Министерство образования Российской Федерации

Нижегородский государственный университет
им.Н.И.Лобачевского

Высокопроизводительные параллельные вычисления на кластерных системах

Материалы Международного научно-практического семинара

20–24 ноября 2001 г.

Издательство Нижегородского госуниверситета
Нижний Новгород
2002

УДК 681.3.012:51
ББК 32.973.26-018.2:22
В93

В93 **Высокопроизводительные параллельные вычисления на кластерных системах.** Материалы Международного научно-практического семинара./ Под ред. проф. **Р.Г.Стронгина**. Нижний Новгород: Изд-во Нижегородского университета, 2002, 217 с.

Сборник сформирован по итогам научного семинара, посвященного теоретической и прикладной проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа. Семинар проходил в Н.Новгороде 20—24 ноября 2001 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**

ISBN 5-85746-681-4 ББК 32.973.26-018.2:22

20–24 ноября 2001 года Вычислительный Центр РАН, Институт математического моделирования РАН, Нижегородский государственный университет им. Н.И. Лобачевского провели в Нижнем Новгороде 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 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 параллельными 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

Р.Г. Стронгин	председатель Оргкомитета, Первый проректор 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

КОМПЛЕКС ПРОГРАММ ЛЭГАК ДЛЯ РАСЧЕТА НЕСТАЦИОНАРНЫХ ТЕЧЕНИЙ МНОГОКОМПОНЕНТНОЙ СПЛОШНОЙ СРЕДЫ И ПРИНЦИПЫ РЕАЛИЗАЦИИ КОМПЛЕКСА НА МНОГОПРОЦЕССОРНОЙ ЭВМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

П.А. Авдеев, М.В. Артамонов, С.М. Бахрах, С.В. Величко, Н.А. Володина, Н.М. Воробьева, С.П. Егоршин, Е.Н. Есаева, А.Д. Ковалева, М.В. Лучинин, С.Н. Проневич, В.Ф.Спиридонов, И.Ю. Тарадай, А.Н.Тарасова, Е.В. Шувалова

Российский федеральный ядерный центр (РФЯЦ) – Всероссийский научно-исследовательский институт экспериментальной физики(ВНИИЭФ), г.Саров

В работе изложены основы методики, реализованной в комплексе программ ЛЭГАК, предназначенном для расчета нестационарных течений многокомпонентной сплошной среды.

Обсуждаются принципы распараллеливания комплекса программ ЛЭГАК на многопроцессорных ЭВМ с распределенной памятью. Приводятся примеры расчетов двумерных задач.

Реализация комплекса ЛЭГАК на многопроцессорных вычислительных системах позволила проводить расчеты с существенно большим числом счетных точек, чем в расчетах на скалярных ЭВМ, что особо существенно при численном моделировании явления неустойчивости контактных границ.

УЧЕБНО-НАУЧНЫЙ ЦЕНТР МГУ ПО ВЫСОКОПРОИЗВОДИТЕЛЬНЫМ ВЫЧИСЛЕНИЯМ*

А.Н.Андреев, Вл.В.Воеводин

НИВЦ МГУ им. М.В.Ломоносова, Москва

Введение

Цель данного сообщения – рассказать о создаваемом в Московском государственном университете учебно-научном центре по высокопроизводительным вычислениям и различных направлениях деятельности в этой области. В 2000–2001 годах при поддержке ФЦП «Интеграция», РФФИ и ректората МГУ в Московском университете создан уникальный и на сегодняшний день самый мощный среди вузов России вычислительный центр. Комбинация вычислительных мощностей с накопленным опытом в области использования высокопроизводительной техники и теоретическими исследованиями в области параллельных вычислений позволяет обеспечить полноценную среду поддержки фундаментальных научных исследований и высшего образования, а также экспериментальный полигон для разработки и апробирования кластерных технологий.

Информационно-аналитический центр по параллельным вычислениям Parallel.Ru

Коллектив лаборатории Параллельных информационных технологий НИВЦ МГУ уже много лет работает в области параллельных вычислений и супер-ЭВМ. Мы достаточно хорошо представляем данную предметную область, знакомы со многими коллективами и отдельными учеными, работающими в данной области, участвуем в совместных проектах. На этой основе в мае 1998 года мы создали страничку по параллельным вычислениям в сети Интернет, в дальнейшем выросшую в Информационно-аналитический Центр PARALLEL.RU. В 1999 году работа Центра была поддержана Российским фондом фундаментальных исследований (РФФИ). Тематика Центра охватывает следующие разделы предметной области: обзор новостей мира высокопроиз-

* Данная работа выполняется при поддержке РФФИ, грант N99-07-90230.

водительных вычислений, описание архитектур компьютеров, технологии параллельного программирования, задачи, решаемые на суперкомпьютерах, обзор источников информации в данной области, списки конференций по данной тематике, списки крупнейших суперкомпьютерных центров, информация о российских ресурсах в данной области, очерки по истории высокопроизводительных вычислений и многое другое. Задача, которая ставилась нами в самом начале – структуризация предметной области и построение справочника на базе HTML, практически выполнена. Задача-максимум для PARALLEL.RU состоит в создании полноценной энциклопедии предметной области на основе веб-технологий, что требует привлечения к проекту гораздо большего числа авторов, специалистов в отдельных областях. Другая задача Центра – это создание среды для постоянного общения и взаимодействия российских специалистов по параллельным вычислениям и пользователей высокопроизводительных вычислительных ресурсов. Собирается информация о российских персоналиях и организациях, вовлеченных в данную деятельность, поддерживается «карта российских проектов». Организована рассылка по электронной почте новостей Центра и мира высокопроизводительных вычислений, охватывающая все основные коллективы России и ближнего зарубежья, занимающиеся данной тематикой (на октябрь 2001 года более 1600 подписчиков). Функционирует дискуссионный клуб, в котором обсуждаются все интересующие вопросы данной тематики, оказывается информационная поддержка научных конференций, семинаров и других мероприятий вокруг тематики параллельных вычислений. На базе Центра реализуются элементы дистанционного обучения. Кроме предоставления учебных материалов и заданий, разработаны также варианты интерактивного взаимодействия с обучаемыми, такие как онлайн-тестирование, сдача зачета через Интернет и другие. В 1999 году был проведен пробный конкурс по параллельному программированию для студентов ведущих российских ВУЗов.

Вычислительные ресурсы Центра МГУ

В феврале 2000 года в НИВЦ МГУ был установлен первый вычислительный кластер, оборудование для которого было закуплено частично за счет ФЦП «Интеграция», частично за счет собственных средств НИВЦ. Этот кластер состоял из 12 двухпроцессорных ПК-

серверов российской сборки (компании «Инел»). Использование двухпроцессорных узлов считается более эффективным по технологическим и экономическим соображениям по сравнению с одно- и четырехпроцессорными узлами. При проектировании кластера с учетом технических требований и имеющихся средств рассматривалось несколько вариантов построения коммуникационной сети, в том числе Fast Ethernet, Gigabit Ethernet, Myrinet и SCI. В результате оценки имеющихся данных по цене, надежности программного обеспечения и производительности этих технологий, была выбрана технология SCI (Scalable Coherent Interface), несмотря на то, что стоимость коммуникационного оборудования в расчете на один узел (1700 долларов США) превосходила стоимость самого вычислительного узла (1300 долларов). Для построения коммуникационной сети были закуплены программно-аппаратные комплекты Wulfkit производства норвежских компаний Dolphin Interconnect Solutions (сетевые адаптеры) и Scali Computer (программное обеспечение). Технология SCI отличается от других тем, что не использует специальных коммутаторов, а узлы кластера объединяются в топологию «двухмерного тора», состоящего из однонаправленных колец SCI. При такой организации сети каждый узел имеет по 2 входящих и выходящих канала SCI с очень высокой скоростью передачи и присутствует одновременно в двух кольцах. Мы благодарим за помощь при установке и настройке программного обеспечения Московский научно-исследовательский центр электронно-вычислительной техники (НИЦЭВТ), который является пионером в использовании SCI-кластеров в России, и лично сотрудников НИЦЭВТ К.А.Ларионова и Л.К.Эйсымонта.

При дальнейшем расширении аппаратных ресурсов в начале 2001 года мы сочли необходимым учесть потребности наших пользователей и провели среди них опрос – какие компоненты нужно нарастить в первую очередь? По результатам этого опроса был увеличен объем оперативной памяти на всех узлах до 1 Гбайта и увеличено количество узлов. Таким образом, кластер SCI в настоящее время включает 18 двухпроцессорных узлов с процессорами Pentium III 500 МГц (и 550 на новых узлах) и на каждом узле установлено по 1 Гбайту оперативной памяти. Узлы объединены в сеть SCI с топологией двухмерного тора 3x6, а управление кластером производится с отдельной головной машины по сети Fast Ethernet. Кроме того, установлен отдельный файл-

сервер с RAID-массивом, на котором хранятся домашние директории пользователей, доступные со всех машин по протоколу NFS. Временные файлы программы пользователей могут создавать на локальных дисках каждого узла. Для поддержки непрерывной работы кластера на всех узлах установлены специальные платы watch-dog, разработанные в ИПС РАН (г. Переславль-Залесский), позволяющие автоматически перезагружать зависшие узлы.

Весной 2001 года был установлен новый 40-процессорный кластер «SKY» с более мощными процессорами (Pentium III/850 МГц) на базе традиционной сети Fast Ethernet. Преимущественно этот кластер используется как «вычислительная ферма», т.е. для запуска большого числа однопроцессорных задач; однако на этой конфигурации успешно работают и многие параллельные приложения, в частности, расчет больших квантовохимических систем с помощью PC-GAMESS. Для целей поддержки больших квантовохимических расчетов на каждом узле кластера установлено по 2 жестких диска, которыми независимо могут пользоваться два процесса параллельной задачи.

Суммарная пиковая производительность ресурсов Центра на сегодняшний день составляет 52 Гфлопс, суммарный объем оперативной памяти 38 Гбайт.

Характеристики производительности

По результатам измерения характеристик сети, скорость обменов данными по сети SCI на уровне MPI-приложений составила более 80 Мбайт/с, а латентность (время задержки сообщений) – примерно 5.6 микросекунд; эти результаты практически не зависят от взаимного расположения узлов. При больших размерах сообщения скорость обменов по сети SCI даже превышает скорость обменов внутри одного SMP-узла. Для сравнения: при обменах по сети Fast Ethernet пропускная способность обычно не превышает 10–11 Мбайт/с, а латентность составляет более 100 микросекунд. Это ограничивает круг параллельных приложений, которые могут эффективно работать на таком кластере.

Традиционным тестом производительности параллельных компьютеров, используемым при составлении списка Top500, является тест решения системы линейных уравнений LINPACK HPC. Лучшие результаты по LINPACK составляют 10 Гфлопс (или 278 Мфлопс на

процессор) для кластера SCI и 13 Гфлопс (или 325 Мфлопс на процессор) для кластера SKY (на задачах размера 44000x44000 и 47000x47000, соответственно). Таким образом, за счет более мощных процессоров кластер SKY на тесте LINPACK показывает лучшую производительность. Эти данные говорят не о том, что сеть SCI плохо справляется с нагрузкой, а о том, что тест LINPACK при больших объемах вычислений не предъявляет высоких требований к производительности коммуникационной сети. Интересно, что при фиксированном размере задачи (16000x16000) кластер на базе SCI показывает лучшую масштабируемость при большом числе процессоров.

За счет высоких коммуникационных характеристик сети SCI достигается очень высокая масштабируемость кластера на многих прикладных тестах. В частности, на некоторых тестах пакета NPB 2.3 результаты кластера оказывались лучше, чем результаты суперкомпьютера Cray T3E/900 с аналогичным числом процессоров.

Технологии программирования

На кластере поддерживаются все стандартные технологии программирования для систем с распределенной памятью. Установлены реализации стандарта MPI (Message Passing Interface) для работы поверх сети SCI и Fast Ethernet. Поддерживается программирование на языках C/C++ и Фортран 77/90. Большинство параллельных программ пользователи создают на языке Фортран с использованием технологии MPI. Кроме этого, установлены и реально используются российские программные системы DVM и НОРМА, разработанные в ИПМ РАН. Проводились эксперименты с системой динамического автоматического распараллеливания программ «Т-система», разработанной в ИПС РАН, а также с ParJava, параллельным расширением языка Java, разрабатываемой в ИСП РАН. Установлены однопроцессорные и параллельные библиотеки процедур линейной алгебры и преобразований Фурье: BLAS, MKL, SCALAPACK, FFTW. Осуществляется перенос на кластер «Библиотеки численного анализа», разработанной в НИВЦ МГУ для машин серии БЭСМ-6.

Особенности работы Центра

Среди особенностей функционирования Центра по высокопроизводительным вычислениям в МГУ стоит отметить следующие:

- Необходимо поддерживать множество пользователей с различными задачами и различными требованиями к вычислительным ресурсам.
- Большинство пользователей работают в удаленном режиме.
- Требуется качественная поддержка учебного процесса.
- Многим пользователям нужно предоставить возможность отладки параллельных программ перед запуском на счет.
- Задачи пользователей написаны на разных языках с применением различных технологий параллельного программирования.
- Нужно поддерживать запуск как параллельных, так и последовательных программ.
- В центре установлены множественные вычислительные ресурсы с различными параллельными средами.

Система управления заданиями

С учетом описанных выше требований нами была разработана и продолжает развиваться система управления заданиями для вычислительного кластера. В качестве вариантов ПО управления кластером рассматривались многие распространенные пакеты, такие как OpenPBS, но по разным причинам они не стали использоваться. Среди возможностей системы управления заданиями, разрабатываемой в НИВЦ МГУ, стоит отметить следующие:

- Поддержка множественных подкластеров с независимыми очередями и с возможностью запуска задач на всем кластере; независимые настройки различных очередей.
- Гибкая настройка системы на различные параллельные среды.
- Поддержка приоритетов задач.
- Возможности временной блокировки очередей, задач и узлов кластера.
- Настройка различных ограничений и прав пользователей.
- Средства оптимального планирования загрузки с учетом предполагаемого времени работы задач.
- Внешние модули для поддержки различных стратегий распределения процессоров.
- Средства сбора статистики по использованию кластера.
- Веб-интерфейс для просмотра состояния кластеров и очередей.

- Переносимость кода – система реализована на языке Perl с отдельными элементами на языке Си.
- Развитая документация для пользователя и администратора.

Поддержка пользователей

Большое внимание в работе Центра уделяется взаимодействию с пользователями и комплексной поддержке их работы. В качестве элементов такой деятельности стоит выделить:

- Подготовку справочной и методической литературы, документации для пользователей; публикация в рамках Интернет-центра Parallel.Ru.
- Оперативную поддержку пользователей по электронной почте и по телефону.
- Список рассылки оперативной информации для пользователей.
- Обязательный учет требований и пожеланий пользователей при расширении аппаратного и программного обеспечения при настройке параметров системы управления заданиями.

ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ ОБРАЗОВАНИЯ СПИРАЛЬНЫХ ГАЛАКТИК НА СУПЕР-ЭВМ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

С.М. Бахрах, А.О.Наумов

РФЯЦ – ВНИИЭФ, г. Саров.

Моделируется формирование спиральных галактик из скопления звёзд, случайным образом расположенных внутри сферы. Всем звёздам сообщается одинаковая угловая скорость. Учитывается гравитационное взаимодействие каждой звезды с другой, т.е. с момента времени $t = 0$ решается следующая система дифференциальных уравнений

$$\frac{\partial^2 \vec{r}_i}{\partial t^2} = -\gamma \sum_{j \neq i}^{N} \frac{\vec{r}_i - \vec{r}_j}{\left|\vec{r}_i - \vec{r}_j\right|^3} \quad \mathrm{i} = 1,2,\ldots,\mathrm{N}.$$

Для решения данной системы был выбран метод «с перешагиванием», учитывающий специфику выписанных уравнений:

$$\begin{cases} \dfrac{\partial \vec{r}_i}{\partial t} = \vec{v}_i \ , \\ \dfrac{\partial \vec{v}_i}{\partial t} = -\gamma \sum\limits_{j \neq i}^{N} \dfrac{\vec{r}_i^{\,n} - \vec{r}_j^{\,n}}{\left| \vec{r}_i^{\,n} - \vec{r}_j^{\,n} \right|^3}. \end{cases}$$

Алгоритм решения задачи о взаимодействии звёзд, входящих в модель галактики, удалось распараллелить на супер-ЭВМ с распределённой памятью. Трудность состояла в том, что необходимо учитывать взаимодействие каждой звезды с каждой другой. Проблема была решена за счёт поэтапного счёта и круговых обменов данными между процессорами.

Разработанный и реализованный оригинальный алгоритм распараллеливания позволил провести расчёты с различным числом звёзд (от $10^3 - 10^5$).

Исследовано влияние скорости вращения на эволюцию хаотического скопления звёзд. Получено, что для каждого скопления звёзд существует согласованная скорость вращения, при которой происходит формирование спирального образования. В других случаях были получены распределения звёзд, также характерные для существующих галактик.

ЧИСЛЕННОЕ МОДЕЛИРОВАНИЕ РОСТА НАЧАЛЬНОГО ВОЗМУЩЕНИЯ ПРИ КОСОМ СОУДАРЕНИИ МЕТАЛЛИЧЕСКИХ ПЛАСТИН

С.М.Бахрах, В.Ф.Спиридонов, Н.А.Володина

РФЯЦ – ВНИИЭФ, г.Саров.

В расчетах моделируется рост начальных возмущений при косом соударении металлических пластин. Известно, что при сверхзвуковом режиме соударения в газодинамическом приближении происходит рост начальных возмущений.

Сложнее обстоит дело при соударении пластин, обладающих прочностью.

При косом соударении слоев металлов в контактной зоне развиваются интенсивные сдвиговые деформации, приграничные слои металлов сильно разогреваются, образуются кумулятивные струи. Эти эффекты приводят к искажению профиля контактной границы. Возникают регулярные волны, несимметричные искаженные волны.

В настоящее время подробно исследован дозвуковой режим косого соударения: $U_{\text{точки контакта}}<C_{\text{звука}}$. В таких условиях нагружения в точке контакта образуется постоянно кумулятивная струя, если давление в окрестности точки соударения превышает прочность металла.

Если $C_{\text{звука}}<U_{\text{точки контакта}}<U_{\text{критическая}}$, в потоке формируются отсоединенные косые ударные волны и возмущения растут как в случае дозвукового соударения.

При $U_{\text{точки контакта}}> U_{\text{критическая}}$ в точке контакта устанавливаются присоединенные косые ударные волны и струеобразование в этом случае не наблюдается. Ранее считалось, что развитие возмущений в этом случае не возможно, так как отсутствует их основной генератор – кумулятивная струя.

В одном из отделений ВНИИЭФ были проведены эксперименты по соударению пластин из алюминия. Предполагалось получить при переходе к бесструйному режиму соударения пластин практически мгновенное прекращение процесса развития возмущений. То есть, как только $U_{\text{точки контакта}}= U_{\text{критическая}}$ возмущения возникать не должны. Однако в экспериментах было получено, что при дальнейшем увеличении точки контакта амплитуда возмущений уменьшается монотонно. Такие данные были получены впервые.

Состояние контактной границы материалов при подобных сверзвуковых режимах нагружения мало изучены. Поэтому в работе проводилось численное моделирование экспериментов с различными условиями соударения пластин, обладающих прочностью. Получено согласие расчетных и экспериментальных данных.

Счетная сетка бралась таким образом, чтобы на длину волны приходилось порядка 20 счетных ячеек. Общее число точек $1120\times740=828800$. Проведение таких расчетов на ЭВМ в скалярном (однопроцессорном) режиме проблематично. Поэтому расчеты проводились на многопроцессорной ЭВМ с разделенной памятью в комплексе ЛЭГАК-МП.

РАСПАРАЛЛЕЛИВАНИЕ ЯВНО-НЕЯВНОГО АЛГОРИТМА СЧЕТА ГАЗОДИНАМИКИ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

С.М.Бахрах, С.В.Величко, О.Н.Кулыгина, М.В.Лучинин, В.Ф.Спиридонов

РФЯЦ – ВНИИЭФ, г. Саров

Использование явных разностных схем для численного решения задач газовой динамики приводит к ограничениям на шаг по времени, которые связаны с устойчивостью разностной схемы. От этого недостатка свободны безусловно устойчивые разностные схемы. Однако объём вычислений в таких схемах существенно больше нежели в явных. Поэтому в комплексе программ ЛЭГАК был реализован явно-неявный алгоритм расчета давления, в котором в целях экономии машинного времени неявная схема используется для тех точек, где шаг по времени оказался значительно меньше, чем в остальных. Данный алгоритм позволяет существенно сократить время счета задач.

Для уменьшения календарного времени счета задач в комплексе ЛЭГАК этот алгоритм был распараллелен.

На ЭВМ с распределенной памятью производилось разбиение задачи на фрагменты по столбцам.

В программе явно-неявной газодинамики полученная схема разрешалась методом прогонки по двум направлениям – вдоль столбца и вдоль строки.

В результате декомпозиции по столбцам распараллеливание прогонки вдоль столбца вносит минимальные изменения в программу.

Прогонку вдоль строки напрямую проводить нельзя, т.к. каждый процесс должен знать прогоночные коэффициенты прямого и обратного хода прогонки, рассчитываемые на соседних процессах. Для реализации распараллеливания прогонки вдоль строки был выбран параллельно-конвейерный метод.

Распараллеленная программа явно-неявной газодинамики тестировалась на задаче об обжатии сферической оболочки газа.

Результаты расчетов не зависят от числа заказываемых процессов. Программа явно-неявной газодинамики даёт полное совпадение результатов в однопроцессорном и многопроцессорных расчетах.

Расчёты, проведенные в многопроцессорном режиме, показывают работоспособность распараллеленного алгоритма программы и приемлемую, теоретически ожидаемую эффективность распараллеливания.

РАСПАРАЛЛЕЛИВАНИЕ ОБХОДА ДЕРЕВА ПОИСКА ДЛЯ РЕШЕНИЯ ЗАДАЧИ О РЮКЗАКЕ НА КЛАСТЕРНОЙ СИСТЕМЕ

В.А.Беляев, Н.Е.Тимошевская

Томский государственный университет

Введение

Задача о рюкзаке относится к классу комбинаторно-логических задач, связанных с нахождением подмножества конечного множества с заданными свойствами, причем она известна как труднорешаемая задача, лежащая в классе *NP*-полных задач [1]. Этот факт обусловил возможность применения данной задачи в криптографии для создания криптосистем с открытым ключом [2]. Необходимость решения задачи о рюкзаке возникает при попытке вскрытия криптосистемы с открытым ключом рюкзачного типа, а также при анализе её стойкости.

В принципе, любую задачу о рюкзаке можно решить тривиальным перебором всех возможных подмножеств. Некоторые из известных последовательных алгоритмов решения задачи о рюкзаке основаны на построении и обходе специального дерева поиска решений[3]. Обход дерева может быть организован по нескольким ветвям одновременно и независимо, что позволяет использовать параллельные вычисления.

Решение задачи о рюкзаке методом обхода дерева поиска

Рассматриваемая задача о рюкзаке заключается в следующем.

Пусть заданы натуральное число ω и некоторое множество $A=\{a_1, a_2, \ldots, a_n\}$ из n натуральных чисел. Найти все подмножества множества A, если таковые существуют, сумма элементов в которых в точности равна ω. Такие подмножества будем называть *ω-совместимыми*.

Предполагается, что элементы в множестве A упорядочены некоторым образом, так что наряду с множеством A будем использовать

обозначение вектор $A=(a_1, a_2, \ldots, a_n)$, называемый в дальнейшем рюкзачным вектором.

Ниже предлагается алгоритм перечисления ω-совместимых подмножеств для заданной пары (A,ω), основанный на выполнении шагов двух типов для некоторого текущего множества B: первый тип – расширение, если это возможно, построенного на данный момент множества B, за счет добавления очередного элемента из рюкзачного вектора, и, если условия расширения не выполняются, то второй тип – удаление последнего добавленного элемента.

Шаги расширения и возврата могут быть наглядно представлены с помощью дерева поиска, которое строится следующим образом. Вершинам дерева сопоставляются подмножества множества A, а ребрам – его элементы. Корню дерева (вершине нулевого яруса) ставится в соответствие само множество A. Пусть построено i ярусов дерева поиска. Рассмотрим вершину v i-го яруса ($i>0$), сопоставим ей множество, элементы которого могут быть использованы для построения ω-совместимого множества без нарушения условия расширения. Если это множество пусто, то v является концевой вершиной, в противном случае ветвям дерева, исходящим из данной вершины, сопоставим элементы данного множества. Построение заканчивается, когда на некотором ярусе все вершины дерева оказываются концевыми.

Изложенный выше алгоритм перечисления всех ω-совместимых подмножеств множества A можно представить в виде специальной процедуры обхода дерева, которая состоит из двух операций – операции спуска (переход с i-го яруса на $i+1$), соответствующей расширению строящегося множества, и операции подъема, соответствующей шагу возврата. Данная процедура обеспечивает левый обход дерева в глубину. Если сумма элементов, сопоставленных ребрам пути, ведущего от корня к концевой вершине, в точности равна ω, то эти элементы образуют ω-совместимое множество.

Решение задачи о рюкзаке с использованием параллельных вычислений

Задача разбивается на подзадачи, каждая из которых заключается в построении ω-совместимого множества, содержащего выделенное подмножество элементов множества A. Каждая из подзадач может быть решена с помощью обхода поддерева в дереве поиска, такого, что

элементы, сопоставленные ребрам пути, ведущего от корня дерева к корню поддерева, составляют выделенное подмножество элементов множества *A*. Таким образом, обход всего дерева поиска может быть заменен обходами поддеревьев, каждый из которых может выполняться независимо от других. Это означает, возможность повышения скорости решения задачи, за счет того, что одновременно (параллельно) может выполняться несколько обходов. С помощью параллельных вычислений обход дерева может выполняться одновременно несколькими процессами, каждый из которых будет выполнять обход не всего дерева целиком, а лишь выделенного ему поддерева. Таким образом, для обхода дерева поиска можно одновременно запустить несколько параллельных процессов. При этом число обменов сообщениями между процессами будет невелико. Параллельные алгоритмы такого типа могут быть реализованы на системах кластерного типа, дешевизна и простота которых очень привлекательны, а эффективность такого подхода достаточно высока.

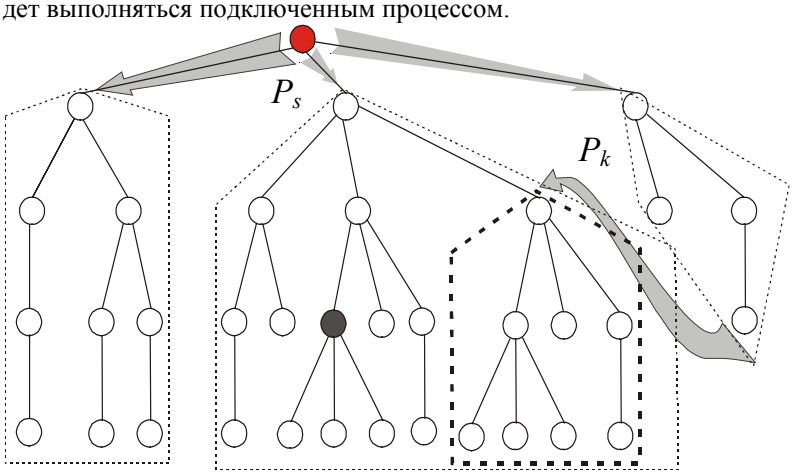
Организация параллельных процессов на системе кластерного типа для решения задачи о рюкзаке

Параллельный алгоритм разрабатывался с учетом следующих факторов:

- алгоритм должен обладать свойством масштабируемости, т.е. способностью сохранять работоспособность независимо от числа процессоров в системе;
- обмен информацией между процессами должен быть минимальным.

Среди параллельных процессов выделяется управляющий процесс, который передает остальным процессам необходимые начальные данные, распределяет подзадачи обхода поддеревьев дерева поиска между процессами и постоянно собирает построенные решения. Распределение подзадач происходит следующим образом. На начальном этапе выделяются поддеревья, корневые вершины которых лежат на первом ярусе и каждый из процессов начинает левый обход в глубину выделенного ему поддерева. Процессы, которым не хватило поддеревьев, корни которых лежат на первом ярусе, а также процессы, закончившие обход своего поддерева подключаются «в помощь» к другим процессам. Для этого задача выбранного процесса дробится на подзадачи, т.е.

в соответствующем дереве выделяется поддерево, обход которого будет выполняться подключенным процессом.



Пусть P_k – освободившийся процесс, а P_s – процесс, задача которого будет дробиться. Задача процесса P_s заключается в левом обходе в глубину выделенного ему поддерева. Определяется максимальный номер яруса, обход всех ветвей которого уже был выполнен процессом P_s. На следующем ярусе выбирается первая непройденная ветвь, и тогда задача процесса P_k заключается в обходе поддерева с корнем, соответствующим концевой вершине выбранной ветви. Другими словами, он решает задачу построения всех ω-совместимых подмножеств, уже содержащих элементы соответствующие ветвям, ведущим от корня всего дерева к корню выделенного поддерева.

При организации такого взаимодействия между процессами возникают проблемы, снижающие эффективность параллельного алгоритма, такие как: определение процесса, задача которого будет разбиваться на подзадачи; организация протокола связи с выбранным процессом; синхронизация работы всех процессов для избежания возникновения состояния дедлока.

Результаты анализа работы параллельного алгоритма

Разработанный параллельный алгоритм был реализован с помощью MPI на системе кластерного типа, состоящей из 18 узлов. Был проведен некоторый анализ эффективности использования параллель-

ных вычислений, результаты которого показали, что применение параллельного алгоритма позволяет в некоторых случаях добиться ускорения в двадцать раз.

Проведенные эксперименты позволяют сделать следующие выводы. Эффективность параллельного алгоритма тем выше, чем больше число множеств просматриваемых по ходу поиска решений, другими словами, чем больше фактический объем перебираемых подмножеств. Кроме того, замечено, что эффективность алгоритма тем выше, чем больше число ветвей, исходящих из корня, в соответствующем условиям конкретной задачи дереве поиска. Предложенный алгоритм является универсальным, т.е. достаточно эффективным во всех случаях и обладает свойством масштабируемости. При увеличении числа процессоров в системе ускорение возрастает.

Литература

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
2. Саломаа А. Криптография с открытым ключом / Пер. с англ. М.:Мир,1989. 264с.
3. Агибалов Г. П., Беляев В. А. Технология решения комбинаторно-логических задач методом сокращённого обхода дерева поиска. Томск: Изд-во Томского ун-та, 1981. 125 с.

РАСПАРАЛЛЕЛИВАНИЕ СЧЕТА ПО ПРОГРАММЕ ДМК НА МНОГОПРОЦЕССОРНЫХ МАШИНАХ С ОБЩЕЙ ПАМЯТЬЮ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА OPENMP

А.А. Воропинов, В.Н. Мотлохов, В.В. Рассказова

РФЯЦ – ВНИИЭФ, г. Саров

Краткое описание комплекса программ ДМК

Комплекс программ ДМК предназначен для решения задач газодинамики с учетом процессов упругопластики и теплопроводности на регулярных и нерегулярных многоугольных сетках в лагранжевой постановке. Для предотвращения возможных перехлестов в процессе

решения задач применяется оригинальный способ локальной коррекции счетной сетки, основанный на механизме упругого соударения.

Совместное использование MPI и OpenMP

Для распараллеливания счета на распределенной памяти используется MPI – распараллеливание по математическим областям. Для распараллеливания в модели общей памяти используется интерфейс OpenMP – распараллеливание внутри математической области. На кластерных системах возможно совместное использование этих двух подходов при условии, что каждый кластер представляет собой многопроцессорный узел с общей памятью.

Преимущества OpenMP перед моделью с передачей сообщений

Преимущества OpenMP:

- отсутствие затрат на подготовку и передачу данных между вычислительными процессорами.
- эффективное использование преимуществ общей памяти;
- простота реализации программ (как правило, незначительная модификация последовательных программ);
- простота освоения (стандарт OpenMP 1.1 содержит всего 12 директив, порядка 10 клауз и 15 подпрограмм и функций);
- возможность распараллеливание сверху вниз.

Хотя модель с передачей сообщений является наиболее универсальной, однако она не лишена недостатков. Особенно ярко некоторые из них проявляются при использовании в модели общей памяти.

Недостатки модели с передачей сообщений:

- затраты на подготовку данных для передачи (в некоторых ситуациях такая подготовка может быть чрезвычайно сложной);
- затраты непосредственно на передачу данных (особенно больших массивов);
- дополнительные затраты оперативной памяти, особенно в модели общей памяти – логически ненужное дублирование данных, но необходимое при использовании такой модели;
- необходимость весьма существенных доработок программ, написание новых модулей и процедур;
- сложна для освоения, так как требует особого «невычислительного» мышления.

Краткое описание OpenMP

Обоснование метода добавления директив в текст программ

Для распараллеливания в модели общей памяти наиболее эффективным считается механизм нитей.

Существует два противоположных подхода для использования этого механизма:

1) автоматическое распараллеливание компилятором (недостатки: недостаточная эффективность из-за ограничений существующих компиляторов);
2) полностью «ручное» добавление команд обращения к библиотеке нитей (недостатки: необходимость привлечения системного программирования, непереносимость между платформами).

«Золотая середина» – добавление специальных директив-подсказок компилятору о распараллеливании.

До появления OpenMP многие производители компиляторов поддерживали подобные наборы директив, однако эти механизмы были неполны и несовместимы между собой.

На этом фоне, при поддержке большого количества производителей компиляторов и вычислительных систем, был создан стандарт OpenMP.

Модель выполнения программ, использующих интерфейс OpenMP

Параллельные программы, использующие интерфейс OpenMP, выполняются в модели порождения параллельных нитей и ожидания завершения их выполнения. Программа начинает выполнение как один последовательный процесс, называемый основной нитью выполнения. Основная нить выполняется последовательно до тех пор, пока не встретится первая конструкция распараллеливания. Пара директив PARALLEL и END PARALLEL составляет конструкцию распараллеливания и определяет параллельную область программы. Как только встречается конструкция распараллеливания, основная нить создает группу нитей и основная нить становится основной нитью группы. Операторы программы, заключенные в конструкцию распараллеливания, включая и вызовы процедур, выполняются параллельно каждой нитью группы.

По завершении конструкции распараллеливания нити в группе синхронизируются, и только основная нить продолжает выполнение. В

одной программе может быть специфицировано любое число конструкций распараллеливания. В результате программа может много раз распараллеливаться и синхронизироваться в течение выполнения.

Интерфейс OpenMP позволяет программисту использовать директивы в процедурах, вызываемых изнутри конструкций распараллеливания. С этой функциональной возможностью пользователи могут программировать конструкции распараллеливания с верхних уровней дерева вызовов в программе и использовать директивы для контроля выполнения в любой вызываемой процедуре.

Основные определения и понятия

Программа, использующая интерфейс OpenMP, содержит специальные директивы, которые компилятором, не поддерживающим OpenMP, будут восприниматься как комментарии.

Директивы – это специальные комментарии в синтаксисе языка программирования Фортран, которые идентифицируются уникальной сигнальной меткой. Метка директивы структурирована так, что директивы трактуются как комментарии. Компиляторы, которые поддерживают интерфейс OpenMP, включают возможность активизировать и разрешать интерпретацию всех директив интерфейса OpenMP в командной строке. Формат директив интерфейса OpenMP следующий:

```
метка директива [клауза [[,] клауза] ...],
```

`метка` – последовательность символов `x$OMP`,

`клауза` задает параметры директивы.

Основные директивы и клаузы

PARALLEL и END PARALLEL – конструкция определения параллельной области.

DO – директива указывает, что итерации, следующие непосредственно за оператором цикла DO, должны быть выполнены параллельно.

PARALLEL DO – обеспечивает краткое определение параллельной области, которая содержит единственную директиву DO.

CRITICAL и END CRITICAL – эти директивы, обрамляющие код, ограничивают доступ к этому коду только одной нити в одно и то же время.

THREADPRIVATE – устанавливает атрибут доступности PRIVATE поименованному общему блоку нити, в то же время, оставляет этот блок общим в пределах нити (контекстной области).

Клаузы области видимости (PRIVATE, SHARED, REDUCTION).

Общие проблемы распараллеливания в модели общей памяти

Факторы, влияющие на производительность в модели общей памяти с использованием интерфейса OpenMP:

- время на организацию и завершение нитей (заведение локальных переменных, установка начальных данных, накопление переменных, перечисленных в клаузе REDUCTION и пр.);
- ожидание в критических секциях;
- операции, требующие синхронизации;
- «последовательные» циклы (параллельные циклы, при описании которых использовалась клауза ORDER);
- работа с внешними устройствами (вывод на экран и пр.);
- задержки при одновременном обращении на чтение несколькими нитями к одной ячейке памяти.

Аварийные ситуации

Одновременное обращение на запись несколькими нитями к одной ячейке памяти.

Одновременное обращение к внешнему устройству ввода или вывода.

Сложность отладки (необходимость одновременного отслеживания исполнения множества нитей).

Преимущество модели с передачей сообщений при отладке (более явные обращения к памяти).

Тестовый расчет

В качестве тестового расчета для проверки правильности работы алгоритма распараллеливания использовалась задача полета несферической оболочки.

Начальная форма оболочки:

$R(t=0,\theta) = 7 + 0{,}875\cos^3\theta - 0{,}525\cos\nu$ (см),

толщина $\Delta R = 0{,}16$ см.

Материал – свинец: $\rho(0) = 11{,}4\frac{\text{г}}{\text{см}^3}$.

В начальный момент оболочка имеет скорость к центру: $v = 5{,}70686\cdot 10^6$ см/с.

Давление снаружи $P_E = 0$, давление внутри $P_i = 3{,}495\cdot 10^6 \cdot V^{-5/3}$ (см^3), где V – объем полости.

Сетка сформирована из 6 счетных областей по 2800 ячеек в каждой (сетка матричного типа 140 на 20).

Расчет проводился на регулярной и нерегулярной сетках.

Представлены замеры времени выполнения параллельных областей на 1 и 2 процессорах.

В качестве характеристик эффективности распараллеливания использовались следующие функции:

$S_p = \frac{t_1}{t_p}$ – ускорение; $E_p = \frac{t_1}{p \cdot t_p} \cdot 100\%$ – эффективность распараллеливания.

На 2 процессорах ускорение составило 1,5 (эффективность 75%).

ОБЩАЯ ХАРАКТЕРИСТИКА ПРОГРАММЫ КУРСА «МНОГОПРОЦЕССОРНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ И ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ»

В.П. Гергель

Нижегородский государственный университет им. Н.И. Лобачевского

Анализируя современное состояние компьютерной индустрии, можно отметить, что применение параллельных вычислительных систем (*ПВС*) является стратегическим направлением развития вычислительной техники. Это обстоятельство вызвано не только принципиальным ограничением максимально возможного быстродействия обычных последовательных ЭВМ, но и практически постоянным существованием вычислительных задач, для решения которых возможностей существующих средств вычислительной техники всегда оказывается недостаточно. Отсюда вытекает значимость проблемы подготовки высококвалифицированных специалистов, способных эффективно использовать высокопроизводительные компьютерные системы для решения важных научно-технических задач с высокой вычислительной трудоемкостью. В свою очередь, это приводит к тому, что учебные дисциплины по тем или иным аспектам параллельных вычислений являются практически обязательными в образовательных программах большинства университетов.

Основная цель данной работы состоит в изложении основных положений учебного курса по параллельному программированию, читаемому в Нижегородском университете для студентов факультета вычислительной математики и кибернетики с 1995 г. Кроме того, в работе дается краткий обзор существующих учебных дисциплин по данной тематике в стране и за рубежом. При рассмотрении вопросов параллельных вычислений основное внимание уделяется использованию кластерных компьютерных систем.

1. Проблема получения решений сложных вычислительно-трудоемких научно-технических задач при реалистических временных затратах на основе организации параллельных вычислений для быстродействующих компьютерных систем приобрела в последнее время особую важность и значимость за счет широкого распространения *кластерных вычислительных систем* (*clusters*) – современное состояние данного подхода отражено, например, в обзоре, подготовленном под редакцией Barker (2000). Под *кластером* обычно понимается (см., например, Xu and Hwang (1998), Pfister (1998)) множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (*single system image*), надежного функционирования (*availability*) и эффективного использования (*performance*). Применение кластеров может в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (*lowly parallel processing*). Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (*coarse granularity*), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организации взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ (минимальность информационных потоков передачи данных).

2. Для получения общего представления о состоянии проблематики по параллельным вычислениям могут быть использованы два основных портала глобальной сети Интернет:

- http://www.parallel.ru (российский портал);
- http://homer.csm.port.ac.uk/ieee-tfcc/ (портал международной рабочей группы по кластерам – TFCC–IEEE Computer Society Task Force on Cluster Computing).

На этих же порталах имеется информация об основных исследователях и специалистах в области параллельных технологий и систем.

По рассматриваемой тематике имеется большое количество публикаций; в числе последних и значимых изданий работы Xu and Hwang (1998), Pfister (1998), Buyya R. (1998), Sterling T., Salmon J., Becker D.J., Savarrese D.F. (1999), Wilkinson B., Allen M. (1999). Среди опубликованных за последние годы отечественных учебных пособий: Комолкин А.В., Немнюгин С.А. (1998), Якобовский М.В. (2000), Фурсов В.А. и др. (2000), Гергель В.П., Стронгин Р.Г. (2000).

3. Обзорам образовательной составляющей проблематики по параллельным вычислениям посвящены отдельные разделы вышеупомянутых информационных серверов сети Интернет.

На портале www.parallel.ru упомянуты 3 учебных курса:

- Воеводин Вл.В. Материалы к курсу лекций «Параллельные вычисления, технологии параллельного программирования, параллельные компьютеры и супер-ЭВМ, анализ структуры программ»;
- Ластовецкий А.Л. Интерактивный учебный курс «Программирование параллельных вычислений на неоднородных сетях компьютеров на языке mpC»;
- Комолкин А.В., Немнюгин С.А. Программирование для высокопроизводительных ЭВМ.

Кроме того, в сети Интернет доступна информация о ряде других учебных программ:

- Крутиков М.П. Параллельные вычислительные системы (Институт высокопроизводительных вычислений и баз данных, Санкт-Петербург, www.csa.ru:81/~mike/edu/parallel.htnl);
- Симонов С.А. Языки и методов параллельного программирования (Институт вычислительной математики и математической геофизики СО РАН, Новосибирск, www.invest.sscc.ru/simonov/exam.htm).

Большую работу по систематизации образовательных программ выполнила экспертная группа Министерства образования РФ по подготовке специалистов в области высокопроизводительных вычислений. В качестве рекомендаций данная группа выдвинула предложения по созданию новых специализаций «Высокопроизводительные вычислительные технологии и системы», «Математическое и программное обеспечение высокопроизводительных вычислительных систем (ВВС)», «ВВС и параллельное программирование», а также сформировала рекомендуемый набор учебных курсов по этим специализациям. Информация о результатах деятельности рабочей группы представлена, например, в работе А.Н.Тихонова, А.Д.Иванникова, В.Г.Домрачева, А.К. Скуратова, И.В. Ретинской (2001).

Более широко представлены в Интернет зарубежные материалы по обучению. В числе возможных примеров:

1. Introduction to High Performance Computing (Ediburch Parallel Programming Center, www.epcc.ed.ac.uk/epcc-tec/courses/HPCintro.html)

2. Practical Parallel Programming (University of Leeds, www.comp.leeds.ac.uk, раздел MSc/modules_99/so31_comp3580.html)

3. Parallel Computing (Boston University) и др.

Анализируя содержание всех перечисленных учебных курсов, можно отметить следующее:

- во многих курсах проблемы параллельного программирования сводятся к задаче освоения средств программирования (типа библиотек MPI);
- вопросы моделирования вычислений с целью выявления возможных способов эффективного распараллеливания рассматриваются в недостаточно полном объеме;
- лабораторный практикум по параллельному программирования имеет в значительной степени иллюстративный характер (использование простых примеров, задач, методов).

4. Характеризуя необходимый набор знаний и умений, требуемый специалисту в области высокопроизводительных вычислений для эффективного решения сложных научно-технических задач, можно выделить следующий возможный набор разделов программы подготовки в указанной области:

- Архитектура параллельных вычислительных систем,
- Модели вычислений и методы анализа сложности,

- Параллельные методы вычислений,
- Параллельное программирование (языки, среды разработки, библиотеки).

Приведенный перечень может быть расширен, но даже и в таком составе становится ясным, что подготовка в области высокопроизводительных вычислений не может быть сведена к нескольким учебным курсам, а требует формирования комплексных учебных программ в виде специализаций существующих или вновь вводимых специальностей.

