину порядка 1,5 Гбит/с. Эти данные были получены для IBHPC 0.5.0. При переходе к IBGD 1.6.1 достигаемая ПС на тестах TCP_STREAM остается примерно на том же уровне. В кластере на базе Nocona/3,2 ГГц удалось достигнуть более высоких показателей – 2366 Мбит/с в TCP_STREAM и 16082 пакета в TCP_RR. Увеличение ПС связано, очевидно, с более высокой производительностью процессоров

Nocona по сравнению с использованными в ЦКОХИ (в тестах TCP_RR производительность процессора не является лимитирующей).

Эти результаты для TCP/IP коррелируют с отсутствием существенного ускорения при переходе от Gigabit Ethernet к Infiniband при распараллеливании в кластерах квантовохимических программ Gaussian-03 [5] и Gamess-US [3], использующих соответственно Linda и DDI, работающие поверх TCP.

Применение SDP [6], позволяющего прозрачным образом заменить обращение исполняемых модулей (прикладных программ) к сокетам TCP вызовами SDP-библиотеки, позволяет увеличить ПС и уменьшить задержки, а также снизить нагрузку на процессор. Измеренная нами ПС не превышала величины порядка 4 Гбит/с, однако при этом процессор оставался почти полностью загружен. В IBGD 1.8.0 реализована версия SDP, поддерживающая прямой обмен данными между буферами приложений при синхронных операциях ввода-вывода, что позволяет, наконец, кардинально уменьшить процессорную нагрузку и еще поднять величину ПС [6].

В собственных тестах производительности Infiniband от Mellanox, perf_main, при использовании сервиса RC (reliable connection), применяемого также в SDP, найденная ПС составила 796 Мбайт/с при задержке для сообщений нулевой длины в 5,9 мкс. Загрузка процессора при выполнении этого теста близка к 100%.

В тестах MPI OSU и ПС, и задержки найдены близкими к аппаратным возможностям. Так, задержка во «встроенном» тесте MPI OSU равна 5,2 мкс. Нами проведены измерения производительности на тестах Pallas 2.2. В SendRecv (см. рис. 1) максимальная ПС (758 Мбайт/с) была получена при длине сообщения в 256 Кбайт (в тесте ping-pong ПС достигает при этом 695 Мбайт/с). Задержки в тесте ping-pong при длине сообщения до 4 байт не превышают 5 мкс, в тесте SendRecv при длине сообщения до 4 байт – не больше 5,85 мкс, в тесте Bcast при длине сообщения до 8 байт – не больше 5,0 мкс; на операцию barrier уходит около 5,8 мкс (все измерения проведены для двух узлов, по 1 процессу на узел).

Измерения производительности OSU MPI проведены также в рамках пакета Presto. В коммуникационном тесте MPI дал максимальную ПС 717 Мбайт/с при однонаправленной передаче (для сообщений размером 4 Мбайт) и 809 Мбайт/с при двунаправленной передаче (для сообщений размером 0,5 Мбайт). Задержки в тесте двунаправленной передачи Send/Recv близки к 8 мкс.

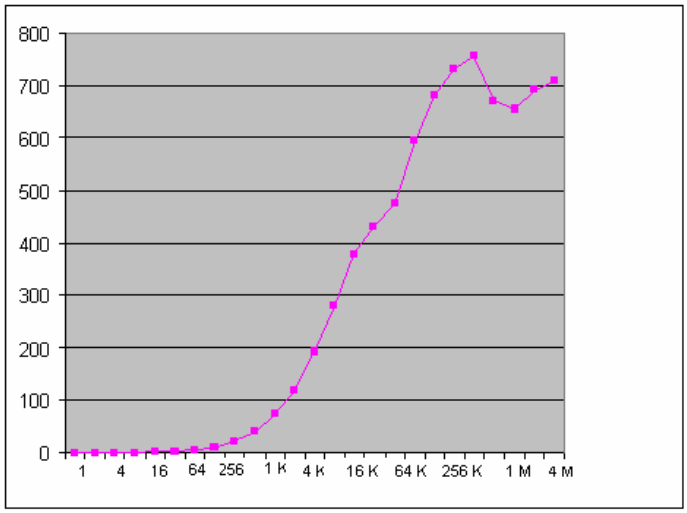


Рис.1. Зависимость ПС (Мбайт/с), измеренной для OSU MPI на тестах Pallas SendRecv, от длины сообщения

Программы вычислительной химии отличаются очень большим разнообразием используемых методов и численных алгоритмов, и соответственно при распараллеливании могут по-разному вести себя по отношению к задержкам и ПС межсоединения. В программах молекулярной механики/молекулярной динамики обычно лимитируют задержки. Примером случая, когда лимитирует ПС межсоединения, является разрабатываемая авторами программа замены диагонализации фокиана в полуэмпирических методах типа AM/1 прямым построением матрицы плотности для больших органических молекул. В использованной в тестах программе применялся подход, основанный на матричных полиномах Чебышева [7].

Нами было проведено сравнение величин ускорений, достигаемых при распараллеливании этой программы в кластере Infiniband на базе Nocona, с аналогичными данными для кластера на базе Myrinet и узлов с Xeon/2.6 ГГц для молекулы с размерностью базиса в 3000 орбиталей. В Infiniband-кластере на 3, 6 и 10 процесорах без применения технологии разреженных матриц нами достигнуто ускорение соответственно в 2,8, 5,3 и 8,0 раз при запуске по 1 вычислительному процессу на узел. Это выше, чем ускорение, достигаемое в лучшем зарубежном программном комплексе MOPAC2002, полученное для SGI Altix 300 [7].

Это также выше ускорений, достигнутых в Myrinet-кластере, причем преимущество Infiniband возрастало с числом процессоров. При 6 процессорах ускорение в кластере Infiniband на 37% выше, чем в кластере Myrinet.

Из-за использования в кластере Infiniband старой версии библиотеки Atlas, не оптимизированной на новые 64-разрядные микропроцессоры Nocona, производительность при вызове подпрограмм умножения матриц dgemm (определяющих процессорное время расчета) на процессорах Nocona была лишь немного выше, чем на Xeon/2,6 ГГц. Это и позволяет сделать корректным сравнение уровней распараллеливания. Напротив, сопоставление распараллеливания нашей программы с MOPAC2002 проведено в условиях, с точки зрения производительности процессоров и ПС межсоединения более выгодных для последней.

Мы протестировали также распараллеливание нашей программы с использованием в кластере Infiniband не по одному, а по 2 процесса на двухпроцессорный узел. Время выполнения возрастает при этом на величину порядка 10%, что связано с проблемой разделения ПС системной шины узлов при обращении к оперативной памяти. В Myrinet-кластере [8], узлы которого имеют системную шину с существенно более низкой ПС, чем в узлах с Nocona, использование при распараллеливании двух процессоров на узел ранее вообще было найдено невыгодным [7].

В кластере ЦКОХИ применение Infiniband с распараллеливанием в Linda типовых методов RHF, DFT, MP2 и CIS с использованием Gaussian-03 вообще не дало преимуществ по сравнению с Gigabit Ethernet. Поэтому мы исследовали характеристики возникающего при распараллеливании трафика на уровне сетевых интерфейсов, в т.ч. с применением iptraf, а также использовали для анализа средства tcpdump. Наши данные показали, что в небольших кластерах для молекул достаточного размера межсоединение вообще не лимитирует. При большом числе узлов, по нашим предварительным оценкам, из-за большого размера сообщений может лимитировать ПС межсоединения.

Работа поддержана РФФИ, проект 04-07-90220. Автор выражает также благодарность компании «Т-платформы» за предоставленную возможность удаленного доступа к кластеру Infiniband на базе Xeon Nocona.

Литература

1. *Кузьминский М.* Открытые системы, N11 (2005), в печати.
2. //www.lindaspaces.com.

3. *Schmidt M.W*., *Baldridge K.K*., *Boatz J.A*., *et.al*. // J. Comp. Chem. V. 14, (1993). P. 1347.

4. *Кузьминский М*., *Мускатин А*. // Открытые системы, N 7–8 (2001). С. 17.

5. Gaussian 03 Revision C.02, M.J.Frish, G.W.Trucks, H.B.Shlegel et.al., Gaussian, Inc., Wallingford CT, 2004.

6. *Goldenberg D*., *Kagan M*., *Ravid R*., *et. al*. Transparrently Achieving Superior Socket Performance using Zero Copy Socket Direct Protocol over 20 Gb/s Infiniband Links. White paper, Mellanox, Inc, 2005.

7. *Кузьминский М.Б*., *Бобриков В.В*., *Чернецов А.М*., *Шамаева О.Ю*. // Высокопроизводительные параллельные вычисления на кластерных системах. 4-й международ. научно-практический семинар, Самара, 2004. С. 141.

8. *Михайлов Г.М.*, *Копытов М.А.*, *Рогов Ю.П. и др*. Параллельные вычислительные системы в локальной сети ВЦ РАН. М.: Изд-во ВЦ РАН, 2003.

ОПЫТ ПОСТРОЕНИЯ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С УДАЛЕННОЙ ЗАГРУЗКОЙ УЗЛОВ

М.Г. Курносов

Сибирский государственный университет телекоммуникаций и информатики, Новосибирск

1. Введение

В настоящее время разработаны различные технологии построения кластерных вычислительных систем (ВС) большая часть, которых основана на использовании свободно распространяемого программного обеспечения (ПО) и операционной системе GNU/Linux.

Важным вопросом в процессе построения кластерной ВС является выбор способа установки операционной системы на вычислительные узлы. Можно выделит несколько устоявшихся подходов [1]:

1. Системы с полной установкой ОС на узлах ВС (diskfull) – вычислительные узлы располагают носителями информации, на которые устанавливается ОС (ядро и корневая файловая система). Загрузка ОС осуществляется с локального носителя информации.

2. Бессистемная конфигурация узлов (systemless) – вычислительные узлы располагают носителями информации, но используются они лишь как хранилище для временных файлов и/или раздела подкачки. Ядро операционной системы и корневая файловая система загружают-

ся удаленно, с центральной машины кластера.

3. Бездисковая конфигурация узлов (diskless) – на вычислительных узлах отсутствуют носители информации. Ядро операционной системы и корневая файловая система загружаются удаленно, с центральной машины кластера.

Полноценную установку ОС на вычислительные узлы можно считать традиционным подходом.

Существует ряд случаев, когда данный подход не применим на практике, например при построении вычислительной системы на базе машин компьютерной лаборатории, в которых могут отсутствовать жесткие диски либо запрещено производить установку дополнительного ПО на узлы.

Подход с использованием бездисковых вычислительных узлов часто используется как способ уменьшения совокупной стоимости разрабатываемой кластерной ВС, за счет отказа от использования носителей информации на узлах.

Одним из рациональных подходов к организации функционирования бездисковых систем является использования механизма удаленной загрузки узлов с центральной машины кластера.

В данной работе рассматривается опыт построения кластерных вычислительных систем с удаленной загрузкой узлов и поддержкой бездисковых конфигураций.

Описываемый подход обеспечивает быстрое развертывание вычислительного кластера на базе парка стандартных ПК (например, на базе компьютерной лаборатории) с преднастроенной центральной машины. В качестве операционной системы узлов кластера используется ОС Slackware GNU/Linux.

2. Организация процесса удаленной загрузки

Для обеспечения функционирования бездискового узла необходимо осуществить доставку ядра ОС и организовать доступ к корневой файловой системе (ФС), содержащей системное ПО. Загрузка ядра ОС на узлы ВС может быть осуществлена удаленно (например, по технологии PXE или Etherboot) или с локального загрузочного устройства (например, с дискеты или USB-накопителя).

В рассматриваемом подходе удаленная загрузка вычислительных узлов осуществляется при помощи технологии PXE или Etherboot. Задача размещения и получения доступа к корневой ФС решается следующим образом – корневая ФС узла размешается на RAM-диске, который создается в процессе загрузки. Для уменьшения размера исполь-

зуемой памяти, из образа корневой ФС исключается прикладное ПО, каталоги с которым монтируются по NFS с центральной машины кластера. На центральной машине устанавливаются следующие службы:

1. DHCP-сервер – обеспечивает функционирование процесса удаленной загрузки и динамического конфигурирования сетевых интерфейсов узлов;

2. TFTP-сервер – реализует возможность удаленного копирования ядра ОС и образа начального RAM-диска;

3. Служба NFS – обеспечивает доступ к домашним каталогам пользователей и каталогам с ПО. В процессе начальной загрузки обеспечивает доступ к сжатому образу корневой ФС узлов;

4. NTP-сервер – обеспечивает синхронизацию системных часов на узлах ВС с часами центральной машины.

5. Сервер безопасной оболочки sshd – обеспечивает удаленный доступ к центральной машине и узлам ВС.

6. Web-сервер Apache – предоставляет доступ к Web-интерфейсу распределенной системы мониторинга Ganglia.

На рис. 1 изображена общая схема организации процесса удаленной загрузки отдельного узла.

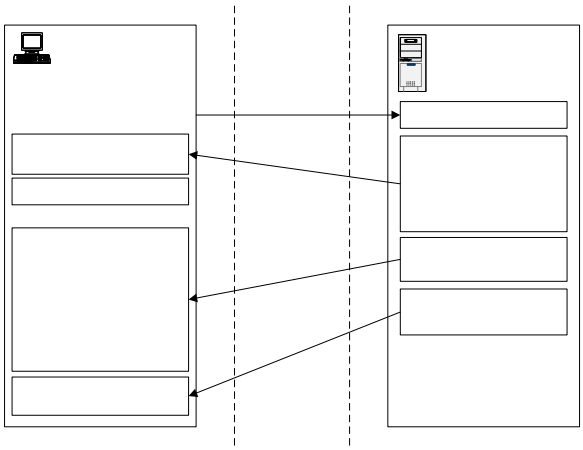


Рис. 1. Схема процесса удаленной загрузки вычислительного узла

В месте с ядром ОС на вычислительные узлы загружается образ начального RAM-диска, с которого осуществляется запуск сценария инициализации, обеспечивающего создание в памяти вычислительного узла RAM-диска, загрузку с центральной машины сжатого образа кор-

невой ФС и размещение полученной ФС на созданном диске.

Инициализация системы продолжается с RAM-диска. В дальнейшем осуществляется синхронизация системных часов и монтирование каталогов с центральной машины для обеспечения доступа к прикладному ПО.

Для уменьшения размера начального RAM-диска в его состав вошел минимально необходимый набор ПО. Стандартные системные утилиты заменены оптимизированными по размеру аналогами из пакета BusyBox.

В случае присутствия в сети двух функционирующих DHCP-серверов клиенты могут получить информацию с IP-параметрами от любого из них. Решением данной проблемы стала модификация DHCP-клиента, используемого узлами в процессе загрузки, что обеспечило возможность игнорировать ответы посторонних DHCP-серверов. Отбрасывание ответов посторонних DHCP-серверов реализовано на базе запроса дополнительных опций протокола DHCP.

3. Опыт внедрения

Описанный подход использован при построении двух экспериментальных кластерных ВС Duron-10 и P4-10. Системы построены на базе существующих компьютерных лабораторий. Вычислительные узлы объединены сетью Fast Ethernet (см. табл.).

Таблица

Параметры кластерных вычислительных систем

№	Кластер	Конфигурация узла	Кол-во узлов	Производитель-ность, Gflops	Латентность, мкс
1	Duron-10	AMD Duron 800 MHz, RAM 128 Mb	10	3.374e+00	102
2	P4-10	Intel Pentium 4 2.0 GHz, RAM 256	10	7.124e+00	81

Оценка производительности развернутых ВС осуществлялась при помощи пакета HPL (High-Performance Linpack) [2] – свободно-распространяемой реализации теста Linpack, являющейся официальным пакетом оценки производительности систем списка TOP500.

Оценка производительности межпроцессорных обменов MPI осуществлялась при помощи набора тестов MPI Benchmark Suite, разрабо-

танных в НИВЦ МГУ [3]. Пакет включает ряд тестов позволяющих оценить латентность и пропускную способность среды передачи данных, а также измерить производительность совместного доступа узлов к NFS-серверу.

На рис. 2 и 3 соответственно приведены графики роста производительности с увеличением количества узлов в ВС Duron-10 и P4-10.

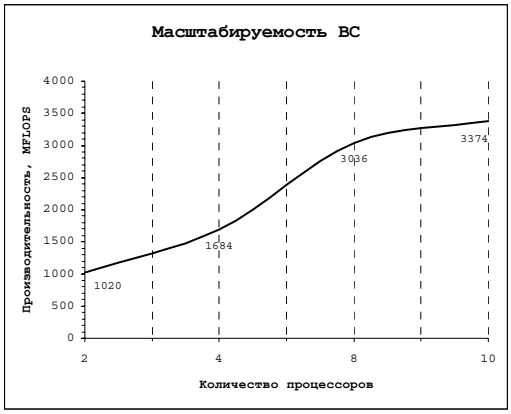


Рис. 2. График роста производительности ВС Duron-10 с увеличением количества узлов

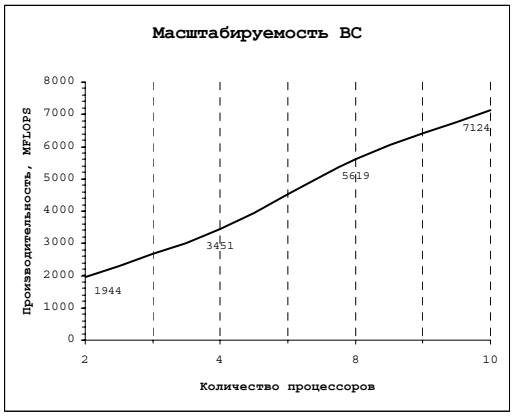


Рис. 3. График роста производительности ВС P4-10 с увеличением количества узлов

3. Заключение

Описанный подход находит применение при построении кластерных ВС с удаленной загрузкой узлов на базе существующих компьютерных лабораторий, а также в случае построения ВС на базе бездисковых рабочих станций.

Отличительной особенностью данного подхода является использование смешанного подхода – размещение корневой файловой системы на RAM-диске и монтирование каталогов с прикладным ПО с центральной машины кластера.

Данное решение позволяет сохранить масштабируемость подхода с использованием RAM-диска и обеспечить приемлемый уровень использования оперативной памяти на узлах ВС.

Модификация DHCP-клиента обеспечила возможность приема IP-параметров от DHCP-сервера центрального узла и игнорирование ответов других серверов на основе запроса дополнительных опций протокола DHCP.

Опыт внедрения показывает, что системы с подобной организацией могут быть успешно использованы для оперативного построения временных и учебных кластерных ВС.

Литература

1. *Benoit des Ligneris*, *Michel Barrette*, *Francis Giraldeau*, *Michel Dagenais*. Thin-OSCAR: Design and future implementation. [Электронный ресурс]: Centre de Calcul Scientifique, Universit´e de Sherbrooke, Quebec, Canada. – Режим доступа: http://thin-oscar.ccs.usherbrooke.ca/, свободный.

2. HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. [Электронный ресурс]: Innovative Computing Laboratory, 2004. – Режим доступа: http://www.netlib.org/benchmark/hpl/, свободный.

3. Система тестов производительности для параллельных компьютеров. [Электронный ресурс]: Информационно-аналитический центр по параллельным вычислениям. – М.: Лаборатория параллельных информационных технологий НИВЦ МГУ. – Режим доступа: http://parallel.ru/testmpi/, свободный.

ИЗМЕРЕНИЕ И ПРОГНОЗИРОВАНИЕ ИЗМЕНЕНИЯ ПАРАМЕТРОВ ЗАГРУЖЕННОСТИ КЛАСТЕРНЫХ СИСТЕМ

В.П. Кутепов, Д.В. Котляров, В.Я. Маркин

Московский энергетический институт

Введение

Проблема измерения параметров загруженности вычислительных систем, в особенности кластеров, чрезвычайно важна [1,2]. Ее состояние подробно обсуждалось в работе [3].

Как показывает анализ, существующие методы и программные средства измерения загруженности компьютеров в различных ОС реализованы по разному, их точность часто требует специальных исследований (достаточно указать на работу [4], в которой весьма критически обсуждаются используемые в ОС методы измерения путем усреднения определенным образом загруженности процессора компьютера).

В действительности, наиболее важными измеряемыми параметрами, по которым можно судить о загруженности компьютеров кластера и самого кластера, являются следующие:

– относительная величина простоя или загруженности процессора,

– количество активных процессов,

– интенсивность обмена между оперативной памятью и жестким диском,

– интенсивность межкомпьютерного обмена,

– объем свободной памяти.

Изменение каждого из этих параметров может происходить непредсказуемым образом и, как правило, определение его значения осуществляется путем усреднения значений на некотором интервале времени, выбор которого определяет точность измерения и существенно зависит от характера (в общем случае частотных характеристик) поведения параметра во времени.

Измерение параметров загруженности

Будем предполагать, что загруженность компьютера определяет он сам, а регулирование загруженности кластера осуществляет сервер (рис. 1):

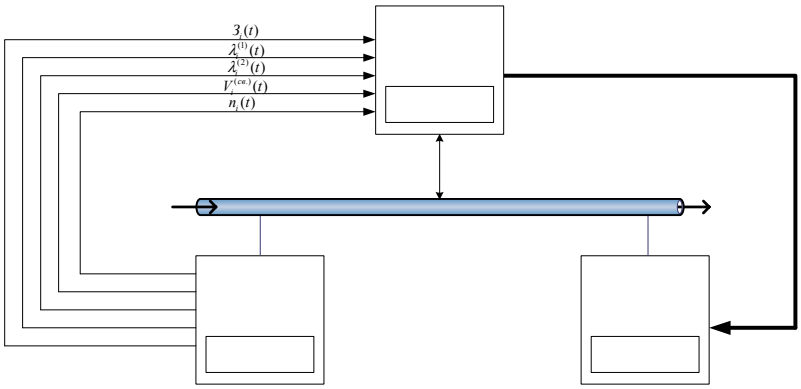


Рис. 1. Схема управления загруженностью кластера

В качестве основных параметров, по которым можно судить о загруженности компьютера кластера, рассматриваются следующие усредненные их значения, определяемые по измерениям (средствами ОС) на определенном интервале времени:

$З_i(t)$ – загруженность процессора (которая, вообще говоря, складывается из двух величин: а) загруженности полезной работой и б) загруженности, связанной с выполнением системных функций)

$n_i(t)$ – количество выполняемых процессов,

$\lambda_i^{(1)}(t)$ – интенсивность обмена ОП/дисковая память,

$\lambda_i^{(2)}(t)$ – интенсивность межкомпьютерного обмена,

$V_i^{(св)}(t)$ – объем свободной памяти.

Введем понятие средней загруженности компьютеров кластера в момент времени t:

$$З(t) = \frac{1}{N}\sum_{i=1}^{N} З_i(t),$$ где N – количество компьютеров, $З_i(t)$– средняя загруженность i-го компьютера, а также определим величину

$$\Delta З_i(t) = \frac{З_i(t) - З(t)}{\max\{З_i(t) \mid i = 1, 2, ..., N\}},$$

характеризующую относительную загруженность i-го компьютера.

Периодически через интервал Δt каждый компьютер посылает серверу значения указанных параметров, по которым сервер вычисляет

от друг
узло

относительную загруженность компьютера кластера, а также относительные значения параметров $n_i(t)$, $\lambda_i^{(1)}$, $\lambda_i^{(2)}$, $V_i^{(св)}(t)$.

Заметим, что величина $\Delta З_i(t)$ – относительная загруженность процессора компьютера позволяет в определенной степени оценивать загруженность безотносительно к «характеру» выполняемой программы (для одной и той же программы для малого числа компьютеров в кластере загрузка будет большой, для большого – малой, однако относительная загрузка может оставаться близкой). Кроме того, по $\Delta З_i(t)$ можно легко упорядочить по степени загруженности все компьютеры кластера.

Вычисляя дисперсию

$$\sigma(t) = \sqrt{\frac{1}{N}(З_i(t) - З(t))^2},$$

можно судить о «разбросе» в загруженности компьютеров (также можно поступить с другими параметрами загруженности). При этом очевидно, что при малой средней загруженности $З(t)$ компьютеров кластера надо либо уменьшать количество компьютеров в кластере, либо увеличивать количество выполняемых программ.

Прогнозирование параметров загруженности

Измерение параметров производится через равные промежутки времени и их значения представляют собой «стационарный» временной ряд, который идентифицируется, как и всякий случайный процесс, следующими основными характеристиками:

– математическим ожиданием,
– дисперсией,
– автокорреляционной функцией,
– спектральной функцией.

Автокорреляционной функцией случайного стационарного процесса $f(t)$ называется функция:

$$r(h) = \frac{\int_{-\infty}^{\infty} f(t) f(t+h)\,dt}{\sigma^2},$$

где σ – дисперсия.

В случае временного ряда это выражение (функция имеет смысл только для натуральных h) принимает вид:

$$r(h) = \frac{1}{\sigma^2} \frac{\sum_{t=1}^{n-h} \xi(t)\xi(t+h)}{n-h}.$$

В случае прогнозирования корреляционная функция даст информацию о том, насколько шагов можно эффективно прогнозировать значение параметра.

Представление временного ряда как суммы случайных гармоник приводит к понятию спектра временного ряда, связанного с корреляционной функцией (косинус-преобразование Фурье):

$$S(\omega) = \frac{2\sigma^2}{\pi} \int_0^{\infty} r(t) \cos \omega t \, dt.$$

В случае временного ряда спектральная функция $S(\omega)$ примет вид:

$$S(\omega) = 1 + 2\sigma^2 \sum_{h=1}^{\infty} r(h) \cos \omega h.$$

Важной характеристикой спектральной функции является эффективная величина спектра:

$$\Delta\omega = \frac{1}{\max\limits_{\omega}(\omega)} \int_{-\infty}^{\infty} S(\omega) \, d\omega = \frac{2\sigma^2}{\max\limits_{\omega}(\omega)}.$$

Мерой эффективной дальности прогнозирования значения случайного процесса является средний интервал корреляционной функции:

$$\Delta\tau = 2 \int_0^{\infty} | r(\tau) | d\tau.$$

Рассмотрим линейный оператор $\Theta(B)$, где B – оператор смещения номера элемента ряда на 1 $Ba_t = a_{t-1}$. Рассмотрим новый временной ряд $x_t = (1 - \alpha)a_t + \Theta(B)x_{t-1}$. Спектр нового ряда получается из спектра старого путем воздействия на него фильтра.

Простейшим примером такого преобразования рядов является метод сглаживания, предложенный в [4]:

$$y_t = y_{t-1} + \alpha(x_t - y_{t-1}) = (1 - \alpha)y_{t-1} + \alpha x_t$$

Данный метод был получен автором работы [4] из электротехнических соображений, в теории временных рядов этот метод называется

прогнозом с экспоненциально взвешенным скользящим средним и является рекуррентной записью усреднения всех членов ряда с экспоненциальными коэффициентами:

$$y_t = (1 - \alpha)(x_t + \alpha x_{t-1} \ldots, \alpha^n x_{t-n} + \ldots).$$

На практике имеет смысл работать не с самими значениями загруженности процессора, а с их разностями. Дискретный оператор разности является аналогом производной первого порядка для непрерывных функций. Преимущества такого подхода заключается в том, что, во-первых, разностная величина уже центрирована, во-вторых, симметрична, и, в-третьих, с ее помощью лучше видна случайная составляющая процесса. Используя разностную схему, можно добиться лучших результатов прогнозирования.

Заключение

На основе проведенных экспериментальных исследований в рамках созданной системы граф-схемного потокового параллельного программирования [2] нами разработаны алгоритмы и программные модули измерения указанных параметров загруженности компьютеров кластера и прогнозирования их изменения. Также разработан модуль для управления загруженностью кластера в целом. В настоящее время программные средства проходят экспериментальную проверку.

Литература

1. *Кутепов В.П.* Об интеллектуальных компьютерах и больших компьютерных системах нового поколения. М: Из-во РАН, Теория и системы управления, 1996, №5.

2. *Котляров Д.В.*, *Кутепов В.П.*, *Осипов М.А.*, Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах. М: Из-во РАН, Теория и системы управления, 2005, №1.

3. *Кутепов В.П.*, *Котляров Д.В*. Управление загруженностью кластерных вычислительных систем. Матер. IV Международ. научно-практического семинара и Всероссийской молодежной школы, Изд-во СНЦ РАН, 2004.

4. *Dr. Neil Gunther*, UNIX Load Average Part 1: How It Works. TeamQuest Corporation, 2005.

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ КАК WEB-СЕРВИС ПЛАТФОРМЫ MICROSOFT . NET

Д.Ю. Лабутин

Нижегородский государственный университет им. Н.И. Лобачевского

Удаленный доступ

Как правило, удаленный доступ к Linux-кластерам предоставляется с помощью SSH (Secure Shell) с интерфейсом командной строки. Рассмотрим положительные и отрицательные стороны данного подхода.

Основным достоинством данного подхода является то, что большинство вспомогательных утилит для управления и мониторингом кластера, разработанных или адаптированных под особенности конкретной архитектуры вычислительного комплекса администраторами, становятся доступными пользователям. Благодаря этому в распоряжении пользователя имеется богатый инструментарий, облегчающий его работу. Например, это могут быть утилиты пакетного запуска заданий. Так же администраторы кластера, часто используют следующий «трюк» – подменяют стандартные утилиты на свои собственные, т.о. их использование для конечного пользователя становится прозрачным. Например, если на кластере для написания параллельных программ используется библиотека MPI, то стандартная команда mpirun может быть заменена собственной версией, которая задачу не отправляет непосредственно на исполнение, а помещает в очередь задач.

Недостатком такого подхода является не очень дружественная среда для пользователя. Ему необходимо помнить названия команд, их параметры и т.п. Да и работа с командной строкой в эпоху графических приложений является непривычной.

Можно разработать графическое приложение, которое будет выполняться на стороне клиента и обеспечит ему удобную среду работы, транслируя действия пользователя в команды запуска тех или иных утилит на удаленном кластере. Наряду с очевидными плюсами в данном решении есть два недостатка. Первый – это проблемы с возможными брандмауэрами, установленными на стороне клиента, которые могут существенно затруднить механизм создания удаленного соединения между графическим приложением и кластером. Второй – это необходимость установки дополнительного программного обеспечения на стороне клиента. Проблему могут создать следующие ситуации: у пользователя установлена операционная система, для которой не суще-

ствует реализации графической среды работы с кластером; пользователь часто работает с кластером с разных компьютеров, на которых не возможности каждый раз устанавливать нужное приложение.

В качестве решения описанных проблем, при реализации системы удаленного доступа к вычислительному кластеру ННГУ, было принято решение, что графическим клиентом на стороне пользователя будет стандартный Интернет браузер (Web-интерфейс), который существует в любой операционной системе.

Очевидно, что для тех, кто пользуется кластером регулярно, вероятнее всего удобней будет пользоваться интерфейсом командной строки, т.к. это в большинстве случаев ускоряет процедуру подготовки задания и размещения его в очереди задач. А для тех пользователей, кто пользуется кластером нечасто – идеальным будет именно Web-доступ с интуитивно-понятным интерфейсом, т.к. для его использования нет необходимости изучать документацию и выяснять какие именно команды и с какими параметрами нужно запускать для выполнения тех или иных действий.

Для создания возможности разработки различных типов клиентов для системы управления кластером, был выделен модуль, представляющий собой точку входа в систему (*Диспетчер доступа*). Данный модуль осуществляет прием команд от пользовательской программы – клиента, их передачу соответствующим модулям системы и возврат клиенту результатов выполнения команд. Это единственный модуль системы, непосредственно взаимодействующий с пользователем, и этот модуль может быть выполнен в виде Web-сервиса.

Диспетчер доступа на основе Web-сервиса

В последнее время механизм Web-сервисов становится все более популярным. Практически во всех языках программирования, даже тех, которые не ориентированы под Microsoft .NET Framework, имеются библиотеки, необходимы для разработки приложений, использующих функциональность Web-сервисов. Предоставляя в качестве внешнего открытого интерфейса функциональность Web-сервиса диспетчера доступа, открывается возможность разработки клиентов системы под любые платформы не только разработчиками системы, но любому желающему. Кроме того, популяризация Web-сервисов ведет к новому подъему интереса к GRID-технологиям.

Так как диспетчер доступа является единственной точкой взаимодействия с пользователем (клиентским приложением), то создается впечатление, что вся функциональность системы управления класте-

ром заложена именно в нем. Но, как было отмечено выше, на самом деле часть команд просто передается другим модулям системы. Естественно, что подобная передача осуществляется только в том случае, если пользователь предварительно прошел аутентификацию.

Основные функциональные возможности менеджера доступа следующие:

- *Аутентификация пользователей* при работе с системой управления вычислительным кластером. Каждый пользовать в начале работы должен ввести «Имя пользователя» и «пароль», известные только ему, которые передаются методу аутентификации, который создает сессию для взаимодействия с клиентом. После этого можно однозначно сопоставлять производимые действия на кластере с конкретным пользователем.
- *Выделение независимого файлового пространства*. Каждому пользователю выделяется индивидуальное место для хранения файлов, необходимых для проведения вычислительных экспериментов. При этом накладываются определенные ограничения на хранимые файлы (максимальный суммарный размер хранимых файлов, максимальный размер каждого файла в отдельности), индивидуальные для каждого пользователя системы.
- *Система задач*. После того как пользователь загрузил необходимые файлы для проведения вычислительных экспериментов, он создает именованные задачи, т.е. задает необходимые параметры (имя задачи, исполняемый модуль, приоритет, имя файла с результатами, необходимое количество вычислительных мощностей, параметры командной строки для исполняемого модуля). Затем пользователь может ставить задачу в очередь на исполнения неоднократно, не утруждая себя повторным заданием параметров. В случае необходимости владелец задачи может снять ее с исполнения или убрать из очереди.
- *Мониторинг*. После того, как одна или более задач поставлены в очередь на исполнение, пользователь может посмотреть их состояние (выполняется, ожидает выполнения в очереди, выполнилась). Так доступна информация о загруженности вычислительного кластера в целом.
- *Протоколирование действий*. Все производимые пользователем системы действия протоколируются. Анализируя протокол можно определить, какие системные ресурсы выделяются тому или иному пользователю.
- *Административный интерфейс*. Так же имеется интерфейс ра-

боты с системой, предназначенный только для администраторов, где они могут производить такие действия, как добавление и удаление пользователей системы, изменение параметров, задающих ограничения на хранимые файлы, контроль хранимых пользователями файлов и т.п.

Стоит отметить, что функциональность мониторинга и системы задач является трансляцией команд и результатов их исполнения между клиентом пользователя и отдельными модулями системы управления кластером, отвечающими за выполнение этих команд.

Перечислим основные преимущества используемого подхода:

- Легко разрабатывать клиентские приложения, как с интерфейсом командной строки, так и графические приложений,
- Благодаря открытости интерфейса Web-сервиса, дополнительные утилиты могут разрабатывать сторонние разработчики,
- Использование в качестве клиента связки Web-браузера и Web-сервера позволяет пользователю работать удаленно с кластером с любого компьютера, подключенного к Интернету

Литература

1. Programming .NET Web Services by Alex Ferrara, Matthew MacDonald Publisher: O'Reilly; 1 edition (October 15, 2002) ISBN: 0596002505

2. Service-Oriented Architecture : A Field Guide to Integrating XML and Web Services by Thomas Erl Publisher: Prentice Hall PTR (April 16, 2004) ISBN: 0131428985

3. Understanding Web Services: XML, WSDL, SOAP, and UDDI by Eric Newcomer Publisher: Addison-Wesley Professional; 1st edition (May 13, 2002) ISBN: 0201750813

4. From P2P to Web Services and Grids: Peers in a Client/Server World by Ian J. Taylor Publisher: Springer; 1 edition (October 21, 2004) ISBN: 1852338695

ПРОГРАММНАЯ СИСТЕМА ДЛЯ ИЗУЧЕНИЯ И ИССЛЕДОВАНИЯ ПАРАЛЛЕЛЬНЫХ МЕТОДОВ РЕШЕНИЯ СЛОЖНЫХ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ

В.П. Гергель, Е.А. Козинов, Д.Ю. Лабутин, А.А. Лабутина

Нижегородский государственный университет им. Н.И.Лобачевского

В докладе представлен программный комплекс ПараЛаб, который отвечает следующим основным требованиям: он позволяет проводить как реальные параллельные вычисления на многопроцессорной вычис-

лительной системе, так и имитировать такие эксперименты на одном последовательном компьютере с визуализацией процесса решения сложной вычислительной задачи.

При проведении имитационных экспериментов, предоставлена возможность для пользователя:

- *определить топологию* параллельной вычислительной системы для проведения экспериментов, *задать число процессоров* в этой топологии, *установить производительность* процессоров, выбрать *характеристики коммуникационной среды* и *способ коммуникации*;
- *осуществить постановку вычислительной задачи*, для которой в составе системы ПараЛаб имеются реализованные параллельные алгоритмы решения, *выполнить задание параметров* задачи;
- *выбрать параллельный метод* для решения выбранной задачи;
- *установить параметры визуализации* для выбора желаемого темпа демонстрации, способа отображения пересылаемых между процессорами данных, степени детальности визуализации выполняемых параллельных вычислений;
- *выполнить эксперимент* для параллельного решения выбранной задачи; при этом в системе ПараЛаб предусмотрена возможность сформировать несколько различных *заданий для проведения экспериментов* с отличающимися типами многопроцессорных систем, задач или методов параллельных вычислений, для которых выполнение эксперимента может происходить одновременно (в режиме разделения времени); одновременное выполнение эксперимента для нескольких заданий позволяет наглядно сравнивать динамику решения задачи различными методами, на разных топологиях, с разными параметрами исходной задачи. Для организации анализа полученных данных при проведении длительных вычислений, система обеспечивает возможность проведения серии экспериментов в автоматическом режиме с запоминанием результатов в *журнале экспериментов*;
- *накапливать и анализировать результаты выполненных экспериментов*; по запомненным результатам система позволяет построить графики зависимостей, характеризующих параллельные вычисления (*времени решения*, *ускорения*, *эффективности*) от параметров задачи и вычислительной системы.

Одной из важнейших характеристик системы является возможность выбора способов проведения экспериментов. Эксперимент может быть выполнен в режиме имитации, т.е. проведен на одном процессоре без использования каких – либо специальных программных

средств типа библиотек передачи сообщений. Кроме того, в рамках системы ПараЛаб должна существует возможность проведения вычислительного эксперимента в режиме удаленного доступа к вычислительному кластеру Нижегородского государственного университета.

Разработка системы на нескольких языках

Изначально система ПараЛаб разрабатывалась на языке C++ в среде Borland C Builder, поскольку на тот момент эта система наиболее подходила для разработки приложений с развитым пользовательским интерфейсом.

В настоящее время ведется параллельная разработка системы ПараЛаб на языке Java с тем, чтобы реализуемые системой ПараЛаб процессы изучения и исследования параллельных алгоритмов решения сложных вычислительных задач были доступны пользователям Интернет посредством технологии Java апплетов.

Кроме того, начата разработка ПараЛаб на языке C# на основе платформы Microsoft .NET. Данная система предоставляет развитые возможности для параллельной разработки программных систем несколькими специалистами на разных языках из числа тех, которые поддерживает .NET. Кроме того, данная технология используется в той компоненте системы управления кластером ННГУ, которая обеспечивает взаимодействие с внешними пользователями и предоставляет удаленный доступ. Поскольку ПараЛаб выступает в качестве внешнего пользователя, происходит «бесшовная» стыковка двух программных систем.

Моделирование параллельных процессов

В рамках системы ПараЛаб параллельная программа представляется набором процессов, которые выполняются в режиме разделения времени на одном последовательном компьютере. Каждый из этих процессов в любой момент времени обладает данными, которые хранятся в специально отведенной области памяти и отображаются рядом с соответствующим процессором в области выполнения эксперимента. Модельное время выполнения локальных вычислений есть произведением количества операций и времени выполнения одной операции. Для оценки коммуникационной составляющей времени выполнения алгоритма используется модель Хокни операции передачи данных.

Основной набор параметров, описывающих время передачи данных, состоит из следующего ряда величин:

- *время начальной подготовки* (α) характеризует длительность

подготовки сообщения для передачи, поиска маршрута в сети и т.п.;

- *время передачи служебных данных* (t_c) между двумя соседними процессорами (т.е. для процессоров, между которыми имеется физический канал передачи данных); к служебным данным может относиться заголовок сообщения, блок данных для обнаружения ошибок передачи и т.п.;
- *время передачи одного слова данных по одному каналу передачи данных* ($1/\beta$); длительность подобной передачи определяется полосой пропускания коммуникационных каналов в сети.

Время пересылки данных объема m по маршруту длиной l определяется выражением

$$t_{пд} = \alpha + \left(\frac{m}{\beta} + t_c \right) l. .$$

При достаточно длинных сообщениях временем передачи служебных данных можно пренебречь и выражение для времени передачи данных может быть записано в более простом виде

$$t_{пд} = \alpha + \frac{m}{\beta} l \ .$$

Были разработаны и протестированы на высокопроизводительном кластере ННГУ реальные параллельные программы, которые решают все задачи, входящие в состав системы ПараЛаб всеми представленными методами.

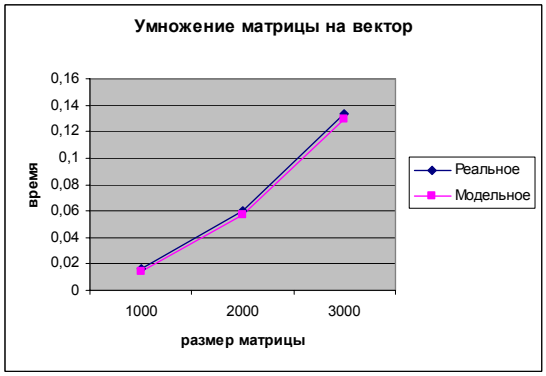


Рис. 1. Сравнение реального и модельного времени выполнения араллельного алгоритма умножения матрицы на вектор на 4 процессорах

Это работа была проделана для того, чтобы обосновать применимость математических моделей вычислений и коммуникации для оценки времени выполнения алгоритма. В результате проведенных экспериментов выяснилось, что описанная математическая модель адекватно отображает реальность.

Расширяемость системы

ПараЛаб – программная система предназначена для иллюстрирования технологии параллельного программирования на сравнительно несложных задачах, и относится, главным образом, к программированию «в малом». Типичными примерами такого рода задач служат программы для реализации отдельных алгоритмов сортировки, матричных операций, обработки графов, алгоритмов из области линейной алгебры.

В настоящее время приоритетным направлением развития системы является реализация возможности добавления в систему новых алгоритмов решения тех задач, которые включены в состав ПараЛаб. Для того, чтобы предоставить пользователю возможность самому сконструировать параллельный метод решения той или иной задачи, необходимо выделить все возможные конструктивные элементы, из которых может состоять алгоритм, и обеспечить возможность соединять эти элементы в корректную последовательность, приводящую к решению задачи. Так, например, для реализации операции матричного умножения ленточным методом, необходимо наличие всего двух конструктивных элементов: локальной операции скалярного умножения векторов и операции передачи данных следующему процессору в структуре сети.

В данный момент можно говорить о расширяемости кластерной компоненты системы ПараЛаб. Для того, чтобы обеспечить возможность работы системы с новым алгоритмом, необходимо поместить соответствующую программу на кластер. Далее запустить эту программу из системы посредством удаленного доступа к кластеру. Таким образом, можно получить представление о динамике решения задачи с помощью данного алгоритма и проанализировать результаты вычислительных экспериментов при помощи компоненты накопления результатов.

Очевидно, что разработка параллельных программ практического уровня сложности представляет собою многоэтапный технологический процесс и не может быть продемонстрирована во всей своей полноте на таких задачах. Говоря о технологии программирования, мы подразумеваем все этапы разработки параллельной программы, начиная с анализа задачи, выбора модели программы, декомпозиции задачи на

параллельные процессы и заканчивая вопросами анализа производительности и организации вычислительного эксперимента. Однако не следует забывать, что без освоения основ параллельного программирования, без глубокого понимания процессов, протекающих при параллельном решении задачи на многопроцессорной вычислительной системе самых простых задач, невозможно приступить к разработке сложных параллельных программных комплексов.

Литература

1. *Hockney R.W.*, *Jesshope C.R*. (1988). Parallel Computers 2. Architecture, Programming and Algorithms. - Adam Hilger, Bristol and Philadelphia. (русский перевод 1 издания: Хокни Р., Джессхоуп К. Параллельные ЭВМ. Архитектура, программирование и алгоритмы. - М.: Радио и связь, 1986).

2. *Kumar V.*, *Grama A.*, *Gupta A.*, *Karypis G.* Introduction to Parallel Computing. – The Benjamin/Cummings Publishing Company, Inc., 1994.

3. *Quinn M.J.* (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.

4. *Гергель В.П.*, *Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных машин: Учеб. пособие – Н. Новгород: Изд-во ННГУ, 2000.

МОНИТОРИНГ МУЛЬТИПЛАТФОРМЕННЫХ УЗЛОВ КЛАСТЕРА

И.В. Лопатин

Нижегородский государственный университет им. Н.Лобачевского

В связи с планируемым расширением кластера Нижегородского государственного Университета за счет узлов, работающих под управлением ОС Linux, возникла задача расширения текущей системы управления кластером. В настоящий момент программный комплекс «Метакластер» ориентирован на работу в среде с узлами на которых запущена ОС семейства Windows (Windows 2000 и выше).

Одной из проблем, которые возникают при расширении системы управления кластером, является задача мониторинга узлов. В текущем варианте программы используется удаленный контроль за состоянием компьютеров с использованием стандартных средств, интегрированных в ОС семейства Windows NT. Windows Management Interface (WMI, интерфейс управления Windows) реализует поддержку мониторинга без необходимости установки дополнительного программного

обеспечения на узлах кластера.

В большинстве вариантов ОС Linux, представленных на рынке, отсутствует соответствующая реализация WBEM, что препятствует разработке удаленной системы мониторинга аналогично Windows-версии . В результате был разработан прототип агента, который расположен на узле и запускается в виде демона (или сервиса в терминологии Windows). Агент ожидает поступления удаленного запроса, после чего формирует информацию о текущем состоянии узла и передает ее удаленной стороне. Возможно протоколирование поступающих запросов и передаваемой информации для отладочных целей и сбора статистики. Со стороны центрального узла, который запрашивает информацию, необходимо реализовать механизм прозрачности мониторинга – то есть, обработка должна происходить единым образом как для Windows, так и для Linux-систем. Вместе с тем, в структуре данных для хранения информации об узлах кластера необходимо ввести специальную метку для каждого компьютера, показывающую запущенную на нем в данный момент ОС (это необходимо для корректной работы планировщика заданий и остальных компонент системы управления).

Одним из интересных направлений в исследовании проблемы мониторинга и управления многоплатформенными кластерами является применение открытой реализации WBEM OpenWBEM [1], созданной для управления компьютерами на множестве платформ, включая различные версии Linux, Sun Solaris и Mac OS. К недостаткам применения OpenWBEM можно отнести то, что она не является встроенной в ОС, как в случае WMI на Windows, таким образом, OpenWBEM придется устанавливать и конфигурировать на каждом узле кластера. Также OpenWBEM и WMI несовместимы по интерфейсам, что заставляет писать отдельные реализации обработчика информации на центральном узле для Windows и Linux.

Другим вариантом решения задачи мониторинга может стать использование открытых реализаций WBEM на языке Java, что обеспечивает кроссплатформенность и единый интерфейс как на клиентской так и на серверной стороне. Примерами таких систем могут служить SNIA CIMOM и WBEM Services [2], разрабатываемые как проекты с открытым исходным кодом.

Литература

1. http://www.openwbem.org OpenWBEM project home page.
2. http://wbemservices.sourceforge.net/ WBEM Services project.

ОПЫТ ИСПОЛЬЗОВАНИЯ КЛАСТЕРА ВЦ РАН В ОБРАЗОВАТЕЛЬНЫХ ЦЕЛЯХ[5]

Г.М. Михайлов, Н.Н. Оленев, А.А. Петров, Ю.П. Рогов, А.М. Чернецов

Вычислительный центр им. А.А. Дородницына Российской академии наук (*ВЦ РАН*), *Москва*

С момента ввода в эксплуатацию в 2003 году вычислительный кластер Вычислительного центра имени А.А. Дородницына Российской академии наук (ВЦ РАН) [1] используется не только для проведения научных расчетов, но и в образовательных целях [2]. Кластер ВЦ РАН (http://www.ccas.ru/acluster/main.htm) с суммарной оперативной памятью в 32 GB и с суммарной памятью на жестком накопителе в 288 GB содержит восемь вычислительных узлов, связанных вычислительной сетью Myrinet 2000 (http://www.myri.com) и управляющей сетью Ethernet 100Base-T. Вычислительные узлы кластера содержат по два процессора Intel Xeon DP 2600 MHz, 512 Kb cache. Управляющий узел кластера имеет процессор Intel Pentium 4 2260 MHz, 512 Kb cache, 533 MHz. На кластере установлена операционная система Linux RedHat 9, различные библиотеки взаимодействий API GM-2; MPICH-GM v.1.2.5-10, библиотеки высокого уровня ATLAS 3.6.0, Intel MKL 7.2.1 и параллельная библиотека ScaLapack. На кластере используется пакетная система очередей OpenPBS (http://www.openpbs.org), входящая в состав проекта OSCAR v 3.0 (http://sourceforge.net/projects/oscar/). Для работы студентов в системе очередей выделена специальная очередь shortest (время выполнения до 1 часа), при этом реализована политика выполнения в первую очередь задач с наименьшим временем выполнения.

Основываясь на характеристиках имеющегося в ВЦ РАН вычислительного кластера, был специально разработан годовой курс обучения «Параллельное программирование в интерфейсе MPI», который излагается студентам 5 курса Московского физико-технического института (государственного университета), обучающимся на базовой кафедре «Математическое моделирование сложных процессов и систем» в ВЦ РАН. Курс рассчитан на начинающего пользователя Message Passing

[5] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (код проекта 04-07-90346).

Interface – интерфейса передачи сообщений MPI – студента или специалиста, не знакомого с методами программирования для параллельных ЭВМ, однако владеющего последовательным программированием на языке программирования C в среде Windows или Linux. В настоящее время в локальной сети ВЦ РАН помещен рабочий вариант полной электронной версии курса, который непрерывно совершенствуется. Некоторые из работ появляются в открытом доступе [2].

С одной стороны, данный курс MPI следует международным образцам достаточно полного изложения MPI, которые даны в западных учебных курсах, например, в курсе Cornell Theory Center (CTC) [3]. С другой стороны, данный курс MPI специально предназначен для специалистов в математическом моделировании сложных процессов и систем. Поэтому к стандартным упражнениям апробированных учебных курсов здесь добавлены упражнения, связанные с применением параллельных вычислений в математическом моделировании. Курс состоит из двух частей: первая часть предназначена для начинающих использовать MPI и содержит восемь лабораторных работ по темам: основы программирования в MPI, попарный и коллективный обмен сообщениями, управление группами и коммуникаторами, производные типы данных и устойчивые запросы связи в MPI, - а вторая часть предназначена для совершенствующихся в его использовании и содержит шесть лабораторных работ: виртуальная топология и топология графа, параллельные математические библиотеки, определяемые пользователем операции приведения, зондирование сообщений и тесты исполнения, – и индивидуальную курсовую работу по решению практической проблемы. В основе первой части лежит курс MPI от CTC, эволюционно совершенствующийся на основе работ ВЦ РАН по мере получения новых результатов в области параллельных вычислений. Например, в первой лабораторной работе (см. [4]) домашнее задание включает составление параллельной версии программы идентификации параметров блока «Производство» в модели экономики России переходного периода (см. [5]).

Изложение курса MPI в виде цикла лабораторных работ идеально подходит для студентов, обучающихся по системе Физтеха, в которой свободное посещение занятий сочетается с обязательным выполнением заданий для самостоятельной работы и сдачей заданий к назначенному сроку. Кроме того, электронная версия курса может быть использована в системе открытого дистанционного образования. Необходимые предварительные требования к подготовке студентов включают знание ос-

нов программирования на языке C или C++ , а также знание операционных систем Linux и Windows NT на уровне пользователя.

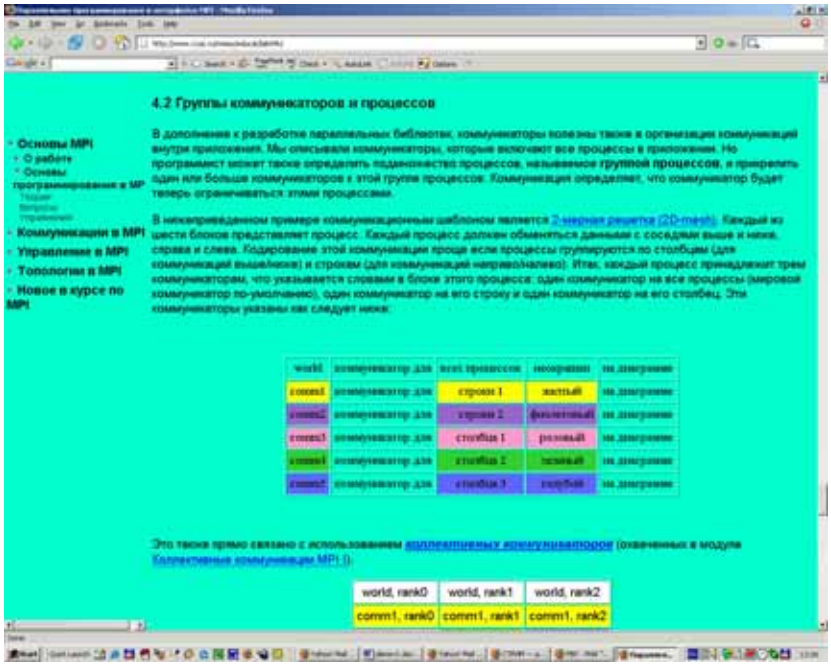


Рис.1. Пример вызова электронной версии курса [2]

Каждая лабораторная работа в курсе представляет собой взаимосвязанный модуль, состоящий из трех частей. В первой части лабораторной работы излагаются необходимые для выполнения заданий теоретические сведения, во второй части приведены контрольные вопросы, проверяющие усвоение теоретического материала, а в третьей, заключительной части, предложены упражнения. Часть упражнений предлагается выполнить в качестве домашнего задания. Предполагается последовательное освоение каждого модуля: вначале следует изучить теорию, затем с помощью контрольных вопросов проверить ваше понимание и, наконец, выполнить практические задания – упражнения. Заметим, что в электронной версии работ исходные файлы в упражнениях можно свободно скачать, что освобождает значительное время от рутинной работы.