Вместе с этим актуальным является наличие и общего (интегрального) учебного курса по параллельному программированию, в рамках которого с единых позиций предпринималась бы попытка общего анализа многих проблем в области высокопроизводительных вычислений. Такой курс может быть использован как систематическое введение в предметную область в рамках специализаций и специальностей соответствующей направленности. Кроме того, подобный курс мог бы оказаться полезным для различных программ переподготовки и повышения квалификации кадров.

Именно к указанному типу относится учебный курс «Многопроцессорные вычислительные системы и параллельное программирование», который читается для студентов факультета вычислительной математики и кибернетики Нижегородского университета с 1995 г.; с 1999 г. этот курс читается также в Нижегородском государственном техническом университете. В 2001 г. данный курс читается (после соответствующей переработки) для научных сотрудников Института прикладной физики РАН и для работников компании «Интел».

В 2001 г. по курсу было подготовлено учебное пособие «Основы параллельных вычислений» (см. Гергель, Стронгин (2001)), которое на данный момент уже используется в учебном процессе более чем в 20 вузах страны.

Для характеристики содержания курса приведем в кратком виде (до уровня подразделов) учебную программу курса (в полном виде программа курса опубликована в пособии Гергеля, Стронгина (2001):

1. *Цели и задачи введения параллельной обработки данных:*
   1.1. Необходимость использования параллельных вычислений.
   1.2. История введения параллелизма .

1.3. Различие многозадачности, параллельных и распределенных вычислений.
1.4. Проблемы использования параллелизма.

2. *Принципы построения параллельных вычислительных систем:*
2.1. Пути достижения параллелизма.
2.2. Способы построения многопроцессорных вычислительных систем.
2.3. Виды параллельных вычислительных систем.
2.4. Классификация МВС.
2.5. Оценка производительности МВС.

3. *Моделирование и анализ параллельных вычислений:*
3.1. Модели параллельных вычислительных систем.
3.2. Модель алгоритма в виде графа «операнд – операции».
3.3. Модель параллельных вычислений в виде сети Петри.
3.4. Модель параллельных вычислений в виде графа «процесс-ресурс».

4. *Принципы разработки параллельных алгоритмов и программ:*
4.1. Оценка эффективности параллельных вычислений.
4.2. Уровни распараллеливания вычислений.
4.3. Этапы построения параллельных алгоритмов и программ.
4.4. Технологические аспекты распараллеливания.

5. *Системы разработки параллельных программ*
5.1. Создание специализированных языков программирования.
5.2. Расширение существующих языков программирования.
5.3. Разработка специализированных библиотек.

6. *Параллельные численные алгоритмы для решения типовых задач вычислительной математики:*
6.1. Общие способы распараллеливания алгоритмов.
6.2. Организация параллельного исполнения рекурсивных вычислений.
6.3. Параллельные численные алгоритмы линейной алгебры.
6.4. Параллельные численные алгоритмы решения дифференциальных уравнений в частных производных.
6.5. Параллельные численные алгоритмы многомерной многоэкстремальной оптимизации.

Для проведения лекционных и практических занятий имеется полный комплект компьютерных презентаций в формате системы MS

PowerPoint; информационное сопровождение курса обеспечивается ресурсом сети Интернет http://www.software.unn.ac.ru/ccam/teach.htm, в котором представлены лекционные презентации и результаты выполнения студенческих заданий.

Литература

1. Гергель В.П., Стронгин Р.Г. *Основы параллельных вычислений для многопроцессорных вычислительных систем*. Н.Новгород: Изд-во ННГУ, 2001.
2. Комолкин А.В., Немнюгин С.А. Программирование для высокопроизводительных ЭВМ. СПб: Изд-во НИИ химии СПбГУ, 1998.
3. Тихонов А.Н., Иванников А.Д.., Домрачев В.Г., Скуратов А.К., Ретинская И.В. Организационные и методические основы подготовки специалистов по высокопроизводительным вычислениям. // Труды Международной научной конференции» Телематика 2001». СПб: Изд-во СПбГИТМО, 2001. С. 151–154.
4. Фурсов В.А. и др. Введение в программирование для параллельных ЭВМ и кластеров / Под редакцией В.А. Фурсова. Самара: Изд-во СНЦ РАН, СГАУ, 2000.
5. Якобовский М.В. Распределенные системы и сети. М.: Изд-во МГТУ «Станкин», 2000.
6. Barker, M. (Ed.) (2000). *Cluster Computing Whitepaper* at http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/.
7. Buyya R. Editor, High Performance Cluster Computing, Vol. 1 System and Architecture, Vol. 2 Programming and Applications, Prentice Hall PTR, Upper Saddle River, 1998.
8. Kumar, V., Grama, A., Gupta, A., Karypis, G. *Introducrion to Parallel Computing*. – The Benjamin/Cummings Publishing Company, Inc., 1994.
9. Pfister, G. P. (1995). *In Search of Clusters*. Prentice Hall PTR, Upper Saddle River, NJ (2^{nd} edn., 1998).
10. Sterling T., Salmon J., Becker D.J., Savarrese D.F. How to Build a Beowulf, The MIT Press, 1999.
11. Xu, Z., Hwang, K. Scalable Parallel Computing Technology, Architecture, Programming. McGraw-Hill, Boston, 1998.
12. Wilkinson B., Allen M. *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, Upper Saddle River, 1999.

ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ INTEL-ПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ*

В.П. Гергель

Нижегородский государственный университет им. Н.И. Лобачевского

1. Широкое использование *кластерных вычислительных систем* (*clusters*) – современное состояние данного подхода отражено, например, в обзоре, подготовленном под редакцией Barker (2000), – является одним из стратегических направлений развития современной компьютерной техники. Под *кластером* обычно понимается (см., например, Xu and Hwang (1998), Pfister (1998)) множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (*single system image*), надежного функционирования (*availability*) и эффективного использования (*performance*). Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров, либо же могут быть сконструированы из новых типовых компьютерных элементов, что обычно не требует значительных финансовых затрат. Применение кластеров может в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (*lowly parallel processing*). Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (*coarse granularity*), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и

* Проведение исследований, по результатам которых была подготовлена данная работа, было поддержано грантом компании Intel.

программ (минимальность информационных потоков передачи данных).

2. Решение проблемы разнообразия архитектур параллельных вычислительных систем и обеспечение возможности создания мобильных (переносимых между различными компьютерными платформами) программ лежит на пути разработки стандартизованного базового системного программного обеспечения для организации параллельных вычислений. Основным стандартом, широко используемым в настоящее времени в практических приложениях, является *интерфейс передачи сообщений* (*message passing interface – MPI*) – см., например, Group и др. (1994). Наличие такого стандарта позволило разработать стандартные библиотеки программ (*MPI-библиотеки*), в которых оказалось возможным скрыть большинство архитектурных особенностей параллельных вычислительных систем и, как результат, существенно упростить проблему создания параллельных программ. Более того, стандартизация базового системного уровня позволила в значительной степени обеспечить мобильность параллельных программ, поскольку в настоящее время реализации MPI-стандарта имеются для большинства компьютерных платформ.

Близкой по назначению и также имеющей широкое распространение в практических приложениях является библиотека PVM (*Parallel Virtual Machine*) – см., например, Geist и др. (1994) (следует отметить, что спецификация библиотеки PVM не регламентируется каким-либо из общепринятых стандартов).

Характеризуя данный подход, следует отметить, что определенным недостатком стандартизации является снижение возможностей по использованию аппаратных особенностей конкретных параллельных вычислительных систем.

3. Использование передачи сообщений для организации параллельных вычислений ориентировано, прежде всего, на *многопроцессорные компьютерные системы с распределенной памятью.* Уменьшение времени расчетов (*ускорение*) при таком подходе может быть достигнуто только для тех научно-технических проблем, в которых объем вычислений превалирует над уровнем необходимых межпроцессорных взаимодействий (т.е. *для вычислительно-трудоемких задач с низкой коммуникационной сложностью*). Эффективность же параллельных вычислений для задач с высокой интенсивностью передач

данных может быть достигнута при использовании другого актуального в настоящее время способа построения параллельных компьютеров – *многопроцессорных систем с общей памятью* (*symmetric multiprocessors – SMP*), для которых взаимодействие процессоров обеспечивается использованием общей памяти для хранения совместно-обрабатываемых данных. Для унификации параллельных программ для такого типа систем также предпринимаются попытки стандартизации; одним из основных результатов в этом направлении является *стандарт OpenMP* – см., например, Chandra, R. and etc. (2000). Следует отметить, что системы с общей памятью имеют, как правило, ограниченное количество процессоров (порядка нескольких десятков), что определяет характер использования таких компьютеров от автономно функционирующих быстродействующих рабочих станций до применения в качестве вычислительных серверов в высокопроизводительных кластерных системах.

4. Возможное решение по снижению трудоемкости разработки параллельных алгоритмов и программ может состоять в разработке и в последующем широком использовании *библиотек параллельных программ*. Этот общепринятый в области разработки прикладного программного обеспечения подход позволяет согласовать два обычно несовместных требования – повысить качество создаваемых программ и значительно снизить сложность программирования. В настоящее время различными группами исследователей разработано большое количество библиотек, реализующих различные методы для выполнения научных и технических расчетов – среди основных обзоров данного вопроса можно привести следующие информационные ресурсы сети Интернет:

- http://parallel.ru/tech/tech_dev/par_libs.html;
- http://www.erc.msstate.edu/labs/hpcl/projects/mpi/tools_libraries.html;
- http://sal.kachinatech.com/C/3/index.shtml;
- http://www.mhpcc.edu/training/workshop/parallel_libs/MAIN.html и др.

В числе широко используемого программного обеспечения:

- Библиотека параллельных программ ScaLAPACK (Scalable LAPACK – http://www.netlib.org/scalapack) для решения систем линейных уравнений, обращения матриц, поиска собственных значений и др.

- Пакет параллельных процедур линейной алгебры PLAPACK (Parallel Linear Algebra Package – http://www.cs.utexas.edu/users/plapack) включает параллельные версии процедур решения систем линейных уравнений с помощью LU и QR-разложений, разложения Холецкого и др.

Применимость подобного *крупно-модульного* способа построения параллельных программ при использовании стандартизованного системного программного обеспечения и типовых библиотек параллельных вычислений полностью определяется эффективностью применяемых программных средств. Необходимость предельного использования всех возможностей компьютерных систем для решения научно-технических задач высокой вычислительной трудоемкости (а именно для решения таких задач чаще всего требуется применение параллельных вычислительных систем) ставит актуальную проблему комплексного анализа эффективности всей совокупности средств (алгоритмов, программ и технологий), применяемых для параллельных вычислений. На проведение исследований в рамках данной значимой проблемы и были направлены исследования, выполненные в Нижегородском университете в последние годы (см., например, Гергель, Стронгин (2000, 2001)). В рамках выполненных работ были *исследованы* вычислительные схемы основных алгоритмов, использованных в библиотеках научных расчетов, *проведена* оценка производительности таких схем на многопроцессорных вычислительных системах кластерного типа и *определены* пути для повышения эффективности параллельных способов решения задач.

5. В данной работе приводятся результаты экспериментального сравнения возможных способов организации эффективных параллельных вычислений для кластерных Intel-процессорных вычислительных систем, построенных на основе операционных систем семейства Microsoft Windows. Для организации параллельных вычислений в рамках данной работы исследуются возможные способы выбора операционной системы, системного программного обеспечения, среды разработки и компилятора, базового математического обеспечения (библиотек параллельных методов).

Для исследования эффективности возможных способов организации параллельных вычислений были проведены следующие серии экспериментов:

- Эксперименты по сравнению двух реализаций MPI для ОС Windows: Argonne National Lab MPICH и RWTH Aachen MP-MPICH с целью выявления наилучшей из них, по таким показателям, как латентность и скорость передачи данных;
- Эксперименты для сравнения наилучшей реализации MPI для ОС Windows (MP-MPICH) с широко распространенной реализацией MPI для ОС Unix Argonne National Lab MPICH;
- Эксперименты для оценки производительности (тест LINPACK) кластерной системы при использовании двух разных операционных систем (Windows и Linux);
- Эксперименты для оценки масштабируемости теста LINPACK при использовании Windows для различных конфигураций кластера.

В работе приводится краткое общее описание результатов выполненных экспериментов; более подробная информация приводится в отдельных работах настоящего издания.

6. Для проведения вычислительных экспериментов использовался вычислительный кластер Нижегородского университета, оборудование для которого было передано в рамках Академической программы Интел в 2001 г. В состав кластера входят:

- 2 вычислительных сервера, каждый из которых имеет 4 процессора Intel Pentium III 700 Мгц, 512 MB RAM, 10 GB HDD, 1 Гбит Ethernet card;
- 12 вычислительных серверов, каждый из которых имеет 2 процессора Intel Pentium III 1000 Мгц, 256 MB RAM, 10 GB HDD, 1 Гбит Ethernet card;
- 12 рабочих станций на базе процессора Intel Pentium 4 1300 Мгц, 256 MB RAM, 10 GB HDD, CD-ROM, монитор 15», 10/100 Fast Etherrnet card.

Важной отличительной особенностью кластера является его неоднородность (*гетерогенность*). В состав кластера входят рабочие места, оснащенные новейшими процессорами Intel Pentium 4 и соединенные относительно медленной сетью (100 Мбит), а также вычислительные 2- и 4- процессорные сервера, обмен данными между которыми выполняется при помощи быстрых каналов передачи данных (1000 Мбит). В результате кластер может использоваться не только для решения сложных вычислительно-трудоемких задач, но также и для проведения различных экспериментов по исследованию многопроцессор-

ных кластерных систем и параллельных методов решения научно-технических задач.

7. В качестве системной платформы для построения кластера выбраны современные операционные системы **семейства Microsoft Windows** (отдельные эксперименты проводились с использованием ОС Unix). Выбор такого решения определяется рядом причин, в числе которых основными являются следующие моменты:

- операционные системы Microsoft Windows (так же, как и ОС Unix) широко используются для построения кластеров; причем, если раньше применение ОС Unix для этих целей было преобладающим системным решением, в настоящее время тенденцией является увеличение числа создаваемых кластеров под управлением ОС Microsoft Windows (см., например, www.tc.cornell.edu/ac3/, www.windowclusters.org и др.);
- выполненные в ходе проекта эксперименты показали преимущество решений с применением библиотеки PLAPACK, реализованной для ОС Microsoft Windows, перед решениями, использующими ScaLAPACK для ОС Unix (см. раздел 2). Исследования системного программного обеспечения (библиотеки MPI) не выявили существенного преимущества реализаций, выполненных для ОС Unix (Linux), перед реализациями для ОС Microsoft Windows (см. раздел 1).
- разработка прикладного программного обеспечения выполняется преимущественно с использованием ОС Microsoft Windows;
- корпорация Microsoft проявила заинтересованность в создании подобного кластера и передала в ННГУ для поддержки работ все необходимое программное обеспечение (ОС MS Windows 2000 Professional, ОС MS Windows 2000 Advanced Server и др.).

В результате принятых решений программное обеспечение кластера является следующим:

- вычислительные сервера работают под управлением ОС Microsoft® Windows® 2000 Advanced Server; на рабочих местах разработчиков установлена ОС Microsoft® Windows® 2000 Professional;
- в качестве сред разработки используются Microsoft® Visual Studio 6.0;

- на рабочих местах разработчиков установлены библиотеки: для платформы **Windows** – Plapack 3.0 (см. www.cs.utexas.edu/users/plapack) и MKL (см. developer.intel.com/software/products/mkl/index.htm); для платформы **Unix** – ScaLAPACK (Scalable LAPACK – http://www.netlib.org/scalapack)
- в качестве средств передачи данных между процессорами установлены две реализации стандарта MPI: для платформы **Windows** – Argonne MPICH (www-unix.mcs.anl.gov/mpi/MPICH/) и MP-MPICH (www.lfbs.rwth-aachen.de/~joachim/MP-MPICH.html); для платформы **Unix** – Argonne MPICH (www-unix.mcs.anl.gov/mpi/MPICH/);
- в опытной эксплуатации находится система разработки параллельных программ DVM (см. www.keldysh.ru/dvm/).

Оптимизация системного программного обеспечения для организации параллельных вычислений

Поддержка стандарта MPI (см. www.mpi-forum.org) обеспечивается при помощи создания программных библиотек, и в настоящий момент известно достаточно большое число реализаций MPI. Эти разработки, обеспечивая поддержку единого стандарта, различаются по способам построения и могут быть ориентированы на различные операционные системы и аппаратные платформы. Как результат, при организации параллельных вычислений важным вопросом является выбор эффективной реализации MPI, наиболее соответствующей условиям применяемой вычислительной системы. При этом, если данный вопрос для операционной системы Unix практически разрешен в пользу библиотеки MPICH (см., например, www-unix.mcs.anl.gov/mpi/MPICH/), то для многопроцессорных (кластерных) систем, построенных на основе операционной платформы Microsoft Windows, проблема выбора эффективного варианта MPI по-прежнему имеет высокую значимость и актуальность.

Выполнение вычислительных экспериментов осуществлялось для определения *пропускной способности* и *времени задержки* (*латентности*) передачи данных коммуникационной среды кластера Нижегородского университета. Данные показатели являются одними из наиболее важных характеристик параллельных вычислительных систем (см., например, Андреев, Воеводин).

В рамках предусмотренных работ были выполнены три группы экспериментов:

1. Эксперименты для сравнения эффективности двух наиболее применяемых в системах под управлением ОС семейства Microsoft Windows свободно-распространяемых реализаций библиотек MPI: Argonne National Lab MPICH (www-unix.mcs.anl.gov/mpi/MPICH/) и RWTH Aachen MP-MPICH (www.lfbs.rwth-aachen.de/~joachim/MP-MPICH.html) с целью выбора наилучшего варианта.

2. Эксперименты для сравнения наилучшей реализации MPI для Windows (MP-MPICH) с широко распространенной реализацией MPI для Unix Argonne National Lab MPICH.

3. Серия экспериментов для сравнения Argonne National Lab MPICH и RWTH Aachen MP-MPICH для кластерных систем различных конфигураций.

Для примера результатов выполненных экспериментов в таблице 1 приведены оценки величины латентности (задержки) перед началом передачи данных для библиотек MPICH и MP-MPICH под управлением ОС Windows.

Таблица 1.

Оценки величины латентности

Тип взаимодействующих узлов	Латентность (мкс)	
	MPICH	MP- MPICH
Процессор двухпроцессорного сервера и рабочая станция кластера, соединенных сетью Fast Ethernet	162.217	109.288

Результаты приведенных экспериментов показывают, что время задержки начала передачи сообщений для варианта MPICH выше (по крайней мере, на 50%) по сравнению с реализацией MP-MPICH.

Результаты экспериментов для оценивания пропускной способности (скорости передачи) данных между процессором двухпроцессорного сервера кластера и рабочей станцией (сеть Fast Ethernet) представлены на рис. 1.

Таким образом, анализируя результаты проведенных экспериментов, можно сделать вывод о том, что библиотека MP-MPICH имеет

очевидное *преимущество* перед библиотекой MPICH по таким важным параметрам, как *латентность* и *пропускная способность*.

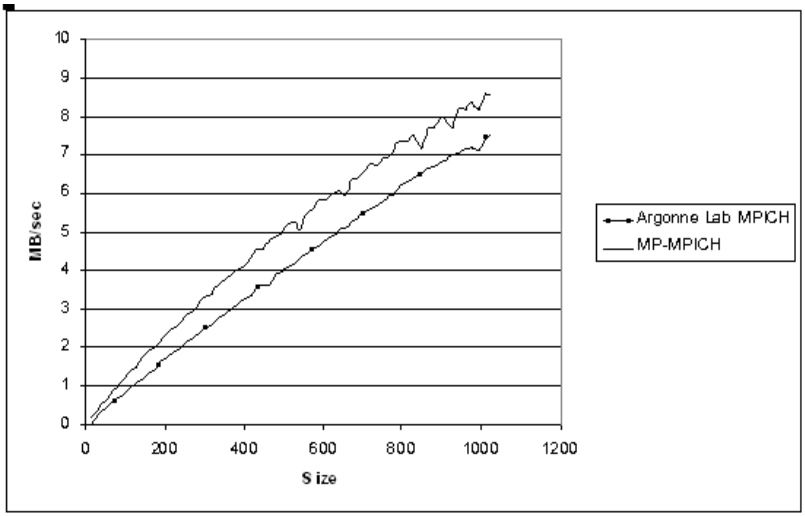


Рис. 1. Скорость передачи данных между процессором двухпроцессорного сервера кластера и рабочей станцией, которые соединены сетью Fast Ethernet, для пакетов данных от 16 до 1024 байтов

Результаты экспериментов для сравнения эффективности передачи данных для разных операционных систем (Windows и Linux) приводятся на рис. 2.

Оценка результатов экспериментов:

1. Обобщая результаты приведенных экспериментов для платформы Windows, в целом можно заключить, что использование реализации MP-MPICH обеспечивает большую скорость передачи данных по сравнению с вариантом библиотеки MPICH. Преимущество MP-MPICH наиболее заметно при относительно небольших размерах пересылаемых пакетов данных при использовании сетей с высокой пропускной способностью (Gigabit Ethernet). Результаты экспериментов позволяют *рекомендовать вариант библиотеки MP-MPICH* в качестве системного программного обеспечения для организации параллельных

вычислений в многопроцессорных (кластерных) системах, построенных на основе операционных платформ семейства Microsoft Windows.

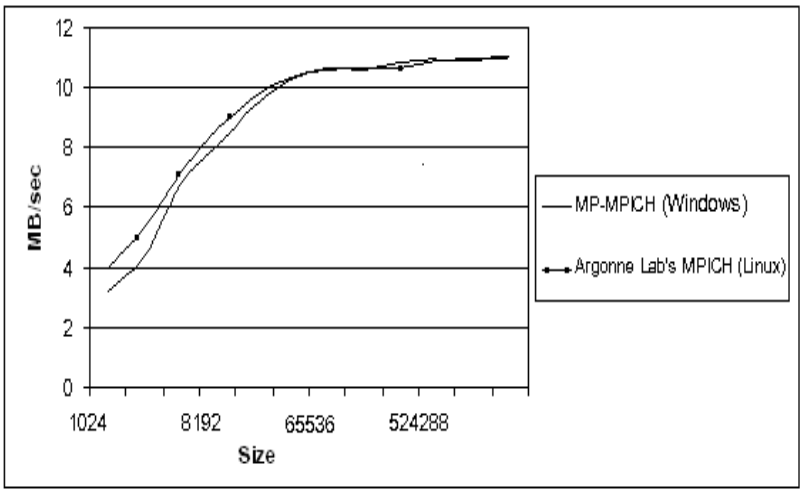


Рис. 2. Скорость передачи данных между рабочими станциями кластера, соединенных сетью Fast Ethernet, для пакетов данных от 1024 до 2097152 байтов

2. Сравнение результатов для реализаций библиотек MPI для платформ Windows и Unix (Linux) показывает, что в случае обменов короткими сообщениями MPICH (Linux) обеспечивает более высокую скорость передачи данных, чем MP-MPICH (Windows). Реализация MPICH (Linux) обеспечивает также лучшие показатели по такому параметру, как латентность. Однако, при увеличении размера передаваемых пакетов, преимущество реализации MPICH (Linux) сходит на нет, и скорости передачи данных для обеих реализаций становятся практически одинаковыми.

Анализ эффективности параллельных вычислений с использованием библиотек численных методов

Для оценки производительности кластерной системы использовался **тест LINPACK**, представляющий собой решение системы линейных уравнений методом LU-разложения. Тест LINPACK на сегодняш-

ний день является признанным тестом производительности параллельных систем, по результатам этого теста составляется список Top500 (http://www.top500.org/) – список пятисот наиболее производительных систем в мире.

В данном разделе описываются результаты, полученные для теста LINPACK при его реализации с широким использованием двух библиотек параллельных вычислений PLAPACK (для ОС Windows) и ScaLAPACK (для ОС Linux).

Библиотека PLAPACK (Parallel Linear Algebra Package) представляет собой набор параллельных процедур линейной алгебры, необходимых при выполнении большого класса научно-технических расчетов. PLAPACK реализован с использованием *библиотеки передачи сообщений MPI* для операционных систем семейства Windows и включает интерфейсы для языков Fortran и C. Для работы PLAPACK требует наличия библиотеки, реализующей функции набора BLAS (Basic Linear Algebra Subprograms). Более подробная информация о библиотеке PLAPACK приведена, например, в документе http://www.cs.utexas.edu/users/plapack.

ScaLAPACK (Scalable Linear Algebra Package – http://www.netlib.org/scalapack) представляет собой набор параллельных процедур, по функциям аналогичный процедурам PLAPACK. Так же как и PLAPACK, ScaLAPACK реализован с использованием *библиотеки передачи сообщений MPI,* но разработка выполнена для платформы Unix.

В экспериментах, проводимых в рамках настоящего проекта под управлением ОС Windows с использованием библиотеки PLAPACK, в качестве реализации BLAS использовалась библиотека MKL (описание библиотеки приведено, например, в http://developer.intel.com/software/products/mkl/index.htm). В качестве реализации MPI использовались две разные реализации (см. раздел 1):

- Argonne MPICH (http://www-unix.mcs.anl.gov/mpi/MPICH/);
- MP-MPICH (http://www.lfbs.rwth-aachen.de/~joachim/MP-MPICH.html).

В экспериментах под управлением ОС Unix (Linux) с применением библиотеки ScaLAPACK в качестве реализации MPI использовалась реализация Argonne MPICH (http://www-unix.mcs.anl.gov/mpi/MPICH/).

Для примера результатов выполненных экспериментов на рис. 3 представлены достигнутые максимальные показатели производитель-

ности с использованием разного числа вычислительных узлов (компьютеров на базе процессоров Pentium 4 с тактовой частотой 1.3 GHz) и для разных реализаций MPI с разными операционными системами.

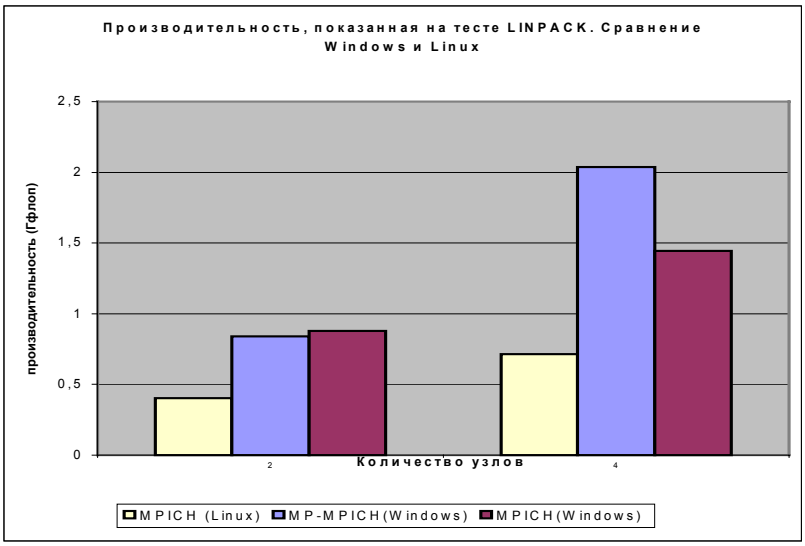


Рис 3. Максимальные показатели производительности на тесте LINPACK

Оценка результатов экспериментов:

1. Преимущество реализации стандарта MPI для ОС Windows в виде библиотеки MP-MPICH увеличивается с ростом числа используемых узлов.

2. Библиотека PLAPACK, реализованная для ОС Windows, **обеспечила в проведенных тестах большую производительность**, чем библиотека ScaLAPACK, реализованная для ОС Unix (Linux).

Оптимизация вычислений для многопроцессорных вычислительных серверов с общей памятью

В данном разделе рассматриваются результаты вычислительных экспериментов, выполненных для определения способов эффективной

реализации параллельных вычислений в кластерных вычислительных системах при наличии многопроцессорных серверов с общей памятью (*мультипроцессоров*).

Разработка параллельных программ для мультипроцессоров может осуществляться в рамках следующих двух основных схем реализации:

- общий подход (как и в случае многопроцессорных вычислительных системах с распределенной памятью) на основе технологии передачи сообщений MPI;
- подход, основанный на использовании возможностей стандарта OpenMP.

Первый способ позволяет создавать мобильные параллельные программы, которые могут использоваться в многопроцессорных системах различного типа; второй подход обеспечивает учет возможностей архитектуры параллельной системы с общей памятью.

Программный интерфейс приложений (API) OpenMP (см., например, Chandra, R. and etc. (2000)) обеспечивает поддержку программирования в модели общей памяти на языках C/C++ и Fortran и представляет собой достаточно простой способ для разработки параллельных программ, являющихся переносимыми (мобильными) между различными типами мультипроцессоров и операционных систем.

Для проведения экспериментов применялись двух- и четырехпроцессорные сервера вычислительного кластера Нижегородского университета. Разработка программ осуществлялась с помощью среды разработки Microsoft® Visual Studio 6.0 с компиляторами Microsoft® 32-bit C/C++ Optimizing Compiler и Intel® C++ Compiler 5.0. При проведении расчетов в качестве контрольного примера использовалась *задача матричного умножения*, для решения которой были подготовлены три различные программные реализации одного и то же алгоритма умножения матриц:

- последовательная программа, разработанная на основе стандартного последовательного алгоритма умножения матриц;
- параллельная программа, полученная из последовательного варианта путем добавления соответствующих директив OpenMP; следует отметить, что выполнение этой работы не потребовала значительных усилий; главное изменение исходной программы свелось к вставке только одной директивы распараллеливания перед основными циклами умножения:

```
#pragma omp for private(i,j,k) nowait
   for (i=0;i<N;i++)
         for (j=0;j<N;j++)
               for (k=0;k<N;k++)
                     C[i][j]+=A[i][k]*B[k][j];
```

- параллельная программа, использующая для организации взаимодействия процессоров механизм передачи сообщений MPI; для распределения элементов перемножаемых матриц между процессорами была задействована ленточная схема организации параллельных матричных вычислений (см., например, Kumar and etc. (1994)), когда на каждый процессор пересылается одновременно k строк матрицы A и k столбцов матрицы B (значение k может быть принято равным $\sqrt{p}$, где p есть число имеющихся в системе процессоров).

Результаты экспериментов для двухпроцессорного (2 процессора Intel® Pentium® III Xeon 1000 MHz, 256 Mb RAM) в числовой форме сведены в таблице 2. В этой таблице приведены данные по времени выполнении операции перемножения матриц разного порядка (от 300 до 2100) для всех трех вариантов программной реализации вычислений. Кроме того, для параллельных программ приведены показатели получаемого ускорения времени решения задачи по сравнению с временем работы последовательной программы

$$S = T_{посл} / T_{пар} .$$

Следует отметить, что при определении времени $T_{посл}$, несмотря на выполнение последовательной программы, используются оба процессора вычислительного сервера (дополнительный процессор применяется, в частности, для исполнения процессов операционной системы). Как результат, реальное время последовательного выполнения является большим, чем приведено в таблице 2 (что, соответственно, увеличивает коэффициент ускорения, обеспечиваемого параллельными вариантами программ).

Для наглядности результаты вычислительных экспериментов представлены также на рис. 4 в графической форме в виде графика коэффициентов ускорения процесса решения задачи при использовании параллельных вычислений.

Таблица 2.

Сравнение времени выполнения последовательного варианта программы с вариантами OpenMP и MPI для 2-процессорного сервера

Порядок матрицы: (N)	Время $T_{посл}$ (последователь-ный алгоритм)	OpenMP		MPI	
		Время $T_{пар}$	Ускорение S	Время T	Ускоре-ние S
300	0.42	0.39	1.08	0.39	1.07
600	4.69	3.55	1.32	3.85	1.22
900	16.20	12.05	1.34	14.17	1.14
1200	38.67	30.00	1.29	33.72	1.15
1500	76.56	58.20	1.32	60.19	1.27
1800	150.08	108.42	1.38	154.73	0.97
2100	258.09	171.75	1.50	177.03	1.46

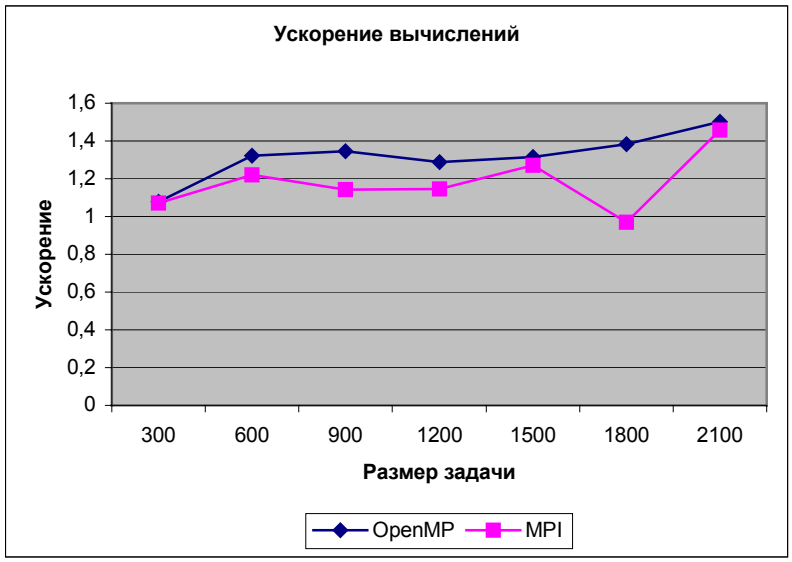


Рис. 4. Ускорение матричного умножения при использовании параллельных вычислений

В результате анализа полученных данных можно заключить, что при организации параллельных вычислений на многопроцессорных системах с общей памятью более эффективным является использование технологии разработки с возможностями OpenMP. Подобный вывод, в свою очередь, позволяет сформулировать предложение о целесообразности применения *комбинированной технологии разработки параллельных программ*, при которой для организации взаимодействия между узлами вычислительной системы используются средства MPI, а для обеспечения эффективных вычислений в пределах отдельных многопроцессорных узлов с общей памятью применяется способ распараллеливания на основе OpenMP.

Таблица 3.

Сравнение времени выполнения MPI и MPI+OpenMP вариантов программы для двух 2-процессорных серверов

Порядок матрицы: (N)	Время $T_{пар}$,с		Ускорение S
	MPI	MPI+OpenMP	
300	0.52	0.28	1.84
600	3.95	2.18	1.82
900	14.03	7.75	1.81
1200	37.36	29.08	1.28
1500	65.39	32.03	2.04
1800	112.75	89.29	1.26
2100	185.85	101.97	1.82

Для оценки подобного подхода были выполнены вычислительные эксперименты с использованием двух 2-процессорных серверов кластера. Для организации параллельных вычислений были разработаны два варианта программ для ленточного алгоритма перемножения матриц:

- программа с использованием только интерфейса передачи сообщений MPI; при выполнении экспериментов эта программа запускалась с генерацией 4 параллельных процессов (по два процесса на каждый двухпроцессорный сервер);

- программа с использованием интерфейса MPI и средств распараллеливания OPenMP; при выполнении этого варианта программы порождались 2 параллельных процесса (по одному процессу на каждый двухпроцессорный сервер), далее на каждом сервере для процессов средствами OpenMP создавались два параллельных потока (по одному на каждый процессор).

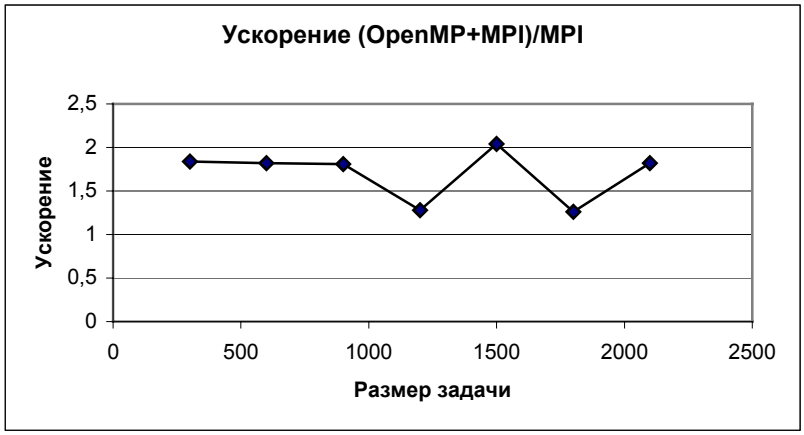


Рис. 5. Ускорение матричного умножения при использовании комбинированного OpenMP+ MPI варианта параллельной программы

Результаты выполненных экспериментов приведены в таблице 3 и представлены в графическом виде на рис. 5. Как следует из приведенных данных, комбинированный вариант параллельной программы имеет заметное преимущество по эффективности в сравнении с программой, разработанной только для одного интерфейса MPI.

Оптимизация вычислений для однопроцессорных компьютерных систем

Целью вычислительных экспериментов, описание которых проводится в данном разделе, являлось определение способов эффективной реализации алгоритмов и сравнение временных характеристик программ для различных C++ компиляторов на примере задачи умножения случайно генерируемых квадратных матриц.

Необходимым условием для создания эффективных программ является наличие оптимизирующих компиляторов для наиболее распространенных языков программирования высокого уровня. В частности, в настоящее время для C++ активно используются компиляторы Intel® C++ Compiler, Microsoft® 32-bit C/C++ Optimizing Compiler, Borland® C++ for Win32 Compiler.

Для примера результатов выполненных экспериментов в таблице 4 приведены временные затраты для выполнения операции умножения матриц разного порядка.

Таблица 4.

Временные затраты для выполнения операции умножения матриц разного порядка

Размер Компиляторы	Время T, с	
	N=1000	N=1500
Borland® C++ for Win32 Compiler 5.5	9,60	32,47
Microsoft® 32-bit C/C++ Optimizing Compiler	3,76	13,86
Intel® C++ Compiler 5.0 (включены оптимизирующие опции)	1,50	4,99
Библиотека PLAPACK (Intel® C++ Compiler 5.0)	2,66	8,87

Заключение

В работе приводится описание выполненных вычислительных экспериментов, по результатам которых оказалось возможным:

1) проанализировать возможности эффективной организации вычислений с использованием архитектурных особенностей современных процессоров производства корпорации Интел; при проведении экспериментов были разработаны программы для контрольной задачи матричного умножения, которые выполняются быстрее не менее чем на 75% по сравнению с соответствующей процедурой библиотеки PLAPACK 3.0 (для проведения расчетов библиотека PLAPACK использует программную систему MKL разработки корпорации Интел);

2) осуществить выбор системного программного обеспечения для организации параллельных вычислений в многопроцессорных (кластерных) системах; выбранный вариант библиотеки MP-MPICH обеспечивает более чем на 50% лучшую производительность по сравнению

с MPICH, являющейся одной из наиболее широко применяемых реализаций стандарта MPI;

3) сравнить эффективность возможных способов разработки параллельных программ для многопроцессорных вычислительных систем с общей памятью; результаты выполненных экспериментов показывают, что использование технологии OpenMP позволяет повысить эффективность параллельных вычислений по сравнению с применением средств передачи сообщений стандарта MPI (эффект ускорения может достигать порядка 40%);

4) оценить возможности *технологии комбинированной разработки параллельных программ*, при которой для организации взаимодействия между узлами вычислительной системы используются средства MPI, а для обеспечения эффективных вычислений в пределах отдельных многопроцессорных узлов с общей памятью применяется способ распараллеливания на основе OpenMP;

5) изучить масштабируемость (изменение производительности) вычислений при увеличении количества процессоров вычислительной системы для широко применяемой в практических приложениях библиотеки параллельных методов PLAPACK 3.0; эксперименты показали, что увеличению числа процессоров соответствует практически линейный рост производительности вычислений (для задач матричного умножения и решения систем линейных уравнений);

6) проанализировать эффективность вычислений при использовании разных параллельных методов для решения одной и той же задачи; на примере задачи матричного умножения проведено сравнение методов Фокса и Кеннона и ленточного алгоритма; результаты экспериментов показали, что при относительно малых размерах решаемой задачи лучшую эффективность имеет ленточный алгоритм; для матриц более высокого порядка более предпочтительными являются блочные алгоритмы Фокса и Кеннона (различие эффективности может достигать более 200%).

Литература

1. Андреев А.Н., Воеводин Вл.В. Методика измерения основных характеристик программно-аппаратной среды. (www.dvo.ru/bbc/benchmarks.html)
2. Воеводин В.В. *Модели и методы в параллельных процессах*. – М.: Наука, 1986.

3. Воеводин В.В. *Математические основы параллельных вычислений*. – М.: МГУ, 1991.
4. Гергель В.П., Стронгин Р.Г. Параллельная глобальная оптимизация в многопроцессорных вычислительных системах. // Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения». М.: Изд-во МГУ, 2000. С. 87–92.
5. Гергель В.П., Стронгин Р.Г. Параллельные вычисления в задачах глобально-оптимальных решений для многопроцессорных кластерных систем // Труды Международной научной конференции «Математическое моделирование». Самара, 2001. С. 89–92.
6. Гергель В.П., Стронгин Р.Г.. Высокопроизводительный вычислительный кластер Нижегородского университета. // Труды Международной научной конференции» Телематика 2001». СПб.: Изд-во СПбГИТМО, 2001. С. 160–161.
7. Гергель В.П., Стронгин Р.Г. *Основы параллельных вычислений для многопроцессорных вычислительных систем*. Н.Новгород: Изд-во ННГУ, 2001.
8. Barker, M. (Ed.) *Cluster Computing Whitepaper* at http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper/. 2000.
9. Bertsekas, D.P., Tsitsiklis, J.N.. *Parallel and Distributed Computation. Numerical Methods*. – Prentice Hall, Englewood Cliffs, New Jersey, 1989.
10. Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.
11. Geist, G.A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, B., Sunderam, V. *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.
12. Group, W., Lusk, E., Skjellum, A. *Using MPI. Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1994.
13. Kumar, V., Grama, A., Gupta, A., Karypis, G. *Introducrion to Parallel Computing*. – The Benjamin/Cummings Publishing Company, Inc., 1994.
14. Pfister, G. P. *In Search of Clusters*. Prentice Hall PTR, Upper Saddle River, NJ (2^{nd} edn., 1998).
15. Xu, Z., Hwang, K. *Scalable Parallel Computing Technology, Architecture, Programming*. McGraw-Hill, Boston, 1998.

16. Zomaya, A.Y. *Parallel and Distributed Computing Handbook*. / Ed. A.Y. Zomaya. – McGraw-Hill, 1996.

МНОГОПРОЦЕССОРНАЯ СИСТЕМА СБОРА И ОБРАБОТКИ ДАННЫХ НАУЧНОГО ЭКСПЕРИМЕНТА

М.Л. Гольдштейн, А.В. Матвеев

Институт математики и механики УрО РАН, Екатеринбург

Рассматривается один из возможных вариантов построения многопроцессорной системы сбора и обработки данных (МССОД) с точки зрения получения заданных временных характеристик на примере автоматизированных систем научных исследований (АСНИ) для физического эксперимента в области высокотемпературной электрохимии и медицинского эксперимента для исследования нервно-мышечного аппарата человека. Конкретные временные характеристики АСНИ обусловлены типом объекта и характером эксперимента и лежат во временном диапазоне $10^{-3} – 10^{-7}$ с. В обоих случаях МССОД является составной частью АСНИ с n количеством входов, регистрирующих ряд показателей. В случае электрохимического эксперимента к основным можно отнести ток, потенциал рабочего электрода, количество электричества, температуру, давление и др, в случае медицинского – информацию о форме воздействующих импульсов и реакцию на них.

Подсистема ввода и первичной обработки информации построена как многопроцессорная среда с использованием интеллектуальных плат ввода-вывода (ИПВВ) с цифровыми сигнальными процессорами (DSP) на борту. В настоящее время на одной плате может быть от 1 до 16 DSP, связь между которыми возможна как слабая, так и сильная. Слабая, когда каждый процессор выполняет свою подпрограмму и обрабатывает данные, хранящиеся в собственной локальной памяти, а обмен данными выполняется через порты ввода-вывода. Сильная, когда каждый процессор обрабатывает данные из некоторого централизованного хранилища, а обмен данными между процессорами выполняется через разделяемую память. Компьютер, содержащий несколько таких плат, обладает достаточно высокой производительностью, однако стоимость подобной системы может оказаться слишком высокой и для многих проектов неприемлемой. Поэтому необходимо разделить

обработку данных между быстрыми и дорогими ИПВВ и более медленными и дешевыми многоканальными платами ввода-вывода без процессора (МПВВ). В качестве ИПВВ нами были использованы платы L-783 производства фирмы L-Card, содержащие один процессор ADSP2186. В качестве МПВВ – платы A812PG фирмы ICP DAS.