Студентам МФТИ в осеннем семестре требуется изучить первые восемь лабораторных работ-модулей, выполнив и сдав практические задания, помещенные в заключительной части каждого модуля. На изучение каждого модуля, как правило, отводится две недели. До конца второй недели следует по электронной почте выслать преподавателю решение домашних заданий лабораторной работы и сдать в очном режиме классные задания. При этом решение заданий по каждой лабораторной работе следует отправить отдельным электронным письмом, а в начале темы письма следует указать номер группы. Крайний срок отправки решений, после которого могут возникнуть трудности с получением зачета в срок: начало девятой, восьмой, седьмой, шестой, пятой, четвертой, третьей и второй недели до Нового года соответственно для 1–8 модуля.

Выполнение лабораторной работы № 1 «Основы программирования в MPI», на изучение и сдачу которой студентам предоставлено четыре недели, дает допуск к выполнению всех последующих работ.

Заметим, что практические упражнения могут быть выполнены не только на кластере ВЦ РАН, но и на виртуальной параллельной машине, устанавливаемой в Windows NT/2000 на вашем компьютере. Для этого перед началом работы вам следует установить переносимую реализацию MPI – MPICH для Windows – на ваш компьютер (http://www-unix.mcs.anl.gov/mpi/mpich/). Предварительно следует изучить руководство по установке MPICH (http://www.cluster.bsu.by/mpich_install.pdf) и руководство пользователя MPICH (http://www.cluster.bsu.by/mpich_userguide.pdf). Кроме того, полезно изучить руководство пользователя МВС 1000М Межведомственного суперкомпьютерного центра (http://www.jscc.ru/), с тем, чтобы вы знали ответы на следующие вопросы:

- компиляция при использовании библиотек MPI;
- использование команды mpirun;
- создание файла;
- понимание механизма выполнения MPI на двух узлах.

Именно необходимость предварительной подготовки к курсу MPI увеличивает общее время выполнения лабораторной работы № 1 примерно в два раза.

Студентам МФТИ в весеннем семестре требуется сдать шесть лабораторных работ и одну индивидуальную курсовую работу.

Для выполнения индивидуальной курсовой работы студенту в начале семестра необходимо получить у своего научного руководителя

тему работы, которая могла бы быть использована как часть магистерской диссертации. Требования к теме: должно существовать или быть близко к завершению решение задачи в виде последовательной версии программы на языке С или С++. При отсутствии задачи на распараллеливание у научного руководителя можно взять тему курсовой работы у преподавателя. Крайний срок отправки решений, после которого будут трудности с получением в срок дифференцированного зачета: начало восьмой, седьмой, шестой, пятой, четвертой, третьей и второй недели до 1 июня, соответственно, для 9-14 модуля и курсовой работы.

Распечатки ответов на контрольные вопросы по всем модулям оказываются востребованными при получении дифференцированного зачета. Итоговая оценка складывается из трех частей: 30% – оценка за осенний семестр, 30% – оценка за весенний семестр и 40% – оценка за курсовую работу.

Изложение данного курса MPI ограничено использованием языка С. Пользователи Фортрана могут изучить его самостоятельно, основываясь на пособиях [6-9] или на знании MPI для С/С++.

Литература

1. *Михайлов Г.М., Копытов М.А.*, *Рогов Ю.П.*, *Чернецов А.М.*, *Аветисян А.И.*, *Самоваров О.И.* Вычислительный кластер ВЦ РАН // Высокопроизводительные параллельные вычисления на кластерных системах: Матер. IV Международ. научно-практического семинара и Всероссийской молодежной школы / Под ред. чл.-корр. РАН В.А. Сойфера. Самара. 2004. С. 193–199.

2. *Оленев Н.Н.* Параллельное программирование в интерфейсе MPI // Сб. лабораторных работ, ВЦ РАН. 2003–2004 (электронная версия на правах рукописи): http://www.ccas.ru/mmes/educat/lab04k/.

3. Cornell Theory Center Topics: Message Passing Interface. http://www.tc.cornell.edu/ctc-Main/services/education/topics/.

4. *Оленев Н.Н.* Основы параллельного программирования в системе MPI. М.: ВЦ РАН, 2005. 80 с.

5. *Оленев Н.Н.* Параллельные вычисления для идентификации параметров в моделях экономики // Высокопроизводительные параллельные вычисления на кластерных системах. Матер. IV Международ. научно-практического семинара и Всероссийской молодежной школы / Под ред. чл.-корр. РАН В.А. Сойфера. Самара. 2004. С. 204–209.

6. *Немнюгин С.А.*, *Стесик О.Л.* Параллельное программирование для многопроцессорных вычислительных систем. – СПб.: БХВ-Петербург, 2002. – 400 с.

7. Домашняя страница MPI в Argonne National Labs http://www-

unix.mcs.anl.gov/mpi/.
8. Часто задаваемые вопросы по MPI http://www.faqs.org/faqs/mpi-faq/.
9. Практическое программирование в MPI на RS/6000: SP: http://www.redbooks.ibm.com/abstracts/sg245380.html?Open

ЦИРКУЛЯНТНЫЕ СЕТИ СВЯЗИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ: СТРУКТУРЫ И ОБМЕНЫ

Э.А. Монахова

ИВМ и МГ СО РАН, Новосибирск

Введение

Рассматриваются проблемы оптимизации структуры и организации обменов в циркулянтных сетях, используемых в качестве сетей связи вычислительных систем. Циркулянтные сети (графы) [1-4] широко изучаются при проектировании и анализе вычислительных систем, в качестве топологий компьютерных систем и сетей. Эти графы реализованы как коммуникационные сети в системах Intel Paragon, Cray T3D, MPP, SONET и др. В настоящее время расширяются возможности практического применения циркулянтных сетей. В модели «малого мира» (small-world networks) [5] базовую основу структуры составляют циркулянтные сети. В работе [6] изучается мультикластерная вычислительная система, построенная из копий циркулянтных сетей.

Пусть $1 \le s_1 \le \ldots \le s_n \le \lfloor N/2 \rfloor$ – множество целых чисел (образующих). *Циркулянтный граф* размерности n, обозначаемый $(N; s_1,\ldots, s_n)$, имеет множество вершин $V=\{0,1, \ldots, N-1\}$ и множество ребер $A = \{(v, v \pm s_i \mod N) \mid v \in V, i=1,\ldots,n\}$. Диаметр графа

$$d(N;S) = \max_{u,v \in V} D(u,v),$$

где $D(u,v)$ – длина кратчайшего пути между вершинами u и v. Пусть d – натуральное число, $S = \{1, s_2, \ldots, s_n\}$ – множество образующих, а $m(d,S) = \max\{N \mid d(N;S) \le d\}$. Для любых натуральных d и n определим точную верхнюю границу $M(d, n) = \max\{m(d, S) \mid \exists$ множество S с $|S| = n\}$, где $|S|$ – мощность S.

Эффективность алгоритмов основных схем обменов данными при параллельных вычислениях непосредственно зависит от топологии

сети связи, в том числе определяется диаметром графа межпроцессорных связей (максимумом минимальных расстояний между каждой парой процессоров). Наиболее исследуемые проблемы оптимизации циркулянтных сетей – получение оптимальных графов с минимальным диаметром для заданного числа вершин и нахождение семейств оптимальных графов с максимальным числом вершин для любого диаметра. Оптимальные сети имеют высокие показатели отказоустойчивости и скорости коммуникаций, максимальную надежность и минимальную задержку при передаче информации [2, 3]. Циркулянтные графы размерностей 2 и 3 являются наиболее изучаемыми циркулянтами благодаря их практическому применению. В [7] получены трехмерные оптимальные циркулянтные сети с аналитическим описанием, которые имеют максимальное число вершин для любого диаметра.

При использовании циркулянтных графов в качестве коммуникационных сетей требуется эффективное решение проблем организации парных и коллективных обменов данными. При парной маршрутизации сообщение должно быть передано из вершины-источника в вершину-приемник. При трансляционном обмене (broadcasting) процессор-источник посылает идентичные данные в сообщении всем другим процессорам системы. Рассматриваемые типы обменов изучаются при различных коммуникационных моделях. В этой работе рассмотрим модель полнодуплексной ненаправленной передачи информации методом коммутации сообщений. В такой модели протокол состоит из последовательности шагов, и в течение каждого шага каждый процессор может посылать (и получать) сообщения от всех своих соседей. Для организации парных обменов требуется определение кратчайших путей в графе. Известны различные распределенные алгоритмы поиска кратчайших путей в двумерных циркулянтных сетях [4, 8–10] и трехмерных циркулянтах с $N = O(d^2)$ [4, 11]. Получены алгоритмы трансляционного обмена для двумерных циркулянтов [4, 9] и трехмерных циркулянтов с $N = O(d^2)$ [4]. В работе [5] даны оценки времени трансляционного обмена для циркулянтов вида $(N; 1, 2, ..., \delta/2)$ при условии, что вершина может обмениваться сообщениями с $k \le \delta$ соседями в течение каждого шага.

В докладе представлено семейство трехмерных оптимальных циркулянтных сетей с максимальным числом вершин для любого диаметра и аналитическим описанием. Для этих сетей разработаны эффективный динамический алгоритм парной маршрутизации и алгоритм трансляционного обмена, обеспечивающий минимумы времени выполнения и

нагрузки сообщений в сети, полученные в полнодуплексной модели коммутации сообщений.

Оптимальные циркулянтные сети степени 6

Максимальное число вершин циркулянтной сети вида (N; 1, s_2, s_3) [7] с любым диаметром $d \geq 1$ равно

$M(d,3) = 4p^3 + 4p^2 + 3p + 1$, если $d \equiv 0 \bmod 3$,

$4p^3 + 12p^2 + 15p + 7$, если $d \equiv 1 \bmod 3$,

$4p^3 + 20p^2 + 35p + 21$, если $d \equiv 2 \bmod 3$

и достигается для образующих

$(s_2, s_3) = (2p + 1, 4p^2 + 2p + 1)$ или $(2p^2 + p, 2p^2 + 3p + 2)$, если $d \equiv 0 \bmod 3$,

$(2p^2 + 3p + 2, 2p^2 + 5p + 4)$, если $d \equiv 1 \bmod 3$,

$(2p + 3, 4p^2 + 14p + 13)$ или $(2p^2 + 5p + 4, 2p^2 + 7p + 6)$, если $d \equiv 2 \bmod 3$, где $p = 2\lfloor d/3 \rfloor$.

Отметим, что полученное семейство сетей с числом вершин, равным $M(d,3)$, значительно превосходит по соотношению N/d все ранее найденные семейства трехмерных циркулянтов (см., например, [2, 3, 12]). В таблице даны описания трехмерных циркулянтных сетей с максимальным числом вершин для диаметров $1 \leq d \leq 18$.

d	$M(d,3)$	s_2, s_3	s_2, s_3	d	$M(d,3)$	s_2, s_3	s_2, s_3
1	7	2, 4		10	1393	92, 106	
2	21	4, 6	3, 8	11	1815	106, 120	15, 241
3	55	10, 16	5, 21	12	2329	136, 154	17, 273
4	117	16, 22		13	2943	154, 172	
5	203	22, 28	7, 57	14	3629	172, 190	19, 381
6	333	36, 46	9, 73	15	4431	210, 232	21, 421
7	515	46, 56		16	5357	232, 254	
8	737	56, 66	11, 133	17	6371	254, 276	23, 553
9	1027	78, 92	13, 157	18	7525	300, 326	25, 601

Алгоритм парной маршрутизации

При организации парных обменов в сети связи требуется решение проблемы поиска кратчайшего пути между взаимодействующими процессорами. Рассмотрим решение этой проблемы для полученных трехмерных циркулянтных сетей. Поскольку циркулянт – вершинно-транзитивный граф, то достаточно определить для любой вершины v кратчайший путь между 0 и v. Для вершины v вектор кратчайшего пути

$P(v) = (P_1, P_2, P_3)$, где $| P_i |$, $i = 1, 2, 3$, задает число шагов по образующей s_i ($-s_i$) в кратчайшем пути из 0 в v, а знак +(–) – движение по s_i ($-s_i$). Для $d \equiv 0 \bmod 3$ рассмотрим семейство оптимальных графов $\Gamma = \{(N(p); 1, s_2(p), s_3(p)), p$ – четное число$\}$, где $N = 4p^3 + 4p^2 + 3p + 1$, $s_2 = 2p + 1$, $s_2 = 4p^2 + 2p + 1$.

Для любой вершины $0 \le v \le N$ графа семейства Γ получен следующий

Алгоритм вычисления вектора кратчайшего пути $P(v)$

begin

$a = 0$; $r = 2p^2 + 2p + 1$; $\Delta = v \bmod s_3$;

if $\Delta > r$ then $\{\Delta = 2r - \Delta$; $a = 1\}$;

$i = v \operatorname{div} s_3$; $j = \Delta \operatorname{div} s_2$; $k = \Delta \bmod s_3 - p - 1$;

if $(p/2 \le i + j \le 3p/2)$ and $k \le -p/2 + i + j)$ or $(i + j > 3p/2)$

then begin $P_3 = i - p$; $P_2 = j - p$; $P_1 = k$; if $a = 1$ then $\{P_2 = - P_2$; $P_1 = - P_1\}$; stop; end

$P_3 = i$; if $(0 \le i + j < p/2)$ and $(k \le 0)$ or $(p/2 \le i + j \le 3p/2)$ and $(k < p/2 - i - j)$

then begin $P_2 = j$; $P_1 = k + p + 1$; end

else begin $P_2 = j + 1$; $P_1 = k - p$; end;

if $a = 1$ then begin $P_3 = P_3 + 1$; $P_2 = -P_2 + 1$; $P_1 = -P_1$; end;

else begin $P_1 = -k - p - 1$; $P_2 = 1 - j$; $P_3 = i + 1$; end; stop;

end

Как видим, полученный алгоритм не зависит от числа вершин в графе и имеет сложность O(1). Он позволяет быстро вычислять в каждой вершине пути между любыми источниками и приемниками сообщений без использования табличного представления сети, помещенного в каждой вершине. Схема *алгоритма парной маршрутизации* (АПМ) для графов рассмотренного семейства:

1. АПМ вычисляет вектор кратчайших путей P в вершине-источнике сообщения, используя номер вершины-приемника.

2. В каждой промежуточной вершине (и вершине-источнике) АПМ выбирает номер выходного направления, принадлежащего кратчайшему пути до вершины-приемника, и вектор P модифицируется так, чтобы модифицированный вектор P' задавал остаток кратчайшего пути до вершины-приемника. При этом выбор номера выходного направления может учитывать не только нулевые координаты вектора P', но также текущие состояния смежных вершин и линий связи (отказы, нагрузку).

3. АПМ завершается, когда все координаты P' равны 0.

Алгоритм трансляционного обмена

Рассмотрим трансляционный обмен при полнодуплексной модели коммутации сообщений для оптимальных циркулянтных сетей семейства Γ: требуется организовать передачу сообщений из любого процессора (PE) во все другие за минимальное время и без дублирования сообщений. Последнее требование вызвано необходимостью минимизировать нагрузку в сети при коллективных обменах. Одно из решений проблемы заключается в построении минимального покрывающего дерева с корнем в источнике и передаче копий сообщения только по направленным ребрам этого дерева. На рисунке слева показана нумерация входных (выходных) полюсов процессора (вершины) и соответствие их образующим графа, справа - дан фрагмент минимального покрывающего дерева (не показаны ребра, соответствующие образующим $\pm s_1$) для циркулянтного графа с $N = 333$ и диаметром 6. Вершина-источник расположена в центре, стрелки на ребрах указывают направления передач копий сообщения между смежными PE по линиям связи, соответствующим образующим s_2 (по горизонтали) и s_3 (по вертикали).

Структура сообщения для трансляционного обмена: *msg*:= {*T, U, V, W, B*}, где *T* – текст сообщения, *U*, *V*, *W* – счетчики числа шагов между PE, выполняющим алгоритм, и PE-источником, по линиям связи, соответствующим трем образующим, *B* – признак трансляционного обмена. Если сообщение прибывает в PE первый раз, тогда оно должно быть принято, в противном случае PE заканчивает алгоритм трансляционного обмена. Это позволяет избежать передач лишних копий сообщения по линиям связи, соответствующим образующим $\pm s_1$, и ограничить требуемое число шагов алгоритма диаметром графа. Алгоритм начинает работу в транзитном PE (или PE-источнике) после получения сообщения с признаком *B* с входа 1–6 (или 0). В транзитном PE параметры U_m, V_m, W_m поступившего с входа *i* сообщения преобразуются в параметры *U*, *V*, *W* для сообщения в выходной очереди.

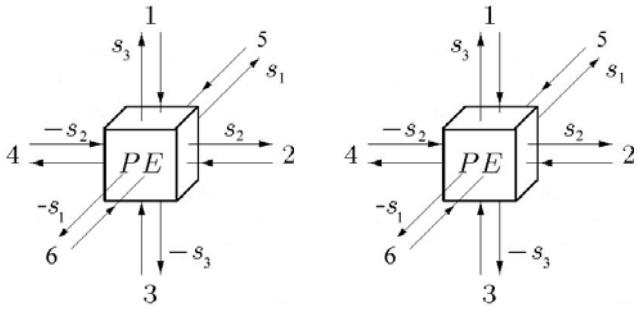
Алгоритм трансляционного обмена в процессоре-источнике.

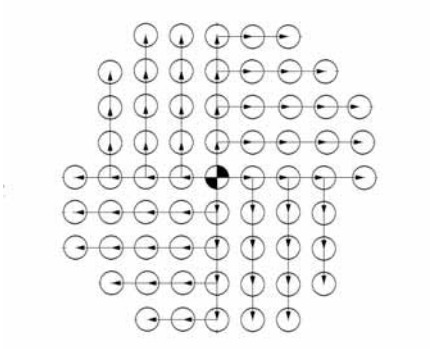
begin
U := 0; *V* := 0; *W* :=0;
for *i* = 1 to 6 do send *msg* to output *i*; stop
end

Алгоритм трансляционного обмена в транзитном процессоре.

begin
if *msg* is received for the first time then send *msg* to output 0

else begin delete *msg* from input *i*; stop end;
if *i*=5 or *i*=6 then begin $U := U_m + 1$; $V := V_m$; $W := W_m$;
if $U + V + W = d$ then stop else begin send *msg* to output *i*; stop end; end
else if *i* mod 2=1 then begin $U := U_m$; $V := V_m$; $W := W_m + 1$; end
else begin $U := U_m$; $V := V_m + 1$; $W := W_m$; end;
for $k = 5$ to 6 do send *msg* to output *k*;
if ($V = 0$) or ($W = 0$ and $V < p$) then send *msg* to output $(i + 2) \bmod 4 + 1$
else if ($V + W = d$) or (*i* mod 2 = 1 and $W = p$) or (*i* mod 2 = 0 and $V = p$) then stop
else send *msg* to output $(i + 1) \bmod 4 + 1$; stop
end





Заключение

Получены оптимальные трехмерные циркулянтные сети, которые в своем классе обладают наилучшими структурными показателями, включая низкую задержку при передаче информации и высокую живучесть (за счет минимизации диаметра), а также имеют простые и эф-

фективные алгоритмы маршрутизации, не использующие таблиц маршрутизации. Построенные алгоритмы обменов инвариантны к числу процессоров в системе и допускают простые модификации, учитывающие отказы процессоров и линий связи. Полученное семейство оптимальных сетей представляет интерес с практической точки зрения, т. к. циркулянтные графы используются в качестве коммуникационных сетей связи для реальных и проектируемых вычислительных и мультикластерных систем.

Литература

1. *Hwang F.K.* A survey on multi-loop networks. Theoretical Computer Science, 2003, **299**, P. 107–121.

2. *Монахов О.Г., Монахова Э.А.* Параллельные системы с распределенной памятью: структуры и организация взаимодействий, Новосибирск, Изд-во СО РАН, 2000.

3. *Bermond J.-C., Comellas F., and Hsu D.F.* Distributed loop computer networks: a survey. J. Parallel Distributed Comput., 1995, **24**, P. 2–10.

4. *Liestman A.L., Opatrny J., and Zaragoza M.* Network Properties of Double and Triple Fixed-Step Graphs. Int. J. Found. Comp. Sci., 1998, **9**, P. 57–76.

5. *Comellas F., Mitjana M., Peters J.G.* Broadcasting in Small-World Communication Networks. Proc. 9th Int. Coll. on Structural Information and Communication Complexity, eds. C. Kaklamanis and L. Kirousis. Proceedings in Informatics, 13, P. 73–85 (2002).

6. *Muga F.P., and Yu W.E.S.* A Proposed Topology for a 192-Processor Symmetric Cluster with a Single-Switch Delay. In Proc. of the Philippine Computing Science Conf. (2002).

7. *Monakhova E.* Optimal Triple Loop Networks with Given Transmission Delay: Topological Design and Routing. In Proc. of the International Network Optimization Conference, (INOC'2003), Evry / Paris, France, 2003, P. 410–415.

8. *Narayanan L., Opatrny J., Sotteau D.* All-to-All Optical Routing in Chordal Rings of Degree Four. Algorithmica, 2001, 31(2), P. 155–178.

9. *Монахова Э.А.* Алгоритмы межмашинных взаимодействий и реконфигурации графов связей в вычислительных системах с программируемой структурой. Вычислительные системы, Новосибирск, 1982, No. 94, С. 81–102.

10. *Robic B. and Zerovnik J.* Minimum 2-terminal routing in 2-jump circulant graphs. Computers and Artificial Intelligence, 2000, **19(1)**, P. 37–46.

11. *Barriere L., Fabrega J., Simo E., Zaragoza M.* Fault-Tolerant Routings in Chordal Ring Networks. Networks, 2000, Vol. 36(3), P. 180–190.

12. *Chen S., and Jia X.-D.* Undirected loop networks. Networks, 1993, **23**, P. 257–260.

СЖАТИЕ НАУЧНЫХ ДАННЫХ БОЛЬШОГО ОБЪЕМА ДЛЯ ОБЕСПЕЧЕНИЯ ВОЗМОЖНОСТИ ИХ ВИЗУАЛИЗАЦИИ

С.В. Муравьев

ИММ РАН, Москва

Введение

Современные компьютерные системы позволяют решать довольно сложные задачи, возникающие как при численном моделировании, так и при обработке данных, получаемых в научных экспериментах. Получаемые результаты (обычно трехмерные скалярные данные) могут занимать довольно большой объем дискового пространства компьютера. Большой объем данных может создавать существенные проблемы во многих областях их применения (визуализация, передача данных по узким каналам Интернета, исследование данных в интерактивном режиме и др.) Таким образом, весьма актуальна проблема сжатия подобных данных.

В данной работе рассматриваются алгоритмы сжатия данных 2-х типов. Первый тип данных – это трехмерные поверхности произвольного вида. Исследование поверхностей может потребоваться, например, при изучении изоповерхностей скалярного физического параметра, распределенного в трехмерной объемной области. Примерами исследуемого параметра могут быть давление, температура, плотность и т.д. Второй рассматриваемый здесь тип данных для сжатия – непосредственно сами трехмерные скалярные данные, заданные на тетраэдрической сетке. К этому типу данных относятся результаты математического моделирования трехмерных физических задач, результаты натурных экспериментов и т.д.

При визуализации изоповерхностей трехмерных скалярных данных большого объема нет необходимости (а иногда и возможности) использовать данные полностью с той же точностью, с которой они были получены. Слишком подробные пространственные распределения, получаемые с помощью современных компьютерных систем, во-первых, слишком велики по объему, а, во-вторых, во многом избыточны для их использования в системах визуализации. Поэтому возникает задача - обработать поверхность каким-либо алгоритмом сжатия, чтобы уменьшить объем и избыточность данных, но сохранить их приемлемый вид.

Сжатие без потерь не решает рассматриваемую проблему визуализации большого объема данных, поскольку оно фактически не умень-

шает размер самих данных, а только уменьшает объем, необходимый для их хранения. Сжатие без потерь используется в случаях, когда необходимо полностью сохранить исходные данные, и проблему избыточности данных оно не решает. Таким образом, для целей визуализации необходимо использовать алгоритм сжатия с потерями.

Существует еще одна важная проблема обработки рассматриваемых данных. Объем данных может быть настолько большим, что их не только нельзя визуализировать, то и поместить в память обычного компьютера. Таким образом, сжать подобные данные можно только на многопроцессорных вычислительных системах с распределенной памятью.

Как показывает практика, однопроцессорная обработка трехмерных скалярных данных (в отличие от обработки поверхностей) занимает существенное время. Указанная проблема является еще одной причиной, по которой для сжатия подобных данных необходимо использовать МВС.

Однопроцессорный алгоритм сжатия поверхностей

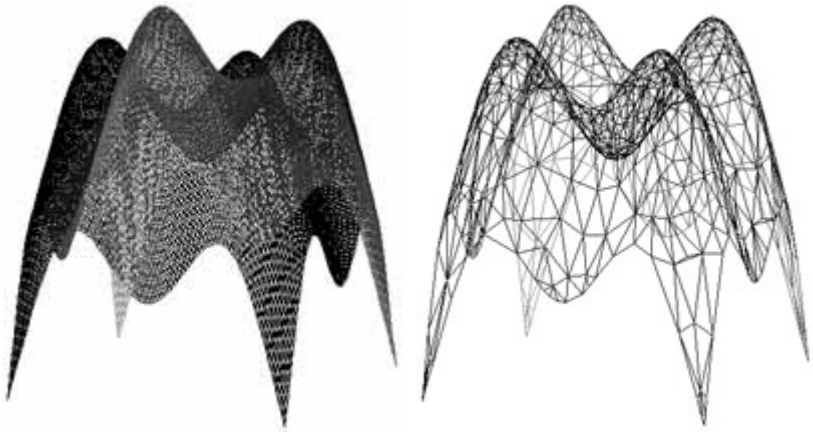
Наиболее распространенным и удобным способом задания поверхности является представление ее в виде триангуляции точек в пространстве. Поэтому именно на этот тип поверхностей и рассчитан используемый алгоритм сжатия с потерями. Общая схема работы алгоритма состоит в том, что на его вход поступает исходная поверхность, а на выходе получается новая поверхность как можно меньшего размера, которая, во-первых, занимает объем, допустимый для использования в системах визуализации, и, во-вторых, стремится аппроксимировать исходную поверхность с требуемой точностью. Алгоритм работает с любыми типами поверхностей: они могут задаваться неоднозначными функциями, иметь полости, не являться гладкими, иметь участки самопересечения, быть несвязными и т.д.

Основная идея алгоритма состоит в последовательном упрощении малых участков поверхности с сохранением требуемой точности. При этом проверяется, чтобы получаемый участок новой поверхности оставался на допустимом расстоянии от соответствующего участка исходной поверхности. Требуемая точность передается алгоритму в виде абсолютной ошибки аппроксимации, с которой новая поверхность должна приближать исходную. После каждого упрощения участка поверхности количество точек и треугольников этого участка уменьшается, за счет чего и выполняется сжатие. Последовательное упрощение всевозможных участков поверхности (удаление избыточности) выпол-

няется до тех пор, пока выполняется условие соблюдения требуемой точности. В процессе модификации поверхности алгоритм также следит за сохранением типа топологии каждого обрабатываемого участка поверхности.