В общем случае, каждая из рассматриваемых АСНИ реализует четыре основные функции:

1. Фиксация откликов объекта и режимозадающего оборудования Ff_x;
2. Формирование управляющих воздействий Fc;
3. Визуализация экспериментальных данных (как прямых, так и расчетных) Fv;
4. Накопление данных Fd.

Обозначим Ff_x для функций 2, 3, 4 Ff_c, Ff_v и Ff_d соответственно. Каждая функция выполняется с определенной частотой. Пусть система имеет n входов: u_1, u_2, … u_n (n внешних сигналов). Обозначим x_{ik} k-тый отсчет (цифровое значение сигнала) с i-го входа системы. Тогда

$$Fc = Fc\ (x_{i1\ 1}, \ldots, x_{i1\ c1}; x_{i2\ 1}, \ldots, x_{i2\ c2}; \ldots; x_{i\alpha\ 1}, \ldots, x_{i\alpha\ ck}),$$

где ij, j = 1,α – номер входа системы; cj, j = 1,k – номер отсчета.

Аналогично можно определить:

$$Fv = Fv\ (x_{j1\ 1}, \ldots, x_{j1\ v1}; x_{j2\ 1}, \ldots, x_{j2\ v2}; \ldots; x_{j\beta\ 1}, \ldots, x_{j\beta\ vl}),$$

$$Fd = Fd\ (x_{y1\ 1}, \ldots, x_{y1\ d1}; x_{y2\ 1}, \ldots, x_{y2\ d2}; \ldots; x_{y\chi\ 1}, \ldots, x_{y\chi\ dm}),$$

Обозначим:

$T(Ff_c)$ – критическое время выполнения функции Ff_c для данных $x_{i1\ 1}, \ldots, x_{i1\ c1}; x_{i2\ 1}, \ldots, x_{i2\ c2}; \ldots; x_{i\alpha\ 1}, \ldots, x_{i\alpha\ ck}$.

T(Fc) – критическое время выполнения функции Fc.

Аналогично определяются $T(Ff_v)$, T(Fv), $T(Ff_d)$, T(Fd).

Важным показателем системы является время ее реакции на внешние события (термин «время реакции» применим не только ко времени формирования управляющих воздействий, то есть к функции Fc, но также и ко времени визуализации информации и времени сбора данных, например, успеть сохранить блок данных до момента поступления следующего блока). Рассмотрим, каким образом можно сократить его, то есть сумму $T(Ff_c) + T(Fc)$, $T(Ff_v) + T(Fv)$, $T(Ff_d) + T(Fd)$.

Функции Ff_c, Ff_v и Ff_d не имеют принципиального отличия, поэтому далее под функцией Ff будем понимать любую из вышеперечисленных.

Обозначим F’f такие функции Ff, для выполнения которых применяются ИПВВ, F’’f – остальные функции Ff. Аналогично вводятся понятия F’c, F’’c, F’v, F’’v, F’d, F’’d.

В случае малых временных диапазонов (T(Ff) порядка микросекунд и менее) ввод данных в реальном времени однопроцессорной АСНИ затрудняется временными задержками, которые возникают при передаче данных по каналам связи из внешнего мира в вычислительную среду, а также задержками операционной системы (современные операционные системы реального времени имеют задержки порядка нескольких микросекунд). Поэтому во многих случаях единственно возможным решением является перенос вычислительной среды как можно ближе к источникам сигналов, то есть на ИПВВ. Тогда решается вопрос о временных задержках, связанных с передачей данных по шинам расширения, таким как ISA, PCI, VME и т.п. В случае электрохимического и медицинского эксперимента применение ИПВВ было необходимым в силу требования непрерывного сбора и обработки группы данных на частоте до 3 МГц (T(F’f) = 0.3 мкс). Однако, в целях снижения стоимости системы, для фиксации относительно медленно меняющихся величин, таких как температура или давление (T(F’’f) = 1 мс), мы применили МПВВ.

Установка в компьютер нескольких плат с одним DSP позволила организовать одновременный сбор и первичную обработку данных на высокой частоте сразу от нескольких источников. До недавнего времени традиционный подход заключался в формировании последовательных управляющих воздействий центральным процессором и не позволял добиться многоканального сбора и обработки данных на высокой частоте из-за относительно больших временных затрат на передачу данных по каналам связи. В случае с DSP управляющие команды на сбор данных, например, в медицинском эксперименте, осуществляет этот процессор, определяя с какого канала и в какое время получить данные. Таким образом, один DSP может последовательно опросить несколько каналов. Вопрос о применении дополнительных плат с DSP решался с учетом времени аналого-цифрового преобразования, которое выполняется последовательно для каждого канала, а также предельно допустимой частоты сбора данных одним DSP. На плате L-783 время составило 0.3 мкс. Таким образом, применение нескольких ИПВВ позволяет существенно сократить время T(F’f).

Следующее преимущество ИПВВ – это первичная параллельная обработка полученных данных. Сокращение общего времени обработки T(F'c) + T(F''c), T(F'v) + T(F''v), T(F'd) + T(F''d) происходит как за счет применения параллелизма, так и за счет снижения объема полезных данных, необходимых для централизованной обработки главным процессором. Так, в медицинском эксперименте ИПВВ выделяет из непрерывного потока данных полезный сигнал, и только эти данные передает центральному процессору системы.

В общем случае время реакции системы определяется суммой T(Ff) + T(F). Это относится ко всем ее функциям: Fc, Fv и Fd. Однако, организовав двухступенчатый конвейер из периферийного и центрального процессора, можно сократить время реакции до следующей величины: max(T(F'f), T(F'')), что и было сделано в разработанных нами АСНИ. Пока центральный процессор выполняет функции F''c, F''v и F''d, DSP выполняет $F'f_c$, $F'f_v$, $F'f_d$.

Для Fv и Fd данные необходимо передать главному процессору системы, так как только через него возможна связь с устройствами визуализации информации и с устройствами хранения данных. Однако, в случае Fc все данные обрабатываются непосредственно на плате ввода-вывода и здесь же может быть сформировано управляющее воздействие. Тогда $T(F'f_c)$ + T(F'c) составляет сотни наносекунд, а применение нескольких жестко связанных процессоров на одной плате позволяет существенно сократить время выполнения определенных алгоритмов. В электрохимическом эксперименте, например, одно из управляющих воздействий вырабатывается при прохождении через электролит заданного количества электричества путем непрерывного подсчета, которым занимается DSP. Время реакции системы в этом случае составляет:

$$T(F'f_c) + T(F'c) = 0.3 \text{ мкс} + 0.1 \text{ мкс} = 0.4 \text{ мкс},$$

что полностью удовлетворяет исходным требованиям.

Алгоритмы, выполняемые на подобных платах, как правило, не являются большими, поскольку основное их предназначение – быстрая обработка данных в непосредственной близости от источников сигналов и средств формирования управляющих воздействий, когда время передачи данных в центральный процессор является критичным.

DSP-мультипроцессинг активно применен в разработанных нами АСНИ для электрохимического и медицинского экспериментов, что

обеспечило получение заданных временных характеристик и позволило избежать использования такого дорогостоящего программного продукта, как операционная система реального времени.

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ РЕКУРСИВНЫХ СХЕМ РЕДУКЦИИ РАЗМЕРНОСТИ*

В.А.Гришагин, В.В.Песков

Нижегородский государственный университет им. Н.И.Лобачевского

Рассмотрим конечномерную задачу оптимизации:

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

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

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

в которой требуется найти глобальный минимум многоэкстремальной целевой функции $f(y)$ в области Q, задаваемой координатными (3) и функциональными (2) ограничениями на выбор допустимых точек (векторов) $y = (y_1, y_2, ..., y_N)$.

Многоэкстремальные оптимизационные задачи являются проблемами высокой вычислительной сложности, причем значительное влияние на сложность задачи (1)–(3) оказывает ее размерность *N*. Так, в случае, когда минимизируемая функция *f*(*y*) удовлетворяет в области поиска условию Липшица

$$|f(y') - f(y'')| \le L\|y' - y''\|,\ y', y'' \in Q, \quad (4)$$

для задач указанного класса характерен экспоненциальный рост вычислительных затрат при увеличении размерности. В связи с этим естественно использовать ресурсы параллельных вычислительных систем для ускорения процесса анализа и расширения круга исследуемых задач, которые без использования потенциала параллелизма решены быть не могут.

Одним из эффективных подходов к численному решению задач (1)–(3) является подход, основанный на использовании схемы вложен-

* Работа выполнена при поддержке РФФИ (грант № 01-01-00587)

ной рекурсивной оптимизации (многошаговой схемы редукции размерности) [1,2], согласно которой решение исходной многомерной задачи заменяется решением семейства рекурсивно связанных одномерных подзадач, что позволяет применить для решения многомерной задачи известные методы одномерного глобального поиска.

Применение в качестве одномерных алгоритмов оптимизации параллельных характеристических методов глобального поиска [3] позволило построить эффективные параллельные алгоритмы для решения многомерных многоэкстремальных задач. Эти алгоритмы позволяют при использовании 2^N процессоров в случае трудновычислимых задач (когда время вычисления функционалов задачи значительно превышает времена обменов данными) добиться эффективности решения, близкой к 95–97%, т.е. экспоненциально ускорить решение задачи за счет, разумеется, экспоненциального увеличения ресурсов (числа процессоров), обеспечивающих процедуру решения.

Настоящая работа предлагает новый механизм формирования одномерных подзадач в структуре многошаговой схемы, позволяющий сократить количество испытаний (вычислений функционалов задачи) и тем самым ускорить решение, сохранив при этом эффективность распараллеливания при многопроцессорной реализации.

Поясним идею алгоритма на примере двумерной задачи (1) при отсутствии функциональных ограничений (2), т.е. для прямоугольной области $Q = D$.

Согласно многошаговой схеме,

$$\min_{y \in Q} f(y) = \min_{a_1 \le y_1 \le b_1} \min_{a_2 \le y_2 \le b_2} f(y_1, y_2). \qquad (5)$$

Введем функцию

$$f^1(y_1) = \min_{a_2 \le y_2 \le b_2} f(y_1, y_2). \qquad (6)$$

Как следует из (5), решение исходной задачи (1) может быть получено посредством минимизации одномерной функции $f^1(y_1)$, т.е. решения одномерной подзадачи

$$f^1(y_1) \to \min, y_1 \in [a_1, b_1]. \qquad (7)$$

При этом каждое вычисление значения функции $f^1(y_1)$ в точке y_1 представляет собой решение новой одномерной подзадачи (y_1 фиксировано)

$$f(y_1, y_2) \to \min, y_2 \in [a_2, b_2] \quad (8)$$

Предположим, что, решая задачу (8) для некоторого y_1^*, мы уже провели решения аналогичных задач для $\underline{y}_1$ и $\overline{y}_1$ таких, что $\underline{y}_1 < y_1^* < \overline{y}_1$ и $\overline{y}_1 - \underline{y}_1 < \delta$, где $\delta > 0$ – некоторый параметр, задающий «меру близости» задач (8), соответствующих координатам $\underline{y}_1$ и $\overline{y}_1$. Предлагается начинать решение задачи (8) для y_1^* не с самого начала, а с некоторого исходного информационного множества, которое строится на основе значений целевой функции, полученных при оптимизации одномерных функций $f(\underline{y}_1, y_2)$ и $f(\overline{y}_1, y_2)$ и рассматриваемых как приближенные значения функции $f(y_1^*, y_2)$. Методология построения такого множества может быть различной. Для примера рассмотрим один из способов его конструирования.

В схеме характеристических алгоритмов в качестве информационного множества поиска рассматривается набор:

$$\omega_k = \{(y^1, z^1), \ldots, (y^k, z^k)\}, \quad (9)$$

где y^i, $1 \le i \le k$, являются координатами проведенных испытаний, а величины z^i, $1 \le i \le k$, – значениями минимизируемой функции в соответствующих точках y^i. В качестве такого начального информационного множества для задачи

$$f(y_1^*, y_2) \to \min, y_2 \in [a_2, b_2] \quad (10)$$

можно выбрать множество

$$\omega_{\bar{k}}^* = \{(y_2^{1*}, z^{1*}), \ldots, (y_2^{\bar{k}*}, z^{\bar{k}*})\}, \quad (11)$$

в котором

$$y_2^{i*} = (y_1^* - \underline{y}_1)(\hat{y}_2^i - \underline{y}_2^i)/(\overline{y}_1 - \underline{y}_1) + \underline{y}_2^i, \quad (12)$$

$$\hat{y}_2^i = \min_{1 \le j \le \bar{k}} \left| \underline{y}_2^i - \overline{y}_2^j \right|,$$

$$z^{i*} = (y_2^{i*} - \underline{y}_2^i)(\hat{z}^i - \underline{z}^i)/(\hat{y}_2^i - \underline{y}_2^i) + \underline{z}^i \quad (13)$$

$$\hat{z}^i = f(\overline{y}_1, \hat{y}_2^i),$$

а величины $\bar{y}_2^i, \bar{z}^i, 1 \le i \le \bar{k}$, и $\bar{\bar{y}}_2^j, 1 \le j \le \bar{\bar{k}}$, относятся к информационным множествам (9):

$$\bar{\omega}_{\bar{k}} = \{(\bar{y}_2^1, \bar{z}^1), ..., (\bar{y}_2^{\bar{k}}, \bar{z}^{\bar{k}})\},$$

$$\bar{\bar{\omega}}_{\bar{\bar{k}}} = \{(\bar{\bar{y}}_2^1, \bar{\bar{z}}^1), ..., (\bar{\bar{y}}_2^{\bar{\bar{k}}}, \bar{\bar{z}}^{\bar{\bar{k}}})\},$$

полученным при решении одномерных задач (8) для $\bar{y}_1$ и $\bar{\bar{y}}_1$ соответственно, причем пара $(\hat{y}_2^i, \bar{z}^i) \in \bar{\bar{\omega}}_{\bar{\bar{k}}}$.

Введение информационного множества (11) позволяет заменить вычисление значений функции $f(y_1^*, y_2)$ в точках $\bar{y}_2^{i*}, 1 \le i \le \bar{k}$ построением приближенных величин z^{i*}, полученных на основе линейной аппроксимации (12), (13). Построив информационное множество (11), можно продолжить решение задачи (10) характеристическим методом оптимизации. При этом метод будет вычислять функцию $f(y_1^*, y_2)$ в меньшем числе точек по сравнению с ситуацией, когда он решает задачу (10) «с чистого листа», ибо в предложенной схеме алгоритм обладает апостериорной информацией (11), т.е. считает, что в точках $\bar{y}_2^{i*}, 1 \le i \le \bar{k}$ значения уже известны.

Вычислительные эксперименты, выполненные на классе сложных двухмерных функций [4], подтверждают эффективность предложенного подхода, обеспечившего существенное сокращение количества поисковых испытаний. При этом наблюдается рост эффективности при увеличении требуемой точности решения задачи. Параллельная реализация, использующая асинхронные алгоритмы из [5], подтвердила теоретические результаты относительно ускорения, достигаемого при использовании рассмотренной рекурсивной схемы редукции размерности.

Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. Информационно- статистический подход. М.: Наука, 1978.
2. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints: Sequential and parallel algorithms, Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

3. 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.
4. Гришагин В.А. Операционные характеристики некоторых алгоритмов глобального поиска. В сб.: Проблемы статистической оптимизации. Зинатне, Рига, 1978. С. 198–206.
5. 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.

СПЕЦКУРС «ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ: ОСНОВЫ ПРОГРАММИРОВАНИЯ И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ» КАК ПРОДОЛЖЕНИЕ И РАЗВИТИЕ ЛИНИИ «МОДЕЛИРОВАНИЕ» В ПОДГОТОВКЕ УЧИТЕЛЯ ИНФОРМАТИКИ

А.Г. Деменев

Пермский государственный педагогический университет

Введение

Применение суперкомпьютеров просто необходимо в ракето- и автомобилестроении, нефте- и газодобыче, фармакологии, биотехнологии и других областях человеческой деятельности. Многие из последних успехов фундаментальных исследований в физике, химии, биологии, которые существенно расширили горизонты наших знаний о мире, обеспечены компьютерным моделированием на суперЭВМ. Широкое внедрение идей параллелизма и конвейерности в вычислительную технику привело к значительному пересмотру всей концепции прикладного программирования.

Технология создания сложных компьютерных моделей требует, чтобы эксперт в данной проблемной области ставил перед программистами задачу реализации их в виде программ для ЭВМ. С одной стороны, требуются специалисты, умеющие так сформулировать задачу в своей области деятельности, чтобы она, в принципе, допускала эффективную реализацию на современных высокопроизводительных ЭВМ. С другой стороны, нужны высококвалифицированные программисты, свободно владеющие технологиями разработки параллельных про-

грамм. Есть основания считать, что для подготовки таких специалистов в требуемых количестве и качестве, следует знакомить наиболее талантливых детей с основами параллельных технологий еще в старших классах специализированных школ. А это означает, что нужны учителя информатики, готовые вести такую работу.

Современный выпускник вуза с квалификацией «Учитель информатики» должен иметь достаточно четкие знания о параллельных вычислительных системах. В рамках подготовки бакалавров образования элементы этих знаний могут быть представлены в рамках дисциплины «Компьютерное моделирование», а при получении специальности «Учитель информатики» и обучении в магистратуре – в виде различных спецкурсов.

Спецкурсы, связанные с параллельными вычислениями, читаются в Пермском государственном педагогическом университете с 1999 года. Слушателями их становятся студенты последних курсов педагогического университета, проходящие обучение с основным профилем «информатика». Знакомство с основами программирования и компьютерного моделирования параллельных вычислительных систем осуществляется в рамках двух спецкурсов для студентов факультета информатики и экономики. Для студентов, обучающихся по специальности «Учитель информатики», автором разработан курс «Параллельные вычислительные системы». Студентам, получающим ученую степень магистра и квалификацию «Учитель информатики, преподаватель высшей школы», адресован разработанный автором курс «Программирование для параллельных вычислительных систем». Объем последнего из них в два раза больше, и содержание первого практически полностью включено в него. Общий материал, используемый при проведении занятий в рамках обоих спецкурсов, представлен в разработанном автором учебном пособии [1].

Основные цели и задачи спецкурса

Спецкурс нацелен на решение следующих проблем:

- подготовка учителя информатики, готового знакомить учеников специализированных школ с основами параллельных технологий в рамках профильных курсов по информатике;
- углубление образования в области информатики;

- развитие практических навыков в компьютерном моделировании, информационной культуры.

Курс включает в себя как лекционную часть, так и лабораторную поддержку.

В лекционной части излагаются основные теоретические понятия и примеры ПВС, излагаются типичные подходы при программировании для таких систем.

Чрезвычайно важную роль в курсе играют лабораторные работы. При их выполнении предусматривается следующие режимы (один из них или сочетание — по выбору преподавателя): выполнение расчетов «вручную» с заполнением всех необходимых таблиц для промежуточных результатов; проведение расчетов в среде, имитирующей ПВС.

При выполнении лабораторных работ должен предусматриваться следующий режим: проведение расчетов в среде, имитирующей ПВС. Работа на реальных параллельных компьютерах не обязательна, но желательна, в первую очередь магистрантам, т.е. студентам, ориентированным на научную деятельность и преподавательскую работу в вузах и специализированных школах.

Иметь доступ к нескольким типам параллельных компьютеров – недешевое удовольствие, а использовать программы, моделирующие ПВС, на персональных компьютерах могут позволить себе все. Можно рекомендовать, например, бесплатно распространяемую среду Multi-Pascal [2].

Методологии программирования обычно иллюстрируются на сравнительно несложных задачах, например, требующих реализации отдельных алгоритмов из области линейной алгебры, вычисления определенных интегралов и т.д.[3] Все это можно сделать, используя компьютерное моделирование в Multi-Pascal. Так как разработка параллельных программ практического уровня сложности представляет собой многоэтапный технологический процесс, то, к сожалению, она не может быть продемонстрирована во всей полноте на таких задачах. В силу серьезной ограниченности аудиторного времени на спецкурс, представляется разумным не ставить задачу разработки параллельных программ практического уровня сложности, а делать это в рамках курсовых и дипломных работ.

Содержание лекций

Тема 1 – Введение в ПВС. Представляется важным в рамках данной темы дать ответы на следующие вопросы: Что такое суперЭВМ? Зачем нужны суперкомпьютеры? Чем определяются перспективы развития суперЭВМ? Что делается в России по развитию суперкомпьютерных технологий? Как устроены современные высокопроизводительные ЭВМ? Как классифицируются компьютерные архитектуры? Каковы основные концепции архитектуры высокопроизводительных вычислительных систем? Что такое конвейер? Какие процессоры называют суперскалярными? Какие процессоры называют векторными? Чем различаются векторные компьютеры между собой? Какая оперативная память используется в современных ЭВМ? Что такое разделяемая память? Что такое распределенная память? Как связаны между собой элементы параллельных вычислительных систем? Что такое кластеры рабочих станций? Как оценивается производительность вычислительных систем?

Тема 2 – Основные парадигмы параллельного программирования. В рамках данной темы освещаются следующие вопросы: Каковы основные подходы к распараллеливанию вычислений? Как реализуется параллелизм данных? Какой набор операций является базовым? Какие способы векторизации и распараллеливания программ применяются, чтобы оптимизировать трудовые затраты? На что влияет применение разных языков программирования? В чем различие и сходство между распараллеливанием и векторизацией программ? Как реализуется параллелизм задач? Какие виды операционных систем используются на параллельных вычислительных системах? На каких принципах построены распределенные ОС?

Тема 3 – Программирование систем с разделяемой памятью. В рамках данной темы дают ответы на следующие вопросы: Какие операционные системы применяют на мультипроцессорных ЭВМ? Что понимается под процессом? Как организуется взаимодействие процессов? Как планирование процессоров влияет на производительность мультипроцессорной системы? Как в среде Multi-Pascal моделируются многопроцессорные компьютеры с разделяемой памятью? Как в Multi-Pascal реализуется блокирующее порождение процессов в системах с разделяемой памятью? Как в Multi-Pascal реализуется неблокирующее порождение процессов в системах с разделяемой памятью? Как в

Multi-Pascal решается проблема взаимодействия процессов в системах с разделяемой памятью?

Тема 4 – Программирование систем с распределенной памятью. Отбор материала для данной темы, в силу своей сложности и ограниченности аудиторного времени на нее, вызывал наибольшие трудности. В текущем варианте спецкурса пришлось остановиться на рассмотрении следующих вопросов: В чем преимущества и недостатки распределенных систем по сравнению с централизованными ЭВМ? Как в среде Multi-Pascal моделируются многопроцессорные компьютеры с распределенной памятью? Как в Multi-Pascal задаются целевые платформы, относящиеся к системам с распределенной памятью? Что такое коммуникационные порты Multi-Pascal? Чем может быть полезно назначение процессов процессорам и как это сделать в Multi-Pascal?

Лабораторный практикум

Каждая из лабораторных работ практикума представляет собой микроисследование. Разработанные задания лабораторных работ по моделированию ПВС предусматривают работу в бесплатно распространяемой среде Multi-Pascal. Эта среда позволяет, используя обычный персональный компьютер, имитировать вычисления на ПВС разных архитектур.

Как известно, простейшая модель, описывающая ускорение (коэффициент эффективности распараллеливания) S, которое теоретически достижимо на компьютере из N процессоров, может быть получена из закона Амдала:

$$S \le \frac{1}{f + (1-f)/N} ,$$

где f – доля участков в программе, которые не могут быть распараллелены. Крайние случаи в значениях f соответствуют полностью параллельным ($f = 0$) и полностью последовательным ($f = 1$) программам. Используя профилировщик (или отладчик), можно оценить время выполнения на однопроцессорной машине последовательного участка программного кода (t) и всей программы (T). Тогда $f = t/T$ и

$$S_{\max} = \frac{1}{t/T + (1-t/T)/N} .$$

Реальная выгода от увеличения числа процессоров всегда меньше:

- во всех параллельных системах – из-за накладных расходов на порождение параллельных потоков;
- в системах с разделяемой памятью – в основном из-за ожиданий процессоров возможности получения доступа к памяти;
- в системах с распределенной памятью – в основном из-за задержек межпроцессорных коммуникаций.

При анализе результатов необходимо сравнивать результаты компьютерного моделирования в среде Multi-Pascal с предсказаниями такой простой математической модели. Где возможно, подбирать более реалистичную полуэмпирическую модель, включающую дополнительно 1 или 2 подгоночных параметров. Ввод параметров должен иметь достаточно правдоподобное обоснование.

Для ускорения проведения расчетов удобно использовать переназначение стандартных ввода-вывода.

Варианты заданий формируются требованием: использовать разные программы типовых алгоритмов (вычисления числа π, перемножения матриц, метода Якоби и т.п.) и/или разные входные параметры (число расчетных точек, размер матриц и т.д.). Число вариантов – в зависимости от имеющихся в наличии готовых программ.

Лабораторная работа «Основы моделирования ПВС в среде Multi-Pascal» – это знакомство со средой Multi-Pascal.

Лабораторная работа «Моделирование ПВС с разделяемой памятью» включает два задания на моделирование вычислений на многопроцессорных компьютерах. Первое задание – это имитация монопольного доступа, второе – имитация коллективного доступа.

Лабораторная работа «Моделирование ПВС с распределенной памятью» в два раза больше предыдущей. Студентам предлагается четыре задания. Задание 1 – моделирование вычислений на многопроцессорных компьютерах монопольного доступа. Задание 2 – моделирование вычислений на кластерах монопольного доступа. Задание 3 – моделирование вычислений на многопроцессорных компьютерах заданной топологии. Задание 4 – моделирование вычислений на компьютерных кластерах заданной топологии.

Заключение

Представленный в данной работе спецкурс помогает решить проблему подготовки учителя информатики, готового знакомить учеников

специализированных школ с основами параллельных технологий в рамках профильных курсов по информатике. Это представляется важным, учитывая принципиальные отличия концепций оптимального программирования для параллельных вычислительных систем от популярных концепций для персональных ЭВМ. Данный спецкурс и разработанные к нему учебно-методические материалы прошли успешную апробацию в Пермском государственном педагогическом университете. Часть этих материалов апробируется в настоящее время в рамках спецкурсов по параллельным вычислениям, читаемых в Пермском государственном университете. Предварительный анализ нашего опыта показывает, что разработанный спецкурс полезен при подготовке не только учителей информатики, но бакалавров прикладной математики и информатики.

Разработка данного спецкурса была поддержана грантом Минобразования РФ в рамках проекта «Разработка учебно-программно-методического комплекса “Компьютерное математическое моделирование” для подготовки студентов педагогических вузов к профессиональной деятельности».

Литература

1. Деменев А.Г. Параллельные вычислительные системы: основы программирования и компьютерного моделирования. Учебное пособие к спецкурсу. Пермь: ПГПУ, 2001. 59 с.+Прил. (6 с.)
2. Lester Bruce P., The Art of Parallel Programming/ Prentice Hall, Englewood Cliffs, NJ 07632, 1993.
3. Программирование на параллельных вычислительных системах/ Пер. с англ. под ред. Р. Бэбба. М.: Мир, 1991. 320 с.

МОДЕЛИРОВАНИЕ ДИНАМИКИ СЛОЖНЫХ КВАНТОВЫХ СИСТЕМ

В.Я. Демиховский, А.И. Малышев

Нижегородский государственный университет им. Н.И. Лобачевского

В последнее время внимание физиков и математиков обращено на исследование сложных (многомерных) квантовых систем. Этот интерес связан, в частности, с изучением эволюции состояний большого числа кубитов, составляющих основу квантового компьютера.

Как известно, при описании динамики N классических частиц объем вычислений (число дифференциальных уравнений первого порядка) растет пропорционально их числу. В то же время, как отмечено в известной работе Р.Фейнмана [1], объем вычислений при расчете совместной эволюции ансамбля N квантовых частиц растет экспоненциально с ростом N. Так, для моделирования поведения N двухуровневых квантовых систем (кубитов) необходимо решать одновременно 2^N дифференциальных уравнений. Определение стационарных квантовых состояний в такой системе требует диагонализации матрицы размером $2^N \times 2^N$. Ясно, что в такой ситуации возможности классических алгоритмов вычислений жестко ограничены.

Аналогичные проблемы возникают уже при исследовании динамики одной квантовой частицы, находящейся во внешнем поле, зависящем от нескольких переменных (например, $U(x, y, z, t)$). Если представить волновую функцию такой системы в виде ряда по M функциям невозмущенного базиса $\psi_n(x)$, $\psi_m(y)$, $\psi_l(z)$, то число коэффициентов этого разложения $C_{n,l,m}(t)$, очевидно, составит M^3. При разумном выборе M (порядка 100), число решаемых уравнений стремится к миллиону.

Приведены конкретные примеры расчетов динамики квантовых систем:

а) квантовая диффузия Арнольда на примере двух взаимодействующих нелинейных осцилляторов, находящихся во внешнем периодическом поле [2];

б) эволюция двухуровневых взаимодействующих квантовых систем (кубитов) [3].

Литература

1. R.P. Feynman // *Int. J. Theor. Phys.*, 21, 467, 1982.
2. V.Ya. Demikhovskii, F.M. Izrailev and A.I. Malyshev // *Int. Conf. «Progress In Nonlinear Science», N. Novgorod, 2001.*
3. G.P. Berman *et.al.*, quant-ph/0110069 v1. (To be published in Phys. Rev. E), 2001.

РЕАЛИЗАЦИЯ АЛГОРИТМОВ МОДУЛЬНОЙ АРИФМЕТИКИ В КЛАСТЕРНЫХ СИСТЕМАХ

М.С. Дрейбанд

Нижегородский государственный технический университет, Институт прикладной физики РАН

Введение

В связи с использованием вычислительных кластеров возникает большое число вопросов, связанных с оптимальным использованием их ресурсов. При создании параллельных программ возникают следующие проблемы:

- Программисту сложно переобучиться, ведь нужно освоить специальные языковые конструкции, или даже новые языки программирования.
- Сложность создания и отладки параллельных алгоритмов.
- Накоплены готовые последовательные программы, которые теперь нужно переводить на язык параллельных вычислений.
- Далеко не все программные и аппаратные платформы, обеспечивающие параллельные вычисления, совместимы друг с другом.

Снять эти проблемы позволяют библиотеки, реализующие «скрытый» параллелизм вычислений базовых арифметических операций.

При создании такой библиотеки был использован следующий подход: все операции над числами выполняются в «модульном представлении». Целое число представляют вычетами по модулям из множества взаимно-простых чисел p_i. Если $p_1,\dots\ p_{k-1}$ – попарно взаимно простые числа и $p=p_1*\dots* p_{k-1}$, то любое целое число u от 0 до p можно однозначно представить множеством его вычетов $u_1,\dots\ u_{k-1}$, где $u_i=u(mod\ p_i)$. Сложение, вычитание и умножение легко выполняются, если эти вычисления рассматривать как операции, определенные в поле классов вычетов по модулям p_i.

При использовании такого подхода в последовательной реализации потенциально можно получить выигрыш на операции умножения, а в параллельной реализации – на всех арифметических операциях. Действия относительно разных модулей могут производиться одновременно и, тем самым, мы получим существенную экономию во вре-

мени выполнения. Добиться же такой экономии с помощью традиционных методов нельзя, так как необходимо учитывать передачу переноса.

Реализация библиотеки

Библиотека, реализующая модульные вычисления, написана на языке C с использованием функций библиотеки MPI.

Все числа представляются в виде переменных типа *long*, т.е. 64 бита на платформе Intel. Модульное представление чисел записывается в виде массива с элементами типа *long*.

Все библиотечные функции можно условно разбить на несколько групп:

- Функции преобразования чисел – переводят число из полиномиального (десятичного) представления в модульное, и наоборот. Необходимо обратить внимание, что функция преобразования числа из модельного представления в десятичное требует «синхронизации» всех компьютеров, т.е. перехода независимо выполняющихся программ в одну точку алгоритма.
- Вычислительные функции – выполняются арифметические операции: умножение, сложение, вычитание чисел, записанных в модульном представлении.
- Вспомогательные функции – используются первыми двумя группами: вывод числа в модульном представлении на экран, нахождение остатка от деления двух чисел.

Показатели эффективности и ускорения

Тестирование программ, написанных с использованием библиотеки модульной арифметики, проводилось в двух системах: вычислительном кластере на базе серверов Compaq, принадлежащем ИПФ РАН (5 узлов Compaq Server DS E20, по 2 процессора Alpha 21264 667MHz), и в учебном классе НГТУ (10 компьютеров Pentium 166).

Из анализа тестов был сделан вывод, что ускорение программ пропорционально логарифму числа процессоров, поэтому использование библиотеки оправдано при небольшом числе процессоров.

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ ИДЕНТИФИКАЦИИ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ НА РЕШЕТКЕ

Н.Ю. Золотых

Нижегородский государственный университет им. Н.И.Лобачевского

Рассматриваются параллельные алгоритмы идентификации геометрических объектов, заданных на целочисленной решетке. Произвольное подмножество c точек n-мерного k-значного гиперкуба $E_k^n=\{0,1, \dots , k-1\}^n$, где $n \geq 2$, $k \geq 2$, назовем объектом. Произвольное семейство C таких подмножеств назовем классом объектов. Одна из задач точного машинного обучения (*machine learning*) заключается в идентификации объекта c из заданного класса C с помощью вопросов оракулу вида `$x \in c$?', где x – точка, произвольно выбираемая из E_k^n. Поставленная задача легко сводится к задаче расшифровки дискретной функции из некоторого класса (см., например, [1], приложение 1 и библиографию).

Для идентификации используется машина UPRAM с p процессорами [2]. За один шаг на такой машине возможно параллельное обращение к оракулу в p точках.

Обозначим через $\mathrm{HALFSPACE}_k^n$ множество объектов c, для каждого из которых найдутся действительные числа $a_0, a_1, \dots, a_n$, такие, что

$$c = \{x \in E_k^n : \sum_{j=1}^{n} a_j x_j \leq a_0\} . \qquad (1)$$

Теорема 1 [3, 4] *Существует алгоритм* A_1, *который по произвольному объекту* $c \in \mathrm{HALFSPACE}_k^2$, *используя не более* $6 \log (k-1)+4$ *обращений к оракулу и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с одним процессором определяет коэффициенты неравенства, при которых выполняется* (1).

Теорема 2 *Существует алгоритм* A_2, *который по произвольному объекту* $c \in \mathrm{HALFSPACE}_k^2$, *используя не более* $3 \log (k-1)+2$ *обращений к оракулу и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с 2 процессорами определяет коэффициенты неравенства, при которых выполняется* (1).

Теорема 3 *Существует алгоритм* A_3, *который по произвольному объекту* $c \in \mathrm{HALFSPACE}_k^2$, *используя не более* $\log (k-1)+1$ *обращений к оракулу и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с 4*

процессорами определяет коэффициенты неравенства, при которых выполняется (1).

Теорема 4 *Существует алгоритм* A_4*, который по произвольному объекту* $c \in \mathrm{HALFSPACE}_k^2$*, используя не более* $6 \log (k-1)+4$ *обращений к оракулу и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с* $\sqrt{k}$ *процессорами определяет коэффициенты неравенства, при которых выполняется* (1).

Результаты, относящиеся к идентификации объектов из класса $\mathrm{HALFSPACE}_k^n$ для $n > 2$, см. в [5, 6, 7].

Обозначим через m–$\mathrm{HALFSPACE}_k^2$ множество объектов $c \subseteq E_k^2$, для каждого из которых найдутся действительные числа $a_{10}, a_{11}, a_{12}, \ldots, a_{m0}, a_{m1}, a_{m2}$, такие, что

$$c = \{x \in E_k^n: a_{11}x_1 + a_{12}x_2 \le a_{10}, \ldots, a_{m1}x_1 + a_{m2}x_2 \le a_{m0}\}. \qquad (2)$$

Теорема 5 *[8] Существует алгоритм* A_5*, который по произвольному объекту* $c \in m$–$\mathrm{HALFSPACE}_k^2$ *и произвольной тройке неколлинеарных точек из c, используя не более* $(10m + 1) \log (k-1) + 34$ *вопросов и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с 1 процессором определяет коэффициенты* $a_{10}, a_{11}, a_{12}, \ldots, a_{m0}, a_{m1}, a_{m2}$*, при которых выполняется равенство* (2).

Теорема 6 *Существует алгоритм* A_6*, который по произвольному объекту* $c \in m$–$\mathrm{HALFSPACE}_k^2$ *и произвольной тройке неколлинеарных точек из c, используя не более* $(11m+1) \log (k-1)+12$ *вопросов и* $O(\log k)$ *арифметических операций, на машине* UPRAM *с m процессорами определяет коэффициенты* $a_{10}, a_{11}, a_{12}, \ldots, a_{m0}, a_{m1}, a_{m2}$*, при которых выполняется равенство* (2).

Обозначим через BOX_k^n множество объектов $c \subseteq E_k^n$, для каждого из которых найдутся числа $a_1, \ldots, a_n$ и $b_1, \ldots, b_n$, такие, что

$$c = \{x \in E_k^n: a_1 \le x_1 \le b_1, \ldots, a_n \le x_n \le b_n\}. \qquad (3)$$

Теорема 7 *Существует алгоритм* A_7*, который по произвольному объекту* $c \in \mathrm{BOX}_k^n$ *и произвольной точке* $x \in c$*, используя не более* $2n \log (k-1)$ *вопросов и* $O(n \log k)$ *арифметических операций, на машине* UPRAM *с 1 процессором находит числа* $a_1, \ldots, a_n$ *и* $b_1, \ldots, b_n$*, при которых выполняется равенство* (3).

Теорема 8 *Существует алгоритм* A_8*, который по произвольному объекту* $c \in \mathrm{BOX}_k^n$ *и произвольной точке* $x \in c$*, используя не более*

log (k–1) *вопросов и* $O(n \log k)$ *арифметических операций, на машине* UPRAM *с* $2n$ *процессорами находит числа* $a_1, \ldots, a_n$ *и* $b_1, \ldots, b_n$, *при которых выполняется равенство* (3).

Обозначим через BALL_k^n множество объектов $c \subseteq E_k^n$, для каждого из которых найдутся точка $y \in E_k^n$ и целое число r, такие, что

$$c = \{x \in E_k^n : \sum_{j=1}^{n} (x_j - y_j)^2 \le r\}. . \qquad (3)$$

Теорема 9 [8] *Существует алгоритм* A_9, *который по произвольному объекту* $c \in \mathrm{BALL}_k^n$ *и произвольной точке* $x \in c$, *используя* $O(n \log k)$ *вопросов, находит* y *и* r, *при которых выполняется равенство* (4).

Часть результатов для случая $n = 2$ сведем в таблицу:

Класс	*Число процессоров*	*Сложность алгоритма*
$\mathrm{HALFPLANE}_k^2$	1	$6 \log(k-1) + 4$
	2	$3 \log(k-1) + 2$
	4	$2 \log(k-1) + 1$
	$\sqrt{k}$	$\log(k-1) + 4$
	k	3
	k^2	1
m–$\mathrm{HALFPLANE}_k^2$	1	$(10m + 1) \log (k-1)+34$
	m	$11 \log(k-1)+12$
BOX_k^2	1	$4 \log(k-1) - 4$
	2	$2 \log(k-1) - 2$
	4	$\log(k-1) - 1$
	k	2
BALL_k^n	1	$O(\log k)$

Литература

1. Шевченко В. Н. Качественные вопросы целочисленного программирования. М.: Физматлит, 1995. 192 с.

2. Bshouty N.H., Cleve R. On the exact learning of formulas in parallel // Proc. of the 33rd Symposium on the Foundations of Comp. Sci. IEEE Computer Society Press, Los Alamitos, CA, 1992. P. 513–522.
3. Золотых Н. Ю. Пороговые функции, зависящие от двух переменных: сложность расшифровки и мощность разрешающего множества // Материалы четвертой молодежной научной школы по дискретной математике и ее приложениям. М.: Изд-во механико-математического факультета МГУ, 2000. С. 48–54.
4. Веселов С. И. Расшифровка одного класса функций // Материалы XI Межгосударственной школы-семинара ``Синтез и сложность управляющих систем". Часть I. М.: Изд-во центра прикладных исследований при механико-математическом факультета МГУ, 2001. С. 39–40.
5. Шевченко В.Н., Золотых Н.Ю. О сложности расшифровки пороговых функций *k*-значной логики // Доклады Академии наук. 1998. Т. 362, 5. С. 606–608.
6. Золотых Н.Ю., Шевченко В.Н. О нижней оценке сложности расшифровки пороговых функций *k*-значной логики // Журнал вычислительной математики и математической физики. 1999. Т. 39, 2. С. 346–352.
7. Shevchenko V.N., Zolotykh N.Y. Lower bounds for the complexity of learning half-spaces with membership queries // Proc. of the 9th International Conference on Algorithmic Learning Theory – ALT'98. V. 1501 Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998, P. 61–71.
8. Веселов С. И., Золотых Н. Ю. Идентификация геометрических образов на целочисленной решетке // Материалы 7-ой международной конференции по дискретной математике и ее приложениям. В печати.

ОРГАНИЗАЦИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА КЛАСТЕРЕ PARC УДМУРТСКОГО ГОСУНИВЕРСИТЕТА

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

Удмуртский государственный университет, г.Ижевск

С момента появления в 1971 году первого микропроцессора i4004 фирмы Intel и создания на его основе микропроцессорных вычислительных устройств развитие вычислительной техники сделало гигантский скачок. Основной прирост вычислительной мощности компьютеров достигался как за счет улучшения технологических норм производства (и, как следствие, тактовых частот), так и за счет улучшения архитектурной организации микропроцессоров и периферийных устройств. При этом, начиная с начальных тактовых частот порядка 750 КГц микропроцессора i4004, микропроцессоры достигли тактовых частот в 1700–2000 МГц. Однако, дальнейшее увеличение производительности за счет повышения тактовых частот становится несколько затруднительным, так как практически достигнуты минимальные размеры транзисторов на кристалле и появились другие сдерживающие факторы. Поэтому современная вычислительная техника все более отходит от традиционной архитектуры и создаются новые направления повышения производительности вычислительных систем.

Достигнутый уровень технологии производства надежных высокоскоростных шин и мощных процессоров, сетевого оборудования и развитое программное обеспечение распределенных и параллельных вычислений позволяют создавать кластеры высокопроизводительных компьютеров, способные обеспечить эффективные расчеты широкого класса научных и технических задач, а также планомерную подготовку специалистов в области высокопроизводительных вычислений. В ноябре 2000 г. по инициативе ректора Удмуртского госуниверситета профессора В. А. Журавлева был создан кластер PARC, состоящий из пяти двухпроцессорных компьютеров. Общая характеристика этому кластеру дана в учебно-методическом пособии [1], описывающем основные особенности и правила работы на кластере PARC.

На примере модельной задачи управления температурой тонкого стержня мы решаем учебно-методические вопросы организации высокопроизводительных вычислений на кластере PARC.

Рассмотрим в течение времени T процесс изменения температуры $u(x,t)$ в тонком стержне длины l с коэффициентом теплопроводности $k(x)$ при воздействии на него тепловыми источниками и стоками плотности $F(x,t)=b(x)v(t)$ в точке $x\in [0,l]$ в момент $t\in[0,T]$. Известно, что температура $u(x,t)$ должна удовлетворять дифференциальному уравнению теплопроводности

$$c\rho(x)\frac{\partial u(x,t)}{\partial t}=\frac{\partial}{\partial x}\left(k(x)\frac{\partial u(x,t)}{\partial x}\right)+b(x)v(t)\,,\ (x,t)\in(0,l)\times(0,T)\,. \qquad (1)$$

Здесь c – удельная теплоемкость, $\rho(x)$ – функция плотности распределения массы стержня.

С помощью двух функций $\alpha(x,t)$ и $\beta(x,t)$ ($\alpha\ (x,t)\le\beta\ (x,t)$) зададим диапазон изменения температуры

$$\alpha(x,t)\le u(x,t)\ \le\beta(x,t). \qquad (2)$$

Задачу управления поставим следующим образом. Требуется построить такое кусочно-постоянное управление $v=v(t)$ $\left(|v(t)|\ \le 1,\ t\in[0,T]\right)$, при котором уравнение (1) имеет гладкое решение $u(x,t)$, удовлетворяющее граничным условиям

$$u(0,t)=\mu(t)\,,\ \ u(l,t)=v(t)$$

и дополнительному ограничению (2) в каждый фиксированный момент времени $t=t_i$ ($0\le t_1<t_2<\ldots<t_{\mathrm{m}}\le T$).