Задача достижения требуемого объема данных в пределах требуемой точности в общем случае может не иметь решения. Чтобы в любом случае получить какое-либо решение, которое можно визуализировать, в алгоритме поставлен более высокий приоритет именно достижению требуемого объема. Если в пределах требуемой точности не удается достичь требуемого объема, то параметр максимально допустимой ошибки начинает постепенно увеличиваться.

Тестирование алгоритма показало, что наиболее высокие коэффициенты сжатия достигаются при сжатии поверхностей, имеющих большую площадь участков с малой кривизной и высокую плотность точек (случаи с наибольшей избыточностью данных). На рисунке можно видеть результат сжатия тестовой поверхности с некоторой высокой степенью точности.

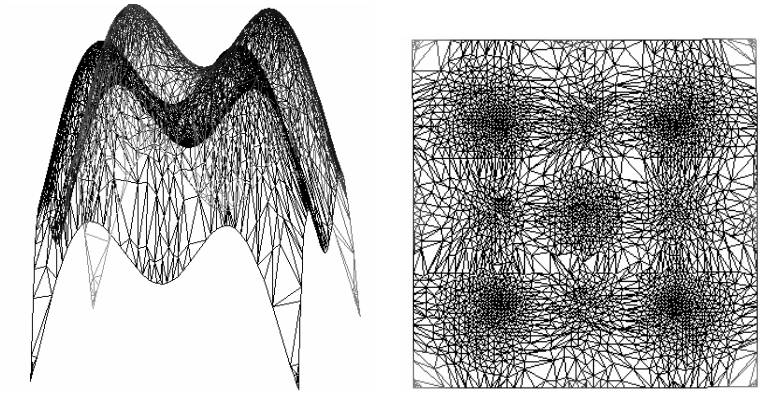


До сжатия После сжатия

Параллельный алгоритм сжатия поверхностей

Обработку поверхностей большого объема эффективнее выполнять на многопроцессорных вычислительных системах. Распараллеливание используемого алгоритма не вызывает проблем, поскольку фактически он работает последовательно с различными малыми участками

поверхности. Каждый участок поверхности обрабатывается независимо от других участков, поэтому в основу многопроцессорной реализации алгоритма был положен метод геометрического параллелизма. В результате при параллельном сжатии исходная поверхность обрабатывается по частям на отдельных процессорах. В течение основного процесса сжатия никакого межпроцессорного взаимодействия не осуществляется. Для обеспечения последующей связности обрабатываемых частей общие точки этих частей предварительно маркируются как не удаляемые. При упрощении малых участков поверхности никаких изменений с маркированными точками не производится. После сжатия все части поверхности собираются со всех процессоров и «сшиваются» по маркированным точкам. На окончательном этапе сжатия все «швы» обрабатываются последовательным алгоритмом с заниженным параметром максимально допустимой ошибки. Как показало тестирование алгоритма, визуальное качество получаемой поверхности вблизи «швов» почти не уступает качеству результатов, получаемых при полной обработке этого же участка поверхности на однопроцессорной системе. На рисунке показан результат сжатия тестовой поверхности, заданной на прямоугольной области (на правом рисунке – вид сверху). Сжатие производилось на 3-х процессорах с некоторой высокой степенью точности.



Результат сжатия поверхности на 3-х процессорах

Параллельный алгоритм сжатия трехмерных скалярных данных

На основе алгоритма сжатия поверхностей разработан алгоритм сжатия трехмерных скалярных данных, заданных на тетраэдрических

сетках. Сжатие таких данных точно так же можно выполнять на многопроцессорных вычислительных системах. Ввиду гораздо большей объемности и более сложной структуры трехмерных скалярных данных, время, требуемое на их сжатие, может в сотни раз превышать время обработки их изоповерхностей. Поэтому, как уже было замечено ранее, использование МВС в данном случае играет очень важную роль.

Основной идеей алгоритма сжатия данных, заданных на тетраэдрических сетках, является представление данных как множество точек в 4-х мерном евклидовом пространстве и последовательное упрощение малых областей данных по аналогии с упрощением поверхностей. В данном подходе имеет место обращение с физическими значениями сеточной функции как с новой геометрической координатой. Для корректной работы данного способа с классическим расстоянием между точками в 4-х мерном евклидовом пространстве необходимо предварительно нормализовать (перемасштабировать) значения сеточной функции.

В целом, алгоритм для обработки трехмерных скалярных данных и алгоритм для обработки поверхностей отличаются только размерностью. Немногочисленные отличия касаются, в основном, способа слежения за сохранением типа топологии.

Заключение

Разработанные методы многопроцессорного сжатия позволяют сжимать исходные данные с любой заданной точностью. Применение данных алгоритмов довольно актуально, поскольку сжатие данных большого объема является необходимым ключом к возможности их исследования в системах визуализации. Разработанные алгоритмы довольно универсальны, поскольку они способны обрабатывать широкий класс исходных данных. Алгоритмы показали свою эффективность на большом количестве результатов численного моделирования различных физических задач, а также на всевозможных тестовых данных.

ДИНАМИЧЕСКАЯ ВИЗУАЛИЗАЦИЯ ВЕКТОРНЫХ ПОЛЕЙ, ЗАДАННЫХ НА ТЕТРАЭДРИЧЕСКИХ СЕТКАХ БОЛЬШОГО РАЗМЕРА

И.А. Нестеров

Институт математического моделирования РАН, Москва

В.Г. Чубченко

Московский государственный инженерно-физический институт

Стремительный рост производительности современных вычислительных систем определяет увеличение объемов данных сопровождающих процессе математического моделирования. Требования к точности расчетов приводят к необходимости использования сеток, содержащих 10^8-10^9 и более узлов. В результате, при численном моделировании задач газовой и гидродинамики, возникает необходимость визуализации получаемых расчетных данных, достигающих гига- и терабайтного порядка. Рост объемов обрабатываемых данных обуславливают актуальность задачи отображения и анализа данных, получаемых в процессе математического моделирования.

Основное внимание в данной работе уделено задаче динамической визуализации трехмерных векторных полей, полученных в результате математического моделирования нестационарного обтекания трехмерных тел, заданных на тетраэдрических сетках большого размера.

В качестве основного метода визуализации векторных полей рассмотрены методы геометрической визуализации данных, основанные на вычислении траекторий невесомых частиц в нестационарном поле скоростей. Их главным отличием является то, что при решении интегрального уравнения движения частицы, производится интегрирование по времени для всего доступного набора векторных полей. Поэтому данные методы применимы для отображения изменения поля скоростей по времени, для выявления особенностей течения. Линии разлета и временные линии при анимации могут эффективно показать образование и распад вихрей, вихревые потоки и ударные волны в нестационарном течении. Эти способы отображения наиболее удобны при сравнении результатов численного моделирования с экспериментальными и имеют четкую физическую интерпретацию. Построение траекторий частиц позволяет отобразить эффекты, связанные с нестационарностью поля скоростей, которые не возможно отобразить, применяя другие подходы.

Траектории частиц, линии разлета и временные линии позволяют

отобразить поля скоростей, определенных на сетках очень большого размера. При этом отображение траекторий возможно с использованием существующего графического аппаратного обеспечения, поскольку требуют вывода несравнимо меньшего набора графических примитивов по сравнению с объемом исходной расчетной сетки и сеточных функций.

Последовательная схема вычисление траектории индивидуальной частицы, помещенной в поле скоростей, включает в себя следующие этапы:

- задание начального положения частицы;
- локализация начального положения частицы в расчетной сетке;
- решение интегрального уравнения с помощью выбранной разностной схемы;
- вычисление дополнительных данных для эффективной визуализации с помощью существующего аппаратного обеспечения.

Цикл расчета представлен в виде графической диаграммы и показывает циклический характер расчета.

Из-за характерных объемов данных, необходимых для современных задач моделирования, расчет траектории частицы на нерегулярной сетке требует существенного вычислительного времени. Например, первоначальная локализация начального положения частицы на тетраэдрической нерегулярной расчетной сетке требует перебора всех тетраэдров и проведения теста на принадлежность точки тетраэдру, что может потребовать значительного времени, напрямую зависящего от размера обрабатываемых данных. Для того, чтобы избежать «узкие места» последовательного алгоритма, таких как первоначальная локализация положений точек и ограниченная скорость работы дисковой подсистемы. требуется разработка параллельного алгоритма расчета. Другой целью разработки параллельного алгоритма вычисления траекторий частиц в поле скоростей является необходимость решать задачи высокой ресурсоемкости, в данном случае обрабатывать большие объемы сеточных данных, а также получить возможность одновременного расчета большого набора траекторий.

Основными преимуществами, предоставляемыми вычислительными кластерами для расчета траекторий частиц, являются: возможности обработки данных существенно большего объема, их распределенного хранения и независимого чтения и обработки (декомпозиция задачи, основанная на параллелизме данных), а также возможность эффективной локализации одновременно множества положений точек на разных

вычислительных узлах в параллельном режиме.

По результатам работы последовательной программы чтение данных может занимать существенное время расчета траектории. Для возможности более гибкого управления чтением и загрузкой сеточных данных, а также компенсации низкой скорости дисковой подсистемы необходимо использование распределенных форматов хранения данных и специальных алгоритмов эффективного управления этими данными. Задача распределенного хранения данных была решена использованием специализированной библиотеки TMLLIB, разработанной в Институте Математического Моделирования РАН. Это библиотека позволяет хранить расчетную сетку и сеточные функции в распределенном формате. Это достигается разбиением сетки на множество подобластей (микродоменов) с сохранением информации о топологии

исходной сетки. Алгоритм балансировки учитывает физическое расположение данных на вычислительных узлах и управляет чтением данных по мере необходимости. Оптимальное управление данными, доступными на индивидуальных вычислительных узлах (например: необходимые сеточные функций для следующих шагов по времени, сеточных данных смежных микродоменов и д.р), делает возможным одновременный расчет траекторий различных частиц на одном процессоре с тем, чтобы необходимость дополнительной загрузки данных была бы минимальной.

Система визуализации была спроектирована в парадигме клиент-сервер. При этом на сервер возлагается вся расчетная часть и управление данными, когда как клиентская часть отвечает за пользовательский интерфейс и отображение получаемых траекторий, используя клиентскую видеоподсистему. Серверная часть запускается на высокопроизводительном кластере и осуществляет обмен данными по протоколу TCP/IP с клиентской частью. Внутренние структуры данных и параллельные алгоритмы расчета реализованы с использованием языка C++ и библиотеки MPI.

Применение разрабатываемой системы визуализации на вычислительных мощностях ИММ РАН для отображения результатов моделирования трехмерного обтекания сферы, описанной тетраэдрической сеткой, сжимаемым теплопроводным газом показывает высокую эффективность параллельного алгоритма. Выбранный способ визуализации позволяет в интерактивном режиме выбирать области интереса, управлять построением и отображением получаемых траектории частиц. Разработанная функциональная часть по визуализации трехмерных векторных полей находится в стадии интегрирования в разрабатываемый комплекс научной визуализации, включающий на сегодняшний момент также отображение скалярных сеточных функций большого размера, заданных на тетраэдрических сетках.

МОДЕЛИРОВАНИЕ РАСПРЕДЕЛЕННОЙ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ НА КЛАСТЕРЕ РАБОЧИХ СТАНЦИЙ

С.И. Олзоева, Д.В. Тюменцев, А.В. Костенко

Восточно-Сибирский государственный технологический университет, Улан-Удэ

Введение

В настоящее время широкое распространение получили автоматизированные системы обработки информации и управления, элементы которых (объекты, узлы) физически рассредоточены в пространстве. Высокая стоимость разработки распределенных автоматизированных систем (РАС) требует постоянного повышения качества их проектирования. Достижение этой цели в значительной степени связано с использованием метода имитационного моделирования, позволяющего изучать систему с учетом всех аспектов ее природы и поведения. Причем имитационные модели (ИМ) должны работать в комплексе со специализированными программными средствами для автоматизации проектирования РАС, обеспечивающими машинный поиск оптимального варианта РАС при ее синтезе при заданных критериях эффективности и системе ограничений. Метод имитационного моделирования характеризуется тем, что с увеличением размерности моделируемого объекта увеличиваются вычислительные затраты для получения статистически обоснованных результатов. Кроме того, необходимо многократное моделирование процесса функционирования РАС с различными исходными данными задачи. Внедрение методов имитационного моделирования в интерактивные системы оптимизации, экспертные системы, требует ускорения вычислительного процесса имитационного моделирования РАС без потери точности и упрощения модели.

Решение проблемы лежит в области эффективного применения современных средств вычислительной техники - многопроцессорных систем, наиболее общедоступными из которых являются кластерные системы и локальные сети.

В работе представлено описание разработки программного продукта по организации имитационного моделирования РАС «Система-Город» на кластере рабочих станций.

Описание моделируемой системы

РАС «Система-Город» предназначена для сбора платежей от населения за различные услуги (коммунальные, услуги связи, электричест-

во, ГИББД и т.п.) с использованием современных сетевых и информационных технологий, основные принципы работы которой состоят в следующем:

– биллинговый центр в городе обслуживает центральный сервер Системы, на котором хранится вся информация о плательщиках;

– поставщики различных услуг (коммунальные службы, компании, предоставляющие услуги связи, ГИБДД и т.п.) передают на центральный сервер по каналам связи информацию о своих абонентах;

– агенты-организации, принимающие платежи (банки, отделения связи и т.п.), получают онлайн – доступ к центральному серверу.

Информационная сеть объединяет эти три основные взаимодействующие группы, каждая из которых обеспечена специальными программными средствами – АРМами, выполняющими соответствующие группе целевые функции, и должна обеспечить эффективное функционирование РАС «Система «Город»».

Внедрение данной технологии в сферу муниципального управления городом требует обоснованного выбора состава и структуры распределенной информационной сети, обеспечивающей надежное выполнение общецелевой функции рассматриваемой РАС. В частности необходимо ответить на вопросы: на базе какой сети можно объединить группу территориально удаленных пользователей (поставщиков услуг, оператора системы, агентов), связанных общими целями в их деятельности, без необходимости внесения в эту систему каких либо изменений; какая телекоммуникационная сеть выгоднее для получения необходимого перечня услуг с приемлемыми затратами?

Основные телекоммуникационные сеансы связи между компонентами Системы «Город», состоят в установлении соединения АРМов Поставщиков услуг и Агентов с сервером БД по протоколу TCP/IP. Связь можно обеспечить либо через программное обеспечение Dial-Up Networking (DUN), устанавливаемое на ПК с АРМом, либо через маршрутизатор, установленный в соответствующей локальной сети Ethernet.

Подключить АРМы Поставщиков Услуг и Агентов к почтовому серверу Биллинг Центра (БЦ) можно либо напрямую к БЦ, либо через сеть Интернет.

В первом случае, при прямом подключении, в качестве достоинств можно выделить отсутствие платы за трафик для клиентов и БЦ в отличие от использования сети Интернет. Кроме того, это единственный вариант для подключения пунктов приема платежей, не имеющих Цен-

трального офиса, через который могли бы объединяться информационные потоки. Но недостатками данного способа являются высокие затраты на сетевое оборудование и телефонные номера ГТС (аналоговые или сети ISDN).

Во втором случае АРМы Поставщиков Услуг подключаются к серверу доступа провайдера сети Интернет (ISP), трафик маршрутизируется по сети Интернет до сетевого оборудования БЦ. АРМы Агентов подключаются к корпоративной сети Агента, т.е. к сетевому оборудованию некоторого Центрального офиса через СПД, оснащенную сервером доступа для подключения коммутируемых соединений, маршрутизатором для подключения выделенных линий и сети Frame Relay. Такая схема необходима в случае, если в городе по Системе «Город» работают несколько банков, каждый из которых может «собирать» информационные потоки своих пунктов приема платежей у себя в центральном офисе, а потом маршрутизировать на сервер БД.

Достоинствами данного способа подключения являются низкие затраты для БЦ на оборудование и подключение к СПД (только оплата подключения к провайдеру сети Интернет). Подключение к СПД используется не только для связи с БЦ, но и для работы в сети Интернет, а также для работы других корпоративных приложений. К недостаткам можно отнести наличие платы за трафик – как для клиентских мест Поставщиков Услуг, так и для БЦ, высокие затраты для Агента по приему платежей на создание корпоративной сети.

Таким образом, стоит задача проектирования информационной сети, способной обеспечить эффективное функционирование рассмотренной системы в заданных конкретных условиях инфраструктуры города. Произвести сравнение перечисленных выше альтернативных вариантов и выбрать наиболее предпочтительный, отвечающий критериям производительности сети и приемлемой стоимости сетевого оборудования. При этом необходимо учесть, что в дальнейшем, при увеличении количества рабочих мест по приему платежей и количества лицевых счетов, может увеличиться нагрузка, следовательно, нужно предусмотреть возможность увеличения количества вычислительных средств, телекоммуникационных каналов и сетевого оборудования.

Проектирование ИС с использованием имитационного моделирования

При построении информационной сети одной из основных задач является синтез структуры сети, обеспечивающей передачу заданных потоков информации по всем направлениям информационного обмена

в приемлемое время. В процессе ее решения необходимо учитывать множество характеристик сети, таких как задержки, надежность, стоимость, а также ограничения накладываемые быстродействием узлов РАС, пропускной способностью каналов, характеристиками трафика и т.д.

Процесс проектирования сети осуществляется в несколько этапов. На первом этапе определяется исходный граф сети. Затем производится анализ исходного варианта с привлечением имитационного моделирования. Определяются основные характеристики сети, область допустимых значений нагрузки, при которых обеспечивается требуемое время передачи информации.

Последующие этапы синтеза выполняются с использованием результатов анализа исходного варианта. Главным параметром оптимизации выступает средняя задержка сообщений в сети. Но сложность задачи заключается в том, что для различных участков сети должны выполняться различные как в качественном, так и в количественном отношении условия. Поэтому целесообразнее в роли критерия эффективности выбрать время доставки сообщений между отдельными корреспондирующими парами «источник – адресат». Тогда задача оптимизации сети будет заключаться в оптимизации сети по направлениям обмена информацией, для решения которой нужно определить среднее время доставки сообщений между всеми корреспондирующими парами узлов при заданных структуре и характеристиках сети; вероятности недоставки сообщений за заданное время межу парами узлов.

Информационная сеть рассматриваемой РАС «Система – Город», внедряемой в г. Улан-Удэ, представляет собой сеть большой размерности, объединяющей более 180 узлов. Программное обеспечение группы «Агенты» должно быть установлено на 149 рабочих местах в пунктах оплаты банков, отделениях почтамта, ЖЭУ города. Количество АРмов Поставщиков услуг составляет 30. Ежедневно через Систему прогнозируется проведение более 12 000 платежей. Количество информационных потоков в этой сети более 9000, причем каждый поток характеризуется своей интенсивностью. Моделирование работы такой сети только для одного варианта исходных данных на компьютере Pentium III потребовало 36 минут процессорного времени.

Распределенное моделирование РАС «Система – Город»

Для организации имитационного моделирования РАС на кластере рабочих станций использовалась парадигма параллельных вычислений MPMD (Multiple Program – Multiple Data), основанная на расчленении

алгоритма программы на крупные, параллельно исполняемые фрагменты, которые существенно отличаются и могут иметь разный объем выполняемых операций. Организация распределенного имитационного моделирования (РИМ) РАС потребовала разработки средств автоматизированного распараллеливания алгоритмов имитационного моделирования, для отделения аспектов, связанных с разработкой программного обеспечения ИМ, от вопросов собственно организации параллельных (распределенных) вычислений.

К рассмотрению предлагается автоматизированный инструментарий для оптимальной, по критерию минимизации времени, организации распределенного вычислительного процесса имитационного моделирования. Функциональное содержание такого инструментария состоит в выдаче результатов измерений отдельных параметров качества проекта распределенной имитационной программы. Технологическими этапами функционирования программного инструментария являются:

1. Декомпозиция ИМ.

2. Оценка качества вариантов проектов распределенной ИМ, определение возможных временных факторов.

3. Выбор метода синхронизации взаимодействий между программными блоками распределенной ИМ.

4. Оптимальное распределение программных блоков по процессорам вычислительной системы.

Алгоритмическим наполнением технологических этапов являются; методы декомпозиции, основанные на теории графов; метод оценки прогнозируемого времени ускорения вычислений ИМ [2]; метод анализа алгоритмов синхронизации модельного времени [1]; метод оптимального размещения программных модулей ИМ на процессорах распределенной вычислительной платформы [3].

Применение предлагаемой методики к организации распределенного имитационного моделирования РАС на 7 персональных ЭВМ (Pentium III) локальной сети с пропускной способностью 100 Мбит/с. позволило существенно ускорить вычислительный процесс ИМ. Ускорение (с учетом времени подготовки РИМ) составляет от 4 до 6,5 в зависимости от вариантов исходных данных. Для организации обмена данными между программами по среде передачи использовались низкоуровневые средства системного программирования – интерфейс сокетов протокола IPX. Предлагаемый способ распределенного имитационного моделирования РАС позволяет использовать существующий парк компьютеров, причем без отрыва последних от повседневного

использования, и не зависит от конфигурации сети и аппаратного обеспечения.

Литература

1. *Вознесенская Т.В.* Исследование эффективности методов синхронизации времени для распределенного имитационного моделирования. // Матер. конф. «Высокопроизводительные вычисления и их приложения». Черноголовка. 2000. С. 208–211.

2. *Кутузов О.И.*, *Олзоева С.И.* Организация вычислительного процесса для распределенного имитационного моделирования дискретно-событийных систем. // Матер. V Международн. конф. по мягким вычислениям и измерениям, Санкт-Петербург, 2002. Т. 1, С. 101–105.

3. *Олзоева С.И.* Метод оптимального распределения программных модулей по процессорам вычислительной системы // Вестник БГУ. Сер. 13 «Математика и информатика». Улан-Удэ. 2005. Вып. 2. С. 257–261.

ПАРАЛЛЕЛИЗМ МОЗГОПОДОБНЫХ ВЫЧИСЛЕНИЙ

Ю.П. Шабанов-Кушнаренко, В.И. Обризан

Харьковский национальный университет радиоэлектроники, *Украина*

Введение

Актуальность исследования определяется необходимостью разработки ЭВМ параллельного действия в целях существенного повышения быстродействия решения задач по сравнению с программной реализацией на компьютере фон-неймановской архитектуры.

Цель исследования – разработка модели ЭВМ параллельного действия, работающей по принципам мозга человека и построенной на современной элементной базе.

Для достижения поставленной цели следует решить следующие *задачи*:

1) разработка нового подхода к созданию искусственного интеллекта: интеллект человека рассматривается как некоторое материальное воплощение механизма логики;

2) выполнение работ по алгебраизации логики;

3) формализация модели логической сети;

4) разработка процедур логического синтеза сети;

5) разработка модели процесса проектирования логической сети; 6) анализ эффективности аппаратной реализации.

Быстро прогрессирующие компьютеризация и информатизация требуют постоянного повышения производительности электронных вычислительных машин. Резервы увеличения быстродействия решающих элементов ЭВМ исчерпываются. Остается путь наращивания числа одновременно работающих элементов в процессоре компьютера. Уже сейчас имеется практическая возможность, опираясь на успехи микроминиатюризации и удешевления электронных элементов и на достижения в области автоматизации проектирования и изготовления ЭВМ, строить компьютеры с числом элементов до 10^{15}. Однако, применительно к нынешним ЭВМ последовательного действия, работающим по принципу программного управления Дж. фон Неймана, делать это не имеет смысла, поскольку в них в каждый момент времени одновременно находится в работе лишь небольшое число элементов. Попытки же перехода к машинам параллельного действия пока не дают ожидаемого роста их производительности. Так, например, производительность многопроцессорных ЭВМ растет не пропорционально числу имеющихся в ней процессоров, как, казалось бы, должно быть, а гораздо медленнее. Возникают существенные трудности также и при попытках создания высокопроизводительных нейрокомпьютеров, которые строятся в виде сетей из формальных нейронов.

Между тем, существует «вычислительная машина», созданная природой, а именно – мозг человека, для которой проблема полноценного распараллеливания обработки информации полностью решена. Мозг человека по сравнению с современной ЭВМ – тихоход. О его «тактовой частоте» можно судить по пропускной способности нервных волокон. Известно, что каждое нервное волокно может пропускать не более 10^3 импульсов в секунду. По проводникам же нынешних ЭВМ передается порядка 10^9 импульсов в секунду. Следовательно, ЭВМ превосходит мозг человека в смысле скорости работы решающих элементов в $10^9 : 10^3 = 10^6$ раз. И тем не менее, мозг, благодаря параллельному принципу действия, способен решать неизмеримо более сложные задачи, чем самые мощные современные вычислительные машины с программным управлением. Это обусловлено тем, что мозг человека имеет в своем составе около 10^{15} решающих элементов (в роли которых принимаются синапсы – стыки между окончаниями нервных волокон), и все они, как свидетельствуют нейрофизиологические данные, работают одновременно. В ЭВМ же последовательного действия в любой момент времени параллельно действует лишь небольшое число элементов.

В Харьковском национальном университете радиоэлектроники на протяжении последних 40 лет разрабатывается научное направление – *теория интеллекта* [1]. Суть подхода состоит в том, что интеллект человека рассматривается как логика в действии, как некоторое материальное воплощение механизма логики. Был разработан специальный математический аппарат для формульного представления отношений и действий над ними, которые называются *алгебро-логическими структурами*. Отношения интерпретируются как мысли интеллекта, а действия над ними – как мышление. Схемная реализация формул, описывающих алгебро-логические структуры, приводит к характерным инженерным сетям (не использовавшимся ранее), которые называются *логическими сетями*. Главное в данном методе – это движение сверху вниз: от общих системных соображений к алгебро-логическим структурам, а от них – к логическим сетям, которые затем отождествляются с биологическими нейронными структурами. Т.о., принципы построения мозгоподобных ЭВМ существенно отличаются от всего того, что до сих пор использовалось при распараллеливании обработки информации, в частности – при создании ЭВМ параллельного действия. Логическая сеть предназначена для выполнения действий над отношениями. Она есть как раз то устройство мозгоподобной ЭВМ, с помощью которого она будет мыслить.

Отношения и предикаты

Произвольно выберем какое-нибудь непустое множество U, элементы которого будем называть *предметами*. Само множество U называется *универсумом предметов.* Оно может быть как конечным, так и бесконечным. Произвольно выберем m каких-нибудь непустых необязательно различных подмножеств $A_1, A_2, \ldots, A_m$ универсума U. Декартово произведение $S = A_1 \otimes A_2 \otimes \ldots \otimes A_m$ множеств $A_1, A_2, \ldots, A_m$ называется *предметным пространством S с координатными осями* $A_1, A_2, \ldots, A_m$ над универсумом U. Введем множество $V = \{x_1, x_2, \ldots, x_m\}$ различных переменных $x_1, x_2, \ldots, x_m$, которые называются *предметными переменными* пространства *S*. Множество V называется *универсумом переменных* пространства *S*. Значениями переменной x_i $(i = \overline{1, m})$ служат элементы множества A_i, так что $x_1 \in A_1, x_2 \in A_2, , \ldots, x_m \in A_m$. Каждой переменой x_i $(i = \overline{1, m})$ ставится в соответствие какая-то фиксированная область задания A_i. Пространство *S* можно рассматривать как совокупность всех векторов вида $(x_1, x_2, \ldots, x_m)$, каждый из кото-

рых удовлетворяет условию $x_1 \in A_1$, $x_2 \in A_2$, , …, $x_m \in A_m$. Любое подмножество P пространства S называется *отношением*, образованным в (или иначе: заданным на) пространстве S.

Предикатом, заданным на декартовом произведении S, называется любая функция $P(x_1, x_2, \ldots, x_m) = \xi$, отображающая декартово произведение $A_1 \otimes A_2 \otimes \ldots \otimes A_m$ множеств $A_1, A_2, \ldots, A_m$ во множество $\Sigma = \{0, 1\}$. Символы 0 и 1 называются *булевыми элементами*, Σ – множество всех булевых элементов. Переменная $\xi \in \{0, 1\}$, являющаяся значением предиката P, называется *булевой*. Между отношениями и предикатами существует взаимно однозначное соответствие.

Пусть L – множество всех отношений на S, M – множество всех предикатов на S. Отношение P из L и предикат P из M называются *соответствующими* друг другу, если при любых $x_1 \in A_1$, $x_2 \in A_2$, , …, $x_m \in A_m$,

$$P(x_1, x_2, ..., x_m) = \begin{cases} 1, \text{ если } (x_1, x_2, ..., x_m) \in P; \\ 0, \text{ если } (x_1, x_2, ..., x_m) \notin P. \end{cases} \quad (1)$$

Предикат $P(x_1, x_2, \ldots, x_m)$, в отличие от соответствующего ему отношения P, есть функция, поэтому появляется надежда, что его удастся выразить в виде формулы.

Алгебра предикатов

Для построения формул нам понадобятся базисные предикаты, играющие роль исходных строительных блоков, и базисные операции, обеспечивающие соединение блоков в единую конструкцию, каковой является искомая формула. В роли базисных предикатов используем предикаты 0, 1, а также специальные предикаты, называемые *предикатами узнавания предмета* **a** *по переменной* x_i $(i = \overline{1, m}, \mathbf{a} \in A_i)$ и записываемые в виде $x_i^{\mathbf{a}}$.

Предикат $x_i^{\mathbf{a}}$ «узнает» произвольно выбранный из множества A_i предмет x_i, сравнивая его с предметом **a**.

$$x_i^{\mathbf{a}} = \begin{cases} 1, \text{ если } x_i = \mathbf{a}, \\ 0, \text{ если } x_i \neq \mathbf{a}. \end{cases} \quad (2)$$

Указание предмета a и значения индекса i полностью определяет предикат вида (2). В роли способов соединения выбранных нами строительных блоков: предикатов 0, 1 и предикатов вида $x_i^{\mathbf{a}}$ – исполь-

зуем операции дизъюнкции, конъюнкции и отрицания предикатов. В результате получаем так называемую *алгебру предикатов*.

Язык алгебры предикатов универсален, на нем можно формально описать структуру произвольных объектов. Предшественниками данной алгебры являются алгебра булевых функций (алгебра логики), созданная Дж. Булем в XIX веке, и многозначная логика (первая четверть XX ст., Пост). Функции алгебры логики принимают те же значения, что и предикаты. Однако аргументы функции алгебры логики двоичны, а предикатов буквенные. Аргументы функции многозначной логики принимают те же значения, что и аргументы предикатов. Однако значения функций многозначной логики буквенные, а у предикатов они двоичны.

Как видим операции $\vee$, $\wedge$, $\neg$ в сочетании с предикатами 0, 1 и всевозможными предикатами узнавания предмета образуют достаточный ассортимент выразительных средств для формульной записи любого предиката произвольного типа, а значит – и любого соответствующего ему отношения. Учитывая, что отношения – это универсальное средство описания любых объектов, мы можем заключить, что на язык формул алгебры предикатов можно перевести любые зависимости, известные в науке.

На универсальном языке (а именно таким языком является алгебра предикатов) можно записать любой закон природы, любую программу для ЭВМ, любое математическое соотношение, вообще – любую мысль, абсолютно все, что только можно наблюдать вокруг и внутри нас.

Модель логической сети

Каждая логическая сеть представляет собой соединение полюсов и ветвей, т.е. описывается парой $\langle X, P\rangle$, где X – конечное непустое множество предметных переменных (вершины), P – совокупность двуместных предикатов (ребра), определенных на множестве X (рис. 1) [2]. В любой логической сети каждый полюс характеризуется своей предметной переменной и ее областью задания, а каждая ветвь – своим двуместным предикатом, который зависит от переменных, указанных на концах этой ветви. В зависимости от сложности решаемой задачи, сеть содержит то или иное число полюсов. Соответственно чем сложнее задача, тем больше полюсов в логической сети. В узлах сети описание предметных переменных происходит на языке алгебры предикатов, когда же происходит процесс обработки информации в ветвях логической сети, здесь приходит на помощь алгебра предикатных операций.

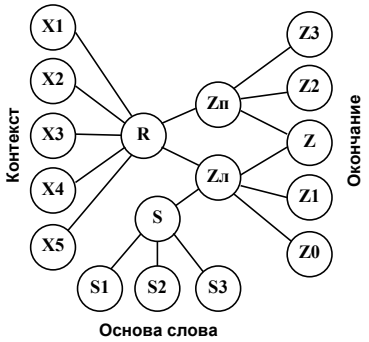


Рис. 1. Логическая сеть анализа полных имен прилагательных русского языка

Модель процесса проектирования

Модель процесса проектирования показана на рис. 2. Здесь входное описание – отношения, записанные на языке алгебры предикатов. На этапе компиляции входное описание преобразуется во внутренние структуры данных – логическую сеть. На этапе логического синтеза происходит трансформация логической сети в системы булевых уравнений, пригодных для реализации в аппаратуре. На выходе получается модель на языке VHDL в виде списка межсоединений логических и RTL элементов.

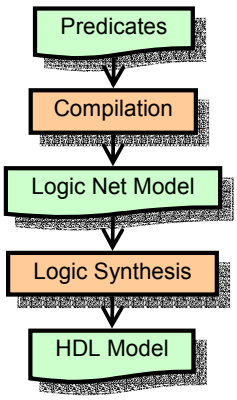


Рис. 2. Модель процесса Проектирования

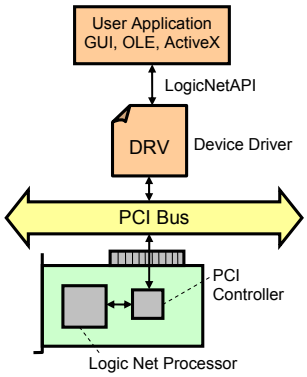


Рис. 3. Взаимодействие прикладного ПО с логической сетью

Использование логической сети, расширяющее интеллектуальные возможности персонального компьютера, показано на рис. 3. Взаимодействие осуществляется следующим образом: процессор логической сети имеет высокоуровневый доступ к шине посредством PCI контроллера. С другой стороны, драйвер устройства имеет программный интерфейс пользовательских приложений (LogicNetAPI), например, компоненты в текстовом процессоре, системе автоматического перевода, распознавания текста или речи.

Экспериментальные результаты

Для анализа эффективности аппаратной реализации разработана и протестирована программа моделирования логической сети на компьютере Intel Pentium IV 2,4 GHz, 256MB DDR RAM. Результат моделирования составляет в среднем 4000 слов в секунду.

В таблице ниже показаны характеристики аппаратной реализации логической сети в различных семействах перепрограммируемых кристаллов Xilinx.

Таблица

Характеристики аппаратной реализации

Part	Slices, %	Luts, %	I/OBs,%	Freq, MHz
xc2s100-6-fg256	24	22	68	47
xcv50-6-fg256	37	34	68	45
xcv100-6-fg256	24	22	68	43
xcv200-6-fg256	12	11	68	41
xc2v250-6-fg256	18	17	71	63
xc2v500-6-fg256	9	8	71	65
xc2v8000-5-ff1152	<1	<1	14	43
xc2vp2-5-fg256	20	18	87	71
xc2vp4-5-fg256	9	8	87	57
xc2vp7-5-fg456	5	5	49	58

Цикл моделирования одного слова в FPGA составляет от трех до семи тактов (включая загрузку начальных значений и генерацию сигнала ready). В самом худшем случае (f = 43 MHz) скорость моделирования составляет $43 \cdot 10^6/7 \approx 6 \cdot 10^6$ слов в секунду, что в $6 \cdot 10^6/4000 \approx$ ≈ 1500 раз быстрее, чем программная реализация.

Выводы

Модель любого механизма языка и мышления описывается с помощью системы отношений, каждое из которых выражается некоторой формулой алгебры предикатов. Их можно представить некоторой схемой, которая без труда реализуется существующими средствами радиоэлектроники. В работе приведены основные принципы построения мозгоподобных ЭВМ параллельного действия. Представлен новый подход к созданию искусственного интеллекта: интеллект человека рассматривается как некоторое материальное воплощение механизма логики. Были выполнены работы по алгебраизации логики. Представлена модель логической сети для полных непритяжательных имен прилагательных русского языка, ориентированная на аппаратную реализацию. Проведен анализ производительности реализации модели в кристалл FPGA по сравнению с программной на компьютере последовательного действия. Выигрыш в производительности вычислений достигается за счет параллельной обработки данных. Программная реализация лишена этого преимущества, поскольку на компьютере Фоннеймановской архитектуры операции в ребрах сети осуществляются последовательно. Выигрыш в производительности будет тем выше, чем выше параллелизм модели.

Литература

1. *Шабанов-Кушнаренко Ю.П.* Теория интеллекта. Харьков: Вища школа, 1984, 1986, 1987. 440 с.

2. Логическая сеть как технология моделирования естественного языка. Шабанов-Кушнаренко Ю.П., Хаханов В.И., Процай Н.Т. и др. // Информационные технологии – в науку и образование. Харьков, ХНУРЭ, 21–22 марта 2005. С. 30–33.

СПОСОБ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ КЛАСТЕРНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

А.А. Потапов

Пензенский государственный университет

Введение

При создании кластерных систем высокой доступности необходимо эффективно решать задачу балансировки вычислительной нагрузки.

Балансировка нагрузки (load balancing) призвана обеспечить равномерную загруженность вычислительных узлов системы подзадачами, обеспечив тем самым максимальную скорость выполнения задачи в целом. При этом необходимо учитывать рейтинги производительности узлов кластера. Подсистема рейтинговой балансировки занимает центральное место в системе управления кластером. Эффективность ее работы определяет эффективность исполнения задачи в целом.

Методика распределения нагрузки

Этап I (выполняется в статике):

1. Производится декомпозиция большой задачи на множество мелких подзадач.

2. Выполняется анализ полученного множества на предмет выявления взаимозависимостей по данным.

3. Предпринимаются попытки реорганизации параллельных алгоритмов, с целью повышения их эффективности путем избавления от избыточных взаимозависимостей.

4. По результатам анализа (шаг 2) и реорганизации (шаг 3) подзадач из общего множества формируются подмножества подзадач. В каждое подмножество войдут задачи, которые должны исполняться последовательно.

Этап II (выполняется в статике):

1. Производится ранжирование подзадач путем вычисления трудоемкостей их выполнения. Результаты представляют собой числовые значения в условных единицах.

2. Осуществляется оценка производительности вычислительных узлов (нахождение рейтингов). Результат представляет собой числовое значение, которое отражает ресурс узла в тех же условных единицах, что и трудоемкость подзадач.

3. Определяется начальная (статическая) стратегия балансировки, например, round-robin [3].

Этап III (выполняется в динамике):

1. В систему поступает подзадача. Диспетчер, согласно выбранной на втором этапе (шаг 3) стратегии, выбирает подходящий узел и передает ему подзадачу на выполнение. Получивший подзадачу узел теряет часть своего ресурса (уменьшая текущий рейтинг узла) на время выполнения подзадачи.

2. Диспетчер динамически корректирует стратегию выборки в соответствии с изменившимся распределением ресурсов. После выполне-

ния подзадачи диспетчер отправляет результаты по месту запроса.

3. Момент поступления подзадачи на выполнение и момент завершения ее обработки фиксирует монитор производительности, тем самым, вычисляя реальное время обработки подзадачи и корректируя значение трудоемкости выполнения конкретной задачи для конкретного узла с учетом его текущей загрузки.

Таким образом, согласно вышеприведенной методике, на первых двух этапах кластер подготавливается к работе, на третьем этапе – работает. Вышеизложенная методика позволяет с течением времени уточнять значение трудоемкости выполнения каждой подзадачи на конкретном узле. В случае перегрузки кластера, когда все узлы исчерпали свой ресурс, стратегия балансировки может изменяться, например, на least-connection [3]. Кроме того, по мере загрузки узла подзадачами, монитор производительности выполняет перерасчет значения производительности узла, сверяет его с рассчитанным в статике, и при необходимости корректирует.

Рассмотрим более подробно методы и способы, применяемые в методике.

Этап I, шаг 1,2,3 – декомпозиция крупной задачи на более мелкие выполняется высокоуровневыми способами распараллеливания (coarse granularity) [1], такими как:

- уровень отдельных заданий (task level);
- уровень подпрограмм (procedure level).

Реорганизация параллельных алгоритмов выполняется по методикам, изложенным в [2].

Этап I, шаг 4 – формирование подмножеств подзадач из основного множества. Разделение подзадач на подмножества выполняется исходя из двух принципов:

- независимости используемых ими данных. То есть, если две подзадачи используют несвязанные между собой данные, то эти подзадачи попадают в различные подмножества;типа производимых операций. То есть, если две подзадачи выполняют:
- операции чтения или добавления над одним и тем же объектом, то они попадают в разные подмножества. Если подзадачи выполняют операции записи или модификации, или удаления одного и того же объекта – они попадают в одно подмножество.

Подзадачи, входящие в состав различных не пересекающихся подмножеств, могут выполняться параллельно. Подзадачи, принадлежащие одному подмножеству (или общей части пересечения с другим

подмножеством), должны выполняться только последовательно. В случае несоблюдения предыдущего требования возможно возникновение конфликтных ситуаций, когда, например, одна подзадача пытается записать результаты в объект, в то время как выполняется процедура его удаления.

Эффективность выполнения задачи напрямую связана с количеством и мощностью подмножеств. Если подмножеств много и их мощность мала, то эффективность высока. Примерами крайних случаев являются две ситуации:

1. Когда подмножество одно и в нем присутствуют все подзадачи – в этом случае задача является последовательной и ее решение на кластерной системе будет несколько менее эффективно (из-за накладных расходов), как если бы она выполнялась на одном, самом производительном узле кластера.

2. Когда количество подмножеств равно количеству подзадач и мощность каждого подмножества, следовательно, равна 1 – в этом случае имеет место идеальное распараллеливание задачи на абсолютно независимые ветви, и производительность в этом случае будет определяться реальной производительностью кластера.

Этап II, шаг 1 – Ранжирование подзадач выполняется при помощи методики, основанной на представлении алгоритма вычислительного процесса в виде Марковской модели [4]. В результате применения методики каждая подзадача получит числовое значение трудоемкости ее исполнения в условных единицах.

Этап II, шаг 2 – Оценка производительности вычислительных узлов производится одним из стандартных контрольно-оценочных тестов: Linpack, Perfect, Spec_89, Spec_92, Spec_SDM, Spec_SFS, AIM. Данные тесты не привязаны к системе команд определенного процессора, поэтому позволяют проводить оценку производительности любой ЭВМ классического типа, кроме гетерогенных вычислительных систем. В результате каждый узел получает эквивалент его производительности в тех же условных единицах, что и подзадачи.

Этап II, шаг 3 – Выбор начальной стратегии балансировки осуществляется из числа общеизвестных статических стратегий: round-robin, least-connection [1] и т.д.

Этап III, шаг 1,2,3 – Работу в динамике рассмотрим на примере, конкретизируя структуру балансировщика следующим образом: подсистема балансировки состоит из двух компонентов: диспетчера задач и монитора производительности.

Монитор производительности обладает распределенной архитектурой и представляет собой систему агентов, каждый из которых выполняется на отдельном узле кластера. Каждый агент выполняет две основные функции:

а) на этапе включения узла в состав кластера формирует оценку производительности узла в условных единицах и сохраняет это значение в системные таблицы диспетчера;

б) в процессе функционирования узла фиксирует время, фактически затраченное на выполнение подзадачи и формирует оценку степени загрузки узла с сохранением в системные таблицы диспетчера.

Таким образом, главная задача монитора производительности – формирование оценок производительности узла и сохранение их в системные таблицы диспетчера для анализа диспетчером задач.

Диспетчер задач также как и агенты монитора производительности располагается на каждом узле кластера по экземпляру. Однако в отличие от агентов, активный экземпляр диспетчера находится лишь на одном узле, все остальные экземпляры пребывают в состоянии ожидания и активизируются только в случае выхода из строя текущего активного экземпляра. Таким образом, в каждый момент времени функционирования системы, только один диспетчер задач выполняет координацию работы узлов кластера. Диспетчер задач состоит из нескольких структурных частей: буфер промежуточного хранения, ядро диспетчера (выполняет основные функции) и системные таблицы диспетчера (хранят данные о работоспособности узлов кластера, например, оценки). Сервер удаленного доступа выполняет прослушивание порта доступа. При появлении запроса сервер принимает его и устанавливает в очередь запросов, отправляя сообщение диспетчеру. Диспетчер, получив сообщение сервера о поступлении нового запроса, просматривает очередь запросов и извлекает поступившую заявку, помещая ее в буфер временного хранения. Затем он обращается к своим системным таблицам, анализирует текущие оценки производительности, сформированные монитором производительности. Далее, согласно действующей стратегии балансировки, диспетчер выбирает наиболее оптимальный узел и передает на него поступившую заявку из буфера временного хранения. Поступившая на узел заявка выполняется под контролем агента монитора производительности.

Заключение

В настоящее время на базе вычислительного центра ПГУ осуществляется практическая реализация подсистемы балансировки и монито-

ринга производительности. Полученное ПО управления кластером будет использовано для эффективного решения задач управления ВУЗом [5].

Литература

1. *Гергель, В.П.* Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие / В. П. Гергель, Р. Г. Стронгин – Нижний Новгород: Изд-во ННГУ, 2003. – 184с.

2. *Антонов, А.В.* Эффективная организация параллельных распределенных вычислений на основе кластерной технологии. Диссертация на соискание ученой степени кандидата технических наук. Пенза, 2005. – 172 с.

3. *Othman, O.*, *O'Ryan, C.*, *Schmid,t D.C.* Strategies for CORBA middleware-based load balancing // IEEE DS Online, 2001. V. 2, № 3.

3. *Князьков В.С.*, *Потапов А.А*. Методика оценки трудоемкости реализации матричных мультипроцессорных систем. – Пенза, Изд-во ПГУ, Сб. тез. АПНО–2003. Т. 2. 2003. – С. 400–402.

4. *Потапов, А.А.*, *Попов, К.В*. Корпоративные информационные системы в информационной образовательной среде. Ж. «Педагогическая информатика» №3.

ИСПОЛЬЗОВАНИЕ ЧИСЕЛ РАСШИРЕННОЙ ТОЧНОСТИ В РЕАЛИЗАЦИИ ИНДЕКСНОГО МЕТОДА ПОИСКА ГЛОБАЛЬНО-ОПТИМАЛЬНЫХ РЕШЕНИЙ

А.В. Сысоев, С.В. Сидоров

Нижегородский государственный университет им. Н.И. Лобачевского

Процессы принятия решений играют важную роль во многих сферах производственной деятельности человека. Одной из типичных моделей таких процессов является модель в виде задачи оптимизации. В общем случае эта модель представляет собой набор функционалов, определенных на пространстве параметров, с помощью которых описывается ситуация принятия решения. При этом целью часто является глобальный оптимум одного – в однокритериальном случае – или нескольких – в многокритериальном – функционалов. Нередко также найденное решение должно удовлетворять некоторым заданным функциональным условиям, которые понимаются как ограничения. Типичным способом решения указанных задач являются различные численные методы, основанные на итерационных процедурах вычисления входящих в задачу функционалов в точках области поиска, выбираемых согласно некоторым правилам.

Простейший подобный метод состоит в вычислении функционалов задачи во всех точках некоторой чаще всего регулярной сетки в области изменения параметров. Однако с ростом размерности, то есть с увеличением числа учитываемых параметров, описывающих ситуацию принятия решения, использование «полного перебора» становится бесперспективным в силу экспоненциального роста числа точек расчета.

Более эффективные методы поиска обычно основываются на некоторой дополнительной информации о характере входящих в задачу функционалов (непрерывность, дифференцируемость, липшицевость, …). При этом часто имеющая место существенная вычислительная трудоемкость функционалов требует для нахождения оптимума при реалистичных временных затратах использования параллельных вычислений в программных реализациях этих методов.

Одним из таких эффективных методов поиска глобального оптимума в многомерных многоэкстремальных задачах является индексный метод [1], важное достоинство которого состоит в возможности создания эффективной параллельной модификации [2]. Вместе с тем при использовании аппаратно поддерживаемых вещественных типов данных реализация индексного метода [3] сталкивается с некоторыми существенными ограничениями. В данной работе обсуждается характер этих ограничений, а также пути их преодоления в виде использования чисел расширенной точности.

Необходимость использования чисел расширенной точности

Индексный метод позволяет находить глобально-оптимальное решение в многомерных задачах оптимизации с ограничениями. При этом одним из основных идейных моментов метода является использование редукции размерности на основе кривых Пеано, в результате чего итерации метода фактически выполняются на отрезке [0, 1] вещественной оси, а для вычисления значений функционалов осуществляется построение образа точки. Проблема состоит в том, что для достижения высокой точности решения по каждой координате в исходном N-мерном пространстве на отрезке [0, 1] требуется точность в 2^N раз большая. Таким образом, если за m обозначить точность построения развертки (то есть погрешность найденного решения по каждой координате будет составлять $1/2^{m+1}$), то на отрезке [0, 1] потребуется точность в 2^{Nm}.

При использовании для хранения точки испытания типа double, мантисса которого имеет 52 бита, максимально возможная точность

ограничена условием $Nm < 52$. Учитывая, что минимальным значением m, дающим разумную точность в исходном N-мерном пространстве является 10, применение прямой реализации индексного метода ведет к ограничению на размерность задачи порядка 5.

Выходом из указанной ситуации является использование чисел расширенной (произвольной) точности для представления точек на отрезке [0, 1].

Проблемы использования чисел расширенной точности

Возможности работы с числами расширенной точности реализованы во многих библиотеках. В частности в работе были проанализированы такие библиотеки как GMP, MAPM, WinNTL. Общим недостатком всех подобных библиотек является уменьшение на два-три порядка скорости выполнения даже простейших арифметических операций в сравнении с типом double, что было подтверждено численными экспериментами. Прямое использование одной из указанных библиотек в реализации индексного метода привело к замедлению процесса поиска глобального оптимума в 500 раз. Следующим естественным шагом стало максимально возможное уменьшение использования чисел расширенной точности в операциях метода.

Использование чисел типа double для вычисления дробных степеней от расстояний между точками

Одной из широко используемых и весьма трудоемких операций в индексном методе является операция возведения в дробную степень, а именно в степень $1/N$ возводится расстояние между двумя точками из отрезка [0, 1]. Как будет показано ниже, данная операция может быть выполнена с использованием типа double без существенной потери точности.

Пусть Ext – тип данных с расширенной точностью.

Рассмотрим пример,

```
Ext m = 0.999999999999999999; // любое "боль-
шое" число цифр
double d, res1, res2;
res1 = m.pow(1.0 / 10.0).toDouble();
res2 = pow(m.toDouble(), 1.0 / 10.0);
```

Зададимся вопросом, насколько значение res1 отличается от res2?

Рассмотрим числа, целая часть которых не превышает 9, доказательство обобщается и на противный случай, но оно менее наглядно.

Представим рациональное число x в виде:

$$x = \sum_{i=0}^{\infty} a_i \cdot 10^{-i}.$$

После преобразования $x_1 = x$.toDouble получим:

$$x_1 = \sum_{i=0}^{14} a_i \cdot 10^{-i}.$$

Возведя x и x_1 в степень $(1/N)$ и взяв логарифм от отношения степеней, получим:

$$\log(\rho_1/\rho_2) = \frac{1}{N} \cdot \log\left(1 + \sum_{i=15}^{\infty} a_i \cdot 10^{-i} \Big/ \sum_{i=0}^{14} a_i \cdot 10^{-i}\right),$$

где

$$\rho_1 = (x_1)^{1/N} = \left(\sum_{i=0}^{\infty} a_i \cdot 10^{-i}\right)^{1/N}, \quad \rho_2 = (x_2)^{1/N} = \left(\sum_{i=0}^{14} a_i \cdot 10^{-i}\right)^{1/N}.$$

Далее, нетрудно показать, что

$$\mathrm{Log}\,(\rho_1/\rho_2) = \frac{1}{N}\mathrm{Log}\,(1 + 0{,}00000000000001/1{,}0) = \frac{1}{N} \cdot \mathrm{Log}\,(1 + (1/10)^{14}).$$

Взяв от обеих частей экспоненту и разложив правую часть в ряд Тейлора, получим, что максимальная разница между ρ_1 и ρ_2 не превышает 10^{-14}. Таким образом, вычисление дробных степеней в типе double не ухудшает качество вычисления расстояний.

Результаты

Рассуждения, показанные в предыдущем разделе, позволяют все операции возведения в дробную степень в реализации индексного метода выполнять в типе double, что обеспечивает существенный выигрыш в скорости. С учетом указанной замены вычислений удалось добиться замедления работы индексного метода с использованием чисел расширенной точности в 10 раз по отношению к реализации, использующей только тип double. При этом ограничение $Nm < 52$ превращается в $2^{Nm} < 10^{308}$ (10^{-308} – минимальное число, представимое в типе double), что дает примерную оценку $Nm < 1000$.

Выполненная реализация была протестирована на модельных задачах, в частности функции Растригина с размерностью N от 10 до 12 и точностью построения развертки m от 10 до 15.

Работа была поддержана грантом Нидерландской Организации

Научных исследований (NWO) № 047.016.014 и грантом Российского Фонда Фундаментальных Исследований № 04-01-89002-НВО_а.

Литература

1. *Strongin R.G.*, *Sergeev Ya.D.* (2000). Global optimization with non-convex constraints: Sequential and parallel algorithms. Kluwer Academic Publisher, Dordrecht.

2. *Гергель В.П.*, *Стронгин Р.Г.* Параллельные вычисления в задачах выбора глобально-оптимальных решений для многопроцессорных кластерных систем // Современные методы математического моделирования / Сб. лекций Всероссий. молодежной школы международ. конф. «Математическое моделирование». Самара. 2001. С. 46–55.

3. *Sysoyev A.V.* (2004). Program system of parallel computations for solving the tasks of global-optimum choice // VI International Congress on Mathematical Modeling / Book of abstracts. P. 62.