По обычной схеме дискретизации пространственной переменной получается конечномерная управляемая система:

$$c\rho(x_1)\frac{\partial u(x_1,t)}{\partial t}=\frac{1}{h}\left(a_2\frac{u(x_2,t)-u(x_1,t)}{h}-a_1\frac{u(x_1,t)-u(0,t)}{h}\right)+b(x_1)v(t)\,,$$

$$c\rho(x_i)\frac{\partial u(x_i,t)}{\partial t}=\frac{1}{h}\left(a_{i+1}\frac{u(x_{i+1},t)-u(x_i,t)}{h}-a_i\frac{u(x_i,t)-u(x_{i-1},t)}{h}\right)+$$

$$+b(x_i)v(t),\qquad i=2,....N-2,$$

$$c\rho(x_{N-1})\frac{\partial u(x_{N-1},t)}{\partial t}=$$

$$\frac{1}{h}\left(a_N\frac{u(l,t)-u(x_{N-1},t)}{h}-a_{N-1}\frac{u(x_{N-1},t)-u(x_{N-2},t)}{h}\right)+b(x_{N-1})v(t),$$

граничные условия для которой имеют следующий вид:

$$\alpha\ (x_i\,,\,t_j)\ \le u(x_i\,,\,t_j)\ \le\beta\ (x_i\,,\,t_j),\ \ j=1,\ldots m,\ \ i=1,\ldots N\text{–}1$$

Для описания свойств требуемого управления данной конечномерной задачи сформулирован принцип максимума, позволяющий определить точки переключения.

В работе получен конструктивный параллельный алгоритм построения оптимального кусочно-постоянного управления, опирающийся на описанный ранее принцип максимума. Вводятся основные понятия, необходимые при описании и построении параллельных алгоритмов, рассматривается решение данной модельной задачи с привлечением различных вычислительных средств, используемых на кластере PARC: сетевые возможности ОС Linux, средства параллельного программирования, использующие механизмы разделяемой (Pthreads) и распределенной (PVM, MPI) памяти.

Литература

1. PARC – кластер высокопроизводительных компьютеров /Исламов Г.Г., Мельчуков С.А., Клочков М.А., Бабич О.В., Сивков Д.А. Учебно-методическое пособие. Ижевск: УдГУ, 2001. 66 с.

СТРУКТУРЫ ДАННЫХ ДЛЯ РЕАЛИЗАЦИИ АДАПТИВНЫХ ДИАГОНАЛЬНЫХ КРИВЫХ*

Д.Е.Квасов

Нижегородский государственный университет им. Н.И.Лобачевского
Калабрийский университет, Козенца, Италия

Рассматривается классическая задача глобальной оптимизации [1]:

$$\min f(x),\ x \in D, \tag{1}$$

$$D = \{ x \in R^N \mid a_i \le x_i \le b_i,\ i=1,\ldots,N \}. \tag{2}$$

Предполагается, что функция $f(x)$ может быть недифференцируемой, и в ходе решения (1)-(2) могут быть найдены лишь значения $f(x)$ в точках D, причем вычисление значений $f(x)$ требует больших затрат времени. Задача (1)-(2) является частным случаем задачи минимального описания функции [2].

* Работа поддержана грантом РФФИ № 01-01-00587.

Одним из подходов к решению задачи (1)-(2) является *диагональный подход* [3], обобщающий характеристические алгоритмы [4] на многомерный случай. Идея данного подхода заключается в последовательном разбиении области поиска D на множество гиперинтервалов D_i и вычислении функции *f(x)* в вершинах главных диагоналей получаемых гиперинтервалов. Для каждого гиперинтервала D_i вычисляется его *характеристика* R_i, отражающая «пригодность» данного гиперинтервала для последующего разбиения. Характеристики выбираются так, что чем больше значение R_i, тем больше вероятность нахождения точки глобального минимума *f(x)* в D_i. На каждой итерации алгоритма очередному разбиению подвергается гиперинтервал D_t с максимальным значением характеристики R_t, т.е.

$$t = \arg\max \{R_i \mid 1 \le i \le m(k)\}, \qquad (3)$$

где $m(k)$ есть число гиперинтевалов D_i на текущей k-й итерации алгоритма. Разбиение D_t осуществляется гиперплоскостями, параллельными координатным плоскостям и проходящими через некоторую точку главной диагонали D_t.

Традиционно применяемыми стратегиями разбиения являются деление пополам и деление на 2^N [3]. Как было показано в [2], при данных стратегиях генерируется множество избыточных точек испытания *f(x)*, что приводит как к замедлению работы алгоритма, так и к увеличению машинной памяти для хранения необходимой информации. Избыточность точек испытания является следствием двух факторов:

1. Каждый гиперинтервал D_i содержит более двух точек, в которых вычисляется функция *f(x)*.

2. Теряется пространственная связь между гиперинтервалами, полученными на разных итерациях алгоритма.

В работе [2] была предложена новая стратегия, которая преодолевает недостатки традиционных стратегий разбиения области поиска D. Данная стратегия генерирует последовательность новых непрерывных кривых, заполняющих пространство («space-filling curves»), – адаптивных диагональных кривых (АДК). Обычно в численных алгоритмах используется аппроксимация некоторой кривой (например, кривой Пеано, см. [1]), заполняющей пространство, с заданным порядком разбиения (одинаковым для всей области D). Порядок же разбиения АДК отличается в разных подобластях D: чем больше точность поиска, тем сильнее АДК разбивается в подобластях,

предположительно содержащих точки глобального минимума функции *f(x)*. Разбиению области D из (2) на текущем шаге алгоритма соответствует АДК с начальной точкой в вершине *a* и конечной – в вершине *b*, которая образована главными диагоналями подобластей D_i. Новая АДК строится путем подразбиения текущей АДК в пределах выбранного гиперинтервала D_t (см. (3)) без изменеия остальных участков кривой, причем D_t разбивается на три гиперинтервала равного объема двумя параллельными гиперплоскостями, проходящими перпендикулярно стороне D_t с наибольшей длиной. Таким образом, АДК остается неразрывной в течение всего процесса разбиения. В силу своего построения АДК является фрактальным объектом.

Методы, использующие АДК, не генерируют избыточных точек испытания *f(x)* и вычисляют *f(x)* непосредственно в N-мерной области D без преобразования аргумента функции, что позволяет в ходе поиска использовать не только значения функции *f(x)* в точках D, но и значения градиента *f(x)*, увеличивая тем самым скорость работы алгоритма поиска.

Теорема [2]. Существует специальная индексация гиперинтервалов D_i, полученных при генерировании АДК на текущей итерации алгоритма, позволяющая по индексу гиперинтервала D_i находить координаты его вершин $a(i)$ и $b(i)$, в которых вычисляется *f(x)*.

Информация о функции в конкретной вершине вычисляется только один раз, а затем просто считывается из некоторой базы данных, причем метод в состоянии определить, была ли *f(x)* уже вычислена в вершинах, сгенерированных на текущей итерации. Так как каждая точка испытания *f(x)* может принадлежать различным (до 2^N) гиперинтервалам, то может возникнуть необходимость считывать 2^N раз одну и ту же информацию о значении функции. Поэтому правильная организация данных в связи с фактом существования индексации гиперинтервалов позволяет существенно повысить эффективность процедуры поиска.

Возможно построение различных структур данных для реализации АДК. Например, будем хранить информацию в специализированной базе данных, структура которой приведена на рисунке 1.

Координаты вершин X и соответствующая им информация F(X) (значения функции, градиента и др.) хранятся в записях v_i массива вершин. Записи массива вершин образуют (при помощи 2N указателей) двунаправленные линейные списки, каждый из которых

соответствует одной из N координат. Информация о гиперинтервалах (значение характеристик R_i, описательная информация $INFO_i$) организуется в виде линейного списка, элементы которого соответствуют данным о каждом из существующих на текущей итерации алгоритма гиперинтервалов. В таком случае гиперинтервалы D_i из списка содержат только указатели A_i и B_i на записи из массива вершин и не дублируют информацию о вершинах $a(i)$ и $b(i)$. Например, подобласти с индексами i и k (см. рис.) имеют общую вершину $b(i)=b(k)$: информация о ней хранится в *единственной* записи v_2 массива вершин, на которую установлены указатели B_i и B_k соответствующих элементов i и k списка гиперинтервалов.

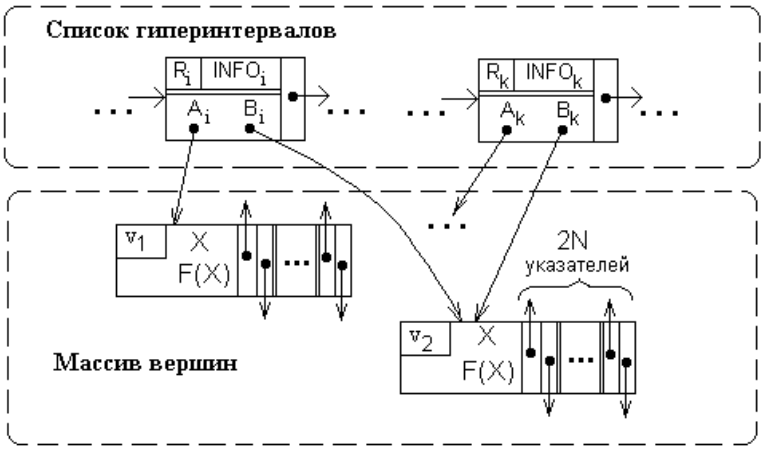


Рис.1. Структура специализированной базы данных

Для обеспечения быстрого доступа к данным необходима эффективная реализация операции вставки новых элементов в список гиперинтервалов с соответствующей настройкой указателей и операции поиска информации о вершинах. Предлагаемая структура данных в связи с процедурой построения АДК отвечает данному требованию.

При параллельной реализации методов [4,5], использующих АДК, на каждой итерации значения характеристик R_i упорядочиваются по невозрастанию и испытания *f(x)* проводятся одновременно в нескольких гиперинтервалах с наибольшими значениями характе-

ристик, что соответствует одновременному подразбиению АДК в пределах данных гиперинтервалов. Предлагаемая структура данных эффективна также при использовании параллельных методов глобальной оптимизации.

Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. М.: Наука, 1978.
2. Sergeyev Ya.D. An efficient strategy for adaptive partition of N-dimensional intervals in the framework of diagonal algorithms //J. Optimizat. Theory Appl. 2000. Vol. 107. N 1. P.145–168.
3. Pintér J. Global Optimization in Action. Dordrecht: Kluwer Academic Publishers, 1996.
4. Grishagin V.A., Sergeyev Ya.D., Strongin R.G. Parallel characteristical algorithms for solving problems of global optimization //J. Global Optimizat. 1997. Vol. 10. P. 185–206.
5. Strongin R.G. and Sergeyev Ya.D. Global Optimization with Non-Convex Constraints: Sequential and Parallel Algorithms. Dordrecht: Kluwer Academic Publishers, 2000.

ПОСТАНОВКА ЗАДАЧИ ОПТИМАЛЬНОГО НАЗНАЧЕНИЯ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ НА ГЕТЕРОГЕННЫЙ ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР

А.В.Киселев, Е.Л. Зверев

Институт криптографии, связи и информатики Академии ФСБ России, Москва

В работе рассматриваются вопросы оптимального назначения параллельных задач на вычислительный кластер, образованный путем объединения нескольких мультипроцессорных вычислительных систем.

Эффективность использования вычислительного кластера во многом зависит от оптимальности распределения его ресурсов между потоком параллельных задач. В случае мультизадачного режима использования кластера, назначение очередной параллельной задачи должно осуществляться с учетом текущего состояния его вычислительных

ресурсов. Степень использования ресурсов кластера (процессоров, коммуникационных каналов) характеризует показатель – загруженность (процессоров, каналов), значения которого для отдельных компонентов кластера могут быть получены от средств мониторинга системы.

В общем случае, при назначении на выполнение очередной задачи, кластер следует рассматривать как неоднородную мультипроцессорную систему, процессоры и каналы которой обладают различными значениями производительности и пропускной способности в силу конструктивных различий или в виду различной загруженности.

Предлагается следующая постановка задачи назначения параллельной программы на гетерогенный вычислительный кластер.

С каждым вычислительным узлом (ВУ) b_i связан описатель – $<l, p, z_p>$, где l – тип процессора, p – производительность процессора (число команд, выполняемых в единицу времени), z_p – текущая загруженность процессора.

Коммуникационная подсистема рассматривается на физическом и логическом уровнях:

- на физическом уровне – как граф $G_f = (B,F)$, где $B=\{b_i\}$ – множество ВУ, а F – множество соединяющих вершины физических каналов связи. С каждым каналом связан описатель $<s, z_s>$, где s – пропускная способность канала, а z_s – текущая загруженность канала. Топология коммуникационной подсистемы задается матрицей смежности $S=\| s_{ij} \|$ графа G_f;
- на логическом уровне – система рассматривается как полный граф G_b, вершинами которого являются ВУ, а ребрами – логические каналы связи. Под логическим каналом понимается путь между вершинами b_i и b_j графа G_f, обладающий максимальной пропускной способностью. С каждым логическим каналом связан описатель $<c,z_c>$, где c – пропускная способность логического канала, определяемая как минимальная из пропускных способностей физических каналов, образующих данный логический канал, а z_c – текущая загруженность логического канала.

Параллельная задача представляется в виде графа $G_a = (A,D)$, где $A=\{A_i\}$- совокупность взаимодействующих процессов, с каждым из которых связан вектор $\boldsymbol{a}=<a_l>$. a_l определяет вычислительную сложность Ai в числе команд процессора l-типа. $D= \{D_i\}$ – множество дуг –

описывает взаимодействия процесса: каждой дуге соответствует число d_j, характеризующее передаваемый объем данных.

Назначение процесса на вычислительный узел приводит к увеличению загруженности процессора и коммуникационной подсистемы. Отличную от 0 загруженность процессора (канала) можно интерпретировать как уменьшение его производительности (пропускной способности). В этом случае будем говорить о приведенной производительности p' (пропускной способности c'): $p' = (1-z_p)\times p;\ c' = (1-z_c)\times c$.

Оценить изменение производительности процессора и пропускной способности канала в случае назначения на ВУ $<l, p, z_p>$ процесса А $<a, d>$ можно по формулам:

$$c \geq c' \geq \frac{ac^2}{dp + ac}; \qquad p \geq p' \geq \frac{p^2 \sum_{(i)} \frac{d_i}{c_i}}{a + p\sum_{(i)} \frac{d_i}{c_i}};$$

Пересчет приведенной пропускной способности осуществляется для всех логических каналов, включающих в свой состав физический канал, используемый процессом A_i для передачи данных.

Обозначим $\varphi : A \to B$, (N = |A|, M = |B|) отображение информационного графа параллельной задачи G_a на структуру вычислительной системы, заданной графом G_b и представим его матрицей $X_\varphi = \{ X_{ij}: i \in A, j \in B \}$, где $X_{ij} = 1$, если $\varphi(i) = j$, и $X_{ij} = 0$, если $\varphi(i) \neq j$. Оптимальное отображение доставляет минимум функционалу

$$F(X\varphi) = F_1(X\varphi) + F_2(X\varphi),$$

где $F_1(X\varphi) = \sum_{i=1}^{N}\sum_{k=1}^{M}(X_{ik}a_i - \frac{\sum_{i=1}^{N} a_i}{\sum_{j=1}^{M} p'_j} p'_k)^2;$

$$F_2(X\varphi) = \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{p=1}^{N}\sum_{k=1}^{N} d_{ij} c'_{pk} X_{pi} X_{kj} .$$

$F_1(X\varphi)$ оценивает равномерность загрузки процессоров, а $F_2(X\varphi)$ оценивает загрузку коммуникационной подсистемы.

Приведенная выше постановка задачи назначения позволяет исследовать вопросы оптимального распределения параллельных процессов в гетерогенных мультипроцессорных системах в условии раз-

деления вычислительных ресурсов (процессорное время ВУ, коммуникационные каналы) между различными задачами.

Поскольку задача оптимального назначения является NP-полной, для ее решения могут быть использованы эвристические алгоритмы, обеспечивающие субоптимальные решения с заданной точностью.

РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ*

В.А. Крюков

Институт прикладной математики им. М.В. Келдыша РАН, г.Москва

1. Введение

Последние годы во всем мире происходит бурное внедрение вычислительных кластеров. Это вызвано тем, что кластеры стали общедоступными и дешевыми аппаратными платформами для высокопроизводительных вычислений. Поскольку единого определения вычислительного кластера не существует, для упрощения дальнейшего изложения введем некоторую классификацию, которая будет отражать свойства программно-аппаратной платформы, существенные с точки зрения разработки прикладных параллельных программ.

Вычислительный кластер – это *мулътикомпьютер*, состоящий из множества отдельных компьютеров (узлов), связанных между собой единой коммуникационной системой. Каждый узел имеет свою локальную оперативную память. При этом общей физической оперативной памяти для узлов не существует. Каждый узел может быть *мультипроцессором* (мультипроцессорный компьютер с общей памятью). Коммуникационная система обычно позволяет узлам взаимодействовать между собой только посредством передачи сообщений, но некоторые системы могут обеспечивать и односторонние коммуникации – позволять любому узлу выполнять массовый обмен информацией между своей памятью и локальной памятью любого другого узла.

* Работа поддержана Российским фондом фундаментальных исследований, грант № 99-01-00209

Если на вычислительном кластере аппаратно или программно-аппаратно реализована DSM (Distributed Shared Memory – распределенная общая память), позволяющая выполняющимся на разных узлах программам (любым, даже ассемблерным) взаимодействовать через общие переменные, то такой кластер будем называть *DSM-кластер*. Если такая DSM отсутствует, и прикладные программы взаимодействуют посредством передачи сообщений, то вычислительный кластер будем называть *DM-кластер* (DM – Distributed Memory).

Если все входящие в состав вычислительного кластера узлы имеют одну и ту же архитектуру и производительность, то мы имеем дело с однородным вычислительным кластером. Иначе – с *неоднородным*.

Неоднородность же вносит следующие серьезные проблемы.

Различие в производительности процессоров требует соответствующего учета при распределении работы между процессами, выполняющимися на разных процессорах.

Различие в архитектуре процессоров требует подготовки разных выполняемых файлов для разных узлов, а в случае различий в представлении данных может потребоваться и преобразование информации при передаче сообщений между узлами (не говоря уже о трудностях использования двоичных файлов). Если существуют различия в представлении данных, то построение неоднородного DSM-кластера представляется просто невозможной.

Тем не менее, любой кластер можно рассматривать как единую аппаратно-программную систему, имеющую единую коммуникационную систему, единый центр управления и планирования загрузки.

С 1992 года, когда мультикомпьютеры стали самыми производительными вычислительными системами, резко возрос интерес к проблеме разработки для них параллельных прикладных программ. К этому моменту уже было ясно, что трудоемкость разработки прикладных программ для многопроцессорных систем с распределенной памятью является главным препятствием для их широкого внедрения.

За прошедший с тех пор период предложено много различных подходов к разработке параллельных программ, созданы десятки различных языков параллельного программирования и множество различных инструментальных средств. Среди них можно отметить следующие интересные отечественные разработки – Норма, Фортран-GNS, Фортран-DVM, mpC, T-система.

Целью данной работы является сравнительный анализ пяти различных подходов к разработке параллельных программ для вычислительных кластеров и сетей – MPI, HPF, OpenMP, OpenMP+MPI и DVM.

Предпочтительность использования того или иного подхода определяют следующие факторы:

- Легкость программирования;
- Эффективность разработанных программ;
- Переносимость и повторное использование программ;
- Качество средств отладки программ (функциональной отладки и отладки рективности).

Именно с этих позиций анализируются указанные подходы в данной работе.

2. Модели и языки параллельного программирования

Как было сказано выше, имеются две основные модели параллельного выполнения программы на кластере – модель передачи сообщений и модель общей. памяти.

В первой модели параллельная программа представляет собой систему процессов, взаимодействующих посредством передачи сообщений. Эта модель может быть использована на любых кластерах.

Во второй модели параллельная программа представляет собой систему нитей, взаимодействующих посредством общих переменных и примитивов синхронизации. Вторая модель доступна для использования только на DSM-кластерах. Можно выбрать в качестве модели программирования одну из этих моделей выполнения.

Однако обе основные модели выполнения являются довольно низкоуровневыми. Поэтому главным недостатком выбора одной из них в качестве модели программирования является то, что такая модель непривычна и неудобна для программистов, разрабатывающих вычислительные программы. Она заставляет его иметь дело с параллельными процессами и низкоуровневыми примитивами передачи сообщений или синхронизации.

Поэтому вполне естественно, что прикладной программист хотел бы получить инструмент, автоматически преобразующий его последовательную программу в параллельную программу для кластера. К со-

жалению, такое автоматическое распараллеливание невозможно в силу ряда причин.

Теперь переходим к рассмотрению подходов, выбранных для сравнительного анализа.

2.1. Модель передачи сообщений. MPI

В модели передачи сообщений параллельная программа представляет собой множество процессов, каждый из которых имеет собственное локальное адресное пространство. Взаимодействие процессов – обмен данными и синхронизация – осуществляется посредством передачи сообщений. Обобщение и стандартизация различных библиотек передачи сообщений привели в 1993 году к разработке стандарта MPI (Message Passing Interface). Его широкое внедрение в последующие годы обеспечило коренной перелом в решении проблемы переносимости параллельных

программ, разрабатываемых в рамках разных подходов, использующих модель передачи сообщений в качестве модели выполнения.

2.2. Модель параллелизма по данным. HPF

В модели параллелизма по данным отсутствует понятие процесса и, как следствие, явная передача сообщений или явная синхронизация. В этой модели данные последовательной программы распределяются программистом по процессорам параллельной машины. Последовательная программа преобразуется компилятором в параллельную программу, выполняющуюся либо в модели передачи сообщений, либо в модели с общей памятью. При этом вычисления распределяются по правилу собственных вычислений: каждый процессор выполняет только вычисления собственных данных, т.е. данных, распределенных на этот процессор.

Обобщение и стандартизация моделей параллелизма по данным привели к созданию в 1993 году стандарта HPF (High Performance Fortran) – расширения языка Фортран 90.

2.3. Модель параллелизма по управлению. OpenMP

Эта модель возникла уже давно как естественная альтернатива явному использованию модели общей памяти при разработке программ для мультипроцессоров. Вместо программирования в терминах нитей предлагалось расширить языки специальными управляющими конструкциями – параллельными циклами и параллельными секциями. Создание и уничтожение нитей, распределение между ними витков парал-

лельных циклов или параллельных секций (например, вызовов процедур) – все это брал на себя компилятор.

Крупнейшие производители компьютеров и программного обеспечения объединили свои усилия, и в октябре 1997 года в результате обобщения и стандартизации моделей параллелизма появилось описание языка OpenMP Fortran -расширение языка Фортран 77. Позже вышли аналогичные расширения языков Си и Фортран 90/95.

2.4. Гибридная модель параллелизма по управлению с передачей сообщений. OpenMP+MPI

Успешное внедрение OpenMP на мультипроцессорах и DSM-мультикомпьютерах резко активизировало исследования, направленные на поиски путей распространения OpenMP на DM-мультикомпьютеры и сети ЭВМ.

Однако ожидать в ближайшее время практического результата от этих исследований очень трудно.

Зато нет никаких препятствий для использования гибридного подхода, когда программа представляет собой систему взаимодействующих MPI-процессов, а каждый процесс программируется на OpenMP.

Преимущества такого смешанного подхода с точки зрения упрощения программирования очевидны в том случае, когда в программе есть два уровня параллелизма – параллелизм между подзадачами и параллелизм внутри подзадачи.

Широкое распространение кластеров, имеющих в качестве узлов мультипроцессоры, также подталкивает к использованию гибридного подхода, поскольку использование OpenMP на мультипроцессоре может для некоторых задач (например, вычислений на неструктурных сетках) дать заметный выигрыш в эффективности.

Основной недостаток этого подхода также очевиден – программисту надо знать и уметь использовать две разные модели параллелизма и разные инструментальные средства.

2.5. Модель параллелизма по данным и управлению. DVM

Эта модель, положенная в основу языков параллельного программирования Фортран-DVM и Си-DVM, объединяет достоинства модели параллелизма по данным и модели параллелизма по управлению. Базирующаяся на этих языках система разработки параллельных программ (DVM) создана в Институте прикладной математики им. М.В. Келдыша РАН.

В отличие от модели параллелизма по данным, в системе DVM программист распределяет по процессорам виртуальной параллельной машины не только данные, но и соответствующие вычисления. При этом на него возлагается ответственность за соблюдение правила собственных вычислений. Кроме того, программист определяет общие данные, т.е. данные, вычисляемые на одних процессорах и используемые на других процессорах. И, наконец, он отмечает точки в последовательной программе, где происходит обновление значений общих данных.

3. Эффективность выполнения параллельных программ

Эффективность выполнения программ всегда являлась очень важным фактором, определявшим в значительной степени успех и распространение языков программирования, предназначенных для создания вычислительных программ.

В докладе приводятся данные об эффективности выполнения шести тестов из пакета NPB 2.3 (BT, CG, FT, LU, MG, SP), реализованных с использованием подходов, выбранных для сравнительного анализа.

Эти тесты хорошо отражают характер вычислительных задач различных классов, за исключением задач с нерегулярными сетками.

4. Переносимость и повторное использование параллельных программ

В настоящее время, когда программист имеет возможность запускать свою программу на разных, порою географически удаленных параллельных системах, важность переносимости программ (их способности выполняться на различных вычислительных системах с приемлемой эффективностью) трудно переоценить.

Создать для новых параллельных систем прикладное программное обеспечение, необходимое для решения важнейших научно-технических задач, вряд ли возможно без повторного использования уже созданных программ, без накопления и использования богатых библиотек параллельных программ.

Поэтому переносимость программ и их способность к повторному использованию должны рассматриваться как самые первостепенные показатели качества параллельных программ.

5. Средства отладки

Отладка параллельной программы является процессом более трудоемким, чем отладка последовательной программы. Причиной этого является не только сложность параллельной программы, но и ее недетерминированное поведение, серьезно затрудняющее и функциональную отладку (достижение правильности результатов), и отладку эффективности программы. Развитые средства отладки могут существенно упростить разработку параллельных программ прикладными программистами.

6. Выводы

На основании проведенного анализа пяти различных подходов к разработке параллельных программ для вычислительных кластеров (MPI, HPF, OpenMP, OpenMP+MPI и DVM) можно сделать следующие выводы.

1. С точки зрения простоты разработки параллельных программ и их повторного использования явное преимущество имеют подходы HPF, OpenMP и DVM. Конечно, очень трудно количественно оценить это преимущество, но в качестве грубой оценки сложности программирования вполне годятся данные о соотношении количества дополнительных операторов, которые пришлось при распараллеливании тестов NPB 2.3 добавить в их последовательные версии -40% для MPI и 4% для DVM.
2. По эффективности выполнения программ HPF заметно отстает от остальных подходов.
3. OpenMP ограничивает переносимость программ мультипроцессорами и DSM-кластерами.
4. Гибридный подход OpenMP+MPI, как и MPI, не может обеспечить эффективного выполнения программ на неоднородных кластерах.

Таким образом, ни один из четырех подходов (MPI, HPF, OpenMP и OpenMP+MPI), базирующихся на имеющихся стандартах, не может рассматриваться в настоящее время как вполне подходящий для разработки параллельных программ для вычислительных кластеров и сетей ЭВМ.

Наверное, пройдет еще немало лет до появления языка программирования, который будет принят сообществом программистов в ка-

честве языка разработки параллельных программ для высокопроизводительных вычислений на кластерах и сетях ЭВМ.

А как же разрабатывать программы для вычислительных кластеров все эти годы до появления нового языка?

Можно использовать MPI, понимая при этом, что будут затрачены большие усилия для написания, отладки и сопровождения программ, которые когда-то все равно придется переписывать на другом языке.

Можно попробовать использовать DVM, поскольку освоение этого подхода может существенно сократить время написания, отладки и сопровождения программ. А в случае появлении нового стандарта языка, DVM-программы смогут быть преобразованы в программы на новом языке автоматически, или с минимальным участием программиста.

Литература

1. Message-Passing Interface Forum, Document for a Standard Message-Passing Interface, 1993. Version 1.0. http://www.unix.mcs.anl.gov/mpi/.
2. High Performance Fortran Forum. High Performance Fortran Language Specification. Version 1.0, May 1993.
3. OpenMP Consortium: OpenMP Fortran Application Program Interface, Version 1.0, October 1997. http://www.openmp.org/.
4. DVM-система. http://www.keldysh.ru/dvm/.
5. Frumkin M., Jin H., and Yan J. Implementation of NAS Parallel Benchmarks in High Performance Fortran. NAS Technical Report NAS-98–009, NASA Ames Research Center, Moffett Field, CA, 1998.
6. Frumkin M., Jin H., and Yan J. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. NAS Technical Report NAS-99–011, NASA Ames Research Center, Moffett Field, CA, 1999.
7. Capello F., Etiemble D. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. In *Proceedings of Super computing '2000*, 2000.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ ДЛЯ ОБРАТНЫХ ЗАДАЧ МАКРОКИНЕТИКИ ХИМИЧЕСКИХ ПРОЦЕССОВ В ПРОТОЧНЫХ АППАРАТАХ

А.М.Кутьин, И.Г. Славутин, Л.В. Смирнов

Нижегородский государственный университет им. Н.И. Лобачевского

Практическая необходимость, определяемая началом проектных и опытно-конструкторских работ по созданию объекта по уничтожению запасов химического отравляющего вещества люизита, обусловила разработку средств математического моделирования процессов и аппаратов.

Математическим ядром для построения такой системы явилась макрокинетическая модель химических и фазовых превращений в реагирующих потоках. При опоре на законы и методы неравновесной химической термодинамики основу этой модели составляют уравнения балансов массы и энергии в потоках, записанные с учетом упрощенной гидродинамической структуры, характеризуемой критериальными соотношениями подобия. При этом протекающие в потоках и на межфазных границах физико-химические процессы рассматриваются как источники вещества и энергии.

По своей математической структуре данная макрокинетическая модель в ее стационарном варианте сводится к системе обыкновенных дифференциальных уравнений достаточно большой размерности с нелинейными правыми частями. Частный вариант этой модели может быть сформулирован как задача Коши, численно решаемая методами Рунге-Кутта (так называемая прямая задача).

Как и любая ориентированная на практическое использование сложная модель, и данная в том числе, на деле нуждается в средствах «настройки» по пробным или тестовым экспериментальным данным. Указанные средства настройки призваны восполнить недостаточность теоретических оценок ряда параметров, а иногда и невозможность таких оценок. Наибольшее число эмпирических параметров используемая модель объективно содержит в кинетической своей части. По сути эмпирическим является и выбор уравнений, определяющих кинетическую схему.

Нахождение всех этих настроечных параметров есть суть решения обратных задач, которые на практике представляют больший интерес,

нежели прямые задачи, так как их решение позволяет глубже изучить процесс.

Решение обратной задачи является намного более трудоемким, т.к. сводится к задаче глобальной оптимизации функции многих переменных (Для вычисления значения функции в точке многомерного пространства необходимо решить прямую задачу).

В существующем варианте программной реализации модели уже предпринята относительно успешная попытка применения параллельных вычислений при решении обратных задач. На базе механизма DCOM реализована возможность использования нескольких компьютеров для решения обратной задачи в случае работы с несколькими наборами экспериментальных значений. Каждый компьютер в этом случае вычисляет значение невязки для своего набора экспериментальных данных, так называемый параллелизм на уровне отдельно выполняемых заданий или распределенные вычисления. Однако, применение более низкоуровневых способов, например, распараллеленных алгоритмов глобальной оптимизации, может существенно расширить круг обратных задач, решаемых с использованием параллельных вычислений.

СОЗДАНИЕ И ПРИМЕНЕНИЕ КЛАСТЕРОВ BEOWULF В СУПЕРКОМПЬЮТЕРНОМ ЦЕНТРЕ ИОХ РАН*

М.Кузьминский, А.Мускатин

Институт органической химии им.Н.Д.Зелинского РАН, г.Москва

Задачи вычислительной химии (в частности, квантовой химии и молекулярной динамики) являются одними из основных мировых потребителей ресурсов суперЭВМ. В случае расчетов больших молекул (или при применении наиболее точных методов неэмпирической квантовой химии) необходимо распараллеливание программ. Соответственно при использовании кластеров необходимо иметь высокий уровень распараллеливания для достижения лучшего соотношения стоимость / производительность по сравнению с суперЭВМ SMP/ccNUMA- или SMP-архитектур.

* Работа финансировалась в рамках проекта РФФИ 01-07-90072.

Поскольку каналы связи между узлами кластеров имеют более низкую пропускную способность и более высокие задержки, чем современные суперЭВМ, в то время как производительности применяемых процессоров обычно близки, распараллеливание в кластерах становится узким местом в ряде задач вычислительной химии [1]. В настоящей работе приведены некоторые данные о создании кластеров с использованием технологий Fast Ethernet и Gigabit Ethernet в суперкомпьютерном центре (СКЦ) ИОХ РАН.

При создании кластеров в СКЦ ИОХ РАН была поставлена естественная задача – достижение приемлемого уровня производительности при минимизации отношения стоимость/производительность. Оптимальными по этим показателям узлами кластера являются ПК-серверы на базе x86-совместимых микропроцессоров. В связи с этим в СКЦ ИОХ РАН было создано 2 Linux-кластера.

В первом используются узлы на базе микропроцессоров Intel Pentium III/600 МГц с внешним кэшем емкостью 512 Кбайт (8 однопроцессорных и 2 двухпроцессорных узла, материнские платы ASUS P3B-F/P2B-D с набором микросхем 440BX, с оперативной памятью ECC PC100 емкостью 256 Мбайт на процессор). Во втором применяются узлы на базе AMD Athlon/700 МГц с внешним кэшем емкостью 512 Кбайт (4 однопроцессорных узла с памятью ECC PC133 емкостью 128 Мбайт на процессор, материнские платы Gigabyte 7VX). Применение более высокочастотных микропроцессоров этих фирм имеет свой недостаток – уменьшенную в 2 раза емкость кэша L2 (серверные варианты микропроцессоров Intel Tualatin в настоящее время недоступны).

В кластере на базе Pentium III для соединения узлов использован Fast Ethernet. Это – наиболее дешевое на настоящий момент решение, обеспечивающее минимально приемлемую для распараллеливания пропускную способность. В узлах кластера инсталлирована ОС Linux RedHat 6.2. Применяемые в кластере программы (Gaussian-98, Gamess) обеспечивают распараллеливание как в модели общего поля памяти (в SMP-узлах), так и в модели обмена сообщениями. В частности, было найдено, что неэмпирический метод ССП удовлетворительно распараллеливается в кластере Fast Ethernet, а уровень распараллеливания метода MP2 существенно хуже [1].

Поскольку обычно в кластере для распараллеливания применяется MPI, работающий поверх TCP/IP, авторами было проведено исследо-

вание производительности стека протоколов TCP/IP для основных типов сетевых плат, применяемых в кластерах (3Com 3c905B, Intel EtherExpress Pro 100, Kingston KNE100TX, CNet CN100TX) на тестах netperf. В этом кластере используется 24-портовый коммутатор Fast Ethernet D-Link DES 3224, который поддерживает режим «коммутации на лету» (cut-through) и имеет пропускную способность 5 Гбит/с, что больше суммарной пропускной способности подсоединяемых каналов Fast Ethernet. Было исследовано влияние различных факторов, в т.ч. версии реализации стека протоколов TCP/IP в ОС Linux. Как и следовало ожидать, большие задержки в протоколах TCP/IP привели к тому, что включение режима cut-through не приводит к существенному повышению производительности, и его следует применять при работе MPI без TCP/IP.

Производительность различных плат на тестах netperf оказалась довольно близкой, однако из соображений хорошей «интероперабельности» для режима channel bonding (по 2 канала Fast Ethernet на узел) были выбраны EtherExpress Pro 100. При этом на тестах TCP_STREAM достигается пропускная способность порядка 150 Мбит/с, а на тестах UDP_STREAM – 190 Мбит/с [2].

Поскольку пропускная способность каналов связи между узлами является узким местом в распараллеливании ряда задач квантовой химии, в кластере на базе AMD Athlon использована технология Gigabit Ethernet на медной проводке. Эта технология считается перспективной в связи с ожиданием резкого падения цен на соответствующую продукцию. В кластере используется 8-портовый коммутатор Gigabit Ethernet – Intel NetStructure ES470T, а в узлах – сетевые карты Intel Pro 1000T.

Однако результаты измерений на тестах netperf показали не очень высокий уровень пропускной способности: 345 Мбит/с для UDP_STREAM и 270 Мбит/с для TCP_STREAM. Переход от RedHat 6.2 (ядро 2.2) к RedHat 7.1 (ядро 2.4) и соответственно к новой версии драйвера не улучшает результаты существенным образом. Так, пропускная способность на тестах TCP_STREAM увеличивается лишь до 305 Мбит/с. Аналогичная ситуация имеет место на минимально коротких пакетах (для TCP_RR – возрастание примерно с 4570 до 4700). Для исследования влияния задержек, вносимых коммутатором, была исследована также производительность на коротких пакетах при соеди-

нении узлов кластера напрямую кабелем cross-over. При этом (для ядра 2.4) результаты тестов UDP_RR (на пакетах длиной 16 байт) возрастают с 5230 до 4590, но эти результаты уступают типовым при работе с картами Fast Ethernet (например, 7210 для EtherExpress Pro 100.

Это говорит о том, что применение Gigabit Ethernet более эффективно при распараллеливании, характеризующемся обменом сообщениями больших размеров. Нагрузка на процессор в netperf-тестах STREAM в среднем не превышала 30%, т.е. производительность процессора не является лимитирующим фактором. С учетом того, что эти сетевые платы стоят на порядок дороже, чем Fast Ethernet (не считая стоимость коммутатора), применение подобных карт в таких ПК-серверах с 32-разрядной шиной PCI (в частности, для многих задач вычислительной химии) не является сегодня эффективным по соотношению стоимость/производительность.

Главным направлением, обеспечивающим эффективное распараллеливание (в частности, в кластерах) задач квантовой химии в применении к сверхбольшим молекулярным системам является, с точки зрения авторов, «пофрагментный» подход с использованием локализованных орбиталей. Так, например, традиционные полуэмпирические схемы метода ССП распараллеливаются плохо. Однако авторами с использованием оригинальной методики применения локализованных орбиталей в полуэмпирических схемах ССП и средств MPI недавно была распараллелена программа, и предварительные оценки показывают на хорошую эффективность распараллеливания в кластере FastEthernet.

Литература

1. Кузьминский М., Мендкович А. Высокопроизводительные вычисления в химии: современные тенденции. Сб.тезисов докладов II Международного симпозиума «Компьютерное обеспечение химических исследований», М., 22–23 мая 2001 г. С.24.
2. Кузьминский М., Мускатин А. Fast Ethernet в кластерах Beowulf // Открытые системы, 2001, N7–8. С.17.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА ДЛЯ ОПЕРАЦИОННЫХ СИСТЕМ WINDOWS 2000 И LINUX REDHAT 7.0

Д.Ю.Лабутин

Нижегородский государственный университет им. Н.И. Лобачевского

С каждым годом увеличивается мощность компьютеров, но очевидно, что классическая фон-неймановская организация вычислений уже не обеспечивает требуемой производительности в некоторых задачах. Это обусловлено как физическими ограничениями (скорость света), так и экономическими, поэтому стимулируется развитие параллельных вычислений. Тот факт, что параллельные вычисления до сих пор не получили должного распространения, объясняется техническими трудностями (необходимость высокоскоростных каналов связи между вычислительными комплексами или процессорами), алгоритмическими трудностями (необходимость разработки алгоритмов, учитывающих параллелизм), и, в особенности отсутствием неких стандартных подходов и программных средств для организации параллельных вычислений. Одним из основных вопросов, возникающих при организации параллельных вычислений – это проблема выбора операционной системы, в которой будут производиться расчеты. Именно этой проблеме и посвящена данная работа.

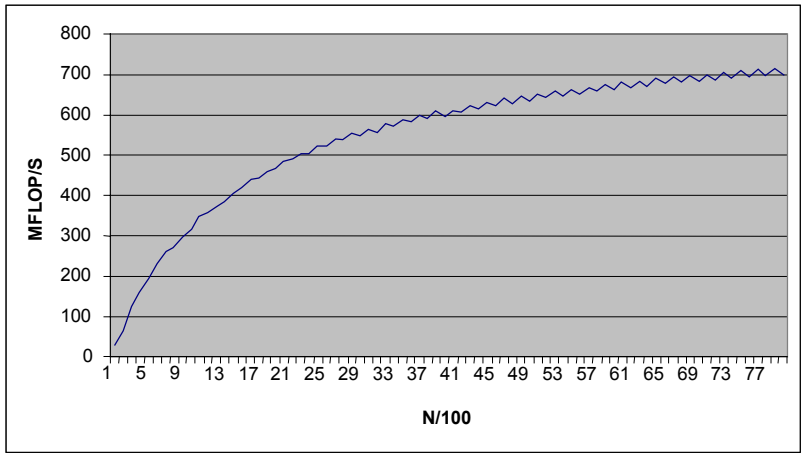
В рамках данной проблемы нужно было провести сравнительный анализ двух операционных систем – Windows 2000 и Linux RedHat 7.0 В качестве основного критерия был выбран показатель производительности кластера на тесте Linpack.

Результаты экспериментов

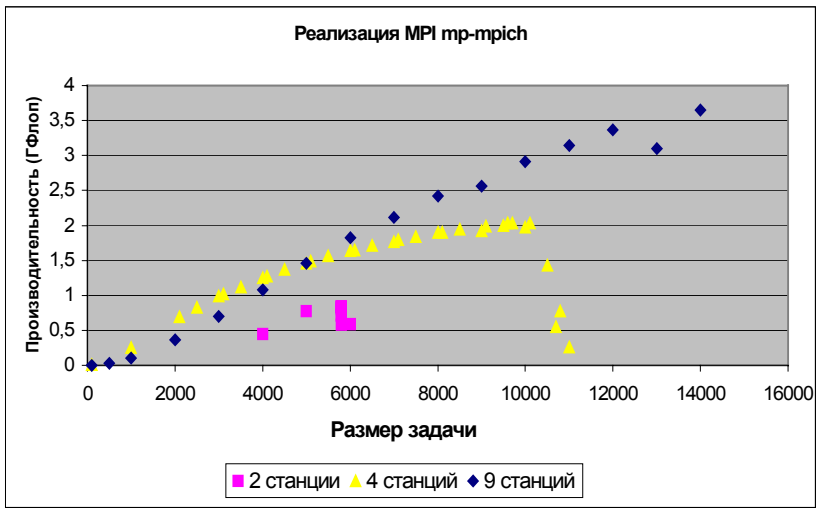
Анализируя результаты экспериментов, хочется сделать вывод, что производительность кластера под управлением операционной системы Windows 2000 почти в 2,5 раза превосходит производительность кластера под управлением операционной системы Linux RedHat 7.0. Но мы пока не спешим делать подобные выводы, т.к. результаты приведенных экспериментов, честно говоря, не совсем корректно сравнивать, т.к. сами эксперименты не были полностью идентичными. Основными отличиями были: различные компиляторы, используемые для подготовки теста, и различные библиотеки. Например, при тестировании кластера в операционной системе Windows 2000 были использо-

ваны PlaPack (в Linux использовалась библиотека ScaLapack) и MKL (в Linux использовалась библиотека BLACS).

Результаты теста Linpack RedHat 7.0 (4 узла)



Результаты теста Linpack Win2000(4 узла)



В продолжение начатых исследований в ближайшее время планируется провести полностью идентичные эксперименты для обеих операционных систем, а в качестве операционной системы семейства Unix использовать Linux RedHat 7.2

ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ ТЕХНОЛОГИЙ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В КЛАСТЕРНЫХ СИСТЕМАХ*

И.В. Лопатин, А.Н. Свистунов, А.В. Сысоев

Нижегородский государственный университет им. Н.Лобачевского

В данном докладе рассматриваются результаты вычислительных экспериментов, выполненных для определения способов эффективной реализации параллельных вычислений на многопроцессорных вычислительных системах с общей памятью.

Одним из подходов к разработке параллельных программ для таких систем является использование возможностей стандарта OpenMP. Такой подход обеспечивает учет возможностей архитектуры многопроцессорных ЭВМ с общей памятью. Программный интерфейс приложений (API) OpenMP [1] является средством компилятора, позволяющим разрабатывать переносимые приложения на языках C/C++ и Fortran.

При проведении экспериментов в качестве примера использовалась задача матричного умножения, для решения которой были подготовлены три варианта реализации одного и того же алгоритма: последовательный, параллельный, полученный добавлением директив OpenMP, и комбинированный, использующий широко известный механизм передачи сообщений MPI. В MPI версии для распределения элементов матриц между процессорами была использован ленточная схема.

В зависимости от размеров перемножаемых матриц и условий проведения экспериментов коэффициент ускорения параллельных версий программы составил от 1.4 до 1.96 на двухпроцессорных серверах

* Проведение исследований, по результатам которых была подготовлена данная работа, было поддержано грантом компании Intel.

и 1.85 – 2.1 для четырехпроцессорных систем. Результаты для двухпроцессорного сервера приведены на рис. 1.

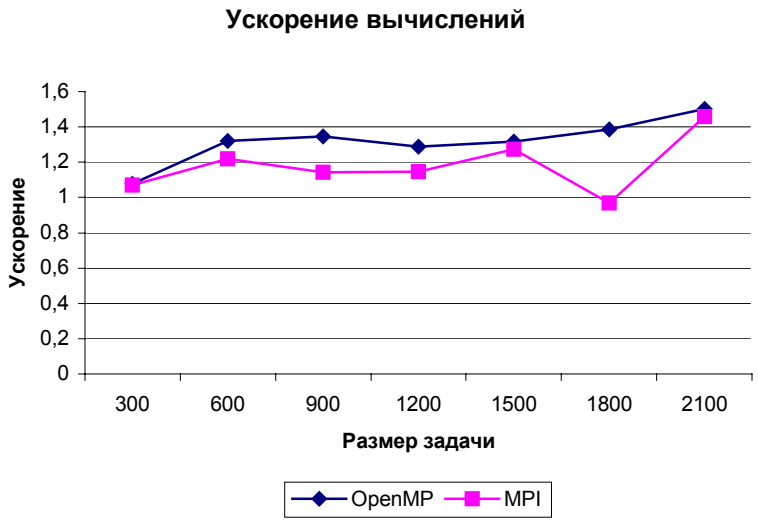


Рис. 1. Ускорение матричного умножения при использовании OpenMP и MPI

Следует отметить, что при определении времени последовательного выполнения программы используются оба (четыре) процессора вычислительного сервера (дополнительный процессор применяется, в частности, для исполнения процессов операционной системы). Как результат, время последовательного выполнения является меньшим, чем на аналогичном однопроцессорном компьютере (что, соответственно, увеличивает коэффициент ускорения, обеспечиваемого параллельными вариантами программ).

Аналогичная ситуация наблюдается при запуске программы под операционной системой Linux (дистрибутив RedHat7.1). Приблизительно совпадает как абсолютное время выполнения последовательной версии, так и получаемый коэффициент ускорения в OpenMP-варианте программы. График ускорения для двухпроцессорного компьютера приведен на рис. 2.

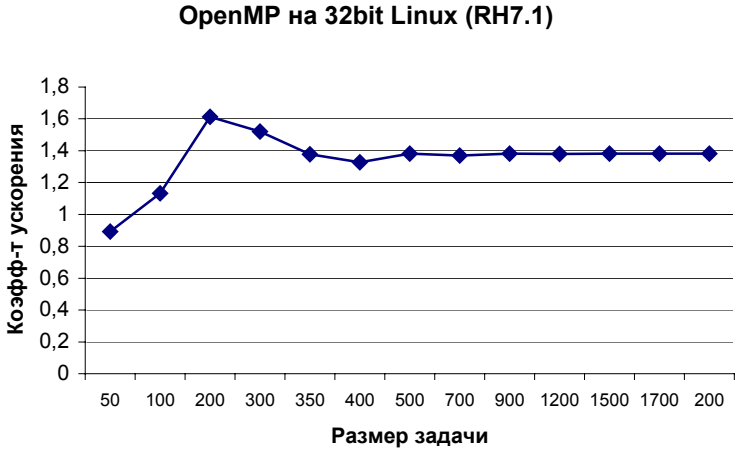


Рис. 2. Ускорение матричного умножения при использовании OpenMP (Linux)

Результаты экспериментов также показали, что, по всей видимости, наличие модифицированного процессора Pentium III (Xeon) влияет на производительность OpenMP при перемножении матриц. Коэффициент распараллеливания на системах, построенных на базе этих процессоров, несколько ниже, чем на аналогичных системах с немодифицированными процессорами. Возможной причиной расхождения результатов является наличие новых технологий, реализованных в Pentium III Xeon, таких как микроархитектура NetBurst™, включающая в себя механизм улучшенного динамического исполнения команд (Advanced Dynamic Execution). [2]

Одним из важнейших факторов, влияющих на производительность данного алгоритма, является частота обращений к памяти и кэшу. Если относительно недорогую в вычислительном плане операцию умножения элементов заменить более трудоемкой операцией перемножения синусов и/или косинусов этих же элементов, наличие кэш-памяти будет оказывать гораздо меньшее влияние на скорость выполнения. Как видно из рис.3, ускорение параллельной версии программы в этом случае приближается к теоретическому максимуму и в среднем составляет 1.93–1.96 на двухпроцессорной системе.

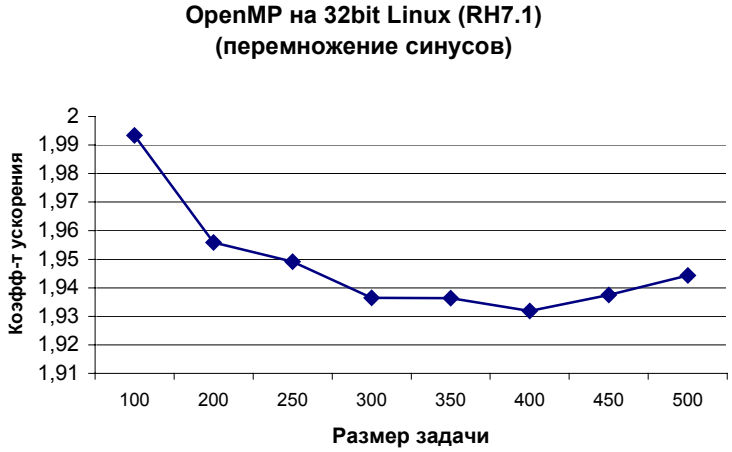


Рис. 3. Ускорение измененной версии программы

Введение промежуточной переменной, в которой накапливаются значения скалярного произведения двух участков матриц, также снижает число обращений к памяти, что также ведет к увеличению производительности.

Многопотоковое программирование с использованием OpenMP приобретает новое значение с началом применения технологии Hyper-Threading (Jackson technology) [3] в новейших моделях процессоров. OpenMP, в отличие от явного, трудоемкого и чреватого ошибками задания потоков, позволяет эффективно использовать возможности, предоставляемые Hyper-Threading.

В результате анализа полученных данных можно заключить, что при организации параллельных вычислений на многопроцессорных системах с общей памятью более эффективным является использование технологии разработки с использованием возможностей OpenMP. Подобный вывод, в свою очередь, позволяет сформулировать предложение о целесообразности применения *гибридной технологии разработки параллельных программ* на кластерах, при которой для организации взаимодействия между узлами вычислительной системы используются средства MPI, а на узлах с общей памятью применяется OpenMP.

Для оценки подобного подхода были выполнены вычислительные эксперименты с использованием двух 2-процессорных серверов кластера. Для организации параллельных вычислений были разработаны два варианта программ для ленточного алгоритма перемножения матриц:

- программа с использованием только интерфейса передачи сообщений MPI; при выполнении экспериментов эта программа запускалась с генерацией 4 параллельных процессов (по два процесса на каждый двухпроцессорный сервер);
- программа с использованием интерфейса MPI и средств распараллеливания OpenMP; при выполнении этого варианта программы порождались 2 параллельных процесса (по одному процессу на каждый двухпроцессорный сервер), далее на каждом сервере для процессов средствами OpenMP создавались два параллельных потока (по одному на каждый процессор).

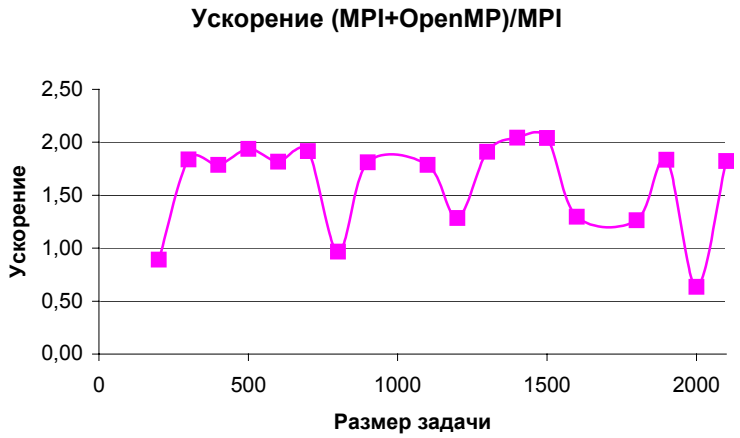


Рис. 4. Ускорение матричного умножения при использовании смешанного MPI+OpenMP варианта параллельной программы

Как следует из рис. 3, комбинированный вариант параллельной программы имеет заметное преимущество по эффективности в сравнении с программой, разработанной только с использованием интерфейса MPI.

Программное обеспечение, использовавшееся при экспериментах: Microsoft Visual C++ 6.0 (IDE), Intel Highly Optimizing Compiler 5.01 (trial version); 6.0beta for Windows and Linux, Linux RedHat 7.1.

Литература

1. OpenMP 2.0 C/C++ specification (draft) http://www.openmp.org/specs/mp-documents/draft_cspec20_bars.pdf.
2. The Intel Xeon Processor. Product overview. http://developer.intel.com/design/Xeon/prodbref/index.htm.
3. Hyper-Threading technology http://developer.intel.com/technology/hyperthread/index.htm.

ОБЩИЕ ПОДХОДЫ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ МЕТОДОВ МОНТЕ-КАРЛО

С.А. Майданов

Нижегородская лаборатория Intel (INNL), Нижний Новгород

Введение

Одной из наиболее перспективных областей развития параллельных вычислений является метод Монте-Карло. За период с 80-х годов XX века до сегодняшнего момента количество работ, посвященных методу Монте-Карло, увеличилось в 2 раза. И это во многом благодаря развитию и совершенствованию параллельных вычислений. Любой метод Монте-Карло может быть тривиальным образом распараллелен как для многопроцессорных систем с общей памятью, так и для систем с распределенной памятью, причем в силу особенностей методов Монте-Карло, эффективность распараллеливания может достигать 100%. При этом основной сложностью является создание «хороших» параллельных генераторов случайных чисел. Существуют, однако, и нетривиальные методы распараллеливания Монте-Карло. Целью данной работы является ознакомление с наиболее важными приложениями Монте-Карло и изучением возможности их распараллеливания.

Метод Монте-Карло и интегрирование функций

Мы рассмотрим задачу интегрирования на примере оценки одномерного определенного интеграла, однако все сказанное достаточно прозрачно переносится и на многомерный случай с очень сложными областями интегрирования (конечными или бесконечными).

Задача оценки определенного интеграла состоит в вычислении значения θ, определяемого как

$$\theta = \int_a^b f(x)dx \tag{1}$$

путем введения случайной величины Y, определенной на интервале (a, b) и имеющей плотность распределения $p(y)$, а также некоторой B-измеримой функции $g(y)$ такой, что математическое ожидание $\mathrm{E}(g(Y)) = \theta$. Очень часто случайная величина Y рассматривается как равномерно распределенная на интервале (a, b). В этом случае θ определяется как $\theta = (b-a)\mathrm{E}(f(Y))$. За оценку θ принимается величина

$$\hat{\theta} = (b-a)\frac{\sum f(y_i)}{n}.$$

Нетрудно убедиться, что оценка *несмещенная*, т.е. $\mathrm{E}(\hat{\theta}) = \theta$. Дисперсия при этом равна:

$$\begin{aligned} \mathrm{D}(\hat{\theta}) &= (b-a)^2 \frac{\sum \mathrm{D}(f(Y_i))}{n^2} = \frac{(b-a)^2}{n}\mathrm{D}(f(Y_i)) = \\ &= \frac{(b-a)}{n}\int_a^b \left(f(x) - \frac{1}{b-a}\int_a^b f(t)dt \right)^2 dx \end{aligned} \tag{2}$$

Дисперсия позволяет оценить вероятность нахождения оценки $\hat{\theta}$ с заданной точностью. Обычная практика состоит в оценке доверительного интервала для вычисляемого интеграла по той же выборке, по которой вычисляется сам интеграл. При этом порядок ошибки (ее называют *вероятностной ошибкой*) $O\left(n^{-1/2}\right)$ не зависит от кратности интеграла. Для сравнения, у квадратурных формул порядок ошибки равен $O\left(n^{-2/d}\right)$, где d – кратность интеграла. Таким образом, при количестве измерений $d > 4$ метод Монте-Карло становится предпочтительнее детерминированных квадратурных формул.

Оценивание дисперсии (2) – задача той же сложности, что и оценка самого интеграла (1). Однако для оценивания дисперсии оценки (как и для оценивания математического ожидания) можно привлечь статистический подход. В случае сильной корреляции соседних членов выборки можно применить методику разбиения исходной выборки на блоки. Оценивание параметров в каждом из блоков осуществляет отдельный узел многопроцессорной вычислительной системы. При таком подходе предложена методика, позволяющая улучшить оценку дисперсии исходной выборки.

Уменьшение дисперсии. Расслоенная выборка

Скорость сходимости метода Монте-Карло порядка $O\left(n^{-1/2}\right)$, безусловно, является неудовлетворительной, и очень часто применяются специальные приемы, позволяющие улучшить сходимость. Один из них основан на разбиении области интегрирования на ряд подобластей (на расслаивании выборки) и организации независимых расчетов в каждой из подобластей.

Представим область интегрирования в виде суммы непересекающихся подобластей $A = \bigcup_{i=1}^{m} A_i$, $A_i \cap A_j = \varnothing$ ($i \neq j$). В этом случае исходный интеграл представляется:

$$\theta = \int_A g(x)p(x)dx = \sum_{i=1}^{m} S_i \int_{A_i} g(x)p_i(x)dx; S_i = \int_{A_i} p(x)dx; p_i(x) \equiv \frac{p(x)}{S_i}. \quad (3)$$

Здесь $p_i(x)$ является плотностью распределения в области A_i. Тогда оценка исходного интеграла складывается из оценок интегралов по подобластям A_i:

$$\hat{\theta} = \sum_{i=1}^{m} S_i \hat{\theta}_i \quad \text{где } \hat{\theta}_i = \frac{1}{n_i} \sum_{k=1}^{n_i} g(x_k^i). \quad (4)$$

(n_i – объем выборки в области A_i, $\{x_k^i\}$ – независимые реализации случайной величины с плотностью распределения $p_i(x)$). Дисперсия оценки равна $D\hat{\theta} = \sum_{i=1}^{m} S_i^2 D\hat{\theta}_i$. При заданном разбиении области интегрирования осуществим оптимальный выбор объема выборки в каждой из

подобластей $D\hat{\theta} \xrightarrow[\sum n_i = N]{n_1, n_2, \ldots n_m} \min$. Нетрудно показать, что оптимальный объем выборки в подобласти должен быть пропорционален стандартному отклонению оценки интеграла в данной подобласти:

$$\frac{n_i}{N} = S_i \sigma_i \Big/ \sum_{i=1}^{m} S_i \sigma_i \qquad (5)$$

Используя данный результат, рассмотрим метод распараллеливания вычислений с декомпозицией по области интегрирования. Исходная область интегрирования разбивается на достаточно большое число подобластей, существенно большее количества вычислительных узлов (процессоров). Каждый из узлов будет осуществлять оценку интеграла в одной из подобластей, причем расчет разделяется на два этапа:

1. оценка стандартного отклонения в подобласти и регулировка объема выборки в подобласти;
2. непосредственный расчет.

Каждому из процессоров назначается область, в которой еще не производились расчеты. Поскольку объем выборки различен в каждой из подобластей, какой-либо процессор закончит расчеты в подобласти раньше остальных. Освободившийся процессор передает результаты расчетов на главную машину, которая осуществляет оценку интеграла по всей области интегрирования, после чего освободившемуся процессору назначается очередная подобласть, в которой не осуществлялось интегрирование. В силу того, что количество подобластей интегрирования существенно больше числа вычислительных узлов, простой процессоров минимален и возможен лишь в самом конце расчетов.

Изингова модель и связь с задачей глобальной оптимизации

Изингова модель позволяет увидеть достаточно широкий спектр эффектов, возникающих в магнетиках. Данный пример является основой моделирования во многих разделах статистической физики по многим причинам:

- Позволяет просто моделировать реальные физические системы.
- Соотносится с другими системами, связанными с теорией критических феноменов и фазовых переходов, например, термодинамика и теория квантовых полей.

- Аналитическое решение существует для нескольких простых моделей, которые полезны для проверки применяемых численных схем.
- Классический пример использования метода Монте-Карло.
- Интересные приложения для параллельных вычислений: была проведена большая работа на параллельных компьютерах (и помогла их совершенствованию).

Непарные спины электронов в кристаллической решетке связываются и упорядочиваются. Сумма их магнитных полей дает макроскопический магнетизм. Низкие температуры приводят к упорядоченности и большой намагниченности, высокие же температуры приводят к неупорядоченным термальным флуктуациям, случайной ориентации, спины и поля прерываются, намагниченность пропадает.

Спины имеют только два состояния +1 и –1. Энергия задается следующим соотношением:

$$E = \sum_{<i,j>} J_{ij} S_i S_j \ , \qquad (6)$$

где S_i – спин i-м в узле кристаллической решетки, $\langle i, j \rangle$ – ближайшие соседи по решетке, J_{ij} – сила взаимодействия спинов i и j. Намагниченность системы определяется как $M = \sum_i S_i$. Взаимное расположение спинов определяет состояние (конфигурацию) системы. Рассмотрим принцип оценивания параметров на примере намагниченности $\langle M \rangle = \sum_C p(C) M(C)$, где $M(C)$ – намагниченность в конфигурации C, $p(C)$ – распределение вероятностей для конфигураций как функция температуры T.

Фундаментальным результатом в статистической механике является физический закон о функции распределения $p(C)$, которое называется *распределением Больцмана*.

$$p(C) = \frac{1}{Z} e^{-E(C)/kT}; Z = \sum_C e^{-E(C)/kT}$$

Здесь E – энергия конфигурации, T – температура, k – постоянная Больцмана. Z часто называют *функцией разбиения*. Ясно, что идеальной ситуацией было бы моделирование конфигураций с вероятностями, определяющимися весами распределения Больцмана $p(C)$, задаю-

щими вклад этих конфигураций в итоговую сумму $\overline{M} = \frac{1}{N}\sum_{i=1}^{N} M(C_i)$.

Однако моделируемая вероятность $p(C)$ зависит от функции разбиения, которую очень сложно вычислить.

Введем фиктивную динамику в моделируемую систему, так называемую Марковскую цепь конфигураций C_t. «Время» t – компьютерное время, реальное время не входит в уравнения, поскольку система инвариантна по времени.

Пусть $P(A,t)$ – вероятность быть в конфигурации A в момент времени t. $W(A \to B)$ – вероятность перехода из состояния A в состояние B. Тогда

$$P(A,t+1) = W(A \to A)P(A,t) + \sum_B \left[W(B \to A)P(B,t) - W(A \to B)P(A,t)\right].$$

Отметим важное условие (не обязательно необходимое) для равновесной системы, так называемое условие баланса $W(A \to B)P(A,t) = W(B \to A)P(B,t)$. Данное условие может быть использовано, в частности, и для распределения Больцмана:

$$\frac{W(A \to B)}{W(B \to A)} = \frac{e^{-E(B)/kT}}{e^{-E(A)/kT}} = e^{-\Delta E/kT}; \Delta E = E(B) - E(A).$$

Есть много способов выбора функции W такой, что удовлетворяется условие баланса. В алгоритме Метрополиса выбор переходной функции осуществлен следующим образом:

$$W(A \to B) = \begin{cases} e^{-\Delta E/kT}, \text{если } \Delta E > 0 \\ 1, \quad \text{если } \Delta E \le 0 \end{cases} \qquad (7)$$

На каждом шаге алгоритма производится пробное изменение конфигурации системы. Для Изинговой модели изменение конфигурации A производится переключением состояния выбранного спина. Полученная конфигурация B принимается с вероятностью Метрополиса (7). По истечении некоторого времени система приходит к равновесной конфигурации с низкой энергией.

Алгоритм Метрополиса для спиновых моделей очень легко поддается распараллеливанию, поскольку он является регулярным и локальным. Хотя алгоритм Метрополиса прост для распараллеливания, невозможно производить обновление всех узлов одновременно, поскольку это будет сказываться на условии баланса: узлы ближайших сосе-

дей зависимы, поэтому должны обновляться по отдельности. Рассмотрим решетку, разделенную в шахматном порядке на черные и красные узлы. Все черные узлы можно обновить параллельно, поскольку они независимы, а затем параллельно обновить все красные узлы.

Так или иначе, всегда нужно помнить, что основная идея метода Монте-Карло состоит в аппроксимации бесконечно большой суммы конечной суммой, взятой по существенным конфигурациям. Ошибка является преимущественно статистической, пропорциональной $1/\sqrt{N}$ для N независимых конфигураций. Таким образом, возможна другая схема распараллеливания: различные процессоры осуществляют полностью независимое решение с различными потоками случайных чисел, чтобы на конечном этапе добавить результат к оцениваемой сумме.

Интересное поведение Изинговой модели наблюдается при очень низких температурах T. Для таких температур вероятность перехода в другое состояние может быть настолько мала, что нереально осуществить моделирование системы за разумное время вычислений. Причина кроется в том, что функция энергии имеет большое число локальных минимумов, и при малых температурах траектории сваливаются не к глобальному минимуму энергии, а лишь к некоторому локальному минимуму. Данное поведение аналогично *закалке металла*, когда, начиная с горячего (неупорядоченного) состояния, металл быстро охлаждается до низкой температуры. Для реальных металлов это приводит к достижению локального минимума энергии, в котором он становится хрупким, некристаллизованным. Чтобы при закалке избежать «сваливания» траекторий в метастабильные состояния, охлаждение производится очень медленно, что дает металлам время на упорядочивание своей структуры до стабильной, структурно сильной конфигурации с низкой энергией. Мы можем применить подобный подход в Монте-Карло, начиная со случайной конфигурации при очень высокой температуре и очень медленно уменьшая ее до тех пор, пока не установится требуемая очень низкая температура.

Нахождение состояния наименьшей энергии является примером *задачи глобальной оптимизации*, т.е. задачей нахождения глобального минимума функции стоимости $C(s)$ для значений s из допустимого множества S. Алгоритм Метрополиса может быть обобщен для решения оптимизационных задач путем введения фиктивной температуры и

трактованием функции стоимости как энергии. Для общей задачи глобальной оптимизации температура является параметром, задающим вероятность увеличения функции стоимости на любом шаге посредством обычного алгоритма Метрополиса в виде $e^{-\Delta C/T}$, где ΔC – изменение функции стоимости при заданном изменении конфигурации (значений параметров). Нулевая температура соответствует алгоритму *наискорейшего спуска*, при котором принимаются только те изменения, которые уменьшают значение энергии. Джеман & Джеман показали, что если температура уменьшается как

$$T_k = T_0 / \log k \tag{8}$$

для достаточно большого значения T_0, есть статистическая гарантия нахождения оптимального значения. Хотя результаты Джемана & Джемана могут показаться достаточно слабыми утверждениями, нужно понимать, что другие алгоритмы оптимизации не могут дать гарантии нахождения статистически оптимального решения для произвольной оптимизационной задачи.

Преимущества модельной «закалки»:

- Может производить оптимизацию произвольных систем и функций стоимости.
- Статистически гарантирует нахождение оптимального решения.
- Относительно проста для программирования даже для сложных проблем.
- Обычно дает «хорошее» решение.

Недостатки модельной «закалки»:

- Многократное повторение «закалки» со скоростью сходимости $1/\log k$ является очень медленным, особенно если функция стоимости сложна для вычислений.
- Для задач, где функция стоимости достаточно гладкая, или же имеются всего несколько локальных минимумов, модельная «закалка» является той самой пушкой, которая стреляет по воробьям: существуют более простые и быстрые методы.
- Эвристические методы, ориентированные на конкретную задачу, часто проявляют себя лучше, хотя часто модельная «закалка» сравнима с эвристическими методами.
- Метод не дает ответа на вопрос, найдено ли оптимальное решение.

Впервые модельная «закалка» была применена для задачи распределения носителей информации между двумя чипами. Данная задача эквивалента *NP*-трудной проблеме разделения графов. Вторым примером, где может быть с успехом применена модельная «закалка», является классическая задача коммивояжера. Как и в предыдущем примере, функция стоимости (длина пройденного пути), может иметь множество локальных минимумов, а мощность множества *S* чрезвычайно большая.

Заключение

Рассмотренные некоторые общие подходы к построению параллельных алгоритмов Монте-Карло являются лишь вершиной айсберга сложных проблем, требующих огромных вычислительных ресурсов в различных областях знаний. На нескольких примерах проиллюстрированы основные идеи и проблемы, возникающие при применении метода Монте-Карло.

АЛГОРИТМ ПАРАЛЛЕЛЬНОГО ПОСТРОЕНИЯ АДАПТИВНОЙ ТРЕУГОЛЬНОЙ КОНЕЧНО-ЭЛЕМЕНТНОЙ СЕТКИ ДЛЯ СТАТИЧЕСКИХ ЗАДАЧ ДЕФОРМИРОВАНИЯ ПЛАСТИН

П.Г. Медведев

Нижегородский государственный университет им. Н.И. Лобачевского

Рассматривается задача упругого деформирования пластины. Решение задачи производится численно с использованием метода конечных элементов на основе принципа виртуальной работы. Конечно-элементная сетка строится адаптивной по итерационному алгоритму [1,2]. При решении используется конечный элемент треугольной формы с узлами в вершинах. При построении сетки учитываются критерии на форму и размер ее элементов [3].

Алгоритм построения адаптивной сетки является итерационным. Суть итерационного алгоритма заключается в следующем. Пусть есть на некотором шаге итерации построенная сетка. Поочередно рассматриваются два элемента, имеющих одну грань. Деление этих элементов производится в случае, когда относительная разность энергий их де-

формаций больше заданной величины, т. е. в случае неудовлетворения «энергетическому критерию».

После деления элементов, производится перестроение сетки по критерию Делоне [3] с учетом критериев на формы и размера. Далее, осуществляется решение задачи и определяется энергия деформации элементов новой сетки. Процесс построения адаптивной сетки заканчивается, когда все элементы удовлетворяют «энергетическому критерию».

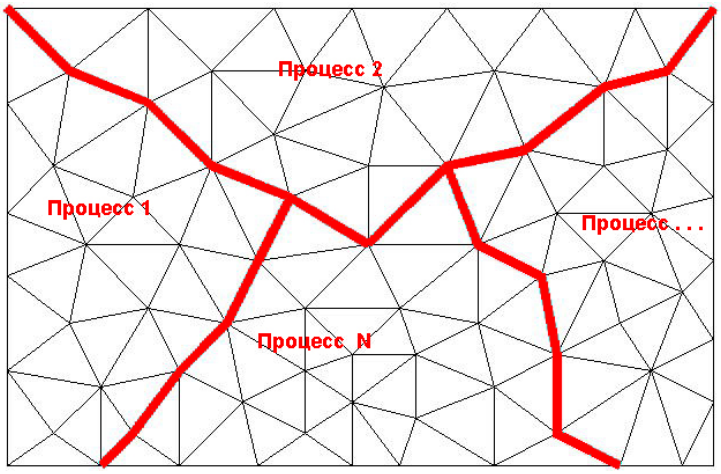


Рис. 1

Для параллельного решения задачи конечно-элементная сетка пластины логически разбивается на части, которые в дальнейшем будут соответствовать различным процессам (рис. 1). На шаге итеративного построения адаптивной сетки производится численное решение задачи МКЭ с использованием принципа виртуальной работы. При этом для каждой части сетки вычисляется свой блок матрицы жесткости конструкции и блок вектора узловых сил. В результате получается система линейных алгебраических уравнений, части которой соответствуют различным процессам. После решения системы на основе вектора узловых перемещений определяется удельная энергия деформации конечного элемента. Согласно распределению энергии деформации, производится новое деление сетки с последующим ее перестроением.

Литература

1. Альес М.Ю., Копысов С.П., Варнавский А.И. Параллельный алгоритм построения сеток Вороного–Делоне в R^3 // Тез. докл. 11-ой зимней школы по механике сплошных сред Пермь , 23 февраля–1 марта 1997 (http://www.icmm.ru/conf/w97).
2. Альес М.Ю., Копысов С.П., Варнавский А.И. Параллельное решение плоской задачи теории упругости на адаптивной сетке. Там же.
3. Jeffrey S. Ely, Anthony P. Leclerc. Correct Delaunay Triangulation in the Presence of Inexact Inputs and Arithmetic // Reliable Computing. 2000. V. 6. P.23–28.

ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ ДЛЯ ОДНОПРОЦЕССОРНЫХ КОМПЬЮТЕРНЫХ СИСТЕМ НА БАЗЕ PENTIUM® 4 В ЗАДАЧЕ МАТРИЧНОГО УМНОЖЕНИЯ

И.Б. Мееров, А.В. Сысоев

Нижегородский государственный университет им. Н И. Лобачевского

В данной статье рассматриваются подходы к оптимизации вычислений [1] для однопроцессорных компьютерных систем на базе Pentium® 4, особенности архитектуры и системы команд которых [2, 3] позволяют решать в режиме реального времени в числе прочих такие сложные расчетные задачи, как обработка сигналов, 3D графика, обработка видео и звука.

Целью настоящего исследования являлось построение эффективной реализации алгоритма и сравнение временных характеристик для различных оптимизирующих C++ компиляторов в задаче умножения случайно генерируемых квадратных матриц.

Соглашения и обозначения

Пусть A, B – исходные матрицы размерности N x N, заполненные случайными числами типа double; C – матрица-результат. T – время работы алгоритма.

Используемые средства

Аппаратное обеспечение: PC на основе Intel® Pentium® 4 1300 MHz, RAM 256 Mb.

Платформа: Microsoft® Windows® 2000 Professional.

Среды разработки: Borland® C++ Builder 5.0, Microsoft® Visual Studio 6.0.

Компиляторы: Borland® C++ for Win32 Compiler 5.5, Microsoft® 32-bit C/C++ Optimizing Compiler, Intel® C++ Compiler 5.0.

Дополнительные средства: библиотека PLAPACK 3.0.

Базовый алгоритм

$$Cij = \sum_{k=1}^{N} AikBkj,\ i = \overline{1,N},\ j = \overline{1,N}$$

Оценка производительности

Будем оценивать производительность по формуле $R = (2*N^3)/T$.

Реализация 1.

Рассмотрим следующую простейшую реализацию алгоритма умножения матриц:

```
for (i=0;i<N;i++)
  for (j=0;j<N;j++)
    for (k=0;k<N;k++)
       C[i][j]+=A[i][k]*B[k][j];
```

В результате проведенных вычислительных экспериментов были получены следующие временные характеристики (время счета):

Размер Компиляторы	N=1000	N=1500
Borland® C++ for Win32 Compiler 5.5	50,67 с	145,53 с
Microsoft® 32-bit C/C++ Optimizing Compiler	37,91 с	94,52 с
Intel® C++ Compiler 5.0 (включены оптимизирующие опции)	39,15 с	93,50 с

Данная реализация имеет следующие показатели (в зависимости от компилятора): минимальная производительность Rmin = 39,5MFp, максимальная производительность Rmax = 72,2MFp.

К очевидным недостаткам алгоритма следует отнести:

- большое количество итераций циклов (проблемы с ветвлениями);
- наличие в качестве тела цикла сложного выражения (явно зависящего от всех трех переменных цикла);
- зависимость текущей итерации от предыдущей итерации (нельзя распараллелить);
- отсутствие ориентации на принципы работы встроенной кэш-памяти Intel® Pentium® 4.

Реализация 2.

Следующая реализация предлагает решение, устраняющее часть из вышеперечисленых недостатков и позволяющее использовать механизм автоматического распараллеливания циклов (векторизации) компилятором Intel® C++ Compiler 5.0.

```
for(i=0,i1=1,i2=2;i<N-2;i+=3,i1+=3,i2+=3){
  for(j=0;j<N;j++){
    p=&B[j][0];
    tmp=A[i][j]; tmp1=A[i1][j]; tmp2=A[i2][j];
    ci=&C[i][0]; ci1=&C[i1][0]; ci2=&C[i2][0];
    for(k=0;k<N;k++){
       temp=p[k];
       ci[k]+=tmp*temp;
       ci1[k]+=tmp1*temp;
       ci2[k]+=tmp2*temp;
}}}
```

Временные характеристики:

Размер Компиляторы	N=1002	N=1500
Borland® C++ for Win32 Compiler 5.5	22,44 с	75,21 с
Microsoft® 32-bit C/C++ Optimizing Compiler	6,70 с	22,89 с
Intel® C++ Compiler 5.0 (включены оптимизирующие опции)	3,36 с	10.33 с

Данная реализация имеет лучшие показатели (в зависимости от компилятора) по сравнению с предыдущей: минимальная производительность Rmin=89,1MFp, максимальная производительность Rmax=653MFp. К достоинствам алгоритма следует отнести:

- устранение зависимостей от внешних индексов во внутреннем цикле;
- возможность распараллеливания команд во внутреннем цикле;
- ориентацию на принципы работы встроенной кэш-памяти Intel® Pentium® 4.

Реализация 3.

Следующая реализация является наиболее эффективной среди рассмотренных.

```
for(k=0;k<N;k+=10) {
    b1=&B[k][0];  b2=&B[k+1][0];
    b3=&B[k+2][0]; b4=&B[k+3][0];
    b5=&B[k+4][0]; b6=&B[k+5][0];
    b7=&B[k+6][0]; b8=&B[k+7][0];
    b9=&B[k+8][0]; b10=&B[k+9][0];
    for(i=0;i<M;i++){
      q=&C[i][0];
      t1=A[i][k]; t2=A[i][k+1];
      t3=A[i][k+2]; t4=A[i][k+3];
      t5=A[i][k+4]; t6=A[i][k+5];
      t7=A[i][k+6]; t8=A[i][k+7];
      t9=A[i][k+8]; t10=A[i][k+9];
        for(j=0;j<M;j++){
           q[j]+=t1*b1[j]+t2*b2[j]+t3*b3[j]+t4*b4[j]+
                 t5*b5[j]+ t6*b6[j]+t7*b7[j]+t8*b8[j]+
                 t9*b9[j]+t10*b10[j];
}}}
```

Временные характеристики:

Размер Компиляторы	N=1000	N=1500
Borland® C++ for Win32 Compiler 5.5	9.60 с	32.47 с
Microsoft® 32-bit C/C++ Optimizing Compiler	3.76 с	13.86 с
Intel® C++ Compiler 5.0 (включены оптимизирующие опции)	1.50 с	4.99 с

Данная реализация имеет следующие показатели (в зависимости от компилятора): минимальная производительность Rmin=208MFp, максимальная производительность Rmax=1353MFp. К достоинствам алгоритма можно отнести:

- устранение зависимостей от внешних индексов во внутреннем цикле;
- уменьшение количества итераций цикла («разворачивание цикла»);
- обращение по индексу ровно столько раз, сколько необходимо;
- возможность распараллеливания команд во внутреннем цикле;

- ориентацию на принципы работы встроенной кэш-памяти Intel® Pentium® 4.

Наилучшие результаты достигаются, прежде всего, из-за эффективного использования кэш-памяти и ориентации на векторизацию внутреннего цикла при использовании Intel® C++ Compiler 5.0.

PLAPACK 3.0

Для оценки эффективности приведенных реализаций их результаты были сопоставлены с результатами, полученными в вычислительных экспериментах, проведенных на базе стандартного теста – библиотеки PLAPACK.

Временные характеристики

Размер Компиляторы	N=1000	N=1500
Intel® C++ Compiler 5.0	2,66 с	8,87 с

Основные выводы

При реализации алгоритмов необходимо обращать внимание на эффективное использование встроенной кэш-памяти Intel® Pentium® 4, а также программировать таким образом, чтобы позволить оптимизирующему компилятору применить векторизацию. При таком подходе становится возможным получение существенного прироста производительности при использовании Intel® Pentium® 4.

Intel® C++ Compiler 5.0 проявил себя наилучшим образом при решении задачи умножения квадратных матриц в плане использования возможностей аппаратуры и системы команд.

Применение оптимизационных решений позволяет получить лучшие результаты по сравнению с PLAPACK 3.0.

Литература

1. Михальчук В.М., Ровдо А.А., Рыжиков С.В. Микропроцессоры 80x86, Pentium®. Архитектура, функционирование, программирование, оптимизация кода. Минск: «Битракс», 1994.
2. Introducing the Streaming SIMD Extensions 2 for the Pentium® 4 Processor http://developer.intel.com/software/products/itc/sse2/sse2down.htm.

3. Шагурин И.И., Бердышев Е.М. Процессоры семейства Intel P6. Архитектура, программирование, интерфейс //М: «Горячая линия – телеком», 2000.

ДИНАМИЧЕСКОЕ УПРАВЛЕНИЕ РЕСУРСАМИ МУЛЬТИКЛАСТЕРНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Ю.А. Мельников

Санкт-Петербургский государственный электротехнический университет

Введение

В докладе рассматриваются вопросы оптимизации распределения ресурсов суперкомпьютерной системы. Формальная постановка задачи (в терминах алгебры логики) делает предлагаемое формальное решение применимым для любых суперкомпьютеров, имеющих структуру процессорной (кластерной) сети вида n-мерный куб. Реализовать предлагаемое формальное решение предполагается на основе ОС PARIX фирмы PARSYTEC.

ОС PARIX имеет ограниченные возможности в области распределения ресурсов суперкомпьютера. В основу механизма распределения ресурсов суперкомпьютера в ОС PARIX положено понятие партиции – заданного на этапе конфигурации объединения узлов (процессоров или кластеров). Таким образом, все узлы суперкомпьютера распределяются между несколькими партициями.

Каждой пользовательской задаче предоставляется партиция с соответствующими или превышающими требования задачи параметрами (количество узлов, наличие устройств ввода-вывода и т.д.). Выделение ресурсов сверх необходимых происходит в случае, когда в конфигурации нет партиции, строго соответствующей требованиям пользовательской задачи.

Вдобавок, ОС PARIX не имеет очереди пользовательских задач, что накладывает дополнительные ограничения на использование ресурсов суперкомпьютера под управлением PARIX.

В качестве решения описанных проблем предлагается механизм динамического выделения ресурсов суперкомпьютера в строгом соот-

ветствии с требованиями пользовательских задач. Для максимизации количества одновременно загруженных узлов суперкомпьютера предполагается использовать очередь пользовательских задач с приоритетами.

Формальная постановка задачи

Сопоставим каждому узлу процессорной сети вершину n-мерного куба. При наличии дополнительных устройств, подключенных к узлу (устройства ввода-вывода и т.д.), определим для соответствующей вершины n-мерного куба необходимые атрибуты. Подкубом размерности k будем называть множество 2^k вершин n-мерного куба, имеющих n-k одинаковых координат.

Каждая пользовательская задача имеет требования к ресурсам: количество узлов процессорной сети, наличие устройств ввода-вывода и т.д. Представим необходимый для исполнения задачи набор узлов в виде подкуба. Требования о наличии дополнительных устройств представим в виде атрибутов подкуба в целом.

Определим максимально полное покрытие n-мерного куба как объединение непересекающихся подкубов, при котором количество вершин n-мерного куба, не принадлежащих ни одному из подкубов, минимально.

Таким образом, задача оптимального распределения ресурсов суперкомпьютерной системы сводится к нахождению максимально полного покрытия n-мерного куба с учетом атрибутов, назначенных вершинам n-мерного куба и подкубам.

Представим очередь пользовательских задач как упорядоченное множество подкубов. Фактором, упорядочивающим множество, является время и дата постановки задачи в очередь. Алгоритм построения максимально полного покрытия (далее покрытия) n-мерного куба из элементов очереди представляется следующим образом:

1. Добавить к покрытию первый находящийся в очереди подкуб.
2. Попытаться добавить к покрытию следующий находящийся в очереди подкуб.
   - Если это возможно, то добавить подкуб к покрытию и перейти к п.2;
   - иначе, если это последний подкуб, то перейти к п. 3;
   - иначе перейти к п. 2.

3. Удалить из очереди все добавленные к покрытию подкубы и перейти к п. 4.
4. Покрытие n-мерного куба сформировано.

Реализация

В ОС PARIX обязанность распределения ресурсов суперкомпьютера между пользовательскими задачами возложена на модуль Network Resource Manager (NRM). NRM взаимодействует с модулем Control Network, загружаемым в процессорную сеть и отвечающим за непосредственное управление ресурсами процессорной сети. Таким образом, написав замену модулю NRM, можно реализовать динамическое распределение ресурсов суперкомпьютера и управление очередью пользовательских задач.

РАСПАРАЛЛЕЛИВАНИЕ ВЫЧИСЛЕНИЙ ТРАНСЦЕНДЕНТНЫХ ФУНКЦИЙ НА СУПЕРСКАЛЯРНЫХ АРХИТЕКТУРАХ INTEL

А.А.Нарайкин, А.В.Ковалёв

Нижегородская лаборатория Intel(INNL)

Введение

Используя компьютеры для решения различных задач, пользователь рано или поздно сталкивается с недостаточностью вычислительной мощности используемых вычислительных средств. Не секрет, что наиболее ресурсоемкие задачи связаны с интенсивными вычислениями как целочисленными, так и с плавающей запятой. Современные суперскалярные архитектуры корпорации Intel предоставляют новые возможности для эффективного решения этих проблем.

Оставляя обсуждение возможности и как следствие – эффективности использования суперкомпьютеров для данного типа задач, предлагается наиболее популярный сейчас подход – построение высокопроизводительной вычислительной системы на базе доступных и широко распространненых персональных ЭВМ. Такие решения достаточно актуальны, так как предоставляют возможность достичь высокой вычислительной мощности при гораздно меньших расходах. Это

очень важно, особенно для образовательных и научных учреждений, которым часто суперкомпьютеры просто не доступны.

В работе подробно рассматривается класс задач, связанных с интенсивным использованием блока вещественной арифметики, особенно такие, в которых значительную долю составляют вычисления элементарных математических функций. Эти задачи условно разделяются на два вида. Первый, наиболее часто встречающийся, составляют задачи, в которых происходит обработка достаточно больших массивов входных аргументов. Второй связан с приложениями, активно использующими скалярные математические функции, в силу таких особенностей алгоритма, как, например, рекурсивность.

Подобная условная классификация весьма удачна в данном случае и позволяет сразу же увидеть важные особенности при распараллеливании математических функций. В большинстве известных сейчас платформ персональных ЭВМ и, в частности, платформ Intel Architecture, поддержка вычислений элементарных и трансцендентных математических функций реализована либо в интегрированном математическом сопроцессоре, либо программной эмуляцией. Как в первом, так и во втором вариантах эти функции достаточно дешевы для того, чтобы заниматься распараллеливанием их алгоритма, даже когда вычисления проводятся над векторами входных аргументов. С другой стороны, обладая кластером независимых машин и производя вычисления над вектором аргументов, мы можем тривиально распараллелить их, выдавая каждому узлу независимые входные значения и производя на каждом из них полностью независимые вычисления. Эффективность распараллеливания в таком случае может достигать теоретических 100% в зависимости от скорости обмена между узлами и «главной» машиной, сохраняющей результаты. Многое на этом этапе зависит от эффективности прикладных пакетов, позволяющих организовать вычисления в сети, базирующихся на стандарте MPI. Если в составе кластера есть многопроцессорные узлы, то потери можно сократить за счет использования в таких машинах общей памяти. В настоящее время для таких целей может быть использован стандарт OpenMP, поддерживаемый рядом компиляторов, в том числе и корпорации Intel, для языков Fortran и C/C++. Кроме того, можно исходный вектор аргументов делить на некоторое подмножество входных векторов и выдавать каждый их них на отдельный узел. Для этого можно использо-

вать любой подходящий алгоритм в зависимости от конкретной организации кластера и с учетом вычислительных возможностей каждого узла.

Однако такие решения не помогут приложениям, основное машинное время которых «съедают» скалярные вызовы математических функций. Как уже было отмечено, трансцендентные математические функции дешевы и их распараллеливание неэффективно. Действительно, для популярного процессора PentiumIII стоимость любой такой функции не превышает 100 процессорных тактов и в то же время их частота достигает 1GHz. Понятно, что получить выигрыш путем замены одного компьютера высокопроизводительной вычислительной сетью в данном случае возможно, лишь запустив несколько копий приложения на разных узлах с различными параметрами, т.е. алгоритмическое распараллеливание на уровне сети исключено.

Возможность использования OpenMP на многопроцессорных узлах для распараллеливания скалярных функций следует, видимо, также считать неэффективной из-за стоимости накладных расходов на организацию вычислений. Тем не менее, можно пытаться улучшить производительность, используя параллелизм на уровне инструкций (Instruction Level Parallelism), повышая таким образом эффективность узлов кластера и удешевляя его стоимость. Это тем более интересно, поскольку, улучшая работу отдельного узла для этого вида задач, мы также автоматически улучшаем производительность рассмотренных задач первого вида, обрабатывающих вектора аргументов. И здесь важную роль могут играть архитектурные особенности используемых для построения кластера персональных машин.