СРАВНЕНИЕ ТРЕХ ПОДХОДОВ К ПОСТРОЕНИЮ ПАРАЛЛЕЛЬНЫХ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ НА ПРИМЕРЕ НЕКОТОРЫХ ЗАДАЧ ФУНКЦИОНАЛЬНОЙ ОПТИМИЗАЦИИ И ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

С.В. Тимченко

Томский государственный университет систем управления и радиоэлектроники

В последние годы при решении различных задач оптимизации и поиска стали широко применяться различные адаптивные процедуры, среди которых особую популярность завоевали эволюционные и, в том числе, *генетические алгоритмы* (ГА) и *эволюционные стратегии* (ЭС). В той или иной мере, они основаны на эволюционных процессах в биологических популяциях, развивающихся поколение за поколением, подчиняясь законам естественного отбора и принципу «выживает сильнейший» (точнее, наиболее приспособленный).

Чрезвычайно важным свойством как ГА, так и ЭС является их естественный внутренний *параллелизм*. При этом, по сравнению с градиентными методами, различие во времени расчета целевой функции при различных значениях ее параметров не оказывает влияния на эффективность распараллеливания (эти времена могут отличаться на порядки – например, если можно определить, что точка в пространстве поиска

не отвечает наложенным ограничениям, не вычисляя целевую функцию или вычисляя ее только частично).

Основная идея большинства параллельных алгоритмов заключается в том, что бы разбить задачу на (сравнительно) небольшие подзадачи и решать их совместно, используя многопроцессорную вычислительную систему. Стратегия «разделяй и властвуй» может быть реализована большим количеством способов. Одни из них лучше подходят для систем с массовым параллелизмом (MPP), другие для небольших параллельных систем.

1. Глобальные ПГА (технология master-slave)

В этом случае существует только одна популяция на master-процессоре, как и в последовательном ГА. При этом остальные процессоры (slave-процессоры или workers-процессоры) используются только для расчета целевых функций. Метод очень просто реализуется (т.к. фактически используется последовательный ГА) и хорош в тех случаях, когда вычислительная часть задачи доминирует над коммуникационной. При построении глобальных ПГА не делается никаких предположений об архитектуре параллельного компьютера. Число индивидуумов, приходящихся на один процессор либо константа, либо может динамически меняться с целью обеспечения равномерности загрузки (например, в многопользовательской среде).

Хотя, как правило, используются синхронные глобальные ПГА, в данной работе рассматривается асинхронный глобальный ПГА. На компьютерах с MIMD архитектурой алгоритм, как правило, состоит в размещении популяции на одном процессоре, рассылки индвидуальностей на workers-процессоры, получение на главном процессоре значений целевой функции и применении генетических операторов для следующего шага.

2. Островная модель (крупно-зернистый ПГА)

В рамках этого подхода на каждом процессоре располагается изолированная сабпопуляция, обменивающаяся особями с другими сабпопуляциями за счет миграции, регулируемой рядом параметров. Каждая сабпопуляция невелика по размеру, поэтому скорость сходимости весьма высока, хотя качество решения оставляет желать лучшего. Существует критическая скорость миграции, позволяющая получить результат, идентичный последовательному ГА. Один из подходов состоит в том, что на каждом шаге отсылается соседям лучшая особь. Однако в этом случае существует опасность преждевременной сходимости к локальному экстремуму.

3. **Клеточная модель (мелко-зернистый ПГА)**

В этом случае используется большое число очень маленьких сабпопуляций (в пределе - одна особь на процесс), которые интенсивно обмениваются индивидуумами. Эта модель наиболее хороша для MPP архитектур, но может применяться на любых мультипроцессорах.

4. Результаты численных расчетов

Проведенные в данной работе исследования показывают, что на системах с относительно небольшим количеством процессоров при решении задач функциональной оптимизации (рассматривались некоторые задачи оптимального профилирования) наиболее оправданным является первый подход к построению ПГА. В тоже время, при решении ряда задач интерпретации расчетных и экспериментальных результатов в рамках парадигмы генетического программирования лучше себя зарекомендовали островная и клеточная модели ПГА.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НАПРАВЛЕННЫХ ОТНОШЕНИЙ

А.М. Толоконников

Московский энергетический институт (*технический университет*)

Введение

Теория направленных отношений [1, 2] является одним из подходов конструктивного построения таких формальных объектов, как функции, предикаты, программы, семантические сети и т.д. В рамках этого подхода каждый из объектов может быть построен из базовых объектов путем операций композиции и оператора наименьшей фиксированной точки. На базе теории направленных отношений создан язык функционально-логического программирования FLOGOL [5] и среда разработки программ, в основе которой графическое представление, разработанное авторами выше указанной теории.

По своей сути отношения встречающиеся повседневно в нашей жизни формализованы как $R^k \subseteq D_1 \times D_2 \times \ldots \times D_k$, где D_i i =1, 2, …, k – множества элементов относящихся к той или иной проблематике. Разделив правую часть (Декартово произведение) на две составляющие и назвав одну входом, а другую выходом, установили соответствие, в общем случае неоднозначное, авторы получили новое понятие, – направленного отношения (НО) $R^{(m,n)} : D'_1 \times D'_2 \times \ldots \times D'_m \rightarrow D''_1 \times D''_2 \times \ldots$

$\times D''_n$, где $D_j^l \in \{D_i \mid i = 1, 2, \ldots, k\}$, где $l \in \{', ''\}$, $j = 1, 2, \ldots, m(n)$. Такое преобразование позволило представить программный продукт в качестве отношения и оперировать с ним как с отношением.

Не для кого не секрет, что программа (или программный комплекс) представляет собой отношение, но задать ее в формализме теории множеств задача достаточно трудоемкая и бессмысленная, для этой цели за весь период существования вычислительной техники разрабатывались многочисленные языки программирования. Посмотрим на программу или ее часть. Мы видим механизм, устанавливающий соответствие между исходными и выходными данными[6]. Тем самым программа это не что иное, как направленное отношение.

Далее используем выходы одного НО в качестве входов другого, или объединить в теоретико-множественном смысле, и получим новое, необходимое нам направленное отношение. Для этого были введены операции композиции и набор необходимых для построения отношений константных отношений[2]. Остановимся немного на базовых операциях композиции НО.

Операции композиции направленных отношений

Пусть R_1 и R_2 – направленные отношения.

$R_1 \cdot R_2 \equiv \{(\alpha,\beta) \mid \exists\gamma\ ((\alpha,\beta) \in R_1\ \&\ (\beta,\gamma) \in R_2)\}$[7] – операция последовательной композиции. Иллюстрация выполнения операции на примере двух отношений представлена на рис. 1. Эта операция позволяет использовать выходы одного отношения в качестве входов другого[8]. Понятно, что данная операция, на первый взгляд, не обладает свойством, позволяющим параллельно вычислять отношения R_1 и R_2 в рамках нового НО. Но учитывая то, что R_1 и R_2 – это не функции, как мы привыкли считать при программировании с помощью процедурно-ориентированных языков, где на сегодняшний день используется функцио-

[6] Нельзя исключать случай, когда исходные и/или выходные данные отсутствуют. Например “программа паразит” ничего не получающая в качестве исходных данных, и ничего не возвращающая в качестве результата; просто занимающая ресурсы вычислительной системы.

[7] В силу требовании к оформления к оформлению тезисов использования шрифта Times New Roman и отсутствия знака «равно по определению», здесь и далее знаком ≡ будем обозначать термин «равно по определению».

[8] Для использования в качестве аргументов НО разной арности по входам и выходам необходимо применение константных НО [2]

нальная или объектная (некоторая модификация функциональной) парадигмы, то в этом случае операция свойству (назовем его свойством параллельности операции) при котором R_1 и R_2 могут вычисляться независимо. Заметим, что на конкретном наборе входных значений, при вычислении (для простоты, можно понимать перечислительную процедуру) мы получим несколько наборов выходных значений и не обязательно конечное. Это свойство направленного отношения дает нам не только больше возможностей при написании программ, но и возможность параллельного вычисления, что при функциональном подходе просто не имеет смысла.

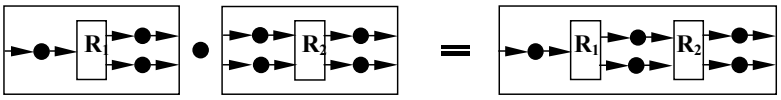


Рис. 1

$R_1 \# R_2 \equiv \{(\alpha_1\alpha_2, \beta_1\beta_2) \mid (\alpha_1, \beta_1) \in R_1 \ \& \ (\alpha_2, \beta_2) \in R_2\}$ – операция параллельной композиции. Иллюстрация выполнения операции на примере двух отношений представлена на рис. 2. Эта операция позволяет получить новое отношение входами и выходами которого являются входы и выходы исходных отношений. При выполнении этой операции необходимо учитывать последовательность соединения входов(выходов). Очевидно, что данная операция обладает свойством параллельности, которое заложено в самом определении операции. Операции подобные приведенным выше используются и в функциональных языках (в том числе [1]) , позволяя задавать функции конструктивно.

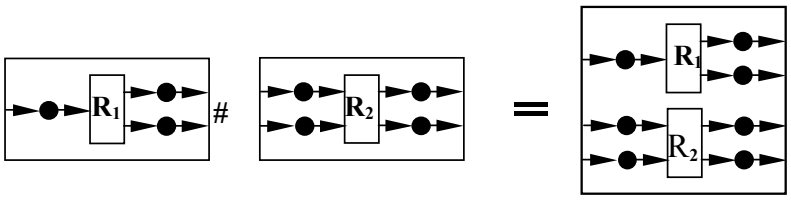


Рис. 2

$R_1 \cup R_2 \equiv \{(\alpha, \beta) \mid ((\alpha,\beta) \in R_1 \ \vee \ (\beta,\gamma) \in R_2)\}$ – операция объединения направленных отношений. Операция представляет собой объединение в теоретико-множественном смысле. По аналогии с операцией # можно сказать, что данная операция обладает свойством параллельности. Она позволяет нам при программировании задавать неоднозначные соответствия.

Оператор наименьшей фиксированной точки. Пусть задана система в общем случае рекурсивных уравнений:

$$\{X_i = \tau_i(X_1, X_2, ..., X_n, r_1, r_2, ..., r_m),\ i = 1, 2, ..., n, \qquad (1)$$

где X_i – переменные, r_j – элементарные НО, а τ_i – термы построенные из символов переменных константных НО и операций композиции.

В интерпретации (1) задает первый компонент кортежа НО $X_1^{\min}$, $X_2^{\min}$, ..., $X_n^{\min}$, являющегося минимальной фиксированной точкой для системы уравнений (1), т.е.

$$\{X_i^{\min} = [X_j^{\min} / X_j \mid j = 1, 2, ..., n]\tau_i,\ i = 1, 2, ..., n$$

и для всякого другого кортежа НО X'_1, X'_2, ..., X'_n такого что X'_i = = $[X_j^{\min}/X_j \mid j = 1, 2, ..., n]\ \tau_i$, $i = 1, 2, ..., n$ выполнено $X_i^{\min} \subseteq X'_i$, $i = 1, 2, ..., n$.

В основе оператора лежит понятие подстановки в терм вместо вхождения переменной соответствующего ей терма. Аналогия с процедурно-ориентированными языками – это понятие процедуры (функции) в языке. Поставили в соответствие набору операторов идентификатор (имя процедуры) и используем это имя в других частях программы.

Разница между языками программирования такими, как C или Pascal, без которых не обходится как правило большинство разработок программного обеспечения, и данным, даже не расширенным в сторону общепринятых типов данных, заключается лишь в том, что здесь присутствуют средства параллельного вычисления, заложенные в самом языке. Нельзя не отметить, что и в C и в языке Pascal были сделаны попытки включить средства распараллеливания, но на сегодняшний день программист вынужден использовать средства предоставляемые операционными системами (ОС) для задания параллельных вычислений. Трудоемкость создания параллельных программ распараллеливания средствами ОС – это большой минус для процедурных языков программирования. Как расширения данного языка можно предложить использование того же C, Pascal и других языков (не обязательно процедурных) в качестве реализации элементарных отношений r_i (i = 1, 2, …, m) в (1).

Далее рассмотрим расширение, где в качестве элементарных от-

ношений выступают конструкторы[9]. Это расширение позволяет работать не просто с данными, структура которых нас не интересует, а с данными в структуре которых наглядно отражен процесс их порождения (однозначно определен). Что позволяет абстрагироваться от конкретных значений. Еще одним плюсом данного расширения является, что и программа и данные представляются одинаково, на основе выбранного расширения. Остается решить вопрос: «Что значит вычислить НО? Программа и данные представляются одинаково. И можно утверждать, что все результаты получены?»

Модель вычислений

В основе модели вычислений, разработанной В.П. Кутеповым и В.Н. Фальком, лежат правила редукции сетей, отражающих фундаментальные свойства используемых при их построении элементарных отношений (в нашем случае конструкторов).

Основным требованием предъявляемым к модели является корректность вычислений. Далее учет и реализация параллелизма при вычислениях. В практическом плане – это создание среды параллельных вычислений, которая позволяла бы реализовывать параллелизм описанного выше языка на распределенных вычислительных системах.

Что значит вычислить НО R?

Первое и самое простое, как упоминалось выше, – это перечислительная процедура. Вычислить – это значит получить все значения наборов данных, принадлежащих НО. Подобного рода задачи и сейчас реализуются, по ошибке человека, по злому умыслу, экспериментально и т.п. Подобный подход имеет право на жизнь и забывать про него не стоит. Следующее, что можно предложить в качестве вычислений – это задать входной набор $a_1, a_2, ..., a_m$ и определить существует ли выходной набор $b_1, b_2, ..., b_n$. Данную постановку можно сформулировать следующим образом $\exists y_1 \exists y_2 ... \exists y_k (a_1 a_2 ... a_m y_1 y_2 ... y_n) \in R$. Эта задача является узкой, т.к. всегда интересует, какие значения будут на входе, если заданы значения на выходе.

Поставим задачу более обще. Пусть заданы значения $a_1, a_2, ..., a_k$ части входов и части выходов. Определить существуют ли значения не заданных частей входов и выходов таких, что набор сформированный из значений принадлежал R. С формальной точки зрения постановка

[9]Конструктор – это НО определенное как $c_i \equiv \{(\alpha_1 \alpha_2 ... \alpha_{ni}, c_i \alpha_1 \alpha_2 ... \alpha_{ni}) | \alpha_j \in \in D_C^{\ni}, j \in 1..n_i\}$

задачи может быть записана следующим образом. Пусть $R^{(m,n)}(x_1x_2...x_m, y_1y_2...y_n)$ что (m,n)-арное НО и aj_1, aj_2, ..., aj_l – заданные значения части входов и выходов $jp \in \{1, 2, \ldots, m + n\}$, $p = 1, 2, \ldots, l$. Т.е. определен набор $(x'_1x'_2...x'_m, y'_1y'_2...y'_n)$, где

$$x'_i = \begin{cases} a_i \in A, \text{ если } x_i \text{ заранее задано,} \\ t_i, \text{ если } x_i \text{ требуется найти,} \end{cases}$$

$$y'_i = \begin{cases} a_{k+m} \in A, \text{ если } y_k \text{ заранее задано,} \\ t_{k+m}, \text{ если } y_k \text{ требуется найти,} \end{cases}$$

$$i = 1, 2, \ldots, m, k = 1, 2, \ldots, n.$$

Определить $\exists t_{j1}\exists t_{j2}...\exists t_{jl}$ $(x'_1x'_2...x'_m, y'_1y'_2...y'_n) \in R$, где $jr \in \{1, 2, \ldots m + n\}$, $r = 1, 2, \ldots, l$.

Для вычисления НО в данной постановке, как упоминалось выше разработана модель. В ее основе лежит операция подстановки термов вместо вхождения переменных соответствующих термам и выполнение правил редукции, приведенных ниже. Правила базируются на локальных трансформациях термов и учитывает такие фундаментальные свойства конструкторов как функциональность, тотальность и др. Данный набор редукций обладают свойством Черча-Россера, т.е. гарантируют получение результата вычислений, если последний существует.

1. Однозначность конструкторов по входам (функциональность), которая выражается равенством $—\langle\bullet(c_i\#c_i) = c_i\bullet—\langle$.

2. Однозначность по выходам (функциональность обратного конструктора): $(c_i\#c_i)\bullet \rangle— = c_i\bullet\rangle—$.

3. Тотальность конструктора: $c_i \bullet —\rangle = —\rangle$.

4. Ортогональность различных конструкторов: $(c_i\#c_j)\bullet \rangle— = \varnothing$[10].

5. Если в сети существует цикл, состоящий только из конструкторов (т.е. в терме существует множество конструкторов и хотя бы один выход каждого конструктора соединен со входом другого), то терм начинающийся с крайне левого и кончающийся крайне правым конструктором заменяется на $\varnothing$.

6. Для полноты картины доопределим правила редукции для операции композиции с аргументом $\varnothing$. Пусть t – терм.

– $t \bullet \varnothing = \varnothing \bullet t = \varnothing$;

[10] $\varnothing$ - нигде не определенное значение

– $t \# \varnothing = \varnothing \# \ t = \varnothing$;
– $t \# \varnothing = \varnothing \# \ t = \varnothing$;
– $t \cup \varnothing = \varnothing \cup t = t$.

Встает вопрос, в какой последовательности применять правила, если нет возможности их применения одновременно. На этот вопрос попытаемся ответить в следующем разделе.

Процесс и стратегии вычислений

В общем случае процесс вычисления НО представляет собой процесс преобразования терма запроса, состоящего как было сказано выше из символа переменной с частью заданных входов и выходов. Сам же процесс вычислений может рассматриваться как процесс развертывания терма в результате подстановки, и процесса свертывания терма с использованием правил редукции.

Рассмотрим несколько стратегий подстановки. Наиболее простая в реализации –это стратегия «полной подстановки». Шаг вычислений в соответствии с этой стратегией представляет: 1) подстановку во все вхождения переменных, термов соответствующие этим переменным; 2) редукцию терма полученного после подстановки и анализ на наличие результата в терме.

Вторая стратегия несколько отличается от первой тем, что подстановка ведется во все вхождения переменных терма, не сразу всех термов соответствующих этим переменным, а вначале только термов, состоящих из конструкторов. Затем идет редукция и анализ на наличие результата. Если результат не достигнут, то осуществляем отложенные подстановки. Эта стратегия ориентирована на подстановку термов которые могут привести к результату на текущем шаге подстановки.

Более сложной с точки зрения реализации, но более эффективной является стратегия, использующая независимость переменных (отсутствие взаимно рекурсивных определений (1)). Этот вопрос будет освещен в следующих публикациях.

Заключение

Рассмотрена модель процесса вычислений направленных отношений. Приведены стратегии, позволяющие более эффективно (с точки зрения времени вычислений) решать задачи.

Литература

1. *Кутепов В.П.*, *Фальк В.Н*. Функциональные системы: теоретический и практический аспекты // Кибернетика, №1. 1979.

2. *Кутепов В.П.*, *Фальк В.Н.* Направленные отношения: теория и приложения // Техническая кибернетика, №4, №6. 1994.

3. *Стерлинг Л.*, *Шапиро Э.* Искусство программирования на языке Пролог / Пер. с англ. М.: Мирб 1990.

4. *Вагин В.Н.*, *Головина Е.Ю.*, *Оськин Ф.Ф.* Модели и методы представления знаний в CASE-технологии. // Интеллектуальные системы. Т. 2. Вып. 1-4. М.: Изд. центр РГГУ, 1997. С. 115–134.

5. *Фальк В.Н.* FLOGOL – входной язык системы функционально-логического программирования, сборник научных статей КНТК МИРЭА (ТУ), 1998. С. 1–10.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА РАСЧЕТА ДИНАМИКИ ПЛАЗМЫ В ПЛАЗМЕННОМ ПРЕРЫВАТЕЛЕ ТОКА

А.В. Толстобров

Московский физико-технический институт
(*государственный университет*)

Введение

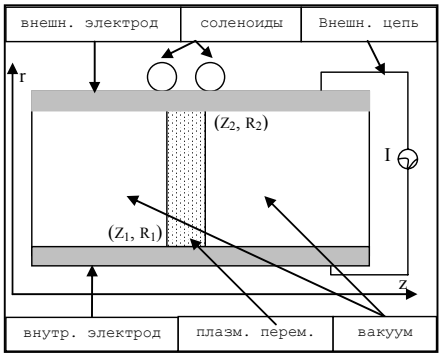
В ходе экспериментального изучения динамики высокотемпературной плазмы существенную роль играет характер проникновения магнитного поля в плазму. Для теоретического исследования протекающих при этом процессов широко применяется численное моделирование.

В ряде теоретических и экспериментальных работ было показано, что в режиме электронной магнитной гидродинамики (когда существенен эффект Холла) поле быстрее проникает в плазму вдоль анода. Образующиеся при ЭМГ-диффузии магнитного поля токовые петли могут приводить к локальному нагреву плазмы и повышению давления вблизи анода – так называемому «анодному взрыву». Обзор эффектов ЭМГ и теоретическое обоснование основных уравнений содержатся в [1].

Цель настоящей работы – численное моделирование динамики плазмы в установке РС-20, созданной в РНЦ «Курчатовский институт». Ввиду того, что при моделировании динамики плазмы в подобных установках существенную роль играют быстрые мелкомасштабные эффекты, необходимо использовать подробные расчетные сетки и малые шаги по времени. Это ведет к необходимости использования параллельных вычислений и построения хорошо распараллеливаемого чис-

ленного метода для использования на системах с распределенной памятью.

Постановка задачи. Установка РС-20 представляет собой систему из двух коаксиальных цилиндрических электродов, окруженных кольцевыми соленоидами прямоугольного сечения, создающими постоянное магнитное поле. Размеры плазменной камеры: длина – 9 см, радиус внутреннего электрода – $R_1 = 5$ см, радиус внешнего – $R_2 = 8$ см. Электроды подключены к электрической цепи, содержащей генератор импульса тока (генератор Маркса). В начальный момент времени плазма занимает следующее пространство между электродами: $R_1 < r < R_2$, $Z_1 < z < Z_2$ (см. рисунок). Ширина плазменной перемычки, расположенной в центре — $Z_2 - Z_1 = 1$ см. Концентрация электронов в плазме составляет $1 \cdot 10^{13}$ частиц (и равна концентрации ионов), концентрация ионов и электронов в вакууме составляет $1 \cdot 10^{13}$ частиц. Начальная температура плазмы 1 эВ, вакуума — 0,5 эВ, температура стенок поддерживается постоянной и равна 0,1 эВ.



Математическая модель. Существенную роль ЭМГ эффекты играют при моделировании проникновения магнитного поля в плазму. Здесь задача сводится к решению системы уравнений Максвелла в двумерном случае с учетом эффектов ЭМГ. Задача обладает цилиндрической симметрией, поэтому рассматриваемые величины не меняются по направлению φ.

Задача решается методом расщепления по физическим процессам. На первом этапе рассматривается магнитогидродинамическая система без учета диссипативных эффектов. На втором этапе решается система для определения электрических и магнитных полей и токов в плазме, также учитывается джоулев нагрев. На последнем этапе проводится

коррекция электронной температуры за счет теплопроводности (с учетом эффектов ЭМГ) и температуры электронов и ионов за счет электрон-ионных обменов.

В рамках гидродинамического блока рассматривается следующая математическая модель движения нейтральной плазмы с учетом токовой скорости электронов, что фактически вводит в рассмотрение двухжидкостную плазму. Система уравнений модели, записанная в безразмерных переменных, имеет вид:

$$\frac{\partial n}{\partial t} + \operatorname{div}(n\mathbf{v}) = 0,$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v},\nabla)\mathbf{v} = \frac{1}{An}\nabla P_\Sigma + \frac{1}{An}(\mathbf{j}\times\mathbf{B}),$$

$$\mathbf{v}_e = \mathbf{v} - \frac{1}{eL_0}\sqrt{\frac{m_p}{N_0}}\frac{\mathbf{j}}{n},$$

$$\frac{\partial T_e}{\partial t} + (\mathbf{v}_e,\nabla)T_e + \frac{2}{3}\operatorname{div}\mathbf{v}_e = \operatorname{div}\mathbf{q}_e + Q_{ie} + Q,$$

$$\frac{\partial T_i}{\partial t} + (\mathbf{v},\nabla)T_i + \frac{2}{3}\operatorname{div}\mathbf{v} = + - Q_{ie}.$$

Модель замыкается уравнениями состояния

$$P_e = nT_e, \quad P_i = nT_i, \quad P_\Sigma = P_e + P_i.$$

Здесь приняты следующие обозначения: n – концентрация ионов в плазме, совпадающая в силу ее нейтральности с концентрацией электронов; $\mathbf{v}$ – скорость ионов, $\mathbf{v}_e$ – электронная токовая скорость; $\mathbf{j}$ и $\mathbf{B}$ – плотность тока и напряженность магнитного поля; P_e и T_e, P_i и T_i – соответственно электронные и ионные давление и температура, P_Σ – полное давление; A – атомный вес иона плазмы, e и m_p – заряд электрона и масса протона соответственно, L_0 и N_0 – характерный размер установки и начальная концентрация ионов.

Для обмена энергиями принято следующее описание:

$$Q_{1e} = -3\frac{m_e}{m_i}n_e\frac{T_1 - T_e}{\tau_e}.$$

Электронный тепловой поток учитывается в следующем виде:

$$\mathbf{q}_e = 0{,}7\ln T_e\mathbf{j}_{\|} + \frac{3}{2}\frac{n_eT_e}{\omega_e\tau_e}\frac{[\mathbf{B}\times\mathbf{v}_e]}{|\mathbf{B}|} - \frac{5}{2}\frac{cn_eT_e}{e}\frac{[\mathbf{B}\times\nabla T_e]}{|\mathbf{B}|^2} - \kappa_{e\|}\nabla T_{e\|} - \kappa_{e\perp}\nabla T_{e\perp}.$$

Соответствующими знаками в последнем соотношении обозначены производные температуры в направлении вдоль и поперек магнитного поля. Для продольной и поперечной составляющих коэффициента теплопроводности используются соотношения

$$\kappa_{e\|} = 3{,}16\frac{n_e T_e \tau_e}{m_e}, \quad \kappa_{e\perp} = 4{,}66\frac{n_e T_e \tau_e}{m_e(\omega_e \tau_e)^2}.$$

Разностная схема для вышеприведенных уравнений строилась с помощью интегро–интерполяционного метода [4]. Для уменьшения осцилляций, возникающих при расчете разрывных решений (ударных волн, контактных разрывов или тангенциальных разрывов) к давлению добавлялась линейная искусственная вязкость [4].

Ниже представлены эволюции магнитного поля и выражения для токов и токовых скоростей (этим уравнениям и будет посвящено дальнейшее рассмотрение):

$$\frac{\partial \mathbf{B}}{\partial t} = -c\,\mathrm{rot}\,\mathbf{E},$$

$$\mathrm{div}\,\mathbf{B} = 0,$$

$$\mathbf{j} = \frac{c}{4\pi}\,\mathrm{rot}\,\mathbf{B},$$

$$\mathbf{E} = -\frac{1}{c}(\mathbf{v}_e \times \mathbf{B}) + \frac{\mathbf{j}}{\sigma} - \frac{1}{en_e}(\nabla P_e - \mathbf{R}),$$

где термосмещения

$$\mathbf{R} = -0{,}71 n_e \nabla_{\|} T_e - 1{,}5\frac{n_e}{\omega_e \tau_e}\frac{[\mathbf{B} \times \nabla T_e]}{|\mathbf{B}|},$$

а токовая скорость

$$\mathbf{v}_e = \mathbf{v} - \frac{\mathbf{j}}{en_e}.$$

Здесь $\mathbf{E}$ – напряженность электрического поля, σ – проводимость, T_e – электронная температура, Q – джоулев нагрев, ω_e – плазменная частота, τ_e - время столкновений, c – скорость света, e – заряд электрона.

Выделение тепловой энергии в плазме (в том числе за счет джоулева нагрева)

$$Q = \frac{(\mathbf{j}, \mathbf{j})}{\sigma} + \frac{1}{e}(\mathbf{j}, \mathbf{R})$$

входит в уравнение изменения энергии электронов (за счет нагрева):

$$\frac{3}{2} n \frac{\partial T_e}{\partial t} = Q.$$

На стенках электродов задавалось условие на φ-ую компоненту магнитного поля: $B_\varphi(R_1) = B_\varphi\ (R_2) = 0$, а также условие на тангенциальную компоненту напряженности электрического поля $E_\tau \equiv 0$. На левой границе области $B_\varphi(z = Z_1) = 0$. На правой границе области ($z = Z_2$) магнитное поле связано с полным током во внешней электрической цепи соотношением (в безразмерном виде):

$$B_\varphi = \frac{0{,}2I(t)}{r}, \quad I(t) = I_0 \sin\frac{\pi t}{2T_{1mp}}.$$

Здесь I – ток во внешней цепи, r – текущий радиус точки границы плазмы.

Для задания начального распределения магнитного поля от внешних соленоидов (r и z компоненты магнитного поля) в расчетной области решалось уравнение для векторного потенциала магнитного поля. Сеточное уравнение решалось методом последовательной верхней релаксации [6], оптимальное значение релаксационного параметра подбиралось на основе тестовых расчетов.

При обезразмеривании приведенных выше уравнений в качестве характерных масштабов использовались концентрация электронов в плазме n_{e0}, длина установки L_0, $Te_0 = 1$ эВ, масштаб времени

$$t_0 = L_0 \sqrt{\frac{m_p}{T_{e0}}},$$

масштаб магнитного поля $B_0 = \sqrt{N_0 T_{e0}}$, тока

$$j_0 = \frac{cB_0}{L_0} = \frac{c\sqrt{N_0 T_{e0}}}{\sqrt{N_0 T_{e0}}},$$

электрического поля

$$E_0 = \frac{B_0 l_0}{ct_0} = \frac{T_{e0}}{c} \sqrt{\frac{N_0}{m_p}},$$

проводимости

$$\sigma_0 = \frac{j_0}{E_0} = \frac{c^2}{L_0} \sqrt{\frac{m_p}{T_{e0}}},$$

где m_p — масса протона.

Построение разностной схемы. В силу цилиндрической симметрии рассматриваем сечение рабочей области плоскостью r–z В рассматриваемой области введем прямоугольную сетку (размером i_m x j_m, i – индекс по z), границы которой совпадают с границами области. При принятом способе расщепления по физическим процессам термодинамические величины (T_e, P_e, σ, n_e, $\omega\tau_e$, $\mathbf{v_e}$) и **B** отнесены к центрам соответствующих ячеек (иногда будем обозначать подобные величины с индексом c), потоки — к серединам соответствующих граней, а скорости – к углам ячеек (иногда будем обозначать подобные величины с индексом n). Векторные величины плотности тока **j**, напряженности электрического поля **E** и термосмещений **R** также будем относить к углам ячейки (см. рисунок). Для удобства реализации однородного алгоритма на границах области, для величин, отнесенных к центрам ячеек, введем фиктивные ячейки с номерами 0, $i_{\max}$, $i_{\max}$. Значения указанных величин определим в фиктивных ячейках особо, чтобы удовлетворить граничным условиям. Используем расщепление по физическим процессам. Отдельно рассматриваем перенос магнитного поля (вмороженного в электронное течение) и диффузию магнитного поля за счет конечной проводимости среды.

Основную трудность при численном решении уравнений эволюции полей представляет корректный учет членов $\mathbf{v}_e \times \mathbf{B}$ при расчете электрического поля **E**. Это приводит к появлению в уравнении для **B** гиперболических слагаемых. Гиперболическая часть уравнения эволюции магнитного поля решается численно с помощью монотонного варианта сеточно-характеристического метода [1] (явного, первого порядка аппроксимации). Для решения параболической части (диффузионные члены) используется схема Алена–Чена.

Оценка эффективности распараллеливания. Теоретические оценки показали пригодность выбранной схемы для эффективного распараллеливания расчетов на кластере из персональных компьютеров (гомогенной системе). Наиболее целесообразным является применение статического варианта геометрического распараллеливания (расчетная область делится между различными процессами с необходимостью обмена данными на границах подобластей) с выделением зон ответственности процессоров по радиальному направлению. Подобное деление зон ответственности обеспечивает равномерную загрузку элементов кластера.

Расчет одного временного слоя для последовательного варианта программы (при размере расчетной сетки 73×73) на компьютере с про-

цессором Intel 3GHZ занимает в среднем 22 мс. При использовании 14 таких процессоров в сети 1 Gbps с пакетом MPI время обмена информацией для одного временного слоя на такой же расчетной сетке составляет примерно 215 мкс. Из них 59 мкс уходит на обмены границами (~17 кб), а 156 мкс уходит на сбор данных для вычисления нового значения шага по времени (для следующего слоя). Таким образом, теоретическое ускорение работы параллельного варианта программы на кластере из 14 компьютеров (процессор Intel 3 Ghz, сеть 1 Gbps, коммуникационный пакет MPI) составляет 7,8 раза. Реальное ускорение во время проведения расчетов следует ожидать в диапазоне 5–6 раз из-за невозможности обеспечения идеальной сбалансированности работы процессоров при выборе статического варианта распараллеливания. Учитывая, что серийный расчет в последовательном варианте на сетке 73×73 до 4 мкс длится 6 суток (без внешних полей), 13 суток с подключенными внешними полями, создание параллельной версии представляет собой актуальную задачу.

Результаты. Был отлажен последовательный вариант программы и проведена серия расчетов; проведены тестовые расчеты для параллельного варианта. Получены следующие результаты расчетов: при отключенных внешних полях плазменная перемычка постепенно размывается, масса начинает движение от центра к нагрузке с последующим отражением ударных волн от стенки. При включенных внешних полях (выполняющих стабилизирующую роль) “размывание” плазменной перемычки происходит более медленно, перемычка принимает S-образную форму (с максимумами на электродах).

Литература

1. *Кингсепп А.С.*, *Чукбар К.В.*, *Яньков В.В*. Электронная магнитная гидродинамика // Вопросы теории плазмы. М.: Энергоатомиздат, 1987. Вып. 16. С. 209–250.

2. *Вихрев В.В.*, *Забайдулин О.З.* Проникновение магнитного поля в плазму вдоль границы двух сред из-за эффекта Холла. // Физика плазмы. 1994, т. 20, № 11. С. 968–972.

3. *Магомедов М.-К.М.*, *Холодов А.С.* Сеточно-характеристические численные методы. М.: Наука, 1988. 290 с.

4. *Самарский А.А., Попов Ю.П.* Разностные методы решения задач газовой динамики. М.: Наука, 1980. 392 с.

5. *Dongarra J.*, *Foster J.*, *Fox G.*, *Gropp W.*, *Kennedy K.*, *Torczon U.*, *White A.* Sourcebook of parallel computing. – Elsiver Science, 2003. – 842 p.

6. *Деммель Д.* Вычислительная линейная алгебра. М.: Мир, 2001.

ПОСТРОЕНИЕ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ НА БАЗЕ ЛВС ПРЕДПРИЯТИЯ С ИСПОЛЬЗОВАНИЕМ GRID ТЕХНОЛОГИЙ

Ю.А. Ушаков

Оренбургский государственный университет

В настоящее время широкое распространение получила идея использования ресурсов компьютеров во время простоя. Такое направление получило название Grid технологий. Основа заключается в использовании ресурсов компьютеров любой сети, когда они простаивают. Ядром такой вычислительной сети является сервер, который координирует выполнение задачи и распределяет части задачи между машинами. Фактически эту структуру можно назвать кластером, но на самом деле есть много различий.

Каждая машина вправе отказать в приеме задания, если она занята или пользователь запретил в данное время выполнение задач. Во-вторых высокопроизводительной сетевой инфраструктуры не требуется, ведь сеть используется для посылки задания и возвращения результата, а не для постоянного обмена информации. Для высокопаралельных алгоритмов технология grid может дать выигрыш в стоимости машинного часа по сравнению с суперкомпьютером до 2 раз. Да, производительность сегмента grid много меньше суперкомпьютера при малом распараллеливании. Но для поточных алгоритмов производительность может быть увеличена почти до бесконечности, ограничиваясь только производительностью сервера и сети. Это определяет более высокую степень масштабируемости системы. Примером такой технологии может быть всероссийская сеть взлома паролей НТВ+. Более 10 000 компьютеров включено в эту программу, большинство из них – домашние пользователи. Они, работая в Интернете даже иногда, и не знают, что их компьютер используется для вычисления паролей НТВ+. Ни один суперкомпьютер не может похвастаться наличием 10 000 процессоров с автономной памятью. И, конечно же, взломом хеша пароля за 6 дней. Конечно, не все процессоры всегда доступны, но их количество определяет надежность такой сети. Если учесть, что сам сервер является такой же сетью grid, то отказоустойчивость системы и ее готовность будет просто потрясающей.

Но есть и существенные недостатки GRID структур. Время ответа отдельной машины может меняться в широких пределах, она может без уведомления покинуть сеть, выгрузить задание и т.д. Также при

просчете больших автономных пакетов заданий сеть будет очень нагружена при передаче и приеме заданий. Например, если распределено кодировать фильм. Сеть из 20 компьютеров, по идее, должна перекодировать фильм в 20 раз быстрее, чем 1 компьютер минус транспортные расходы. Но, так как объем задания велик, то при сети 100 мбит на 20 компьютеров задание будет уходить со скоростью 5 мбит на компьютер, то есть при размере файла 4,7 ГБ это теоретически 8 минут. Возвращение задания тоже потребует около 2 минут. Зато процесс кодирования займет всего 5 минут против 100 на 1 компьютере. То есть на этом задании выгода достигает 600 процентов. И это при том, что не надо ни настраивать ОС, ни устанавливать ПО, достаточно в автозагрузке прописать 1 файл, запускающий по сети клиента. Для сетей организаций, где занимаются в основном учебной или канцелярской деятельностью это реальная экономия денег на серверах вычислений. Особенно, если это какой либо проект по обработке экспериментальных данных. Обычно такие задачи очень ресурсоемки, однако, специальные алгоритмы позволяют итерационно разбивать на блоки и распараллеливать вычисления.

Но есть один критический момент в высоконагруженных grid-сетях (при больших объемах передаваемой информации). Если каналы данных имеют малую скорость но и приемлемое время ответа, нецелесообразно использовать TCP/IP протоколы, которые до 30 процентов трафика используют на служебные цели. Тут можно использовать магистральную технологию ATM, которая, ко всему прочему позволяет более экономно расходовать широковещательный трафик. Но, если корпоративные пользователи могут позволить себе ATM (просто установив требуемые драйверы ATM over Ethernet) и использовать , например, голосовой шлюз, то домашние пользователи не имеют поддержку ATM со стороны провайдера. Тут приходят на помощь поточные технологии вещания. Можно последовательно вещать в сеть задание, части которого помечены маркером и иметь отдельный поток маркеров, показывающий текущую часть. Каждый компьютер имеет свой маркер и ждет только свою часть, не принимая все остальное. Маркеры он принимает все. Такая технология используется при спутниковой связи.

Другой способ повышения эффективности - это локализация трафика заданий в одной рабочей группе. Один из клиентов становится «прокси-сервером» для заданий, то есть сохраняет несколько пакетов для разных компьютеров, являясь при этом также активным клиентом.

При такой организации трафик локализуется в рабочих группах намного больше времени, чем на магистрали, что влечет более полное использование локальной сети и разгрузку магистрали.

Единая операционная среда, созданная на базе сети Grid, может быть использована для большого круга вычислительных, и не только, задач. Структурные преобразования, использование других протоколов, высокая степень параллельности алгоритмов может повысить (или понизить) эффективность системы до 50 процентов. Сложность первоначальной настройки и интеграции средств разработки с grid системами, невозможность запуска традиционных приложений в этой системе делают пока ее не такой распространенной, как кластеры, но использования кроссплатформенных виртуальных машин (технологии PVM) позволяют решать эти проблемы. Пока еще нет широко известных пакетов для полной интеграции всех этих средств со средами разработки типа Delphi или C++. Однако, простота работы с клиентами и масштабируемость таких систем не могут не подкупать перспективами.

К сожалению, очень сложно написать программу, эффективно работающую на 100 или более процессорах. Для проектирования алгоритмов можно применять UML средства для параллельного программирования, но генерация кода тоже будет нелегка. Библиотеки параллельного программирования эффективно не справляются даже с 50 процессами, языки параллельного программирования базирующиеся на C++, тоже требуют особых навыков параллельного программирования. Только последние реализации Fortran, незаслуженно не используемого в России в последние годы, могут похвастаться более простой и понятной реализацией языка для параллельных вычислений. Fortran успешно загружал все 50 тестовых компьютеров при обработке экспериментальных данных объемом 10 Гб со сложными математическими вычислениями рядов Фурье и Колмогорова. В тестовом стенде использовались 2 рабочие группы, 50 компьютеров Celeron 1.7, 256 ОЗУ с ОС Windows 2000 professional, сеть 100 Мбит, 1 главный сервер. Для поддержки сети GRID использовалось обеспечение Condor 6.6.10. Общее время распределения заданий с лимитом размера 100 мб на компьютер за 1 раз было равно 21 минуту, при этом первые результаты начали поступать уже через 1 минуту. И начали формироваться следующие задания по обработке уже полученных результатов. В результате весь тест был закончен за 31 минуту при средней загрузке процессора 60-80 процентов и памяти 30-40 мбайт. На выделенном сервере 2xXeon 2.6, 8 Гб ОЗУ обработка происходила 82 минуты при загрузке процессоров

100% и памяти 6 Гб. Выигрыш составил более 2 раз. И это при том, что компьютеры не изымались из учебного процесса и мало кто заметил нагрузку.

На рис. 1 показан график производительности сети GRID, на рисунке 2 – критическая точка, когда транспортные расходы превышают выигрыш в производительности. Однако, при большом превышении времени транспортирования над временем работы выигрыш производительности сводится на нет уже к 12-ому компьютеру (рис. 2). При повышении трудоемкости задания в 100 раз нужно уже в 2 раза больше компьютеров, а для гарантированного взлома пароля (на 1 компьютере около 140000 лет) за неделю нужно более 1 000 000 компьютеров. И это при минимальной сетевой загрузке. Такие суперсистемы уже существуют, например проект НАСА для расшифровки звездного шума, взлом НТВ+ и т.д.

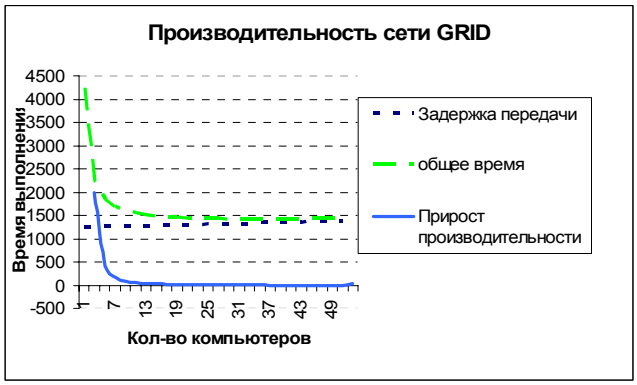


Рис. 1. График производительности сети GRID при 100 мбит сети

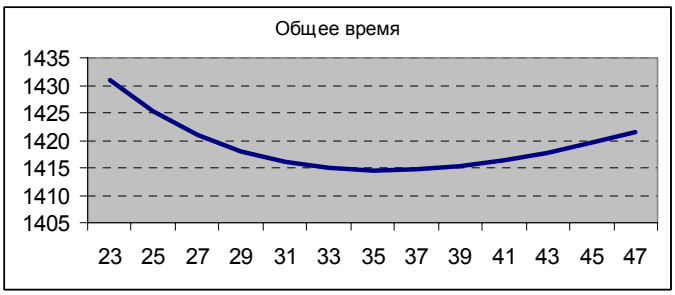


Рис. 2. Критическая точка

МОДЕЛИРОВАНИЕ СТОЛКНОВЕНИЙ АВТОМОБИЛЕЙ НА МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

С.К. Черников, А.Н. Ашихмин, А.М.Файзуллин

Казанский физико-технический институт

Введение

Проектирование кузова автомобиля, удовлетворяющего в полной мере современным требованиям обеспечения пассивной безопасности – сложная инженерная задача. Решение этой задачи только экспериментальной доводкой конструкции практически невозможно из-за высокой стоимости и отсутствия в нужном количестве натурных образцов, особенно на ранних стадиях проектирования, а также из-за большого количества параметров, влияющих на результаты. При этом натурный эксперимент часто может дать лишь конечные характеристики разрушения конструкции при отсутствии информации о характере протекания процессов деформирования. Использование численных методов лишено отмеченных недостатков и позволяет оперативно исследовать изменения конструкции с целью поиска наиболее рационального варианта. При этом конструкция автомобиля рассматривается как совокупность большого количества конечных элементов.

Конечные элементы

Модель автомобиля обычно строится с использованием многих типов элементов: 3-х мерных элементов для массивных сплошных тел типа двигателя, оболочечных элементов для тонколистовых частей, из которых преимущественно состоит кузов, балочных элементов для частей, поведение которых допустимо описывать балочной теорией, а также так называемых дискретных элементов – пружин, демпферов и масс.

Пожалуй, основным требованием к используемым в описываемых задачах конечным элементам является их высокая вычислительная эффективность. Она достигается, во-первых, простотой используемых функций формы (линейных для 3-х мерных элементов и билинейных для оболочечных) и, во-вторых, снижением порядка интегрирования (как правило, используется одноточечное интегрирование). Упрощенное описание поверхностей элементов снижает также сложность решения задач, связанных с контактным взаимодействием. Вычислительной эффективности, кроме всего прочего, способствует использование диагональных матриц масс. За использование высокоэффективного одно-

точечного интегрирования приходится расплачиваться вероятностью появления при расчете так называемых форм с нулевой энергией. В иностранной литературе данное явление носит название hourglassing из-за визуального сходства структуры деформированной сетки с формой песочных часов (англ. «hourglass»). Для борьбы с этим явлением разработаны достаточно эффективные алгоритмы, идея которых состоит во введении в рассмотрение системы нефизичных дополнительных сил упругости или вязкости, которые с одной стороны гасят hourglass-деформации элементов, а с другой не влияют на их реальные деформации. Тем не менее, эти силы совершают некоторую работу, которая вычитается из общей энергии системы, и величину которой надо контролировать на всем ходе решения.

Геометрическая и физическая нелинейность

Для учета геометрической нелинейности обычно используется модифицируемое описание Лагранжа (updated Lagrangian), при котором опорное состояние обновляется на каждом шаге. Благодаря малому размеру шага по времени при явной схеме интегрирования, а, следовательно, и малых изменениях деформаций на шаге, допустимо использовать линейную зависимость приращений деформаций от приращения перемещений или, что эквивалентно, линейные тензоры скоростей деформаций.

Малая величина приращений деформаций на шаге позволяет также довольно просто вводить в рассмотрение произвольные законы деформирования материалов. Существует несколько теорий, описывающих пластическое поведение металлов, но, как показывают эксперименты, лучше всего согласуются с экспериментальными данными результаты расчетов по теории пластического течения.

Контакт

При ударном деформировании конструкций значительную роль играет контактное взаимодействие. При решении контактной задачи необходимо выполнение двух условий – механических условий контакта и условия непроникновения. Первые заключаются в том, чтобы (а) контактирующие точки имели одинаковое перемещение и скорость в направлении, перпендикулярном к поверхности контакта, (б) силы на контактирующих поверхностях были самоуравновешенными (нормальные силы и силы трения равны на обеих сторонах контакта) и (в) на контактных поверхностях не должно возникать растягивающих напряжений по направлению нормали к поверхности контакта. Для вы-

полнения условий непроникновения и вычисления контактных сил в коммерческих пакетах используется, как правило, метод штрафов из-за относительной простоты его реализации и малого влияния на величину необходимого шага интегрирования по времени. Большое влияние на эффективность решения имеет выбор алгоритма поиска контакта. Из всего многообразия существующих на сегодня алгоритмов [1], стоит выделить работы Холквиста [2] и Жонга [3], как получившие практическую реализацию в реальных программных продуктах. Для определения тангенциальных сил на поверхностях контакта было предложено множество законов трения (см. например [4] или [5]), однако по причине того, что шероховатость поверхности при реальном контакте можно оценить только весьма приближенно, для большинства задач достаточна классическая кулоновская модель трения.

Интегрирование по времени

В общем виде уравнение равновесия системы конечных элементов, находящихся в состоянии движения можно записать так:

$$M\ddot{u}(t) + C\dot{u}(t) + Ku(t) = R(t), \qquad (1)$$

где M, C и K соответственно матрицы масс, демпфирования и жесткости; **R** – вектор внешней узловой нагрузки; u, $\dot{u}$ и $\ddot{u}$ – векторы узловых перемещений, скоростей и ускорений ансамбля конечных элементов. Уравнения (1) получены из рассмотрения равновесия в момент времени t, когда согласно принципу Д'Аламбера тело находится в равновесии под действием внешних сил, внутренних сил, сил демпфирования и сил инерции.

Математически (1) представляет собой систему дифференциальных уравнений второго порядка с переменными коэффициентами (в случае учета физической и геометрической нелинейности). Для адекватного моделирования объектов типа автомобиля требуется использовать достаточно мелкую сетку, что ведет к очень большому размеру участвующих в решении матриц. Еще в 90-е годы стали обычными задачи с размерностью порядка 200 000 неизвестных. Сейчас счет уже идет на миллионы неизвестных. Отсюда очевидны высокие требования к эффективности используемой схемы интегрирования. В настоящее время с этой стороны наилучшим образом зарекомендовала себя схема прямого явного интегрирование по времени методом центральных разностей. Алгоритмически она реализуется использованием численной пошаговой процедуры. При этом равновесие рассматривается в дис-

кретных точках временного интервала, что позволяет эффективно использовать весь вычислительный аппарат статического анализа. Для дискретного момента времени t система (1) разрешается относительно ускорений:

$$\ddot{u}_t = M^{-1}[R_t - Ku_t - C\dot{u}_t]. \quad (2)$$

Скорости и перемещения для следующего шага рассчитываются по формулам:

$$\begin{aligned} \dot{u}_{t+\Delta t/2} &= \dot{u}_{t-\Delta t/2} + \ddot{u}_t \Delta t; \\ u_{t+\Delta t/2} &= u_t + \dot{u}_{t+\Delta t/2} \Delta t. \end{aligned} \quad (3)$$

Обновление скоростей в моменты времени, сдвинутые на половину шага по времени улучшает точность и сходимость решения. Необходимые для вычисления ускорений $\ddot{u}_t$ скорости $\dot{u}_t$ в момент времени t можно найти с использованием разного вида экстраполяций [6], в коммерческих пакетах обычно реализован простейший способ – $\dot{u}_t$ принимается равной $\dot{u}_{t-\Delta t/2}$.

Главным недостатком метода центральных разностей является его условная устойчивость. Шаг интегрирования Δt должен быть меньше критического значения, Δt_{cr}, вычисляемого исходя из инерционных и жесткостных свойств всего ансамбля элементов. В линейных задачах для получения достоверного решения необходимо выполнение условия

$$\Delta t \le \Delta t_{cr} = \frac{T_n}{\pi}, \quad (4)$$

где T_n – наименьший период собственных колебаний ансамбля конечных элементов; n – порядок системы. Таким образом, необходимо избегать появления в модели элементов с очень малой массой или очень большой жесткостью. В случае наличия того или иного типа нелинейности считается достаточным уменьшить шаг по времени до величины 0,8-0,9Δt_{cr} [7]. В реальных задачах при длительности ударного импульса 50-100 мск шаг интегрирования по времени составляет порядка 1 мкс.

Расчетные случаи

В настоящее время требования к пассивной безопасности автомобиля прописаны достаточно четко и подробно, и в то же время не прекращается периодическая их корректировка и ужесточение, как в части имеющихся расчетных случаев, так и введением новых. Основной иде-

ей такого рода нормативных документов является задание условий того или иного динамического воздействия на конструкцию автомобиля, а также перечень контролируемых параметров, по которым оценивается пассивная безопасность, и диапазон их допустимых значений. При этом моделируются такие сценарии как лобовое столкновение (прямое и кософронтальное), удар сзади, удар сбоку другим автомобилем или удар бортом о столб ограждения и т.п. Существует также ряд специальных испытаний, моделирующих столкновение с пешеходом.

Программные средства

Описанные выше алгоритмы и методы решения реализованы на сегодня в нескольких коммерческих программных продуктах. Сейчас в автомобилестроении при решении задачи моделирования краш-тестов наиболее широко используется имеющий почти 30-летнюю историю развития комплекс LS-DYNA фирмы Livermore Software Technology Corp. Пакет MSC.Dytran фирмы MSC.Software применяется главным образом в авиакосмической отрасли, а применение обладающего рядом уникальных возможностей пакета PAM-CRASH фирмы ESI-Group ограничено в основном европейским рынком. Нами был использован конечно-элементный пакет LS-DYNA, как хорошо зарекомендовавший себя при решении сложных задач моделирования высоконелинейных быстротекущих процессов.

Аппаратные средства

Стремление получить более достоверную информацию при численном анализе столкновения автомобилей приводит к использованию все более и более громоздких (насчитывающей 1 млн. и более неизвестных) моделей, вызывающих необходимость увеличения мощности применяемых для этой цели вычислительных устройств. Элегантным решением наращивания мощности вычислительных средств является кластерная технология.

Авторами доклада для решения задач механики методом конечных элементов разработан и изготовлен вычислительный кластер АРКО-9А2200 (рис. 1), представляющий собой 9-процессорную вычислительную систему, построенную на базе процессоров AMD Athlon. Кластер включает в себя один двухпроцессорный и семь однопроцессорных вычислительных узлов, объединенных с помощью 8-ми портового Gigabit Ethernet-коммутатора. Все элементы кластера смонтированы в открытой стойке. На однопроцессорных узлах установлена операционная система Windows 2000 Professional SP3, на двухпроцессорном узле

установлена операционная система Windows Server 2003 Enterprise Edition SP1. В качестве коммуникационной среды используется библиотека MPICH 1.2.5 свободно распространяемая Техническим университетом г. Аахен, Германия.

Рис. 1

Производительность кластера определялась при решении системы уравнений утилитой Linpack пакета PLAPACK в зависимости от размерности решаемой задачи, точности вычислений и степени оптимизации вычислений под тип используемого процессора. При использовании всей доступной оперативной памяти максимальная производительность кластера составляет 15,05 гигафлоп для четырехбайтовой точности решения и 8,11 гигафлоп для восьмибайтовой точности. Пиковая производительность кластера (сумма производительностей всех узлов кластера) при использовании для вычислений 8 узлов составит 17,84 гигафлоп для четырехбайтовой точности решения и 11,04 гигафлоп для восьмибайтовой.