Большинство современных процессоров так или иначе используют технологии, перекликающиеся с идеями параллельной организации вычислений. В данной работе сделан упор на современные платформы Intel Architecture (IA). Семейство 32х-битных процессоров этой архитектуры (IA-32) имеет нескольких ярких черт подобного рода. Прежде всего, это наличие инструкций, работающих по принципу SIMD(Single Instruction Multiple Data), которые позволяют выполнять однотипные операции над несколькими аргументами за одну команду. Так, например, набор SSE содержит в себе операции сложения, вычитания и вычисления квадратного корня согласно стандарту IEEE-754 для 4 чисел в формате плавающей запятой с одинарной точностью, а SSE2 позво-

ляет оперировать также 2 числам двойной точности. Другой характерной особенностью является возможность так называемого динамического запуска команд: процессор определяет зависимые инструкции и запускает вне очереди следующие команды, не дожидаясь окончания выполнения текущей инструкции. Это позволяет уменьшить потери производительности из-за зависимого характера вычислений, особенно учитывая наличие аппаратного кэширования, а также многоступенчатого конвейера исполнения инструкций. Еще дальше продвинуты в этом направлении представители нового 64х-битного семейства IA-64. Система команд первого процессора этого семейства Itanium построена по принципу EPIC (Explicitly Parallel Instruction Computing), дословно, допускающая параллелизм в явном виде. Особенности архитектуры и машинного кода данного процессора позволяют выполнять любые 2 независимые операции над числами с плавающей запятой, в том числе и SIMD-подобные инструкции.

Все эти возможности позволяют повысить эффективность вычислений узлов кластера, основываясь на мелкозернистом распараллеливании, которое в данном классе задач является наиболее выгодным подходом и практически единственным, позволяющим добиться заметных результатов в общем случае.

В докладе авторы данной работы на основе опыта разработки скалярных и векторных математических функций с использованием архитектурных особенностей IA-32 и IA-64 приводят общий алгоритм для трансцендентных функций и подробно рассматривают возможности по его распараллеливанию, ограничения общего случая и дополнительные преимущества в случае обработки векторов данных.

Литература

1. Scientific Computing on the Itanium™ Processor, SuperComputing 2001 Bruce Greer, John Harrison, Greg Henry, Wei Li, Peter Tang. Computational Software Lab, Intel Corporation.
2. Table-Driven Implementation of the Exponential Function in IEEE. Floating-Point Arithmetic, ACM. Ping Tak Peter Tang, Argonne National Laboratory .
3. Introduction to Parallel Programming with OpenMP. Tutorial session at SuperComputing 2001. Tim Mattson, Intel Corporation; Rudolf Eigenmann, Purdue University.

4. Optimizing Your Intel® Pentium® 4 and Itanium™ Applications Using Intel® Compilers and Performance Libraries. Hands-on Lab IDF Fall 2001, Aug 27–30, San Jose CA. Andrey Naraikin, Intel Nizhny Novgorod Lab.
5. Itanium math runtime library (opensource) http://developer.intel.com/software/products/opensource/libraries/num.htm.

ПРОБЛЕМЫ ЭФФЕКТИВНОГО ИСПОЛЬЗОВАНИЯ УЗЛОВ НА ОСНОВЕ АРХИТЕКТУР INTEL В ГЕТЕРОГЕННЫХ КЛАСТЕРАХ

А.А. Нарайкин, И.В. Лопатин

Нижегородская лаборатория Intel (INNL)

В данной работе исследованы некоторые пути увеличения производительности и возможности распараллеливания прикладных программ на архитектурах IA-32 и IA-64. Также рассмотрено применение функций, входящих в состав библиотек Intel® Performance Libraries. На примере программы моделирования водной поверхности Sunset исследованы возможности эффективного применения этих средств для научных приложений на архитектурах IA-32 и IA-64. Получены результаты для операционных систем Windows 2000 Professional и Linux (дистрибутив RedHat 7.1).

Несмотря на бурное развитие вычислительной техники, увеличивающаяся сложность прикладных задач заставляет максимально использовать возможности, предоставляемые архитектурой и программными средствами разработки. При этом часто бывает важно обеспечить соответствие кода стандартам и сохранить переносимость программ. Этих целей можно достичь, применяя специализированные библиотеки и задействуя возможности оптимизирующих компиляторов.

При выполнении программ на гетерогенных кластерах можно выделить три уровня параллелизма – команд процессора, многопроцессорных узлов с общей памятью и, наконец, коммуникаций между узлами. На общую производительность кластера существенно влияет эффективность выполнения программ на всех трех уровнях, однако в данной работе проблемы коммуникации узлов не рассматриваются. Использование оптимизации, специфической для архитектуры процес-

сора, поддержка векторных вычислений и стандарта OpenMP позволяют достичь максимальной производительности узлов с минимальными затратами времени и изменениями исходного кода. Эти возможности реализованы как в компиляторах Intel® C/C++ и Intel® Fortran [1], так и в наборе специализированных библиотек Intel® Performance Libraries [2].

Для иллюстрации применения данных технологий было рассмотрено приложение Sunset [3], осуществляющее численное моделирование изображения водной поверхности в реальном времени. Его достоинствами для такого рода экспериментов являются относительно небольшой размер ядра (около 50 килобайт) в сочетании с интенсивными вычислениями с плавающей точкой, которые занимают более 98% времени работы приложения (остальное время уходит на вывод изображения). Существуют как C, так и Fortran-версия вычислительного ядра, что позволяет оценить работу компиляторов сразу для двух языков. Приложение работает на архитектурах IA-32 и IA-64 под управлением как Windows, так и Linux. Производительность легко измеряется количеством кадров в секунду, которое показывается в нижней части окна программы.

Рассмотрение использования параллелизма на разных уровнях логично начать с самого нижнего – команд процессора. Поддержка параллелизма на уровне инструкций, осуществляемая компилятором, позволяет избежать трудоемкого низкоуровневого кодирования, обеспечивая при этом сопоставимую производительность. На архитектуре IA-32 это, прежде всего, выражается в возможности автоматической поддержки расширенных наборов инструкций MMX, SSE и SSE2, работающих по принципу SIMD (Single Instruction Multiple Data), которые позволяют выполнять однотипные операции над несколькими аргументами за одну команду. На IA-64 компилятор позволяет программисту использовать систему команд, построенную по принципу EPIC (Explicitly Parallel Instruction Computing), дословно, допускающую параллелизм в явном виде[4].

На графиках* 1,2 виден эффект от применения данного подхода на IA-32, достигаемый с помощью опций компилятора (QxK – оптимизация для Pentium III, QxW – Pentium 4). Столь существенный

* Результаты приведены для версии, реализованной на языке C. Аналогичные результаты получены для Fortran 90.

прирост производительности достигнут во многом благодаря использованию потоковых операций. Однако, следует помнить о возможных зависимостях внутри циклов. Для диагностики подобных проблем у компилятора существует соответствующий аппарат (например, опция Qvec_report [1,2,3]). В случае, если пользователь, зная алгоритм, уверен в отсутствии проблемы, о которой сигнализирует компилятор, то с помощью специальных директив возможно форсированное использование векторных инструкций независимо от результатов автоматического анализа зависимостей.

Применение директив компилятора и функций времени выполнения, описываемых в стандарте OpenMP[5], позволяет распараллелить программу для запуска на системах с общей памятью (в случае с архитектурой IA-32 – SMP от двух до четырех процессоров, Itanium – до 512 процессоров). OpenMP является переносимым промышленным стандартом, поддерживаемым большим количеством производителей.

Внедрение в программу директив OpenMP позволило получить почти линейное ускорение на всех рассмотренных платформах, как это видно из графиков. Ввиду переносимости стандарта и реализации его в Fortran и C, один и тот же набор директив использовался как в 32-битной Linux и Windows-версиях приложения, так и на 64-битной архитектуре. Следует отметить относительную простоту использования OpenMP по сравнению с реализацией той же функциональности явным заданием потоков с помощью вызовов функций операционной системы. Как правило, объем дополнительного кода невелик, что позволяет без больших затрат распараллеливать уже существующие последовательные приложения. В рассматриваемом примере для параллелизации цикла размером около 700 строк потребовалось 20 строк кода, описывающего директиву. Таким образом, применение OpenMP обеспечивает не только эффективное использование многопроцессорных узлов кластера, но и высокую переносимость программ между операционными системами и различными архитектурами.

Набор специализированных библиотек Intel® Performance Libraries также позволяет эффективно использовать аппаратные возможности архитектур Intel. Библиотеки предоставляют интерфейс к высокооптимизированным подпрограммам, которые находят применение в таких областях, как цифровая обработка сигналов, решение систем линейных уравнений, обработка аудио- и видео информации. Кроме того, функ-

циональность библиотек расширяет возможности компилятора, ограниченного рамками стандарта языка. В качестве примера можно привести шкалируемость точности а, следовательно, и скорости вычисления трансцендентных математических функций. Эффект от использования библиотек отражен на графиках 2 и 3 и особенно ярко проявляется на IA-64, что обусловено наличием технологии EPIC.

Описанные выше подходы могут быть применены как по отдельности, так и совместно. Последнее приближает к максимуму общую эффективность распараллеливания на многопроцессорных компьютерах. Глядя на графики, можно оценить влияние различных комбинаций на производительность.

В заключение авторы хотят еще раз подчеркнуть важность полноценного использования каждого вычислительного узла, входящего в состав кластера. Полученные результаты говорят о том, что учет особенностей архитектуры в сочетании с возможно более полным задействованием возможностей мультипроцессоров с общей памятью позволяет на порядок повысить эффективность работы вычислительной системы в целом. Это позволяет понизить стоимость кластера при сохранении сложности решаемых задач, либо использовать уже имеющуюся аппаратуру для решения задач большей сложности.

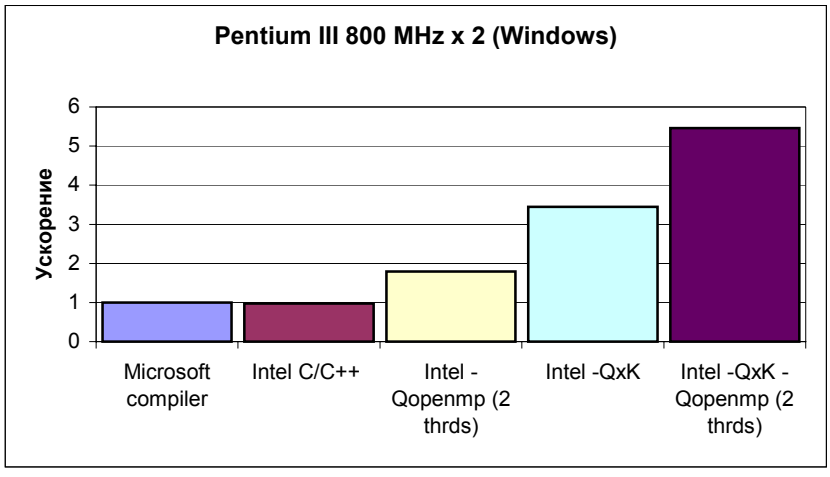


График 1. Ускорение работы программы Sunset на Pentium III

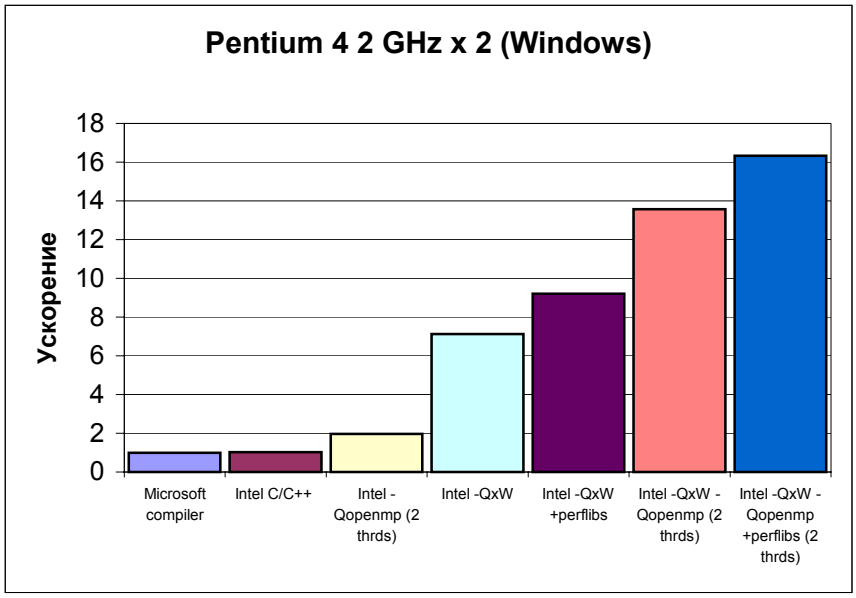


График 2. Ускорение работы программы Sunset на Pentium 4

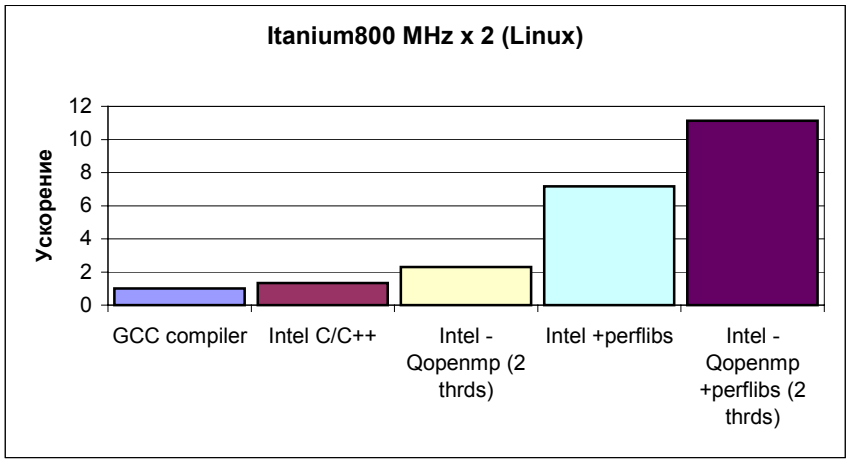


График 3. Ускорение работы программы Sunset на Itanium.

Литература

1. Intel compilers home page. http://developer.intel.com/software/products/compilers/.
2. Intel performance libraries home page. http://developer.intel.com/software/products/perflib/.
3. Численное моделирование изображения водной поверхности. GraphiCon'99, Москва, 26 августа – 1 сентября. Д.Абросимов, В.Зеленогорский, М.Крюков. Нижегородская лаборатория программных технологий.
4. Scientific Computing on the Itanium™ Processor. SuperComputing 2001 Bruce Greer, John Harrison, Greg Henry, Wei Li, Peter Tang. Computational Software Lab, Intel Corporation.
5. OpenMP C/C++ Application Program Inreface. Version 1.0. http://www.openmp.org.
6. Optimizing Your Intel® Pentium®4 and Itanium™ Applications Using Intel® Compilers and Performance Libraries. Hands-on Lab IDF Fall 2001, Aug 27–30, San Jose CA. Andrey Naraikin, Intel Nizhny Novgorod Lab.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В МНОГОСЛОЙНЫХ ДИНАМИЧЕСКИХ СИСТЕМАХ*

К. Невидин

Волжская государственная академия водного транспорта.

Введение

В последние несколько десятилетий в различных областях знаний большой интерес вызывают явления в связанных хаотических системах. Исследования пространственно временной динамики ансамблей связанных элементов обнаружило различные типы когерентных структур, паттернов и явлений синхронизации. В ранних работах рассматривались системы с небольшим количеством элементов, в настоящее время все возрастающий интерес вызывают большие решетки или *сети* связанных хаотических элементов, особенно при изучении процес-

* Работа выполнена при поддержке фонда РФФИ, грант № 99-01-01126.

сов синхронизации. Простейший режим пространственно-временного поведения, который может появиться в решетке связанных идентичных элементов,– это режим *полной синхронизации*. В данном случае все элементы системы демонстрируют идентичное поведение независимо от начальных условий. Явление хаотической синхронизации имеет несколько возможных областей применения. Например, хаотическая синхронизация может использоваться как метод передачи информации при помощи хаотического сигнала. *Кластерная синхронизация* наблюдается в решетке, когда элементы синхронизованы в группы. Причем между группами синхронизация отсутствует. Кластеризация обычно определяется в более широком смысле, как возникновение когерентных структур больших масштабов с различной степенью корреляции между взаимодействующими системами. Нами обнаружены семейства вложенных линейных инвариантных многообразий пространственных решеток диффузионно связанных идентичных динамических систем с периодическим циклом или хаотическим аттрактором. Будучи тесно связанными с числом элементов в решетке и с граничными условиями, данные инвариантные многообразия образуют полный набор всех допустимых режимов кластерной синхронизации, которые могут возникать в решетках. Под кластерной синхронизацией в данном случае понимается пространственно-временной режим, при котором элементы, составляющие один и тот же кластер, имеют идентичное поведение, т.е. полностью синхронизованы между собой, и это поведение является устойчивым по отношению к возмущениям.

В работе изучается трехмерная решетка идентичных, диффузионно связанных элементов, описываемая следующей динамической системой:

$$\begin{cases} \dot{x}_{i,j,k} = f(x_{i,j,k}, y_{i,j,k}) + \varepsilon_1 (x_{i-1,j,k} - 2x_{i,j,k} + x_{i+1,j,k}) + \\ \quad + \varepsilon_2 (x_{i,j-1,k} - 2x_{i,j,k} + x_{i,j+1,k}) + \varepsilon_3 (x_{i,j,k-1} - 2x_{i,j,k} + x_{i,j,k+1}) \\ \dot{y}_{i,j,k} = g(x_{i,j,k}, y_{i,j,k}), \quad i = \overline{1, N_1},\ j = \overline{1, N_2},\ k = \overline{1, N_3} \end{cases} \quad (1)$$

с неймановскими или периодическими граничными условиями. Здесь $x_{i,j,k} \in R^1, y_{i,j,k} \in R^{m-1}$ – переменные, описывающие поведение системы, $f(x_{i,j,k}, y_{i,j,k}) : R^m \to R^1$ – скалярная, $g(x_{i,j,k}, y_{i,j,k}) : R^m \to R^{m-1}$ – векторная функции, задающие поведение индивидуального элемента

решетки, m – размерность индивидуального элемента, $D = N_1 \times N_2 \times N_3 \times m$ – размерность фазового пространства системы (1), ε_1, ε_2, ε_3– параметры, определяющие связь между элементами по трем направлениям решетки, N_1, N_2 и N_3 определяют размер решетки. Заметим, что система (1) представляет класс трехмерных уравнений реакции-диффузии нелинейных клеточных сетей, который может рассматриваться как универсальная динамическая система, моделирующая распределенные среды, в которых может происходить образование паттернов, волн и других явлений синхронизации в физике, биологии, химии, экологии и других областях науки.

Введем следующие обозначения: $X_{i,j,k} = \begin{pmatrix} x_{i,j,k} \\ y_{i,j,k} \end{pmatrix}$ – m-мерный вектор, $M_0(N) - N_1 \times N_2 \times N_3 \times m$-мерное фазовое пространство с координатной матрицей $X_{i,j,k}$, $M(d) \equiv M(n_1, n_2)(d = n_1 \times n_2 \times n_3)$ – $d \times m$-мерное евклидово пространство с координатной матрицей $(U_1, U_2, ..., U_d) \in R^{d \times m}$.

Определение. Если существует отображение T: $M(d) \to M(0)$, такое, что система (1) под действием отображения T преобразуется к $m \times d$ линейно независимых уравнений, то будем говорить, что система (1) имеет инвариантное многообразие $M(d)$ соответствующее D подсистемам в (1), синхронизованным в d кластеров.

Пример 1. При $d = 1$ в решетке (1) существует многообразие полной синхронизации $M(1) = \{X_{i,j,k} = U_1\}$

$$\tilde{M}(1) = [\underbrace{\{r_1, r_1, ..., r_1\}}_{N_1}; \underbrace{\{c_1, c_1, ..., c_1\}}_{N_2}; \underbrace{\{p_1, p_1, ..., p_1\}}_{N_3}].$$

При $d = N_1 \times N_2 \times N_3$ многообразие $M_0(N)$ есть фазовое пространство системы (1)

$$\tilde{M}(N) = [\underbrace{\{r_1, r_2, ..., r_{N_1}\}}_{N_1}; \underbrace{\{c_1, c_2, ..., c_{N_2}\}}_{N_2}; \underbrace{\{p_1, p_2, ..., p_{N_3}\}}_{N_3}].$$

Утверждение 1.

1. В системе (1) существует симметричное инвариантное многообразие $M_s^r(n_1, N_2, N_3)$ с $n_1 = \text{int}((N+1)/2$, задаваемое отображением:

$$T_s^r : \{X_{N_1-i+1,j,k} = X_{i,j,k} = U_{i,j,k},$$
$$i = 1,2,...,n_1,\ j = 1,2,...,N_2,\ k = 1,2,...,N_3\};$$

для нечетного $N_1 = 2n_1 - 1$

$$\widetilde{M}_s^r(n_1, N_2, N_3) =$$
$$= [\underbrace{\{r_1,...,r_{n_1-1}, r_{n_1}, r_{n_1-1},..., r_1\}}_{N_1}; \underbrace{\{c_1, c_2,..., c_{N_2}\}}_{N_2}; \underbrace{\{p_1, p_2,..., p_{N_3}\}}_{N_3}];$$

для четного $N_1 = 2n_1$

$$\widetilde{M}_s^r(n_1, N_2, N_3) =$$
$$= [\underbrace{\{r_1,...,r_{n_1}, r_{n_1},..., r_1\}}_{N_1}; \underbrace{\{c_1, c_2,..., c_{N_2}\}}_{N_2}; \underbrace{\{p_1, p_2,..., p_{N_3}\}}_{N_3}]\cdot$$

2. В системе (1) существует асимметричное инвариантное многообразие $M_a^r(n_1, N_2, N_3)$ с $N_1 = n_1 \cdot p_1$, n и p – произвольные целые числа, задаваемое отображением:

$$T_a^r : \{X_{N_{i+2n_1l},j,k} = X_{-i+1+2n_1l,j,k} = X_{i,j,k} = U_{i,j,k},$$
$$i = 1,2,...,n_1,\ l = 1,2,..,p_1,\ j = 1,2,...,N_2,\ k = 1,2,...,N_3\};$$

$$\widetilde{M}_r^a(n_1, N_2, N_3) =$$
$$= [\underbrace{\{r_1,...,r_{n_1}, r_{n_1},...,r_1, r_1...,r_{n_1},...\}}_{n_1 \cdot p_1}; \underbrace{\{c_1, c_2,..., c_{N_2}\}}_{N_2}; \underbrace{\{p_1, p_2,..., p_{N_3}\}}_{N_3}]\cdot\cdot$$

Утверждение 2.

В системе (1) существует пересечение многообразий $M_{s,a}^{\times}(n_1, n_2, N_3) = M_{s,a}^r(n_1, N_2, N_3) \bigcap M_{s,a}^c(N_1, n_2, N_3)$, задаваемое системой отображений (T^r, T^c) $\widetilde{M}^r(n_1, n_2, N_3) = \widetilde{M}^r \bigcap \widetilde{M}^c$, размерность многообразия $\dim M^{\times} = n_1 \times n_2 \times N_3 \times m$.

Пример 2.

Рассмотрим двумерную решетку, содержащую $N_1 = 3$ и $N_2 = 5$ элементов, и используем целые числа для обозначения элементов, так что элементы, обозначаемые одним и тем же числом, принадлежат одному и тому же кластеру и демонстрируют идентичное поведение. Два главных многообразия синхронизации для чисел N_1 и N_2 имеют вид:

$$M_S^{\times}(2,3)=\begin{pmatrix}1,2,3,2,1\\4,5,6,5,4\\1,2,3,2,1\end{pmatrix}\qquad M_S^{c}(2,5)=\begin{pmatrix}1,2,3,4,5\\6,7,8,9,10\\1,2,3,4,5\end{pmatrix}.$$

В случае решетки с числами $N_1 = 3$ и $N_2 = 6$ асимметричные многообразия имеют вид:

$$M_a^{\times}(2,2)=\begin{pmatrix}1,2,2,1,1,2\\3,4,4,3,3,4\\1,2,2,1,1,2\end{pmatrix}\qquad M_S^{\times}(2,3)=\begin{pmatrix}1,2,3,3,2,1\\4,5,6,6,5,4\\1,2,3,3,2,1\end{pmatrix}.$$

Проведено исследование последовательности вложений многообразий. Обнаружено, что многообразия образуют полный набор всех допустимых режимов кластерной синхронизации, которые могут возникать в решетках в зависимости от размеров решетки и типа граничных условий.

Проведено исследование условий глобальной устойчивости режима полной синхронизации. Найдены достаточные условия, при которых в решетке реализуется режим полной синхронизаци. Дана оценка необходимых условий устойчивости режима полной синхронизации. Найдены достаточные условия устойчивости многообразий. Подробное изложение приведенных выше результатов см. в [1].

Проведено исследование процесса образования пространственных структур при помощи численного моделирования в решетках небольших размеров (6×6, 7×7) состоящих из элементов, описываемых известными системами Лурье и Ресслера. Обнаружено, что в зависимости от начальных условий в решетке реализуются пространственные профили, соответсвующие описанным выше многообразиям синхронизации.

Численное исследование проводилось следующим образом. Задавались параметры элементов, коэффициенты связи между элементами и произвольные начальные значения амплитуд элементов. Численный расчет проводился до 10^6 итераций, за время, необходимое для формирования стационарного профиля. Пространство параметров имело достаточно подробное разбиение ~1000 точек. Для проведения рассчетов решеток больших размеров применялось распараллеливание по пространству параметров задачи.

Параллельный алгоритм

Условия задачи, а именно независимость рассчетов от точек пространства парамеров, предполагают простой путь для увеличения производительности за счет применения кластера из нескольких компьютеров. Для реализации алгоритма была написана программа, использующая интерфейс MPI – Message Passing Interface, реализованный в библиотеке MPICH. Программа состоит из двух частей потоков. Главный, управляющий поток формирует задание, т.е. проводит разбиение области параметров в зависимости от числа компьютеров в кластере, производит рассылку параметров дочерним потокам, собирает результаты рассчетов. Дочерние потоки принимают задания, проводят рассчеты и отсылают результаты рассчетов главному потоку.

Оценим эффективность примения параллельных вычислений для нашей задачи. Учтем несколько факторов:

1. Большое число точек области параметров.
2. Большое число итераций для получения стационарного профиля.
3. Малое время передачи параметров от главного к дочерним процессам.

Учитывая это, можно пренебречь временем передачи данных по сравнению со временем проведения рассчетов. Получаем, что ускорение от применения параллельных вычислений равно числу компьютеров в кластере.

Таким образом, применение параллельных вычислений приводит к существенному увеличению скорости вычислений (~10–15 раз), что дает возможность исследовать процессы синхронизации в системах больших размеров.

Дальнейшее развитие алгоритма

Кластер Нижегородского университета имеет в своем распоряжение 12 двухпроцессорных компьютеров. Для увеличения эффективности рассчетов на таких системах предполагается проводить интегрирование задачи параллельно на обоих процессорах, используя стандарт OpenMP. Для исследования сверхбольших решеток (десятки и более тысяч элементов) предполагается проводить интегрирование задачи на нескольких компьютерах одновременно, используя как OpenMP, так и MPI.

Автор выражает благодарность научному руководителю проф. В.Н. Белых за помощь и руководство работой, а также проф. В.П. Гергелю, предоставившему возможность использования компьютерного кластера Нижегородского государственного университета.

Литература

1. Belykh I.V., Belykh V.N., Nevidin K.N., Hasler M., Partial synchronization in two-dimensional lattices of coupled nonlinear systems, proceedings of the 9th workshop on Nonlinear Dynamics of Electronic Systems, Delft, The Netherlands, DUP Science, 2001. P. 197–200.

ПРЕОБРАЗОВАНИЯ ЦВЕТОВЫХ ПРОСТРАНСТВ И ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Д.Ю.Петров

Нижегородский государственный университет им. Н И. Лобачевского

Введение

В настоящее время прогресс программных и аппаратных средств ПК позволяет обеспечить одинаковое, не зависимое от условий, в которых оно происходит, отображение цвета для всех без исключения категорий пользователей. Эта задача решается проведением последовательности цветовых преобразований. Основой для построения таких преобразований служит ряд международных стандартов.

Поскольку операция преобразования цветов изображения требует проведения большого числа однотипных вычислений, использование параллельных вычислительных систем позволяет повысить скорость ее выполнения. При этом возможно эффективное использование как систем с разделяемой (shared), так и с распределенной (distributed) памятью. На практике возможно применение параллельных алгоритмов на всех уровнях от специализированных приложений, где размеры цветных изображений могут быть очень большими и имеются дорогие многопроцессорные вычислительные системы, до уровня рядового пользователя поскольку в качестве простейшей параллельной вычис-

лительной системы с разделяемой памятью может выступать локальная сеть.

Основы построения цветовых преобразований

Глаз человека ощущает лучистую энергию в виде света, если эта энергия имеет спектр с длинами волн от 380 до 780 нм, которым соответствуют частоты (7.89 ... 3.85) 10^{14} Гц (1 нм = 10^{-9} м). При этом цветовые ощущения зависят от спектрального состава видимого излучения (света), попадающего в глаз наблюдателя, от свойств его зрения, от условий наблюдения и от ряда других факторов. Установлено, что дно глаза содержит три вида светочувствительных приемников, у которых различны области спектральной чувствительности. Поэтому в учении о цвете принято считать цвет трехмерной величиной. Таким образом, цвет определяется тремя параметрами, которые могут быть объективными и субъективными. Объективные параметры цвета: яркость, преобладающая длина волны, чистота. Эти параметры могут быть измерены. Субъективные параметры цвета характеризуют ощущение цвета человеком и являются описательными. К ним относятся яркость (светлота) цвета, его насыщенность и цветовой тон.

Необходимость создания международной стандартной колориметрической системы назрела в первой четверти XX века. Две такие системы были разработаны и утверждены МКО в 1931 г. Это были системы RGB и XYZ. Они основаны на том принципе, что для однозначного задания цвета необходимо и достаточно иметь три соответственно выбранных основных (первичных) цвета. Их аддитивная смесь позволит получить цвет, эквивалентный данному при зрительном восприятии (визуально) по яркости и цветности. На этом, однако, работа над стандартизацией методов задания цветов не завершилась. Развитие новых отраслей человеческой деятельности, в первую очередь телевидения и компьютерной техники, потребовало создания новых цветовых моделей (UVW, sRGB и др.).

Стандартная международная колориметрическая система RGB

Основными цветами в системе RGB являются спектральные (реально существующие в спектре Солнца) цвета R (red – красный), G (green – зеленый), B (blue – синий). Под количествами основных цве-

тов, образующих данный цвет, понимают их яркости (в абсолютных или относительных единицах) в аддитивной смеси, образующей этот цвет. Единичными количествами основных цветов называются их абсолютные или относительные яркости в смеси, эквивалентной визуально по цветности стандартному равноэнергетическому белому цвету, имеющему однородный спектр. Они обозначаются круглыми скобками (R), (G), (B).

Пусть $L_{(R)}$, $L_{(G)}$, $L_{(B)}$ – абсолютные значения яркостей (кандела/кв. м) единичных количеств основных цветов в смеси, образующей белый цвет с произвольной яркостью. Тогда $L'_{(R)}$, $L'_{(G)}$, $L'_{(B)}$ – относительные значения яркостей основных цветов могут быть получены по формулам (например, относительно яркости $L_{(R)}$): $L'_{(R)} = L_{(R)} / L_{(R)} = 1$, $L'_{(G)} = L_{(G)} / L_{(R)}$, $L'_{(B)} = L_{(B)} / L_{(R)}$. Найденные экспериментально относительные значения яркостей основных цветов R, G, B в смеси, образующей стандартный равноэнергетический белый цвет, равны (относительно яркости цвета R): $L'_{(R)} = 1$, $L'_{(G)} = 4{,}5907$, $L'_{(B)} = 0{,}0601$.

Координатами цвета называются величины единичных количеств основных цветов R, G, B в смеси, визуально эквивалентной по яркости и цветности этому цвету. Координаты цвета определяют в трехмерном пространстве точку. Однако в некоторых случаях более удобно представление цвета точкой на плоскости независимо от его яркости. Координаты этой точки на некоторой плоскости (в введенной на этой плоскости системе координат) называются координатами цветности. Они определяются по формулам: $r = R / m$, $g = G / m$, $b = B / m$, где $m = R + G + B$ – цветовой модуль.

Основным недостатком цветовой системы RGB является существование отрицательных координат цвета. Они образуются при получении координат цвета спектральных (монохроматических) цветов. Для устранения этого недостатка в 1931 г. была создана стандартная международная колориметрическая система XYZ.

Стандартная международная колориметрическая система XYZ

На цветовой плоскости может быть построена так называемая диаграмма цветностей (хроматическая диаграмма). Она представляет из себя множество точек, ограниченных подковообразной кривой, соединяющей точки, соответствующие спектральным цветам и прямой, со-

ответствующей пурпурным цветам. Кривая называется спектральным локусом (локус спектральных цветов).

Для того, чтобы координаты цветности всех спектральных цветов были положительными, в качестве основных цветов этой системы были выбраны условные (реально не существующие в природе) цвета X, Y, Z, таким образом, чтобы треугольник основных цветов XYZ охватывал спектральный локус. Преобразование координат цвета из системы RGB в систему XYZ может быть записано в матричном виде следующим образом:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.490 & 0.177 & 0.000 \\ 0.310 & 0.812 & 0.010 \\ 0.200 & 0.011 & 0.990 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

Система sRGB – новый стандарт задания цветов от HP и Microsoft

Спецификация цветового пространства sRGB была разработана компаниями Microsoft и Hewlett-Packard как еще одна альтернатива для аппаратно-независимого задания цветов, прежде всего, в изображениях, предназначенных для передачи или демонстрации во всемирной компьютерной сети Интернет. sRGB не является цветовым пространством в обычном понимании этого слова, поскольку вместе с колориметрическим описанием содержит определения параметров среды просмотра изображения. Эти параметры устанавливают показатели гамма-коррекции изображения, координаты «белой точки» изображения и др. на основе их типовых значений.

Необходимость создания такого пространства возникла из-за того, что способ управления цветами, определенный спецификацией ICC (International Color Consortium) на профайлы, не всегда является приемлемым. Так, не всем пользователям необходима такая точность цветовых преобразований. Использование профайлов приводит к увеличению размера передаваемых по сети данных. Не все форматы файлов графических изображений разрешают включение профайла. Происходит снижение производительности приложений из-за необходимости проводить более сложные цветовые преобразования. Между тем стандартизация процесса задания цветов позволила бы уменьшить размеры

передаваемых цветовых данных и уменьшить время, необходимое на проведение цветовых преобразований.

Преобразование из вещественных координат цвета в пространстве XYZ в целочисленные координаты цвета (обычный способ задания цветов изображения) в пространстве sRGB происходит в несколько этапов. На первом этапе вычисляются вещественные координаты цвета в пространстве sRGB с помощью следующей матрицы:

$$\begin{pmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{pmatrix} = \begin{pmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

Значения, не попадающие в отрезок [0, 1], устанавливаются соответственно либо в 0, либо в 1. Затем эти вещественные координаты цвета преобразуются в нелинейные следующими соотношениями: $R'_{sRGB} = 12.92\ R_{sRGB}$, $G'_{sRGB} = 12.92\ G_{sRGB}$, $B'_{sRGB} = 12.92\ B_{sRGB}$, если R'_{sRGB}, G'_{sRGB}, $B'_{sRGB} \le 0.00304$ и $R'_{sRGB} = 1.055\ R_{sRGB}^{(1.0/2.4)} - 0.055$, $G'_{sRGB} = 1.055\ G_{sRGB}^{(1.0/2.4)} - 0.055$, $B'_{sRGB} = 1.055\ B_{sRGB}^{!(1.0/2.4)} - 0.055$, если R'_{sRGB}, G'_{sRGB}, $B'_{sRGB} > 0.00304$. Эти соотношения аппроксимируют функцию гамма-коррекции с показателем степени 2.2. Это сделано для облегчения построения в дальнейшем обратного преобразования. Спецификация sRGB объясняет причины выбора такого показателя степени [3]. После этого нелинейные вещественные значения $sR'G'B'$ преобразуются в целочисленные, которыми обычно задается цвет в компьютерных системах: $R_{8bit} = ((WDC - KDC)\ R'_{sRGB}) + KDC$, $G_{8bit} = ((WDC - KDC)\ G'_{sRGB}) + KDC$, $B_{8bit} = ((WDC - KDC)\ B'_{sRGB}) + KDC$, где WDC – двоичный уровень белого (White Digital Count), а BDC – двоичный уровень черного (Black Digital Count). Текущая спецификация sRGB предлагает использовать соответственно значения 255 и 0 в качестве этих параметров при кодировании цвета изображения 24 битным числом. Тогда финальное преобразование будет иметь вид: $R_{8bit} = 255.0\ R'_{sRGB}$, $G_{8bit} = 255.0\ G'_{sRGB}$, $B_{8bit} = 255.0\ B'_{sRGB}$.

Некоторые вопросы распараллеливания алгоритмов преобразования цветов между различными цветовыми пространствами

Как показано выше, при преобразовании цветов широко используются операции линейной алгебры. Это, прежде всего, операции перемножения матриц и умножения вектора на матрицу. Распараллели-

вание алгоритмов выполняющих основные операции линейной алгебры, обсуждается в большом количестве литературы, посвященной параллельному программированию. Одним из наиболее эффективных, с теоретической точки зрения, параллельных алгоритмов перемножения двух квадратных матриц порядка n можно считать следующий, основанный на идее, изложенной в [2].

1. В узлах трехмерной решетки процессоров ($n \times n \times n$) вычисляются произведения элементов перемножаемых матриц B и C следующим образом: если (i, j, k) координаты процессора в решетке, то в нем происходит вычисление произведения элементов b_{ik} и c_{kj}.

2. После этого процессоры решетки, начиная с уровня k = 1, последовательно аккумулируют результаты предыдущего этапа следующим образом: пусть a^k_{ij} – результат предыдущего этапа алгоритма в процессоре с координатами (i, j, k), новый результат вычисляется как $a^k_{ij} = a^{(k-1)}_{ij} + a^k_{ij}$, далее происходит передача результата на следующий уровень (k+1).

В результате на уровне k = n будет получена матрица A = BC. Эту же идею можно использовать для построения эффективного параллельного алгоритма умножения вектора на матрицу. В этом случае можно разместить в узлах прямоугольной решетки процессоров ($n \times n$) произведение элементов a_{ij} и x_j перемножаемых матрицы и вектора, а вычисление результата проводить аналогичным способом, аккумулируя результирующий вектор на уровне j = n.

С практической точки зрения представляется удобным распараллеливать алгоритмы преобразования цветов покоординатно, выделяя для операций над каждой координатой свой процессор. Такой подход позволяет увеличить скорость выполнения преобразования, не предъявляя при этом особых требований к мощности параллельной вычислительной системы (числу процессоров) и не требуя значительных усилий на разработку параллельного алгоритма. Для исследования ускорения выполнения операции преобразования цветов изображения в этом случае был проведен ряд вычислительных экспериментов. Была написана программа на языке C в среде MS Visual Studio C++, реализующая параллельные вычисления с помощью функций стандарта MPI (Message Passing Interface – стандарт для написания параллельных алгоритмов на системах с распределенной памятью). Использовалась свободно доступная реализация стандарта MPI – MPICH для локальной сети на базе Windows NT/2000 (разработана в Argonne National

Laboratory совместно с Mississippi State University). Результаты экспериментов приведены в таблице ниже. В столбце I указано время работы параллельного алгоритма (в секундах), в столбце II – соответствующего последовательного и в III – отношение двух предыдущих величин.

Размеры изображения	Результаты работы на процессоре 0 (Intel Pentium III 550MHz)			Результаты работы на процессоре 1 (Intel Pentium II 350MHz)			Результаты работы на процессоре 2 (Intel Pentium III 550MHz)		
	I	II	III	I	II	III	I	II	III
100000×10000	32	51	1.59	46	72	1.56	29	47	1.62
200000×10000	63	97	1.53	91	146	1.6	58	94	1.62
300000×10000	91	143	1.57	136	218	1.6	87	140	1.6
400000×10000	119	192	1.61	181	290	1.6	116	186	1.6

Литература

1. Новаковский С.В. Цвет на экране телевизора. Основы телевизионной колориметрии. М., 1997.
2. Воеводин В.В. Математические модели и методы в параллельных процессах. М., 1986.
3. Stokes Michael, Anderson Matthew, Chandrasekar Srinivasan, Motta Ricardo. A Standard Default Color Space for the Internet – sRGB, http://www.color.org.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ФОРМАЛЬНОЙ МОДЕЛИ ВЗАИМОДЕЙСТВУЮЩИХ ПОСЛЕДОВАТЕЛЬНЫХ ПРОЦЕССОВ

С.Б. Попов

Институт систем обработки изображений РАН, Самара

Введение

Во многих практически важных случаях процесс вычислений представляется в виде неоднородной программной сети, в каждом узле которой программа обработки существенно отличается как по количеству внешних связей, так и по характеру вычислений. Модель, необхо-

димая для исследования таких программных сетей, должна отвечать требованиям адекватности и конструктивности, то есть должна описывать все стороны функционирования программной системы, все аспекты взаимодействия программ, составляющих её, и предлагать, по возможности, способы реализации системы.

В качестве формальной модели процессов обработки информации в многопроцессорных вычислительных системах предлагается взять математическую теорию взаимодействующих последовательных процессов (алгебра CSP – communicating sequential processes) Ч. Хоара[1]. Основными позитивными чертами этой теории являются:

- развитое понятие процесса, учитывающее особенности синхронизации и параллельного выполнения процессов, включая модель обмена данными между различными процессами;
- удобная техника работы с протоколами (последовательностью событий) процессов, позволяющая легко выделять и анализировать результаты работы модели;
- возможность эквивалентного отображения теории на языки высокого уровня.

Основная идея теории CSP заключается в том, что любую систему можно разложить на параллельно работающие подсистемы, взаимодействующие как друг с другом, так и со своим системным окружением. Основным объектом рассмотрения является *процесс*, описываемый как последовательность ограниченного набора событий, выбранных для его описания. Данное множество событий называется *алфавитом* процесса. Алфавит считается постоянным, заранее определенным свойством объекта. При выборе алфавита проводят некоторое упрощение: не рассматриваются многие действия и свойства, представляющие меньший интерес.

Принципы реализации формальной модели взаимодействующих последовательных процессов

В работе Хоара [1] рассматриваются вопросы реализации рассматриваемой формальной модели на языке программирования ЛИСП. В основе такой реализации лежит функциональный подход. Согласно этому подходу, каждый процесс представляется в виде функции, аргументы которой – события из алфавита этого процесса, а результат функции – другая функция, определяющая последующее поведение

процесса. Для описания процесса, определяемого рекурсивно, используется рекурсивный вызов функции, где в качестве аргумента, описывающего дальнейшее поведение процесса, используется сама эта функция. Операции последовательной и параллельной композиции также определяются как функции, аргументами которой являются объединяемые процессы. ЛИСП-реализация модели позволяет проводить доказательство беступиковости сложного составного процесса, проверку соответствия модели требуемой спецификации. Однако такую реализацию очень трудно модифицировать в направлении, позволяющем учесть фактор времени.

В основе предлагаемой программной реализации формальной модели лежит объектно-ориентированный подход, который прямо соотносится с основными понятиями CSP теории. Основной единицей моделирующей программы является объект *событие*. *Процесс* – это также объект, который содержит упорядоченный список событий, в которых он участвует. Рассмотрим основные моменты предлагаемого алгоритма моделирования CSP-алгебры. Функционирование возможно в двух вариантах: верификация процесса на предмет отсутствия тупиков и моделирование процесса обработки со сбором статистики времени исполнения.

В режиме верификации процесс, получая управление, последовательно запускает собственные события. Простое событие устанавливает следующее в списке событие в качестве текущего и передает управление процессу. Процесс проверяет свое состояние и, если он не приостановлен, запускает текущее событие, если процесс приостановлен, то управление передается на верхний уровень. При исследовании только одного изолированного процесса моделирование на этом заканчивается. В случае параллельной композиции нескольких процессов на верхнем уровне формируется список процессов, исполняемых параллельно. При возврате управления на этот уровень запускается любой следующий процесс, готовый к исполнению. Если готовых к исполнению процессов нет, причем процессы не завершили свою работу (состояние «Завершен» отлично от состояния «Приостановлен»), значит система пришла в тупиковое состояние – дедлок. Состояние процесса изменяют события из класса взаимодействий. Событие-взаимодействие присутствует во всех содержащих его процессах, причем, *происходит* оно в них *одновременно*. Если какой либо процесс

первым переходит к такому событию, то он приостанавливается до тех пор, пока остальные, участвующие во взаимодействии процессы не дойдут до момента взаимодействия. Как только *все* процессы, участвующие во взаимодействии, доживут до него, событие происходит в каждом процессе, т.е. состояние «Приостановлен» сбрасывается, и последующее событие устанавливается в качестве текущего во всех процессах, а управление передается любому из взаимодействующих процессов, который и выполняется до наступления следующего события-взаимодействия, либо до своего завершения. Последнее событие из списка процесса устанавливает состояние «Завершен» и передает управление на верхний уровень. Далее опять запускается любой следующий процесс, готовый к исполнению. Верификация завершается успешно, если все процессы, составляющие исследуемую потоковую сеть, перешли в состояние «Завершен».

В режиме моделирования берется за основу квантовая модель времени, то есть наименьшим промежутком времени, которым оперирует моделирующая программа, является квант. Все события, происходящие в системе, имеют длительность, кратную выбранному кванту времени. В объект *событие* добавляется параметр, определяющий количество квантов времени, потребляемых событием до своего завершения. Процесс моделирования несколько отличен от верификации. На верхнем уровне формируется циклический список процессов, составляющих исследуемую сеть. Затем программа начинает последовательно генерировать временные кванты и передавать их очередному процессу. Процесс получает один квант и либо поглощает его, либо возвращает неиспользованным, в последнем случае этот неиспользованный квант передается для обработки следующему по списку процессу. Если процесс использовал выделенный ему квант времени, указатель в очереди смещается на следующий процесс, которому и будет передан очередной квант для обработки. На уровне процесса обработка кванта выполняется следующим образом: если процесс приостановлен, то квант возвращается неиспользованным, если процесс находится в рабочем состоянии, то квант передается для обработки в текущее событие процесса. Событие потребляет необходимое для завершения количество квантов и при обработке последнего кванта устанавливает следующее в списке процесса событие в качестве текущего. Событие-взаимодействие представляется в виде двух событий: собственно со-

бытия-взаимодействия, обрабатываемого аналогично режиму верификации, и простого события, следующего за взаимодействием и определяющего его длительность. После обработки последнего события процесс удаляется из циклического списка, описывающего сеть. На каждом шаге, при генерации очередного кванта проверяется количество приостановленных процессов, если это количество равно числу процессов в списке, то значит сеть находится в заблокированном состоянии. Когда список становится пустым, процесс моделирования завершается, количество сгенерированных квантов определяет общее время обработки. В процессе моделирования собирается информация о времени выполнения каждого процесса и его простоя, времени, которое потребляют выбранные события, и т.д.

Помимо простых событий и событий-взаимодействий для моделирования процессов в системе используются события, реализующие оператор присваивания, условный оператор и цикл. Предполагается, что эти служебные события не потребляют временной квант. Условный оператор и цикл просто переустанавливают указатель текущего события в соответствии со значением логической функции. Оператор присваивания изменяет значение некоторой переменной процесса и устанавливает следующее событие как текущее. Если необходимо учесть время выполнения таких конструкций, то в список событий процесса за служебным событием (или перед ним) вставляется простое событие соответствующей длительности.

Данная квантовая модель естественным образом обобщается на случай выполнения потоковой сети на нескольких процессорах. При этом список процессов, составляющих такую распределенную сеть, подразделяется на несколько подсписков, соответствующих конкретному компьютеру. Внутри каждого подсписка независимо формируется собственный квант времени, который потребляют составляющие его процессы.

Программные средства верификации и моделирования взаимодействующих последовательных процессов

В соответствии с алгоритмом функционирования формальной модели задач обработки информации, разработаны программные средства верификации и моделирования программных сетей взаимодейст-

вующих последовательных процессов, описываемых в соответствии с правилами алгебры Хоара.

Язык программирования – Java.

Основу системы составляет библиотека классов, реализующих различные типы событий, процессов и алгоритмов передачи данных. Библиотека моделей программ обработки информации содержит определения сложных процессов, моделирующих наиболее типичные задачи обработки информации, через список внутренних событий и событий-взаимодействий. Для использования программных средств моделирования в интерактивном режиме служит подсистема организации диалога, позволяющая задавать описание процессов через список составляющих их событий, сохранять эти описания в библиотеке моделей программ обработки информации для дальнейшего использования, составлять программные сети, объединяя процессы через события-взаимодействия.

Подсистема моделирования запускает процесс, определяющий исследуемую сеть, проводит трассировку процесса моделирования, сбор информации о динамике заполненности буферов программных каналов передачи данных и передает собранную информацию в подсистему организации диалога.

Рассмотрим структуру классов моделирующего комплекса. Каждый новый производный класс обладает как свойствами, унаследованными им от родительского класса, так и приобретает новые, характерные только для данного класса.

Класс *PBase* – это абстрактный базовый класс, который является предшественником всех других классов. *PBase* определяет общее для них поведение через методы, позволяющие классам создавать, изменять и удалять внутренние структуры данных.

Класс *PEvent* – базовый класс, определяющий поведение простого события какого-либо процесса. В качестве параметра *PEvent* содержит ссылку на следующее в списке событие процесса. При запуске объекта из данного класса событие всегда успешно завершается и устанавливает следующее из списка событий процесса. Производными от этого класса являются классы, описывающие более сложные события.

Класс *PEventDo* определяет конструкцию бесконечного цикла. Параметрами являются ссылка на начало списка событий данного цикла и ссылка на событие, которому передается управление при необходи-

мости выхода из цикла. При запуске объекта данного класса управление передается текущему событию из внутреннего списка. Внутренний список является циклическим.

Класс *PEventWhile* является производным от предыдущего класса и определяет конструкцию обычного цикла.

Класс *PEventIf* определяет конструкцию условного оператора. Параметрами класса являются ссылка на переменную, определяющую условие, ссылки на события, выполняющиеся соответственно при отличии значения условия от нуля (ветвь – *then*) и при равенстве его нулю (ветвь – *else*). При запуске объекта данного класса проверяется значение условия, и в соответствии с этим устанавливается текущее событие процесса.

Классы *PEventIn* и *PEventOut* определяют события-взаимодействия.

Класс *PProcess* – базовый класс, определяющий поведение простого процесса. В качестве параметра *PProcess* содержит ссылку на внутренний упорядоченный список событий процесса. Объект из данного класса событий может находится в одном из трех состояний: «Приостановлен», «В работе» и «Завершен», которые изменяются его внутренними событиями.

Производными от этого класса являются классы, описывающие укрупненные процессы, являющиеся комбинацией нескольких простых процессов. В качестве базового для таких классов используется абстрактный класс *PGroupProc*, который отличается от своего предшественника тем, что вместо списка своих событий содержит список процессов.

Класс *PParalProc* определяет параллельную композицию нескольких процессов. Класс *PSequenProc* задает последовательный характер выполнения содержащихся в нем процессов. Класс *PNetProc* моделирует параллельное выполнение процессов на нескольких процессорах, путем формирования нескольких внутренних списков процессов.

Класс *PPipe* обеспечивает работу событий-взаимодействий, моделируя как буферизованную, так и небуферизованную передачу данных между процессами.

Библиотека моделей программ обработки информации содержит уже сформированные объекты процессов, моделирующих наиболее

типичные задачи обработки информации. Набор базовых моделей процессов обработки информации включает в себя:

- модель программы-источника одного потока данных;
- модель программы–преобразователя с произвольной задержкой данных;
- модель программы блочной обработки с произвольным размером блока;
- модель программы дублирования входного потока данных в два выходных;
- модель программы обработки двух потоков данных с выводом результата в один поток ;
- модель программы обработки двух потоков данных.

Подсистема организации диалога работает в двух режимах: создание и редактирование моделей программ обработки и составление программных сетей, реализующих параллельную обработку.

В *режиме создания и редактирования моделей* программ обработки информации пользователь может определить как простой процесс, который имеет список событий, совпадающий с иерархией классов, производных от класса *PEvent*, так и сложный процесс, являющийся последовательной комбинацией нескольких простых. Результатом является модель последовательной программы обработки информации, у которой внешние события-взаимодействия выступают как формальные параметры.

В *режиме составления и редактирования программных сетей* пользователь задает параллельную композицию используемых моделей программ путем их перечисления с явным именованием каналов передачи данных (событий-взаимодействий). Далее выбирается способ реализации каналов передачи данных (буферизованная передача или небуферизованная) и запускается процесс моделирования исполнения сформированной сети программ обработки информации либо на одном компьютере (в режиме разделения времени), либо на нескольких процессорах (в режиме параллельного исполнения). В случае выбора буферизованной реализации каналов передачи данных дополнительно указывается размер буфера каждого канала (возможно установить одинаковый размер буфера для всех каналов). В результате моделирования дается сообщение о реализуемости исследуемой сети программ

обработки информации: успешное завершение или состояние взаимной блокировки. В случае успешного завершения пользователь может просмотреть информацию о пиковых значениях заполняемости буферов программных каналов передачи данных и, при необходимости, трассировку процесса моделирования. В случае блокировки исследуемой потоковой сети пользователю выводится информация о заполнении буферов на момент блокировки и распечатывается трассировка процесса моделирования.

Литература

1. Хоар Ч. Э. Р. Взаимодействующие последовательные процессы. М: Мир, 1989. 264с.

СТАТИЧЕСКОЕ ПОСТРОЕНИЕ РАСПИСАНИЯ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

А.Н. Сазонов

Московский Государственный университет им. М.В. Ломоносова

1. Составление расписаний

1.1. Проблемы составления расписаний

Загрузка процессоров, распределение задач между процессорами, накладные расходы на пересылку сообщений являются главными проблемами, возникающими в процессе составления расписаний.

1.1.1. Загрузка процессоров

В процессе работы распределенной вычислительной системы, из-за ограниченности скоростей связи, в общем случае будут происходить простои процессоров. На самом деле избежать этих простоев можно, только специальным образом создав алгоритм так, что данные всегда приходили бы с опережением. Ясно, что далеко не во всех случаях это возможно.

Таким образом, задача составления расписаний, в частности, решает проблему увеличения нагрузки на вычислительные устройства. Ясно, что точное составление оптимального расписания совершенно необязательно, т.к. в процессе вычислений будут неизбежно прояв-

ляться недетерминированные отклонения в работе системы, связанные с невозможностью точно оценить время выполнения каждой задачи, предугадать все столкновения данных и задержки в буферах.

1.1.2. Назначение задач на процессоры

В процессе составления расписания выполнения задач одной из главных проблем является распределение задач на конкретные вычислительные устройства. Если задачи таким образом распределены так, что все необходимые данные находятся на процессоре, исполняющем каждую задачу, тогда процессоры системы лишь читают локальные данные, обеспечивая максимальный параллелизм. В реальных приложениях такое распределение встречается весьма редко, т.к. требуется распараллелить задачи таким образом, чтобы они были полностью независимы друг от друга. Ясно, что такая ситуация вряд ли возможна в реальных задачах.

Но если процессор не имеет нужных ему данных, требуется пересылка информации от процессора, который содержит нужные данные. Только после такой пересылки можно производить вычисления. Плохое распределение задач между процессорами вызывает всевозможные задержки данных. Таким образом, от распараллеливающей системы требуется, чтобы она составляла расписание задач и распределяла их между вычислителями так, чтобы минимизировать задержки. Следовательно, требуется обеспечить максимальное упреждение данных.

1.1.3. Накладные расходы на пересылку информации

Как было описано выше, если процессор не обладает нужными ему данными, это вызывает задержку выполнения задач. Время, требующееся на пересылку требуемой информации, называется накладными расходами на пересылки. Для сокращения таких расходов требуется оптимально распределять задачи между процессорами и составлять расписание их выполнения.

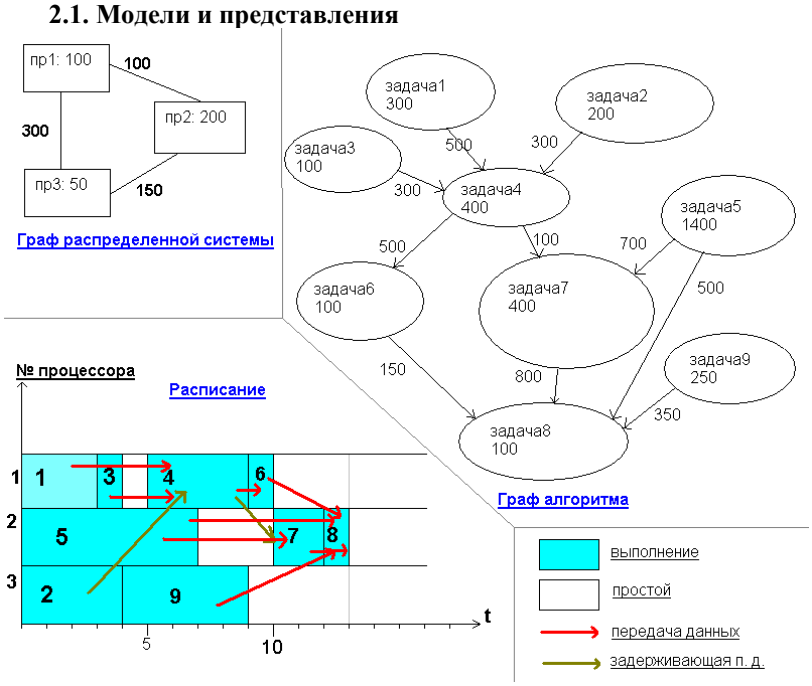
1.1.4. Накладные расходы на составление расписаний

Алгоритм назначения задач на параллельную систему и упорядочивания их во времени требует значительных вычислительных ресурсов, особенно, если требуется получить весьма точное решение этой оптимизационной задачи. Это время называется накладными расходами на распараллеливание. Системы, в которых такое распараллеливание происходит динамически, должны работать таким образом, чтобы

не вызывать задержки выполнения задач. Хотя для статического распараллеливания, когда расписание составляется еще до выполнения параллельной системы, имеется значительно большее количество времени, очевидно, оно должно быть много меньше самого выигрыша от такого распределения, иначе он будет просто бессмысленным.

2. Постановка задачи

2.1. Модели и представления



2.1.1. Модель программы

Распараллеливаемая программа представляется как взвешенный направленный ациклический граф G(V, E, μ, ν), где $V=\{T_1, T_2, \ldots, T_n\}$ – исходное частично упорядоченное множество задач, $E=\{e_{ij}\}=(T_i,T_j)\}$ – набор дуг, описывающих передачу информации между задачами и указывающих на информационные зависимости между ними. Вес задачи $\mu(T_i)$ соответствует времени ее выполнения на процессоре с единичным быстродействием, а вес дуги $\nu(e_{ij}\})$ – объему передаваемой информации.

Если $e_{ij}=(T_i, T_j)$ принадлежит E, это означает, что T_j информационно зависит от T_i. Тем самым выполнение T_j может начаться только после того, как задача T_i будет завершена, и вся информация, соответствующая данной дуге, будет передана с процессора, исполняющего задачу T_i, на процессор исполняющий T_j.

Для множества предшественников задачи T_i введем обозначение $pred(T_i)$.

Во время выполнения каждая из задач исполняется одним процессором и не перемещается на другие процессоры. После выполнения вычислений, задача может пересылать информацию другим, зависимым от нее, задачам. Задача не может дважды передавать информацию, производя вычисления в промежутке между пересылками – в этом случае она должна быть разбита на две.

2.1.2. Модель распределенной системы

Распределенная система $M=\{P, \eta, C, \theta\}$ представлена как множество процессоров $P=\{P_1, P_2, \ldots, P_m\}$, имеющих быстродействие $\eta(P_i)$ вычислений в единицу времени, некоторые из которых соединены каналами $C=\{c_{ij}=(P_i, P_j)\}$ с быстродействием $\theta(c_l)$ информации в единицу времени. Каналы образуют межпроцессорную коммуникационную сеть. Таким образом, может быть задана гетерогенная система.

В зависимости от способа передачи информации между процессами, аппаратного обеспечения передачи информации, типа ОС и др. причин, выбирается функция $send(P_i, P_j, v)$, которая равна количеству единиц времени, необходимых для передачи v единиц информации с процессора P_i на процессор P_j. В одном из простых случаев, $send(P_i,P_j,v)=v(c_{i,p1}+c_{p1,p2}+\ldots+c_{pk,j})$, где P_i, P_{p1}, P_{p2}, … ,P_j есть некоторый (обычно, минимальный) путь из процессора P_i в процессор P_j. Для кластеров рабочих станций в простейшем случае можно использовать такой вид функции send: $send(P_i,P_j,v)= v*c_{i,j}$, что оправдано для реальных систем вследствие наличия большого числа практически случайных факторов, влияющих в этом случае на производительность среды передачи данных.

Однако наиболее важным свойством предлагаемого в данной работе алгоритма является его независимость от способа представления данной функции. Если точности, задаваемой таким представлением функции send недостаточно, можно использовать любое другое представление функции, в общем случае зависящее не только от объема

пересылаемой информации, но и от нагрузки на сеть. Более того, величина send(P_i, P_j,v), в общем случае, может зависеть от времени. Например, имея полную информацию о поведении распределенной системы, способах маршрутизации данных, алгоритмах буферизации данных и разрешения коллизий, можно построить моделирующую функцию send, наиболее точно определяющую время передачи информации. Следует отметить, что функция send активно используется при вычислении функции качества, поэтому чрезмерное ее усложнение нежелательно, т.к. это резко отрицательно отразится на скорости работы программы.

2.1.3. Представление расписания

Расписание выполнения задач представляется в виде списка $S=\{s_i=(T_i, P_i, \tau_i)\}$, упорядоченного по τ_i, где P_i – это процессор, исполняющий задачу T_i, начиная с момента времени τ_i после старта распределенной программы. Задача T_i может исполняться, только если все ее предшественники по направленному графу алгоритма завершили свое выполнение, и все данные, необходимые для ее выполнения, получены процессором P_i.

Таким образом,

$\forall T_i, \forall T_j\ e_{ij}=(T_i, T_j) \in E \Rightarrow \tau_i + \mu(T_i) + send(P_i, P_j, v(e_{ij})) \le \tau_j$.

Если данный предикат верен для некоторого расписания S, говорят, что S – допустимо. Алгоритм не должен выдавать в качестве своего результата недопустимые расписания.

Длина расписания $length(S)=\max\{\forall\ s_i \in S\}\tau_j + \mu(T_i)$ есть общее время выполнения распределенной программы на данной многопроцессорной системе, при выполнении в порядке, задаваемым расписанием S.

3. Решение задачи построения расписания

Для решения задачи построения расписания был использован генетический алгоритм. Далее изложены основные решения, примененные при его создании.

3.1. Схема кодирования

ГА-строка $G=\{G_1, G_2, \dots G_n\}$ состоит из символов $G_i=(P_i, R_i)$, где P_i – это процессор, на котором будет исполняться задача T_i, а R_i – приоритет задачи. При составлении расписания по ГА-строке, значе-

ние R_i используется тогда, когда в один и тот же момент времени на исполнение процессором P_i претендует более одной задачи.

3.2. Функция качества

В реализованном алгоритме, значением функции качества была выбрана длина соответствующего расписания. Вместо нее может быть выбрана любая другая функция, например, средняя загрузка системы, ускорение относительно последовательного выполнения программы, стоимость счета распределенной задачи при оплате процессорного времени и др.

Таким образом, каждый раз, когда вычисляется функция качества, генетический алгоритм, используя закодированную ГА-строку, строит по ней расписание выполнения задач.

Для построения расписания используется дискретный счетчик «текущего» времени, который принимает те значения, при которых освобождается очередной процессор. Также используются 3 пула задач: 1) уже распределенных на процессоры, 2) еще не распределенных, но уже (в текущий момент времени) готовых к выполнению и 3) не готовых к выполнению. Сначала выбираются все задачи, не имеющие внешних входных данных, и по возможности (с использованием приоритетов в случае коллизий) распределяются на «свои» процессоры. Затем увеличивается счетчик, перестраиваются пулы 1,2,3 и по возможности (с использованием приоритетов в случае коллизий), готовые задачи распределяются на «свои» свободные процессоры. Процесс заканчивается в случае отсутствия направленных циклов в графе алгоритма, когда все пулы, кроме 1, пусты. В случае некорректного графа алгоритма все процессоры освободятся, но ни одной задачи в пуле 2 не будет, при том, что пул 3 будет не пуст. Этим можно воспользоваться, чтобы определять корректность расписания. Также при построении расписания активно используется функция send (на самом деле время старта задачи с максимальным приоритетом на «ее» процессоре после его освобождения откладывается до прихода всех данных). Последний факт не означает, что в случае больших объемов передаваемых данных алгоритм обязательно будет строить неоптимальные расписания по генетическому коду, т.к. существует система приоритетов, изменение которых в коде могут привести к перестановке задач в расписании. Проблема не является критичной, т.к. при наличии хотя бы 2-х процессо-

ров в системе перемещение конфликтующих задач с процессора на процессор может дать оптимальный результат.

3.3. Использованная модель генетических вычислений

В силу сравнительно высокой вычислительной сложности функции качества, в данном алгоритме не может быть применен алгоритм, полностью заменяющий следующее поколение индивидов и перевычисляющий все значения функции качества. Использован параметризованный ГА, при котором на каждом шаге порождается случайное число особей [в среднем (AVG_MUTATE_PERCENT+ AVG_RECOMBINATE_PERCENT)*N_CREATURES] потомков, полученных путем скрещивания или мутации родителей (родителя). Константы здесь и далее берутся из ini-файла при запуске программы. При этом сначала порождаются мутантные особи, и затем они сразу же могут стать родителями рекомбинантных особей. Затем из всей пополненной популяции выбирается не более MAX_CREATURES особей, при этом отбираются лучшие, но с учетом случайного «штрафа». Средняя величина «штрафа» задается константой RANDOM_SELECTING_PERCENT в ini-файле.

3.4. Оператор мутации

Реализованный оператор мутации имеет несколько «внешних» – прописываемых в ini-файле параметров: у особи мутируют в среднем AVG_GENS_MUTATE_PERCENT процентов генов, причем процент мутировавших значений P_i в среднем равен PROCESSOR_MUTATE_PERCENT, а процент мутировавших значений R_i в среднем равен PRIORITY_MUTATE_PERCENT.

3.5. Оператор рекомбинации

Реализован параметризованный оператор рекомбинации. Количество точек скрещивания в среднем равно AVG_RECOMBINATE_POINTS, однако указанных точек всегда не менее одной. Рекомбинируют гены целиком, т.е. особи обмениваются значениями $G_i=(P_i, R_i)$ целиком, а не P_i или R_i в отдельности. Точки скрещивания распределяются по хромосоме с помощью датчика равномерно распределенных случайных чисел.

3.6. Старт и окончание поиска

При старте программы случайным образом, используя несколько различных типов генерации кода, порождаются

[START_CREATURES_PERCENT*MAX_CREATURES/100] особей. Далее размер популяции может варьироваться случайным образом или в зависимости от параметров списка значений функции качества особей.

Процесс поиска заканчивается, когда значение функции качества изменяется незначительно (MIN_DIFFERENCE_100/100) за STOP_IDLE_ITERATIONS или время простоя всех процессоров невелико по сравнению с общим временем выполнения. Кроме этого, ограничивается число поколений (MAX_CYCLES). Результатом работы является расписание, имеющее наименьшую длину для данной популяции, т.е. выбирается лучшая особь.

В программе предусмотрена возможность расширения алгоритма при помощи т. н. островной схемы, однако ее реализация резко увеличила бы количество вычислений функции качества, что крайне нежелательно ввиду ее сложности.

4. Тестовые данные (тест генетического алгоритма)

Получено расписание:
Возраст популяции: 67
Время выполнения расписания: 222.000000

Время старта	задача	процессор
0.000	20	1
0.000	11	2
0.000	18	3
0.000	4	4
0.000	1	5
4.000	14	1
10.000	2	5
30.000	3	5
74.000	12	3
47.000	17	2
40.000	5	4
91.000	6	1
73.000	16	5
90.000	19	4
91.000	9	4
192.000	13	2

151.000	7	1
191.000	10	4
208.000	15	4
221.000	8	1

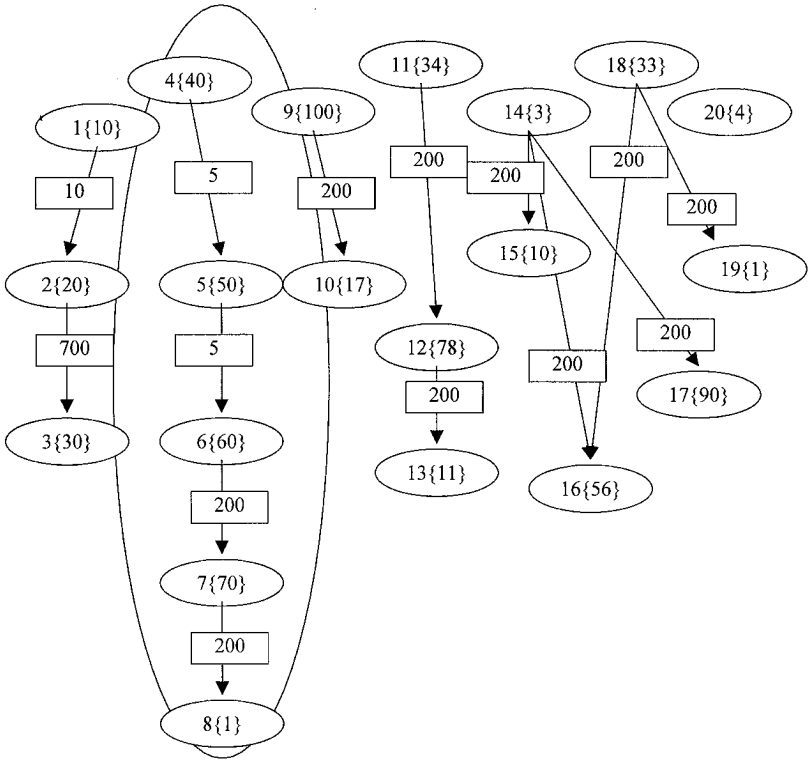


Рис.1. Граф тестовой параллельной программы. Параллельная система гомогенна – 8 процессоров производительностью по 1, связанных друг с другом линиями пропускной способностью 5 в случае различных процессоров и с нулевым временем пересылки между одним и тем же процессором

Максимальное время выполнения последовательности задач «без пересылок» – в обведенной цепочке. Оно равно 221 (если всю цепочку

выполнить на одном процессоре), что практически и сделано ГА: осталась только одна пересылка, потребовавшая единицу времени. Кроме того, в силу достаточно большого количества пересылок задействовано всего 5 процессоров из 8 и при этом расписание практически оптимально. Аналогичная ситуация наблюдалась при правильном подборе параметров ГА и на других тестах.

Таблица

Параметры данного теста

Кол-во особей	50
% мутации	60
% рекомбинации	30
% мутации P_i	10
% мутации R_i	10
Средний % мутирующих генов	50
Среднее кол-во точек скрещивания	1
Мин. допустимая разница значений	0.01
Кол-во «неулучшенных» популяций	32
Макс. размер «штрафа», %	150

Таким образом, реально решение было найдено за 35 циклов (67–32=35), а потом 32 цикла ГА не смог улучшить значение более чем на 0.01 и остановился. Лишних циклов можно избежать, включив систему контроля времен простоя процессора.

В соответствии со свойствами генетических алгоритмов, данный алгоритм также имеет свойство улучшать априори найденное решение, т. е. если решение примерно известно и подано на вход алгоритма как одна или несколько особей в начальной популяции, алгоритм быстрее (в среднем) надет нужное решение.

Литература

1. *Теория расписаний и вычислительные машины* / Под ред. Э.Г Коффмана).
2. Grant K. *An introduction to genetic algorithms* //C/C++ Users Journal, March 1995. P. 45–58.
3. Fogel David B. *Using Evolutionary Programming to Schedule Tasks on a Suite of Heterogeneous Computers* // Computers & Operations research, Vol. 23:6. P. 527–534.

РАЗРАБОТКА ИНСТРУМЕНТАЛЬНОЙ СИСТЕМЫ ДЛЯ ДИНАМИЧЕСКОЙ БАЛАНСИРОВКИ ЗАГРУЗКИ ПРОЦЕССОРОВ И КАНАЛОВ СВЯЗИ

А. Сальников

Московский Государственный университет им. М.В. Ломоносова

На основе графа алгоритма строится параллельная программа, которая будет выполняться на многопроцессорной системе, причём узлы графа алгоритма выбираются для выполнения на процессоре динамически. Разрабатываемые программные утилиты призваны частично снизить потерю производительности при переносе параллельной программы на другую платформу. Предлагаемый инструмент поддержки параллельного программирования возьмёт на себя часть функций операционной системы и самостоятельно позаботится о выполнении параллельной программы на конкретной многопроцессорной системе с учётом информационной структуры задачи, самостоятельно решая задачу составления расписания работы многопроцессорной системы, а также задачу балансировки загрузки процессоров и загрузки каналов связи.

В современном мире существует множество задач, решение которых требует большого количества ресурсов. В качестве примера можно привести задачу моделирования климата на планете, или задачу построения больших молекул, задачу построения реалистических изображений и компьютерных фильмов. Список таких задач можно продолжать, по-видимому, почти бесконечно. Практика показывает, что использование однопроцессорной системы даже очень высокой производительности для таких задач может быть весьма затруднительным. Однако большинство из этих «тяжёлых» задач можно свести к набору более мелких подзадач, зависимости по данным между которыми можно представить в виде макро – графа. В качестве простого примера можно привести многослойный персептрон. Задача вычисления выхода нейронной сети сводится к задаче вычисления выходов каждого из нейронов (рис. 1).

Описание графа программы

Граф алгоритма полностью соответствует парадигме графа зависимости по данным в ярусно-параллельной форме, определённым Вое-

водиным В.В. [Информационная структура алгоритмов. Издательство Московского университета, 1997. С. 6–35.]. Алгоритм решаемой задачи представляется в виде ориентированного графа, где в вершинах сосредоточенны действия над данными, а рёбра символизируют зависимость по данным. При этом дуга направлена от вершины источника данных к вершине, принимающей данные. Следует отметить, что данные операции вовсе не обязательно являются простейшими элементарными операциями, и инструментальная система в первую очередь предназначена для работы в случае, когда вершины графа алгоритма являются полновесными, тяжёлыми операциями, возможно, даже целыми программами, для которых зачастую бывает сложно оценить время их исполнения на одном процессоре. В графе не должно быть ориентированных циклов. Граф выстраивается в ярусную форму, где каждый уровень получает данные от предыдущего уровня. Вершины графа на каждом уровне независимы между собой. Вершины графа с большего уровня используют данные, полученные в процессе выполнения действий, определённых в вершине графа с меньшего уровня.

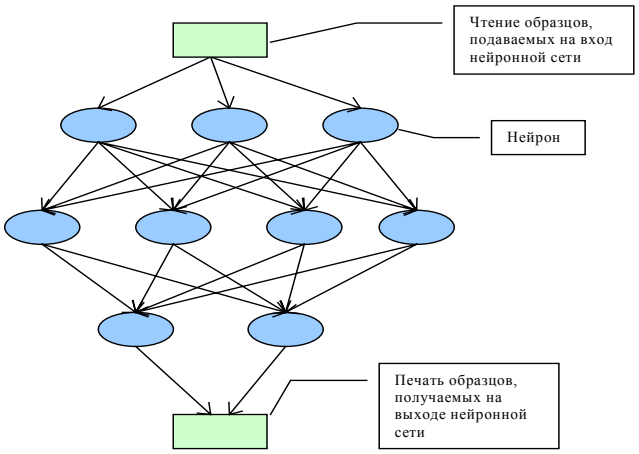


Рис. 1. Нейронная сеть

Узлы графа алгоритма превращаются в функции, и затем каждая функция может быть выполнена на любом процессоре многопроцес-

сорной машины. Узлы графа алгоритма только в процессе выполнения узнают, где находится их окружение. Данное свойство делает возможным гибкое управление процессом загрузки, не привязанное жёстко к расписанию.

Каждый узел графа выполняется в локальной памяти процессора многопроцессорной машины, причём заранее неизвестно какого, поэтому состояние памяти, изменённое одним узлом графа алгоритма на конкретном процессоре, в общем случае не удастся использовать другим каким-либо конкретным узлом. Если необходимо получить данные, изменённые другим узлом графа, необходимо декларировать соответствующую зависимость по данным.

Кода узла графа, выполняется только в тот момент времени, когда получены все данные, входящие в него по дугам графа. В такой постановке не может возникнуть ситуации, когда данные частично поступают в узел. Кроме того, данные по дугам, входящим в узел графа, могут поступать в произвольном порядке и вообще одновременно.

На рисунке 2 представлены компоненты многопроцессорной машины и связи между ними.

Пользовательское описание системы

С точки зрения пользователя, инструментальная система устроена следующим образом:

1. Программное средство *graph2c++* преобразования заданного на вход текстового представления графа алгоритма (графа зависимостей по данным, макро – граф программы) в параллельную программу с параллельными вызовами, например *MPI,* на языке программирования *C++*.
2. Исходные тексты ядра системы и Make файла для совместного компилирования ядра и исходных текстов программы, полученных с помощью утилиты (*graph2c++*). Ядро управляет непосредственно процессом выполнения полученной параллельной программы. После совместной компиляции исходных файлов ядра системы с полученным представлением графа алгоритма, в виде программы, написанной на языке *C*, будет получен исполнимый файл, который можно запустить на многопроцессорной системе.
3. Утилиты *processor_test* и *network_test* для измерения производительности системы. В результате работы данных утилит должно

быть получено 2 файла. Матрица со скоростями передачи данных между процессорами, а также вектор производительности процессоров.

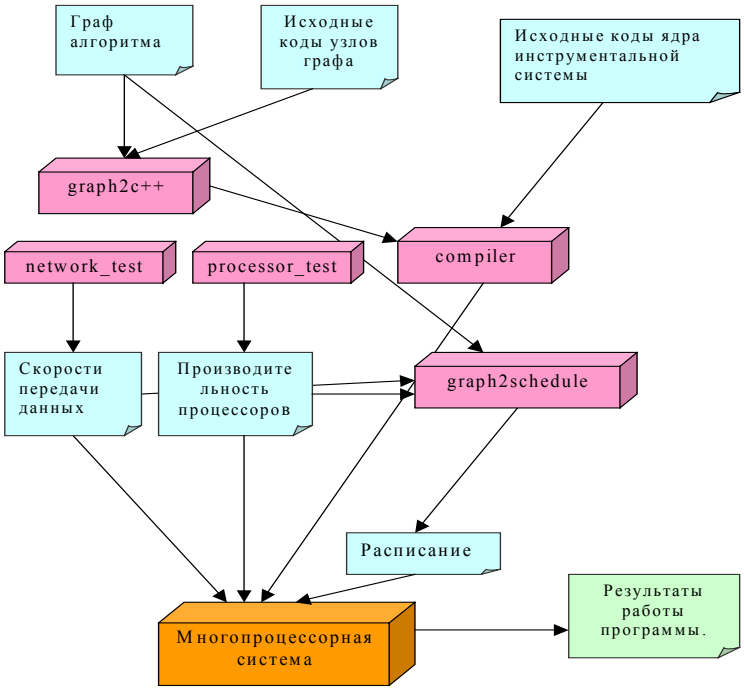


Рис.2. Компоненты инструментальной системы

4. Утилиты для получения статического расписания загрузки процессоров многопроцессорной системы (в текущий момент не реализовано).
5. Утилиты автоматического получения графа зависимости по данным по исходному коду однопроцессорной программы. В текущий момент пользователю предлагается самостоятельно описать собственный граф алгоритма, воспользовавшись форматом текстового представления графа алгоритма. Описание в приложениях. Однако

планируется использование графического интерфейса для создания и редактирования графа алгоритма. Кроме того, планируется автоматическое получение графа алгоритма по программе на языке *C* .

Итак, для работы системы, пользователю необходимо сформировать несколько нижеописанных файлов. При этом он может воспользоваться соответствующими утилитами, предоставляемыми системой:

1. Файл графа алгоритма (*graph.grf*) и файлы с исходными кодами узлов в графе алгоритма.
2. Расписание загрузки узлов графа алгоритма по процессорам, как желаемую последовательность выполнения на процессорах данных узлов графа алгоритма, а также желаемое прикрепление узлов графа к конкретным процессорам. (*schedule.sch*).
3. Вектор производительности процессоров многопроцессорной системы. (*procs.vct*).
4. Матрицу скоростей пересылок данных между процессорами в многопроцессорной системе. (*link.mtr*).

Последовательность действий при использовании инструментальной системы.

Работа с системой разбивается на 4 этапа:

1. Настройка инструментальной системы под конкретную архитектуру многопроцессорной машины;
2. Подготовка задания;
3. Статическая настройка задания на архитектуру;
4. Выполнение задания.

Настройка системы

1. Получить файл *procs.vct* с оценками производительности процессоров, воспользовавшись утилитой *processor_test.*
2. Получить файл *link.mtr* с оценками скоростей передачи данных между процессорами, воспользовавшись утилитой *network_test*.

Подготовка задания

1. Подготовить файл графа алгоритма. В случае отсутствия утилиты автоматического получения графа алгоритма, пользователь системы должен вручную составить файл графа алгоритма (описание и требования к нему представлены ниже).

2. После этого, воспользовавшись программой *graph2c++*, получить исходный файл на языке программирования *C++,* в котором содержатся параллельные вызовы *MPI*. Вершины графа в полученном файле, интерпретируются как функции, где в качестве кода подставляется текст из файлов, связанных с текстовым представлением графа алгоритма. Рёбра при этом интерпретируются как передача данных.

Статическая настройка задания на архитектуру

1. Получить файл *schedule.sch,* в котором указано, на каком процессоре и в каком порядке желательно выполнение узлов графа алгоритма. Желательно, чтобы данный файл был получен автоматически, но в текущий момент такое средство не реализовано.

Выполнение задания

1. Необходимо скомпилировать полученный *C++* файл совместно с ядром системы, например, воспользовавшись стандартной утилитой *make.* В результате получить исполняемый файл *prog*.
2. При помощи стандартной утилиты запуска программы на выполнение в многопроцессорной системе запустить полученную программу на выполнение. Например:
   mpirun –np 8 prog graph.grf schedule.sch procs.vct link.mtr ,
   указав в качестве параметров:
   8 – количество процессоров;
   prog – имя исполняемого файла;
   graph.grf – текстовое представление графа алгоритма;
   schedule.sch – файл расписания;
   procs.vct – файл производительностей процессоров;
   link.mtr – файл скоростей передачи данных.

Подсистема организации процесса выполнения параллельной программы

Система состоит из узлов графа алгоритма, которые оформлены как функции с именами *px_node_xxx*, функции *px_demon*, которая загружается на 0 процессор и занимается организацией порядка выполнения узлов графа, и управлением передачи данных. Функции *main* - головной функции, которая запускается на каждом процессоре и осуществляет непосредственную загрузку узлов графа по процессорам.

Программная часть инструментальной системы основана на принципе обмена сообщениями. В системе присутствует 2 различных типа сообщений. Первый тип – управляющие сообщения, сообщения между узлами графа алгоритма. Все эти сообщения короткие, их по сравнительно немного, следовательно, когда узлы графа алгоритма достаточно тяжёлые, управляющие сообщения не должны нагружать коммуникационную среду многопроцессорной системы. Второй тип – сообщения, посредством которых происходит передача данных между узлами графа алгоритма. Эти сообщения могут быть достаточно тяжёлыми и, как следствие, могут существенно загружать коммуникационную среду многопроцессорной системы.

В узлах графа программы сначала происходит приём данных, затем их обработка и передача другим узлам графа.

Организация выбора загружаемого узла графа на процессор

Выбор узла графа для выполнения на освободившемся процессоре из списка узлов, заявленных на исполнение, производится, исходя из соображения минимизации времени выполнения данного узла графа на данном процессоре.

Каждый узел характеризуется весом. Вес показывает объём операций, необходимых для выполнения узла, в эквиваленте числа операций по перемножению действительных чисел (*node_weight*).

Каждое ребро характеризуется весом. Вес показывает объём в байтах (длина сообщения) передаваемых данных (*edge_weight*).

Производительность процессора задаётся как время выполнения эталонного числа (*num_operation*) операций с плавающей точкой (*processor_weight*).

link_weight – время передачи сообщения с длиной, близкой к *edge_weight.*

Среди всех узлов, заявленных на выполнение, производится поиск узлов с минимальным временем выполнения на процессоре, где время считается по формуле:

$$time = \frac{node\_weight \times processor\_weight}{num\_operation} + \\ + \max_{1 \le i \le N} (link\_weight(edge\_weight_i)),$$

где N – число входящих рёбер.

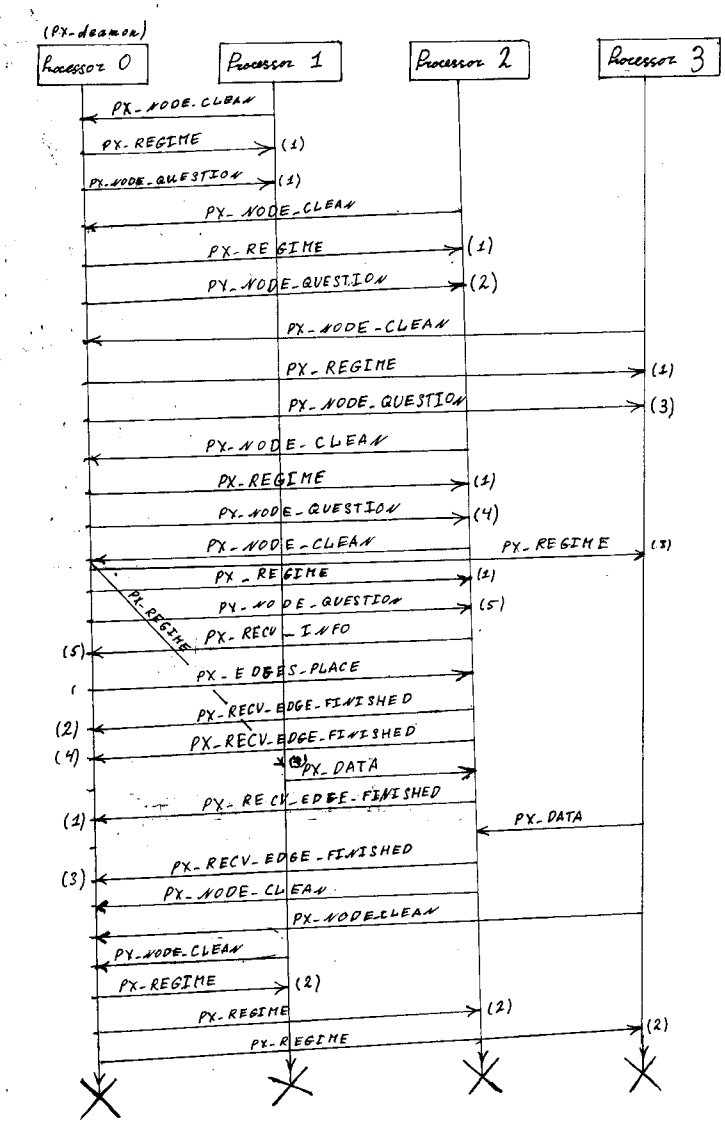


Рис.3. Протокол обмена сообщениями

Если минимум достигается для нескольких узлов графа одновременно, то выбирается тот, который заявлен в расписании на выполнение на данном процессоре.

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ ПАРАЛЛЕЛЬНЫХ МЕТОДОВ PLAPACK ПРИ РАЗРАБОТКЕ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

А.Н. Свистунов

Нижегородский государственный университет им. Н.И.Лобачевского

Создание *многопроцессорных (параллельных) вычислительных систем (ПВС)* является стратегической линией развития компьютерной техники, обусловливаемой существованием в любой текущий момент времени актуальных задач фундаментальной и прикладной науки, для анализа и исследования которых производительности существующих средств вычислительной техники оказывается недостаточно. Тем не менее практическое использование параллельных вычислительных систем не является столь широким, как это могло бы быть потенциально возможным. Одним из сдерживающих факторов, стоящих на пути широкого применения параллельных вычислительных систем, является высокая трудоемкость разработки программного обеспечения для них.

Возможное решение по снижению трудоемкости разработки параллельных алгоритмов и программ может состоять в разработке и в последующем широком использовании библиотек параллельных программ. Этот общепринятый в области разработки прикладного программного обеспечения подход позволяет согласовать два обычно несовместных требования – повысить качество создаваемых программ и значительно снизить сложность программирования.

В настоящее время для использования доступен довольно широкий спектр библиотек параллельных методов, облегчающих решение самых разных задач.