Пример

В качестве примера использования описанного программно-аппаратного комплекса приведем результаты расчетного моделирования кософронтального удара с перекрытием 40% в деформируемое препятствие на скорости 64 км/ч автомобиля «Ока». Расчетная модель автомобиля включала модель кузова, силового агрегата и ходовой части, двери задка и запасного колеса. В расчетную модель не включались

боковые двери и капот. Их масса считалась сосредоточенной в узлах навески. Для учета влияния боковых дверей на деформации кузова ограничивались взаимные перемещения некоторых точек дверных проемов. Панели кузова, подрамник и колеса моделировались конечными элементами оболочки в формулировке Belytschko-Tsay. Массы ряда агрегатов и полезная нагрузка (массы водителя, пассажиров, багажа и пр.) считались сосредоточенными в узлах. При моделировании элементы подвески (рычаги, пружины и амортизаторы) заменялись жесткими балочными элементами. Для моделирования двигателя, КПП и ступиц с тормозами использовался сплошной 3D-элемент. Всего модель автомобиля насчитывала более 46000 элементов. Вид конструкции в деформированном состоянии после удара показан на рис. 2.

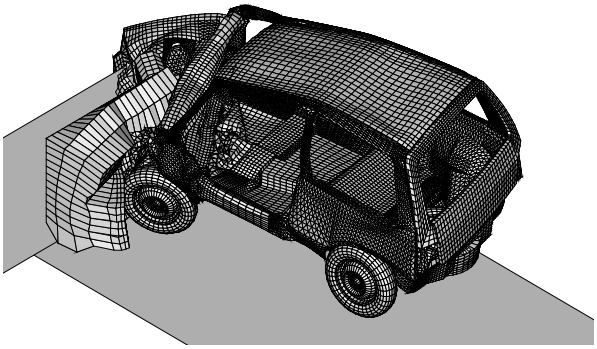


Рис. 2

На рис. 3 приведено время решения задачи (t) в зависимости от количества использованных узлов (N) кластера.

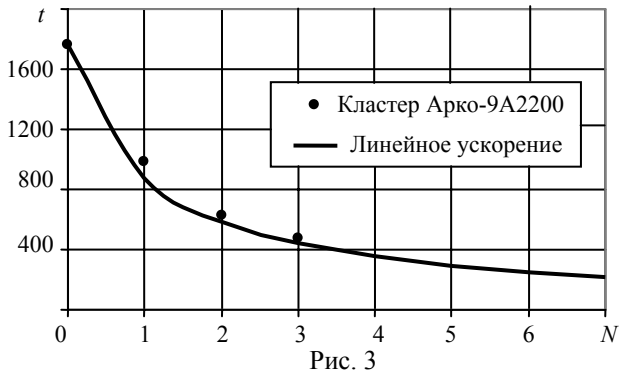


Рис. 3

Сплошная линия соответствует линейному ускорению.

Литература

1. *Бураго Н.Г.*, *Кукуджанов В.Н*. Обзор контактных алгоритмов// МТТ, 2005, №1. С. 45–87.

2. *Hallquist J.O.* (1977) A numerical procedure for three-dimensional impact problems. Proceedings of the ASCE Fall Convention, San Francisco. P. 17–21.

3. *Zhong Z.H.* (1988) On contact-impact problems, doctoral thesis, Div. Solid Mechanics and Strength of Materials. Dept. Mech. Eng., Linkoping University, Linkoping.

4. *Oden J.T. and Martins J.A.C*. (1985) Models and computational methods for dynamic friction phenomena, *Comp. Meth. App. Mech. Eng.* P. 527–634.

5. *Oden J.T. and Pires E.B.* (1983) Algorithms and numerical results for finite element approximations of contact problems with non-classical friction laws, *Computers and Structures,* 19. P. 137–147.

6. *Park K.C. and Underwood P.G.* (1980) A variable-step central difference method for structural dynamics analysis. Part I. Theoretical aspects. *Computer Methods in Applied Mechanics and Engineering*, 22:241–258.

7. *Belytschko T. and Mullen R.* (1977) Explicit integration of structural problems, in Finite *Elements in Nonlinear Mechanics,* (ed.) by P. Bergan *et al.* P. 672–720.

T# - СРЕДА ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ ПЛАТФОРМЫ MICROSOFT .NET

А.М. Чудинов

ИПС РАН, Переславль-Залесский

Введение

Платформа .NET содержит богатые и гибкие библиотеки работы с потоками обеспечивающие эффективную реализацию многопоточных приложений. Однако эти библиотеки могут быть использованы лишь на симметричных мультипроцессорах (SMP) имеющих очень ограниченную масштабируемость. Для более крупных систем широко известных как SMP кластеры приходтся исользовать другие подходы, которые в основном более сложны в реализации и требуют существенного изменения кода последовательных программ.

Данная работа представляет расширение платформы .NET (T#) реализующее концепцию автоматического динамического распаралле-

ливания программ. Расширение сочетает простоту реализации с хорошей масштабируемостью. Параллельных программ, что может существенно облегчить использование кластеров и открыть кластерную архитектуру для целого ряда новых пользователей и приложений.

Концепция автоматического динамического распараллеливания программ

Все современные технологии распараллеливания призваны автоматизировать переход от программы, записанной на языке, удобном для человека, к параллельному алгоритму, который при соответствующей аппаратной поддержке позволяет получить выигрыш в производительности по сравнению с последовательным выполнением аналогичной программы на монопроцессоре.

В силу сложности этой задачи, на пути распараллевания возникают те или иные проблемы, и никем пока не предложен универсальный способ для их решения. Различные подходы разнятся своими ключевыми идеями, которые в совокупности и обуславливают сильные и слабые стороны каждой конкретной технологии, и как следствие этого, имеют ту или иную область своего эффективного применения.

В данной работе рассматривается подход автоматического динамического распаралеливания программ, при котором процесс вычислений представляется в виде графа, вершинами которого являются вызванные функции, а ребрами – отношения «поставщик-потребитель» между вызванными функциями.

Заметим, что для обеспечения детерминированости вычислений, необходимо выполнение требования отсутствия сторонних эффектов у вызываемых функций. Для таких функций можно реализовывать вызов по состоянию и мемоизацию. Под вызовом по состоянию понимается то, что результатом работы функции может быть некоторая фиктивная величина, которая означает, что над глобальными данными произведено некоторое преобразование. Мемоизация – возможность запоминать результаты вычислений функций и затем при повторных вызовах от одних и тех же аргументов просто брать соответствующее значение из таблицы.

В качестве вычислительной модели при этом может быть использован алгоритм параллельной редукции графов При редукции графа представляющего вычисление, в силу прозрачности ссылок, присутствующие в этом графе вершины будут вычисляться с одинаковым результатом независимо от времени и места вычисления. Это позволяет вычислять такие вершины одновременно на отдельных процессорах,

каждый из которых может сгенерировать новые процессы.

Наиболее явно преимущества описаного подхода проявляются на задачах, в которых:

- априорно (до начала счета) неизвестно, как распараллелить работу;
- вычислительная схема близка к функциональной модели, то есть может быть эффективно представлена с помощью совокупности функций, рекурсивно вызывающих друг друга.

T# - система автоматического динамического распараллеливания для платформы Microsoft.NET

В данной работе рассматривается реализация расширения платформы Microsoft.NET, реализуещего описаный выше подход.

Модель вычисления. В процессе описания T# мы будем использовать следующие понятия:

Поставщик – метод возвращающий данные.

Потребитель – метод использующий данные полученые от *поставщика*.

Отношение «поставщик-потребитель» – связь между двумя методами один из которых (*поставщик*) возвращает данные, а второй (*потребитель*) использует их.

Граф вычислений – ориентированый граф, вершинами которого являются *поставщики* и *потребители*, а ребрами – *отношения «поставщик-потребитель».*

Заметим, что поставщик может имет несколько потребителей. При установлении отношения «поставщик-потребитель» потребитель получает *неготовое значение.* При вычислении вершины графа обращение к неготовому значению ведет к приостановке вычисления данной вершины. Когда поставщик заканчивает вычисление, он информирует всех потребителей, при этом возобновляется приостановленое вычисление соответвующих вершин.

Синтаксис и семантика. T# использует следующие основные структуры:

- T-метод
- T-объект

Синтаксически T# обеспечивает работу с данными структурами за счет использования .NET аттрибутов:

```
public class MyObj {[parallel]
virtual public object m1() {…}}
```

Т-метод описывается с помощью указания атрибута [parallel], при этом существует ограничение – все Т-методы должны быть виртуальными, а возвращаемое значение должно иметь ссылочный тип данных.

Т-объекты создаются прозрачно в виде результатов возвращаемых Т-методами.

При вызове Т-метода создается новый узел графа вычислений, при этом вычисление текущей вершины не останавливается, возвращается Т-объект с неготовым значением.

При попытке Т-метода обратиться к переменной содержащей неготовое значение процесс вычисления останавливается, до тех пор, пока значение не станет готовым.

Создание объекта содержащего Т-методы должно осуществляться с помощью вызова метода CreateInstance служебного объекта TProxyFactory:

```
   MyObj obj =
(MyObj)TProxyFactory.CreateInstance(typeof(MyObj),
new object[]{});
```

,где первый аргумент метода – тип создаваемого объекта, а второй – аргументы конструктора.

Ниже приведен пример программы вычисляющей числа фиббоначи (неэффективная реализация):

```
   public class Fib {
      [Parallel]
      public virtual object Fib(int n) {
            if(n <= 1) {
                  return n;
            } else {
                  object n1 = Fib(n-1);
                  object n2 = Fib(n-2);
                  return (int)n1 + (int)n2;
            }
      }
   }
   public class CMain {
   static void Main(string[] args)  {
            Fib f =
(Fib)TProxyFactory.CreateInstance(typeof(Fib), new
```

```
object[]{});      Console.WriteLine("Result: " +
(int)f.Calc(5));
      }
   }
```

Архитектура и реализация T#. Архитектура T# состоит из нескольких уровней:

1. **Пользовательский уровень –** представляет собой T# программы.

2. **Прокси-уровень** – скрывает от пользовательского уровня уровень суперструктуры.

3. **T-суперструктура** – содержит описание основных T-структур и реализует T-семантику (алгоритм параллельной редукции графа вычисления программы, логику работы с готовыми и неготовыми значениями)

4. **T-runtime (среда исполнения)** – содержит низкоуровневые детали реализации, такие как транспортный уровень, планировщик задач.

Пользовательский уровень

Состоит из атрибутов:

[parallel] – для описания T-методов

[memoizable] – помечает методы к которым можно применять мемоизацию

И интерфейса доступа к Прокси-уровню, с единственным методом обеспечивающем создание экземляров объектов содержащих T-методы

```
   TProxyObject.CreateInstance(System.Type instan-
ceType, object[] constructorArguments)
```

Прокси-уровень

Обеспечивает прозрачный доступ из Пользовательского уровня к уровню T-суперструктуры. Содержит функциональность создания прокси-объектов, для произвольного .NET типа. Это достигается за счет использования библиотеки System.Reflection.Emit, которая позволяет динамически создавать MSIL код в процессе работы приложения. Процесс создания прокси-объекта проходит следующим образом:

- создается новый тип данных пронаследованый от базового типа;
- для каждого виртуального метода помеченого атрибутом [parallel] создается метод двойник переопределяющий базовый и вместо непосредсвенного его выполнения формирует T-метод используя функ-

циональность уровня Т-суперструктуры;

• создает и возвращает пользовательскому уровню экземпляр нового типа (прокси-объект).

Т-суперструктура

Содержит описание основных Т-структур и реализует Т-семантику (алгоритм параллельной редукции графа вычисления программы, логику работы с готовыми и неготовыми значениями).

Структура представляющая собой вызов Т-метода. Во время обращения к ней создает и запускает Т-поток.

```
   public struct MethodCall {
            public MethodInfo methodInfo;
            public object target;
            public object[] parameters;
           public object retVal;
           public MethodCall(MethodInfo mi, object
t, object[] p, object r);
   }
```

Реализует концепцию Т-потока – не привязаную к конкретной реализации. Для непосредвенной инициации вычисления использует функциональность T-runtime. По завершении вычислений информирует об этом все ждущие Т-объекты.

```
   public abstract class AMThread {
      public static AMThread Current;
      public MethodCall MethodCall;
      abstract public void Activate();
      abstract public void Suspend();
      abstract public void Resume();
      abstract public void Run();
      abstract public void Join();
   }
```

Т-объект реализует в себе логику работы с неготовыми значениями, синхронизации вычислений.

```
   public class TObject {
      public object Value;
      public bool IsReady = false;
      public bool IsLocked = false;
```

```
      public AMThread ParentThread;
      public AMThread ProducerThread;
   }
```

Уровень T-runtime (среда исполнения)

Содержит низкоуровневые детали реализации T#. Доступен другим уровням посредством следующего интерфейса

```
   public abstract class TRuntime {
            private static TRuntime _currentRuntime
= null;

            public static TRuntime CurrentRuntime {
                  get {
                        return _currentRuntime;
                  }
            }

            public abstract void Initialize();
            public abstract void EnqueueMethod-
Call(MethodInfo mi,
           object target, object[] parameters, ob-
ject retVal);
            public abstract object Wait(ref object
o);
   }
```

Ключевыми являются методы EnqueueMethodCall и Wait.

Метод EnqueueMethodCall реализует выполнение T-метода в контексте текущей среды исполнения, передавая вызов метода планировщику задач, который в последствии разместит метод на соответствующем узле кластера используя Транспортный уровень.

Метод Wait реализует механизм синхронизации вычислений.

Транспортный уровень

В текущей версии T# существуют два варианта транспортного уровня:

- основаный на потоках – полезен при изначальной отладке программы на одном компютере;
- основаный на .NET Remoting – реализует выполнение методов в пределах кластера. .NET Remoting реализует большую часть низко-

уровневых деталей транспортного уровня. Однако, есть возможность и переопределить часть реализации транспортного протокола, например создать свой алгоритм сериализации данных (Formatters) и реализацию Каналов (Channels).

Помимо этого возможна реализация транспортного уровня на основе Веб-сервисов, для развертывания T# в распределенных гетерогенных сетях.

Планировщик задач

Реализован ленивый алгоритм планировки задач:
Имеются 3 очереди.

```
   // Задачи ожидающие запуска (глобальная очередь)
   public Queue PrenatalTasks = new Queue();
   // Активные задачи текущего узла
   static public Queue RunningTasks = new Queue();
   // Приостановленые задачи (обратившиеся к него-
товым значениям)
   public Hashtable WaitingTasks = new Hashtable();
```

По завершении текущей задачи на каком-либо из узлов, планировщик выбирает следующую задачу по следующему принципу:

- если очереди активных и пренатальных задач пусты – это означает что новых задач на выполнение нет, а все текущие приостановлены. В этом случае планировщик ожидает пока в какой-либо из очередей не появятся задачи;
- если эти очереди не пусты, то впервую очередь запускаются задачи из очереди активных задач на текущем узле;
- в случае если очередь активных задач пуста, планировщик выбирает задачу из очереди пренатальных задач и помещает ее в очеред локальных активных задач;
- в случае если выполняемая задача приостанавливается в связи с обращением к неготовому значению, она перемещается в очередь приостановленых задач и планировщик пытается выбрать очередную задачу из очередей активных или пренатальных задач;
- по завершении вычисления задачи, производится оповещение всех потребителей данной задачи и все соответствующие присотановленые задачи перемещаются обратно в очередь активных задач.

Мемоизация

Для улучшения производительности и поддержки отказоустойчивости реализована поддержка мемоизации, для Т-методов помеченых соответствующим атрибутом. При вызове такого метода, проверяется наличие соответствующего вызова (пара – метод, параметры) в глобальной мемотаблице.

```
//Memo Table
public Hashtable MemoTable = new Hashtable();
```

Заключение

Рассмотреная в данной работе система T# обладает следующими свойствами:

- легкость обучения для пользователя;
- снижение трудозатрат при преобразовании существующих последовательных программ на C# в параллельные;
- разделение системы на разные уровни обеспечивает гибкость системы и возможность создания специализированых версий, оптимизированых под конкретные условия.

Дальнейшие направления работы включают в себя:

- реализацию распределенной памяти;
- поддержка ленивых вычисления;
- реализация транспортного уровня системы на основе вебсервисов.

РЕШЕНИЕ ЗАДАЧ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА ПРИМЕРЕ РАСПРЕДЕЛЕННОГО СЖАТИЯ МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ

Э.М. Шагатдинова

Оренбург

В последнее время все большие требования предъявляются к производительности микропроцессоров, реализующих непосредственное сжатие мультимедийной информации. Порой информация занимает внушительные объемы (наиболее актуально для видео), тем более если это добро хранится без компрессии, то есть без сжатия. Самый простой путь сокращения объема – использовать популярные архиваторы WinRar или WinZip. Но, увы, даже с включенной опцией «мультиме-

дийное сжатие» при архивации видеоинформации они дают выигрыш не более 30% от исходного объема, затрачивая при этом ощутимое время на процесс компрессии. Выход в том, что для многих задач, например, обработки аудио и видеоданных, производительность алгоритмов может быть увеличена за счет использования параллельной обработки. Одной из актуальных проблем при разработке программ параллельной обработки является высокая трудоемкость распараллеливания программы на уровне отдельных инструкций. Большинство традиционных технологий и средств параллельного программирования основаны на принципе выделения в программе относительно больших независимых блоков, которые могут выполняться параллельно и передавать данные через определенную систему коммуникаций. Примерами таких систем являются MPI, PVM. Такой подход предполагает наличие определенного числа обычных последовательных процессоров. В случае программирования для одного процессора, допускающего параллельное исполнение инструкций, такие системы не могут эффективно применяться, поскольку явное описание параллелизма на уровне отдельных инструкций является неприемлемым решением.

Создание программы для параллельной (многопроцессорной) вычислительной системы включает в себя, как правило, две основных стадии. Во-первых, последовательный алгоритм подвергается декомпозиции (распараллеливанию), т.е. разбивается на независимо работающие подзадачи (ветви параллельного алгоритма), а для обмена информацией между подзадачами (или для взаимодействия между ветвями) вводятся две дополнительных операции – прием и передача данных. Во-вторых, распараллеленный алгоритм записывается в виде программы, в которой операции приема и передачи представляются в терминах конкретной системы программирования, обеспечивающей связи между подзадачами. Наш подход состоит в использовании библиотеки PVM для автоматического распараллеливании мультимедийной информации, и ее последующего сжатия стандартными кодеками или средствами IPP@Intel. Выбор был сделан в пользу PVM с учетом гетерогенности, размерами кластера, на котором реализуется поставленная задача, и степенью устойчивости программы, обеспечивающейся PVM, программа реализуется на языке высокого уровня C++, т.к. достаточно качественно реализована привязка к нему. Кластерный комплекс, на котором ведутся работы, работает под управлением ОС Linux. Кластеры представляют собой многопроцессорные компьютеры, образованные из вычислительных модулей высокой степени готовности, которые

связаны между собой системой связи или разделяемыми устройствами внешней памяти. Сейчас часто для кластеров используют в качестве вычислительных узлов обычные серийно выпускаемые компьютеры, а также высокоскоростное сетевое оборудование и специальное программное обеспечение. Из-за малой стоимости комплектующих изделий, возможности постоянного обновления и применения, свободно распространяемого программного обеспечения эти системы являются наиболее перспективными с точки зрения получения высокой производительности.

Для решения задачи привлекается встроенная функция transcode, – она позволяет распараллелить сжатие в среде PVM, в частности, для перевода DVD->DivX используется GUI-интерфейс. Мультимедийный файл разбивается на n блоков, в зависимости от числа составляющих кластера, далее программа-клиент обращается к серверу с запросом, получает свой пакет из активного буфера (int ierr = pvm_send (int tid, int msgtag), PVMFSEND (TID, MSGTAG, IERR)). И сжимает его соответствующими средствами: инструментами MPEG, основанными на эталонном кодировщике MPEG Software Simulation Group, обеспечивающие более быстрое сжатие, или аналогичными из IPP, делающее упор на качество сжатия, – это делается в целях качественного сравнения использующихся средств. Все блоки пронумеровываются и при получении ответных сообщений, результатов со сжатым материалом, собираются в заданном порядке. Примечательно то, что прием сообщений ведется без учета разнородности платформ, с которых они отправляются. Операция считается завершенной, когда будет получен последний сжатый файл. В случае неудачной попытки, вновь совершается запрос и повторное сжатие.

Хотя результатами параллельного программирования являются более высокая производительность программ, при этом отмечается и более высокая трудоемкость. В параллельной модели программирования появляются проблемы нетипичные для последовательного: управление работой множества процессоров, организации межпроцессорных пересылок данных. И прежде чем приступить к написанию параллельной программы для исследовательских целей необходимо для себя выяснить следующие вопросы: будет ли созданная параллельная программа работать быстрее, чем ее последовательные варианты и соизмеримы ли полученный выигрыш во времени, который дает параллельная программа, с затратами на программирование. В нашей задаче отношение процессорного времени t_1 на выполнение последовательной

программы к времени t_p выполнения вычислений параллельной программой приближается к числу N – количеству кластерных элементов. Таким образом, программа по сжатию мультимедийной информации современна и востребована всевозрастающими нуждами пользователей.

СОДЕРЖАНИЕ

Оргкомитет семинара 4
Аветисян А.И., Самоваров О.И., Грушин Д.А. Архитектура и системное программное обеспечение вычислительных кластерных систем 5
Адуцкевич Е.В. Организация обмена данными на параллельных компьютерах с распределенной памятью 12
Амосова О.Е., Ткачев Ю.А. Моделирование гравитационного терригенного осадка 20
Бажанов С.Е., Воронцов М.М., Кутепов В.П., Шестаков Д.А. Интегрированная среда анализа структурной и вычислительной сложности функциональных программ и их целенаправленных эквивалентных преобразований 26
Баркалов К.А. Параллельный алгоритм глобальной оптимизации с адаптивным порядком проверки ограничений 31
Barkalov K.A., Markine V.L. About mars method and parallel index method integration 34
Бугаев Д.П. Принципы построения систем оперативной аналитической обработки данных на гетерогенных кластерах 37
Владова А.Ю. Технологии параллельного программирования для идентификации технического состояния трубопроводов 42
Гаева З.С., Гасников А.В. Распараллеливание уравнения коагуляции 48
Гергель В.П., Свистунов А.Н. Разработка интегрированной среды высокопроизводительных вычислений для кластера нижегородского университета 56
Горбунова А.С., Козинов Е.И., Мееров И.Б., Шишков А.В., Николаев А.Ф. Параллельная реализация одного алгоритма нахождения цены опционов бермудского типа 60
Гордиенко А.В., Дудник А.В., Кибец А.И., Кибец Ю.И. Анализ эффективности распараллеленного алгоритма конечно-элементного решения трехмерных нелинейных задач динамики конструкций на кластерах 67
Гришагин А.В., Курылев А.Л., Линев А.В. Оценка трудоемкости алгоритмов коллективных операций mpi для кластеров многоуровневой архитектуры 71
Гришагин В.А., Сергеев Я.Д. Параллельный метод решения многомерных многоэкстремальных задач с невыпуклыми ограничениями 74
Гришин А.В., Курылев А.Л., Коновалов А.В., Пегушин А.Г. Свободный параллельный отладчик на основе GNU DDD 83
Данилкин Е.А. Численное решение адвективно-диффузионных уравнений на многопроцессорной технике с распределенной памятью 88
Зимин Д.И., Фурсов В.А. Технология распределенной обработки цветных изображений 96

Иванников В., Гайсарян С., Аветисян А., Бабкова В. Модель параллельной программы и ее использование для оценки времени выполнения на инструментальном компьютере 102
Исламов Г.Г., Коган Ю.В., Сивков Д.А., Бабич О.В., Мельчуков С.А., Клочков М.А.Об одном методе поиска базисного минора матрицы 109
Исламов Г.Г., Сивков Д.А. Стратегическое направление Удмуртского госуниверситета 112
Карпов В.Е., Лобанов А.И. Параллельные вычисления в задачахфизико-химической гидродинамики: подходы и идеи 116
Ковальчук С.В., Владова А.Ю. Разработка одноранговой распределенной вычислительной системы 124
Котляров Д.В., Маланин В.Н. Разработка среды параллельного распределенного программирования ГСПП.NET 130
Кудерметов Р.К. Распределенная компьютерная среда для поддержки системного инжиниринга космических систем 135
Кузьминский М.Б. Двухъядерные процессоры как платформа для создания будущих HPC-кластеров. тестирование производительности 138
Кузьминский М.Б., Чернецов А.М., Шамаева О.Ю. Практика использования в кластерах аппаратного и программного обеспечения infiniband от mellanox. Распараллеливание в задачах вычислительной химии 143
Курносов М.Г. Опыт построения кластерных вычислительных систем с удаленной загрузкой узлов 149
Кутепов В.П., Котляров Д.В., Маркин В.Я. Измерение и прогнозирование изменения параметров загруженности кластерных систем 155
Лабутин Д.Ю. Высокопроизводительные вычисления как WEB-сервис платформы MICROSOFT.NET 160
Гергель В.П., Козинов Е.А., Лабутин Д.Ю., Лабутина А.А. Программная система для изучения и исследования параллельных методов решения сложных вычислительных задач 163
Лопатин И.В. Мониторинг мультиплатформенных узлов кластера 168
Михайлов Г.М., Оленев Н.Н., Петров А.А., Рогов Ю.П., Чернецов А.М. Опыт использования кластера вц ран в образовательных целях 170
Монахова Э.А. Циркулянтные сети связи вычислительных систем: структуры и обмены 175
Муравьев С.В. Сжатие научных данных большого объема для обеспечения возможности их визуализации 182
Нестеров И.А., Чубченко В.Г. Динамическая визуализация векторных полей, заданных на тетраэдрических сетках большого размера 187
Олзоева С.И., Тюменцев Д.В., Костенко А.В. Моделирование распределенной автоматизированной системы на кластере рабочих станций 191
Шабанов-Кушнаренко Ю.П., Обризан В.И. Параллелизм мозгоподобных вычислений 196

Потапов А.А. Способ организации взаимодействия компонентов классterной вычислительной системы 203
Сысоев А.В., Сидоров С.В. Использование чисел расширенной точности в реализации индексного метода поиска глобально-оптимальных решений 208
Тимченко С.В. Сравнение трех подходов к построению параллельных генетических алгоритмов на примере некоторых задач функциональной оптимизации и генетического программирования 212
Толоконников А.М. Параллельные вычисления направленных отношений .. 214
Толстобров А.В. Параллельная реализация алгоритма расчета динамики плазмы в плазменном прерывателе тока 221
Ушаков Ю.А. Построение высокопроизводительных вычислительных комплексов на базе лвс предприятия с использованием GRID технологий 228
Черников С.К., Ашихмин А.Н., Файзуллин А.М. Моделирование столкновений автомобилей на многопроцессорных вычислительных системах .. 232
Чудинов А.М. T#-среда параллельных вычислений для платформы MICROSOFT.NET 239
Шагатдинова Э.М. Решение задач параллельных вычислений на примере распределенного сжатия мультимедийной информации 247