В докладе рассказывается об опыте использования в ННГУ одной из распространенных библиотек параллельных вычислений PLAPACK. Библиотека PLAPACK (Parallel Linear Algebra Package) представляет собой набор параллельных процедур линейной алгебры, необходимых при выполнении большого класса научно-технических расчетов.

PLAPACK реализован с использованием библиотеки передачи сообщений MPI для операционных систем семейства Windows и включает интерфейсы для языков Fortran и C.

Также в докладе рассказывается о результатах, полученных при тестировании кластера ННГУ с помощью теста LINPACK, реализованного с использованием библиотеки PLAPACK.

ПРИНЦИПЫ ПОСТРОЕНИЯ УЧЕБНО-НАУЧНОГО КОМПЛЕКСА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ СПбГУ

В.Ю. Сепман, В.И. Золотарев, Ю.П. Галюк

Санкт-Петербургский государственный университет

Учебно-научный комплекс высокопроизводительных вычислений Санкт-Петербургского государственного университета (СПбГУ), созданный по проекту В0008 Федеральной целевой программы «Интеграция», представляет собой многоуровневую систему компьютерных кластеров, распределенных по территории Петродворцового кампуса. В состав комплекса также входят кластеры, установленные в Петербургском институте ядерной физики им. Б.П. Константинова РАН (ПИЯФ РАН) в г. Гатчине и Междисциплинарном центре СПбГУ на Васильевском острове СПб. В качестве нижнего уровня системы используются учебные (малые) кластеры, установленные на естественных факультетах – физическом, прикладной математике – процессов управления, математико-механическом и химическом. Все учебные кластеры соединяются оптоволоконной сетью с основным вычислительным ядром системы и представляют собой достаточно мощные вычислители, ресурсы которых могут использоваться автономно, как отдельные восьмипроцессорные кластеры, и все вместе, образуя единый 32-х процессорный кластер.

Учебные кластеры имеют в своем составе 8 компьютеров Celeron 400, объединенных коммутаторами 3Com SuperStack 3300. Эти кластеры построены по типу Cluser of Workstation (COW). Производительность каждого учебного кластера на тесте Linpack на матрице 6000*6000 – 734 Mflops.

Программное обеспечение кластеров направлено, прежде всего, на создание комфортной среды даже для непрофессионального пользова-

теля, впервые занимающегося параллельным программированием. Вот основные принципы, реализуемые при построении учебных кластеров:

- Одинаковость операционных систем. На всех машинах кластеров установлена операционная система Linux RedHat 6.2/7.2.
- Наличие графического интерфейса с пользователем. Графическая оболочка KDE по внешнему виду и интерфейсу очень сильно напоминает интерфейс WINDOWS95.
- Единая система аутентификации. Обеспечивается использованием системы NIS.
- Отказ от принципа закрепленности определенного компьютера за пользователем. Обеспечивается наличием выделенного файл-сервера, раздающего через систему NFS рабочие каталоги пользователя на все компьютеры кластера.
- Однородность программного обеспечения. Все прикладные пакеты (Matlab, Mathematica) и трансляторы (Lahey/Fujitsu Fortran 95) приобретаются в сетевых вариантах и доступны к использованию на всех машинах всех кластеров.
- Максимальная WINDOWS-совместимость программного обеспечения. Практически все компиляторы (C, Fortran), прикладные пакеты (Matlab, Mathematica), библиотека параллельных расчетов MPICH-1.2.1, библиотека научных программ IMSL имеют свой WINDOWS-аналог. Этот факт позволяет полностью отлаживать свою программу на компьютере в системе WINDOWS, а затем с минимальной доработкой, а часто и без нее, после перетрансляции запустить параллельную версию задачи на машинах кластера.
- Максимальная скорость обменов по сети. Обеспечивается как аппаратной реализацией (100 Mbps коммутируемый Ethernet), так и применением наиболее эффективного протокола связи – RSH, с разрешением беспарольного входа на все машины кластеров (это необходимо для реализации многопроцессорной работы в наиболее популярном пакете распараллеливания – MPICH).
- Безопасность. Обеспечивается отсутствием физического контакта с сетью Интернет. Вход из Интернета возможен только через специальный компьютер-шлюз и только по шифрованному протоколу SSH. Доступ из кластеров в Интернет запрещен.

Следующий уровень системы – основное вычислительное ядро – состоит из кластера повышенной мощности, работающего в режиме

круглосуточного счета. Кластер состоит из 8 двухпроцессорных рабочих станций Pentium III 933 с памятью 1 GB. Программное обеспечение кластера отвечает следующим принципам:

- Полная переносимость исполняемых файлов, исключающая необходимость перетрансляции программ.
- Отсутствие системы очередей, при этом ответственность по распределению задач между компьютерами кластера ложится на программиста.
- Наличие эффективных средств отладки программ, как последовательных, так параллельных.

Наиболее высокопроизводительный ресурс – SCI-кластер, состоящий из 20 двухпроцессорных серверов Pentium III 933 с памятью 1 GB и дисковой подсистемой 40 GB. Производительность его на тесте Linpack – 17.1 Gflops (матрица 43000*43000). Для управления кластером используется сервер Celeron 500. Для поддержки пользователей в кластер включен двухпроцессорный сервер Pentium II 450 с памятью 256 MB и 20 GB HDD. Управляющая сеть кластера создана на базе коммутатора 10/100 Cisco Catalyst WS-C2924-XL-EN. При построении кластера высокопроизводительных вычислений были заложены следующие принципы:

- Прямой доступ к машинам кластера запрещается.
- Для запуска задач используется система очередей PBS.
- Правом постановки задач в очередь пользуется ограниченный круг пользователей, прописанный в специальном списке (ACL-list).
- Задачи, решаемые на кластере высокопроизводительных вычислений, должны быть перетранслированы с подключением специальных библиотек, поддерживающих сеть SCI.

В настоящее время расширяется число учебных кластеров, подключенных к комплексу, установленных в сети университета. Пять кластеров построены сотрудниками факультета прикладной математики – процессов управления в новом здании в Петергофе. Все кратеры соединяются 100 Mbps каналами коммутатором Cisco Catalyst WS-C3548, который подключен к сети комплекса скоростной 1 Gbps оптоволоконной линией.

Вычислительные мощности комплекса используются для выполнения расчетов по крупным научно-исследовательским проектам. Сре-

ди этих проектов – работы по исследованию свойств материалов с использованием программ GAMESS и CRYSTAL, расчеты жидкокристаллических структур, исследования неравновесных и нестационарных процессов в газах и многофазных средах, расчеты по физике высоких энергий (проект ALICE). В число организаций, проводящих исследования на высокопроизводительном кластере, входят: Санкт-Петербургский институт информатики и автоматизации РАН, Петербургский институт ядерной физики им. Б.П. Константинова РАН, Институт высокомолекулярных соединений РАН, Государственное унитарное предприятие «Государственный оптический институт им. С.И. Вавилова» и Институт физиологии им. И.П. Павлова РАН, Институт проблем машиноведения РАН.

ПРОСТЕЙШИЙ ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР НА БАЗЕ УЧЕБНОГО КЛАССА

А. Шишкин, С. Гельбух

Поволжский региональный центр новых информационных технологий СГУ, г. Саратов

В последнее время острая потребность в высокопроизводительных вычислительных комплексах появляется во многих областях научной деятельности. В Саратове к ним относятся нанофотоника, лазерная физика, квантовая химия, физическая химия. Между тем до сих пор не существовало ни одной подобной системы в Саратове и регионе, несмотря на сформировавшуюся потребность в высокопроизводительных вычислительных ресурсах.

Основной проблемой при создании подобного комплекса является его высокая стоимость, поэтому предпочтительным было начать с использования имеющегося оборудования СГУ. Простейший кластер создан на базе одного из имеющихся в университете учебных компьютерных классов. Каждый узел кластера имеет следующую конфигурацию: Celeron 433/64 Mb RAM/HDD Fujitsu MPF3102AT 9.54Gb/S3 Trio 3D2X 4Mb/Realtec RTL8139(A) PCI Fast Ethernet. Для изучения особенностей коммуникационной среды кластера использовались концентратор Compex MX2216SB (100 Мбит/с)и коммутатор IBM-8271. На

каждый компьютер установлена операционная система RedHat Linux 6.2, в стандартный дистрибутив которой входит реализация MPICH интерфейса MPI. Основной задачей первого этапа работ было определение параметров полученной системы. Для оценки производительности кластера использовалась параллельная версия теста LINPACK.

По результатам тестов были построены графики зависимости вычислительной мощности кластера от числа узлов. При включении всех 6 узлов максимальная производительность увеличивается в 4.14 раз по сравнению с одним компьютером.

При тестировании были выявлены «узкие места» системы. Замечено, что при построении локальной сети кластера с использованием концентратора Compex MX2216SB, при числе узлов более трех сеть не справлялась с нагрузкой. В результате производительность системы понижалась. Использование коммутатора IBM-8271 позволило увеличить число узлов до 6. При решении задач большой размерности серьезным ограничением стал объем оперативной памяти, которая используется для хранения временных результатов вычислений. Эту проблему можно решить лишь расширением ОЗУ каждого узла.

Для эффективного использования вычислительных ресурсов кластера необходимо организовать доступ к нему большего числа пользователей. Так как все узлы кластера подключены к сети СГУ и Интернет, то существует возможность организации удаленного доступа к системе. При этом пользователю необходимо скопировать свою программу на все узлы системы и запустить ее. В перспективе будет разработан web-интерфейс для удаленной установки вычислительной задачи на кластер.

ВЫБОР СИСТЕМЫ УПРАВЛЕНИЯ КЛАСТЕРОМ

В.Е.Спирин

Нижегородский государственный университет им. Н.И. Лобачевского

В ННГУ им. Лобачевского в рамках гранта фирмы Интел проводились исследования в области параллельных технологий (результат можно увидеть на неофициальном сайте проекта http://unn_cluster.chat.ru/). Актуальность этих исследований обуславли-

валась повышением интереса компании Интел к этой тематике. Мы проводили теоретический обзор систем управления кластером. В начале работы не было ясности с архитектурой поставляемой вычислительной техники (Itanium, P4, P!!!) и с операционной системой (Linux, Windows), поэтому исследования проводились во всем возможном диапазоне. Основными требованиями к системе управления кластером были поддержка MPI, диспетчеризации, балансировки, графического интерфейса. Второстепенным требованием была поддержка миграции задач. В результате анализа более 10 систем и консультаций с ведущими исследовательскими центрами и производителями (ИПМ им. Келдыша, МГУ, NCSA, Sun, Intel, Portland Group и др.) были выбраны следующие системы: для Linux – OpenPBS, для Windows – LSF. Кроме вышеперечисленных особенностей, стоит отметить очень хорошую техническую поддержку этих систем (по опыту установки MPI можно сказать, что оперативные консультации производителя – очень важный момент).

Литература

1. Кузьминский М. NQS и пакетная обработка в Unix// Открытые системы. 1997. № 1. http://www.osp.ru/os/1997/01/18.htm
2. Коваленко В., Корягин Д. Вычислительная инфраструктура будущего // Открытые системы. 1999. № 11–12. http://www.osp.ru/os/1999/11–12/045.htm.
3. Коваленко В., Коваленко Е. Пакетная обработка заданий в компьютерных сетях // Открытые системы. 2000. № 7–8. http://www.osp.ru/os/2000/07-08/010.htm.
4. Владимиров Д. Кластерная система Condor // Открытые системы. 2000. № 7–8. http://www.osp.ru/os/2000/07-08/020.htm.
5. Общий обзор систем управлений кластером. http://nhse.cs.rice.edu/NHSEreview/CMS/).

LAWRA
LINEAR ALGEBRA WITH RECURSIVE ALGORITHMS*

Jerzy Wasniewski[1], Bjarne S. Andersen[1],
Fred Gustavson[2], Alexander Karaivanov[1],
Minka Marinova[1], Plarnen Yalamov[3]

Abstract

Recursion leads to automatic variable blocking for dense linear algebra algorithms. The recursion transforms LAPACK level-2 algorithms into levels codes. For this and other reasons recursion usually speeds up the algorithms.

Recursion provides a new, easy and very successful way of programming numerical linear algebra, algorithms. Several algorithms for matrix factorization have been implemented and tested. Some of these algorithms are already candidates for the LAPACK library.

Recursion has also been successfully applied to the BLAS (Basic Linear Algebra Subprograms). The ATLAS system (Automatically Tuned Linear Algebra Software) uses a recursive coding of the BLAS.

The Cholesky factorization algorithm for positive definite matrices, LU factorization for general matrices, and LDL^T factorization for symmetric indefinite matrices using recursion are formulated in this paper. Performance graphs of our packed Cholesky and LDL^T algorithms are presented here.

* This research was partially supported by the UNI•C collaboration with the IBM T.J. Watson Research Center at. Yorktown Heights. The second and fifth authors was also supported by the Danish Natural Science Research Council through a grant for the EPOS project (Efficient Parallel Algorithms for Optimization and Simulation).

[1] Danish Computing Center for Research and Education (UNI•C), Technical University of Denmark, Building 304, DK-2800 Lyngby, Denmark, email: jerzy.wasniewski@uni-c.dk.

[2] IBM T.J. Watson Research Center, P.P. Box 218, Yorktown Heights, NY 10598, USA.

[3] Center of Applied Mathematics and Informatics, University of Rousse, 7017 Rousse, Bulgaria.

1. Introduction

This work was started by Gustavson and Toledo described in [8, 16]. These papers describe the application of recursion to the numerical dense linear algebra algorithms. Recursion leads to automatic variable blocking for the dense linear-algebra algorithms. This leads to modifications of the LAPACK [2] algorithms. LAPACK's level-2 version routines are transformed into level-3 codes by using recursion.

Fortran 90 allows recursion (see [15]). The programs are very concise and the recursion part is automatic as it is handled by the compiler. The intermediate subroutines obey the Fortran 90 standard too (see [5]).

Section 2 shows the recursive Cholesky factorization algorithm. Section 3 formulates the recursive algorithm of Gaussian elimination without pivoting and LU factorization with partial pivoting. Section 4 explains two recursive BLAS: RTRSM and RSYRK. Section 5 demonstrates the factorization algorithm using the pivoting strategy introduced by Bunch-Kaufman for symmetric indefinite matrices (see [6, pp. 161–170]).

2. Cholesky Factorization

We would like to compute the solution to a system of linear equations $AX = B$, where A is real symmetric or complex Hermitian and, in either case, positive definite matrix, X and B are rectangular matrices or vectors. The Cholesky decomposition can be used to factor A, $A = LL^T$ or $A = U^TU$, where U is an upper triangular matrix and L is a lower triangular ($L = U^T$). The factored form of A is then used to solve the system of equations $AX = B$.

A recursive algorithm of Cholesky factorization is described in detail in [18, 8]. Here we give the final recursive algorithms for the lower triangular and upper triangular cases. We assume that A is n by n.

Recursive Algorithm 2.1. *Cholesky recursive algorithm if lower triangular part of A is given* (*rcholesky*)*:*

Do recursion

- *if* $n > 1$ *then*
  - L_{11} := *rcholesky of* A_{11}
  - $L_{21}\ L_{11}^T = A_{21} \rightarrow$ **RTRSM**
  - $\hat{A}_{22} := A_{22} - L_{21}\, L_{21}^T \rightarrow$ **RSYRK**

• L_{22} := *rcholesky of* $\hat{A}_{22}$

• *otherwise*

• $L := \sqrt{A}$

End recursion

The matrices A_{11}, A_{12}, A_{21}, A_{22}, L_{11}, L_{21}, L_{22}, U_{11}, U_{12}, and U_{22}, are submatrices of A, L and U respectively.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad L = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix}, \quad U = \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}.$$

Recursive Algorithm 2.2. *Cholesky recursive algorithm if upper triangular part of A is given* (*rcholesky*)*:*

Do recursion

• *if* $n > 1$ *then*

• U_{11} := *rcholesky of* A_{11}

• $U_{11}^T \; U_{12} = A_{12} \rightarrow$ **RTRSM**

• $\hat{A}_{22} := A_{22} - U_{11}^T \; U_{12} \rightarrow$ **RSYRK**

• U_{22} := *rcholesky of* $\hat{A}_{22}$

• *otherwise*

• $U := \sqrt{A}$

End recursion

The sizes of the subrnatrices are: for A_{11}, L_{11} and U_{11} is $h \times h$, for A_{21} and L_{21} is $(n - h) \times h$. for A_{12} and U_{12} is $h \times (n - h)$, and for A_{22}, L_{22} and U_{22} is $(n - h) \times (n - h)$, where $h = n/2$. Matrices L_{11}, L_{22}, U_{11} and U_{22} are lower and upper triangular respectively. Matrices A_{11}, A_{12}, A_{21}, A_{22}, L_{21} and U_{12} are rectangular.

The RTRSM and RSYRK are recursive BLAS of _TRSM and _SYRK respectively. _TRSM solves a triangular system of equations. _SYRK performs the symmetric rank k operations (see Sections 4.1 and 4.2).

The listing of the Recursive Cholesky Factorization Subroutine is attached in the Appendix A.

2.1. Full and Packed Storage Data Format

The Cholesky factorization algorithm can be programmed either in "full storage" or "packed storage". For example the LAPACK subroutine POTRF

works on full storage, while the routine PPTRF is programmed for packed storage. Here we are interested only in full storage and packed storage holding data that represents dense symmetric positive definite matrices. We will compare our recursive algorithms to the LAPACK POTRF and PPTRF subroutines.

The POTRF subroutine uses the Cholesky algorithm in full storage. A storage for the full array A must be declared even if only $n \times (n+1)/2$ elements of array A are needed and, hence $n \times (n-1)/2$ elements are not touched. The PPTRF subroutine uses the Cholesky algorithm on packed storage. It only needs $n \times (n+1)/2$ memory words. Moreover the POTRF subroutine works fast while the PPTRF subroutine works slow. Why? The routine POTRF is constructed with BLAS level 3, while the PPTRF uses the BLAS level 2. These LAPACK data structures are illustrated by the figs. 1 and 2 respectively.

Lower triangular case

$$\begin{pmatrix} 1 & & & & & & \\ 2 & 9 & & & & & \\ 3 & 10 & 17 & & & & \\ 4 & 11 & 18 & 25 & & & \\ 5 & 12 & 19 & 26 & 33 & & \\ 6 & 13 & 20 & 27 & 34 & 41 & \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 \end{pmatrix}$$

Upper triangular case

$$\begin{pmatrix} 1 & 8 & 15 & 22 & 29 & 36 & 43 \\ & 9 & 16 & 23 & 30 & 37 & 44 \\ & & 17 & 24 & 31 & 38 & 45 \\ & & & 25 & 32 & 39 & 46 \\ & & & & 33 & 40 & 47 \\ & & & & & 41 & 48 \\ & & & & & & 49 \end{pmatrix}$$

Figure 1. The mapping of 7×7 matrix for the LAPACK Cholesky Algorithm using the full storage

Lower triangular case

$$\begin{pmatrix} 1 & & & & & & \\ 2 & 8 & & & & & \\ 3 & 9 & 14 & & & & \\ 4 & 10 & 15 & 19 & & & \\ 5 & 11 & 16 & 20 & 23 & & \\ 6 & 12 & 17 & 21 & 24 & 26 & \\ 7 & 13 & 18 & 22 & 25 & 27 & 28 \end{pmatrix}$$

Upper triangular case

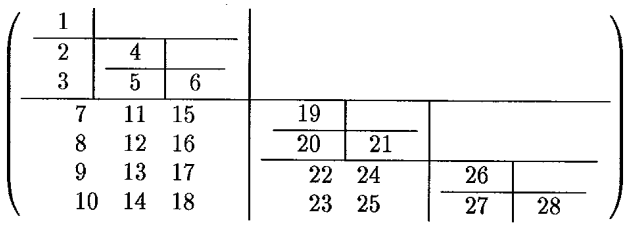
$$\begin{pmatrix} 1 & 2 & 4 & 7 & 11 & 16 & 22 \\ & 3 & 5 & 8 & 12 & 17 & 23 \\ & & 6 & 9 & 13 & 18 & 24 \\ & & & 10 & 14 & 19 & 25 \\ & & & & 15 & 20 & 26 \\ & & & & & 21 & 27 \\ & & & & & & 28 \end{pmatrix}$$

Figure 2. The mapping of 7×7 matrix for the LAPACK Cholesky Algorithm using the packed storage

2.2 The Recursive Storage Data Format

We introduce a new storage data format, the recursive storage data format, using the recursive algorithms 2.1 and 2.2. Like packed data format this recursive storage data format requires $n \times (n-1)/2$ storage for the upper or lower part of the matrix. The recursive storage data format is illustrated in fig. 3. A buffer of the size $p \times (p-1)/2$, where $p = [n/2]$ (integer division), is needed to convert from the LAPACK packed storage data, format to the recursive packed storage data, format, and back. No buffer is needed if data is given in recursive format.

Lower triangular case



Upper triangular case

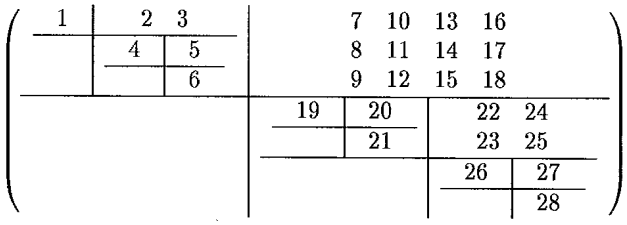


Figure 3. The mapping of 7×7 matrix for the LAPACK Cholesky Algorithm using the packed recursive storage

We can apply the BLAS level 3 using the recursive packed storage data format. The performance of the recursive Cholesky algorithm with the recursive packed data format reaches the performance of the LAPACK POTRF algorithm. A graphs with the performance between different storages is presented in fig. 4.

The Figure 4 has two subfigures. The upper subfigure shows comparison curves for Cholesky factorization. The lower subfigure show comparison curves of forward and backward substitutions. The captions describe

details of the performance figures. Each subfigure presents six curves. From the bottom: The first two curves represent LAPACK PPTRF performance results for upper and lower case respectively. The third and fourth curves give the performance of the recursive algorithms, *L* and *U,* respectively. The conversion time from LAPACK packed data format to the recursive packed data format and back is included here. The fifth and sixth curves give performance of the *L* and *U* variants for the LAPACK POTRF algorithm. The IBM SMP optimized ESSL DGEMM was used by the last four algorithms.

The same good results were obtained on other parallel supercomputers, for example on Compaq *a* DS-20 and SGI Origin 2000.

The paper [1] gives comparison performance results on various computers for the Cholesky factorization and solution.

3. LU *factorization*

We would like to compute the solution to a system of linear equations $AX = B$, where A is a real or complex matrix, and X and B are rectangular matrices or vectors. Gaussian elimination with row interchanges is used to factor A as $LU = PA$, where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. The factored form of A is then used to solve the system of equations $AX = B$.

The recursive algorithm of the Gauss LU factorization is described in detail in [3, 8]. We give two recursive algorithms here. They are listed in figs. 3.1 and 3.2.

Recursive Algorithm 3.1 *Recursive LU factorization without pivoting* (*rgausslu*)*:*

Do recursion

- *if* $\min(m,n) > 1$ *then*
  - (L_1,U_1) = *rgausslu of* A_1
  - $L_{11}U_{21} = A_{12}$ → **RTRSM**
  - $\hat{A}_{22} := A_{22} - L_{21}U_{12}$ → **_GEMM**
  - (L_{22},U_{22}) = *rgausslu of* $\hat{A}_{22}$
- *otherwise*
  - $L_1 := A_1/a_{11}$ *and* $U_1 = a_{11}$

End recursion

- *if* $n > m$ *then* ,
  - $LU_3 = A_3$ → **RTRSM**

The matrices A_1, A_2, A_3, A_{12}, A_{22}, L_1, L_{11}, L_{21}, L_{22}, U_1, U_3, U_{12} and U_{22} are submatrices of A, L and U respectively, $a_{11} \in A_1$.

Recursive Algorithm 3.2. *Recursive LU=PA factorization with partial pivoting* (*rgausslu*)*:*

Do recursion
- *if* $\min(m,n) > 1$ *then*
  - (P_1,L_1,U_1) = *rgausslu of* A_1
  - *Forward pivot* A_2 *by* $P \rightarrow$ **_LASWP**
  - $L_{11}U_{21} = A_{12} \rightarrow$ **RTRSM**
  - $\hat{A}_{22} := A_{22} - L_{21}U_{12} \rightarrow$ **_GEMM**
  - (P_2,L_{22},U_{22}) = *rgausslu of* $\hat{A}_{22}$
  - *Back pivot* A_1 *by* $P_2 \rightarrow$ **_LASWP**
  - $P = P_2P_1$
- *otherwise*
  - *pivot* A_1
  - $L_1 := A_1/a_{11}$ *and* $U_1 := a_{11}$

End recursion
- *if* $n > m$ *then*
  - *Forward pivot* A_3 *by* $P \rightarrow$ **_LASWP**
  - $LU_3 = A_3 \rightarrow$ **RTRSM**

4. Recursive BLAS and Parallelism

Two recursive BLAS (Basic Linear Algebra Subprograms, see [20]) subroutines are used in our recursive Cholesky and recursive *LU* algorithms: RTRSM and RSYRK. These two routines will be explained below.

4.1 RTRSM

RTRSM is a recursive formulation of _TRSM, where _ is a precision and arithmetic indicator: S, D, C or Z. _TRSM subroutine solves one of the matrix equation

$$AX = \alpha B, \ A^{\mathrm{T}}X = \alpha B, \ XA = \alpha B, \ \text{or } XA^{\mathrm{T}} = \alpha B,$$

where α is a scalar. X and B are $(m \times n)$ rectangular matrices. A is a unit, or non-unit, upper or lower (m xm) triangular matrix. The matrix X is overwritten on B. We have 16 different triangular equations because A and A^T can be either upper or lower triangular matrices, and the diagonal is normal or unit.

We will introduce the recursive formulation only for one case $AX = \alpha B$, where A is lower triangular. The other cases will be similar.

The matrices A, B, and X can be partitioned into smaller submatrices, thus

$$\begin{pmatrix} A_{11} & \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} = \alpha \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

The matrices $A_{11} = A(1 : h, \quad 1 : h)$, $A_{21} = A(h+1 : m, \quad 1 : h)$, $A_{22} = A(h+1 : m, \quad h+1 : m)$, $B_{11} = B(1 : h, \quad 1 : p)$, $B_{12} = B(1 : h, \quad p+1 : n)$, $B_{21} = B(h+1 : m, \quad 1 : p)$, $B_{22} = B(h+1 : m, \quad p+1 : n)$, $X_{11} = X(1 : h, \quad 1 : p)$, $X_{12} = X(1 : h, p+1 : n)$, $X_{21} = X(h+1 : m, 1 : p)$ and $X_{22} = X(h+1 : m, p+1 : n)$ are submatrices of A, B and X respectively, and $h = m/2$ and $p = n/2$. Multiplying the matrix A by X gives:

$$\begin{pmatrix} A_{11}X_{11} & A_{11}X_{12} \\ A_{21}X_{11} + A_{22}X_{21} & A_{21}X_{12} + A_{22}X_{22} \end{pmatrix} = \begin{pmatrix} \alpha B_{11} & \alpha B_{12} \\ \alpha B_{21} & \alpha B_{22} \end{pmatrix}.$$

We have got two independent groups of triangular systems:

$$A_{11}X_{11} = \alpha B_{11} \qquad A_{11}X_{12} = \alpha B_{12}$$

$$A_{22}X_{21} = \alpha B_{21} - A_{21}X_{11} \qquad A_{22}X_{22} = \alpha B_{22} - A_{21}X_{12}$$

We could do a double recursion on m and n; i. e. on h and p. However, we do not do the recursion on p. This results in the following algorithm:

Recursive Algorithm 4.1. *Recursive algorithm for the $AX = B$ operation* (*one group only*)*, where A is a lower triangular matrix* (*rtrsm*)*:*

Do recursion

- *if* $m > 1$ *then*
  - $A_{11}X_1 = \alpha B_1 \rightarrow$ **RTRSM**
  - $B_2 := \alpha B_2 - A_{21}X_1 \rightarrow$ **_GEMM**
  - $A_{22}X_2 = \alpha B_2 \rightarrow$ **RTRSM**
- *otherwise*
  - $a_{11}X_1 = \alpha B_1$

End recursion

4.2 RSYRK

RSYRK is a recursive formulation of _SYRK, where _ is a precision and arithmetic indicator: S, D, C or Z. _SYRK performs one of the symmetric rank k operations:

$$C := \alpha AA^{\mathrm{T}} + \beta C \text{ or } C := \alpha A^{\mathrm{T}}A + \beta C$$

where α and β are scalars. A is a rectangular matrix ($m \times n$). C is a square symmetric matrix.

We will introduce the recursive formulation only for one of the four cases of _SYHK:

$$C := \alpha AA^{\mathrm{T}} + \beta C.$$

The matrices A and C can be partitioned into smaller submatrices:

$$\begin{pmatrix} C_{11} & \\ C_{21} & C_{22} \end{pmatrix} = \beta \begin{pmatrix} C_{11} & \\ C_{21} & C_{22} \end{pmatrix} + \alpha \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}^T.$$

The matrices $A_{11} = A(1 : h, \quad 1 : p)$, $A_{12} = A(1 : h, \quad p+1 : n)$, $A_{21} = A(h+1 : m, 1 : p)$, $A_{22} = A(h+1 : m, \quad p+1 : n)$, $C_{11} = C(1{:}h, \quad 1{:}h)$, $C_{21} = C(h+1 : m, \ 1 : h)$, $C_{22} = C(h+1 : m, \ h+1 : m)$ are submatrices of A and C respectively, and $h = m/2$ and $p = n/2$. The recursion could be done again on two variables, ft and p, but we do recursion on ft only.

In terms of the partitioning we have three independent formulas:

$$C_{11} = \beta C_{11} + \alpha A_{11} A_{11}^T + \alpha A_{12} A_{12}^T$$

$$C_{21} = \beta C_{21} + \alpha A_{21} A_{11}^T + \alpha A_{22} A_{12}^T$$

$$C_{22} = \beta C_{22} + \alpha A_{21} A_{21}^T + \alpha A_{22} A_{22}^T$$

These three computations can he done in parallel. We now formulate a recursive algorithm as follows:

Recursive Algorithm 4.2 *Recursive algorithm for the $C := aAA^T + \beta C$ symmetric rank k operations (rsyrk):*

Do recursion

• *if* $m \geq 1$ *then*

Perform computation C_{11}*:*

• $\hat{C}_{11} := \beta C_{11} + \alpha A_{11} A_{11}^T \rightarrow$ **RSYRK**

• $C_{11} := \hat{C}_{11} + \alpha A_{12} A_{12}^T \rightarrow$ **RSYRK**

Perform computation C_{21}:

• $\hat{C}_{21} := \beta C_{21} + \alpha A_{21} A_{11}^T \rightarrow$ **_GEMM**

• $C_{21} := \hat{C}_{21} + \alpha A_{22} A_{12}^T \rightarrow$ **_GEMM**

Perform computation C_{22}*:*

• $\hat{C}_{22} := \beta C_{22} + \alpha A_{21} A_{21}^T \rightarrow$ **RSYRK**

• $C_{22} := \hat{C}_{22} + \alpha A_{22} A_{22}^T \rightarrow$ **RSYRK**

End recursion

4.3. A fast _GEMM algorithm

The _ of _GEMM is a precision and arithmetic indicator: S, D, C or Z. _GEMM subroutine does the following operations

$$C := \alpha AB + \beta C,\ \ C := \alpha AB^{T} + \beta C,\ \ C := \alpha A^{T}B + \beta C,$$
$$C := \alpha A^{T}B^{T} + \beta C, \text{ or } C := \alpha AB^{C} + \beta C,\ \ C := \alpha A^{C}B + \beta C,$$
$$\text{and } C := \alpha A^{C}B^{C} + \beta C,$$

where α and β are scalars. A, B and C are rectangular matrices. A^{T}, B^{T}, A^{C} and B^{C} are transpose and conjugate matrices respectively.

The GEMM operation is very well documented and explained in [9, 10, 20]. We can see that work is done by _GEMM for both our BLAS RTRSM Section 4.1 and RSYRK Section 4.2. The speed of our computation depends very much from the speed of a good _GEMM. Good _GEMM implementations are usually developed by computer manufacturers. The model implementation of _GEMM can be obtained from netlib [20]; it works correctly but slowly. However, an excellent set of high performance BLAS, called _GEMM based BLAS was developed by Bo Kågström at the University of Umeå in Sweden, see for example [14, 10]. A key idea behind _GEMM based BLAS was to cast all BLAS algorithms in terms of the simple BLAS _GEMM. Recently, the Innovative Computing Laboratory at University of Tennessee in Knoxville developed a system called ATLAS which can produce a fast _GEMM program.

4.3.1 ATLAS

Automatically Tuned Linear Algebra Software (ATLAS) [19]. ATLAS is an approach for the automatic generation and optimization of numerical software for processors with deep memory hierarchies and pipelined functional units. The production of such software for machines ranging from desktop workstations to embedded processors can be a tedious and time consuming task. ATLAS has been designed to automate much of this process. So, having a fast _GEMM means our RTRSM and RSYRK routines will be fast. The ATLAS GEMM is often better than _GEMM developed by the computer manufacture. What is important, the ATLAS software is available to everybody, free of charge. Every personal computer can have a good GEMM.

5. LDL^{T} *Factorization for Symmetric Indefinite Matrices*

It is well-known that the Cholesky factorization can fail for symmetric indefinite matrices. In this case some pivoting strategy can be applied (e. g.

the Bunch-Kaufman pivoting [6, §4.4]). The algorithm is formulated in [4, 6, 13, 17]. As a, result we get

$$PAP^{\mathrm{T}} = LDL^{T}.$$

where L is unit lower triangular, D is block diagonal with 1×1, or 2×2 blocks, and P is a permutation matrix.

Now let us look at a recursive formulation of this algorithm. This is given below. The recursion is done on the second dimension of matrix A, i.e. the algorithm works on full columns like in the LU factorization.

The LAWRA project on LDL^{T} is still going on. We have developed several perturbation approach algorithms for solving linear systems of equations, where the matrix is symmetric and indefinite. The results are presented in two papers, [7, 12].

We have also obtained very good results for the Bunch-Kaufman factorization, where the matrix A is given in packed data format with about 5% extra space appended. We call it a blocked version. The recursion is not applied. The speed of our packed blocked algorithm is comparable with the speed of LAPACK full storage algorithm. The algorithm is described and the comparison performance results for the factorization and solution are presented in [11].

Recursive Algorithm 5.1. *Recursive Symmetric Indefinite Factorization* (*RSIF*) *of* $A_{1:m,1:n}$, $m = n$:

$k = 1$
if ($n = 1$)
    Define the pivot: 1×1, *or* 2×2.
    Apply interchanges if necessary
    $k = k + 1$, *or* $k = k + 2$
    If the pivot is 2×2: $FLAG = 1$
else
    $n1 = n/2$
    $n2 = n - n1$
    RSIF of $A_{:,k:k+n1-1}$
    if ($FLAG = 1$)
        $n1 = n1 + 1$
        $n2 = n - n1$
    end
    update $A_{:,k:k+n2-1}$
    RSIF of $A_{:,k:k+n1-1}$
end

References

1. B.S. Andersen, F. Gustavson and J. Waśniewski: "A recursive formulation of Cholesky factorization of a matrix in packed storage", University of Tennessee, Knoxville, TN, Computer Science Dept. Technical Report CS-00-441, May 2000, also LAPACK Working Note number 146 (lawn146.ps), and submitted to the Transaction of Mathematical Software (TOMS) of the ACM.
2. E. Anderson, Z. Bai. C. H. Bischof. S. Blackford. J. Demmel, J. J. Dongarra, J.Du Croz, A.Greenbaum, S.Hammairling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide Release 3.0.* SIAM, Philadelphia, 1999.
3. B.S. Andersen. F. Gustavson, J. Waśniewski, and P. Yalamov: Recursive formulation of some dense linear algebra algorithms, in *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, PPSC99,* B. Hendrickson, K.A. Yelick, C.H. Bischof, I.S. Duff, A.S. Edelman, G.A. Geist, M.T. Heath, M.A. Heroux, C. Koelbel, R.S. Schreiber, R.F. Sincovec, and M.F. Wheeler (Eds.), San Antonio, TX, USA, March 24–27, 1999, SIAM, Scientific Computing, CDROM.
4. J.W. Demmel, *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.
5. J. Dongarra and J. Waśniewski, High Performance Linear Algebra Package - LAPACK90, in Advances in Randomized Parallel Computing, Kluwer Academic Publishers, Combinatorial Optimization Series, P.M. Pardalos and S. Rajasekaran (Eds.), 19139 and available as the LAPACK Working Note (Lawn) Number 134: http://www.not Iib.org/lapack/lawns/lawn134.ps
6. G.H. Golub and C.F. Van Loan, *Matrix Computations* (*third edition*). Johns Hopkins University Press, Baltimore, MD, 1996.
7. A. Gupta, F. Gustavson, A. Karaivanov, J. Waśniewski, and P. Yalamov, *Experience with a Recursive Perturbation Based Algorithm for Symmetric Indefinite Linear Systems,* in: Proc. EuroPar'99, Toulouse, France, 1999.
8. F. Gustavson, *Recursive Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms.* IBM Journal of Research and Development, Volume 41, Number 6, November 1997.

9. F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling, Recursive Blocked Data Formats and BLAS' for Dense Linear Algebra Algorithms, in *Proceedings of the 4th International Workshop, Applied Parallel Computing, Large Scale Scientific and Industrial Problems, PARA '98,* B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski (Eds.), Umeå, Sweden, June 1998, Springer, Lecture Notes in Computer Science Number 1541. pp. 195–206.
10. F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling, Superscalar GEMM-based Level 3 BLAS - The On-going Evolution of Portable and High-Performance Library, in *Proceedings of the 4th International Workshop, Applied Parallel Computing, Large Scale Scientific and Industrial Problems, PARA'98,* B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski (Eds.), Umeå, Sweden, June 1998, Springer, Lecture Notes in Computer Science Number 1541, pp. 207–215.
11. F. Gustavson, A. Karaivanov, M. Marinova, J. Waśniewski, and Plamen Yalamov, *A Fast Minimal Storage Symmetric Indefinite Solver,* in Lecture Notes in Computer Science, Springer Volume 1947, Editors: T. Soerevik, F. Manne, R. Moe, A.H. Gebremedhinr,. PARA'2000, Bergen, Norway, June, 2000.
12. F. Gustavson, A. Karaivanov, J. Waśniewski, and P. Yalamov, *A columnwise recursive perturbation based algorithm for symmetric indefinite linear systems*, in: Proc. PDPTA'99, Las Vegas. 1999.
13. N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, 1.996.
14. B. Kågström, P. Ling, and C. Van Loan, *GEMM-based level 3 BIAS: High-performance model implementations and performance evaluation benchmark.* ACM Trans. Math. Software, 1997.
15. S. Metcalf and J. Reid. *Fortran 90/95 Explained.* Oxford, New York, Tokyo, Oxford University Press, 1996.
16. S. Toledo, *Locality of Reference in LU Decomposition with Partial Pivoting.* SIAM Journal on Matrix Analysis and Applications, Vol. 18, No. 4, 1997.
17. L.N. Trefethen and D. Bau. III. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.
18. J. Waśniewski, B.S. Andersen, and F. Gustavson. Recursive Formulation of Cholesky Algorithm in Fortran 90, in *Proceedings of the* 4ift *In-*

ternational Workshop, Applied Parallel (*Computing, Large Scale Scientific and Industrial Problems. PARA'98.* B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski (Eds.), Umeå, Sweden, June 1998, Springer, Lecture Notes in Computer Science Number 1541, pp. 574–578.

19. R.C. Whaley and J. Dongarra, Automatically Tuned Linear Algebra Software (ATLAS), in *Ongoing Projects, The Innovative Computing Laboratory, Distributed Network Computing, Numerical Linear Algebra, Software Repositories, and Performance Evaluation,* http://www.netlib.org/atlas/, Knoxville, Tennessee, USA, 1999.
20. BLAS (Basic Linear Algebra Subprograms), in *Ongoing Projects, The Innovative Computing Laboratory, Distributed Network Computing, Numerical Linear Algebra, Software Repositories, and Performance Evaluation,* http://www.netlib.org/blas/, Knoxville, Tennessee, USA, 1999.

Appendix A

Recursive Cholesky Factorization Subroutine

```
RECURSIVE SUBROUTINE RPOTRF( A, UPLO, INFO )
    USE LA_PRECISION, ONLY: WP => DP
    USE LA_AUXMOD. ONLY: ERINFO, LSAME
    USE F90_RCF, ONLY: RCF => RPOTRF, RTRSM, RSYRK
    IMPLICIT NONE
    CHARACTER(LEN=1), OPTIONAL, INTENT(IN) :: UPLO
    INTEGER, OPTIONAL, INTENT(OUT) :: INFO
    REAL(WP), INTENT(INOUT) :: A(:,:)
    CHARACTER(LEN=*), PARAMETER :: SRNAME = 'RPOTRF'
    REAL(WP), PARAMETER :: ONE = l.0_WP
    CHARACTER(LEN=1) :: LUPLO; INTEGER :: N, P, LINFO
    INTEGER, SAVE :: IC = 0, NMAX = 0
      N = SIZE(A,1); LINFO = 0; IF( NMAX == 0 )NMAX = N
      IF( PRESENT(UPLO) )THEN; LUPLO = UPLO
        ELSE; LUPLO = 'U'; ENDIF
      IF( N < 0 .OR. N / = SIZE(A,2) )THEN; LINFO = -1
      ELSE IF( .NOT. (LSAME(LUPLO,'U').OR.LSAME(LUPLO,'L')) )THEN LINFO = -2
      ELSE IF (N = = 1) THEN: IC = IC + 1
        IF( A(l,l) > 0.0_WP )THEN: A(1,1) = SQRT(A(1,1))
          ELSE; LINFO = 1C: ENDIF
      ELSE IF( N > 0 )THEN; P=N/2
        IF( LSAME(LUPLO,'L') )THEN
          CALL RCF( A(1:P,1:P), LUPLO, LINFO )
          IF( LINFO == 0 )THEN
            CALL RTRSM( A(1:P,1:P), A(P+1:N,1:P), UPLO=LUPLO, &
```

```
            SIDE='R', TRANSA='T' )
          CALL RSYRK( A(P+1:N,1:P), A(P+1:N,P+1:N), ALPHA=-ONE, &
              UPLOC=LUPLO )
          IF( LINFO == 0 )CALL RCF( A(P+1:N,P+1:N), LUPLO, LINFO )
        ENDIF
      ELSE
        CALL RCF( A(1:P,1:P), LUPLO, LINFO )
        IF( LINFO == 0 )THEN
          CALL RTRSM( A(1:P,1:P), A(1:P,P+1:N), TRANSA='T' )
          CALL RSYRK( A(1:P,P+1:N), A(P+1:N,P+1:N), ALPHA=-ONE, &
              TRANSA='T'
          IF( LINFO == 0 ) CALL RCF( A(P+1:N,P+1:N), LUPLO, LINFO)
        ENDIF
      ENDIF
    ENDIF
    IF( NMAX == N )THEN: NMAX = 0; IC - 0; ENDIF
    CALL ERINFO( LINFO, SRNAME, INFO )
END SUBROUTINE RPOTRF
```

PARALLEL GLOBAL OPTIMIZATION ALGORITHMS*

Yaroslav D. Sergeyev

University of Calabria, ITALY
and University of Nizhni Novgorod, RUSSIA

Abstract

In this paper two types of global optimization problems with the objective function determined over a hyperinterval are considered. The first one is the class of Lipschitz functions and the second one is its subclass: functions having Lipschitz first derivatives. For solving these problems we use sequential and parallel characteristical algorithms with and without derivatives. Convergence conditions and conditions, which guarantee significant speed up in comparison with the sequential versions of the parallel methods, are investigated from a general viewpoint. Numerical experiments executed on parallel computers illustrate performance of the parallel methods.

* This work was supported by the grant № 01-01-578 of Russian Fund of Basic Research.

This paper deals with the following well known global optimization problem of finding the global minimum of a function $\Phi(z)$ of N variables over a hyperinterval D:

$$\Phi(z^*) = \min\{\Phi(z) : z \in D\}, \tag{1}$$

$$D = \{z \in R^N : a_j \le z_j \le b_j,\ 1 < j < N\}, \tag{2}$$

where $\Phi(z)$ is multiextremal. If $\Phi(z)$ is continuous then, for the function

$$\phi(x) = \Phi(z(x)),\ x \in [a,b], \tag{3}$$

where $z(x)$ is the continuous Peano-type mapping of closed interval [a, b] onto the hyperinterval D, we have

$$\min\{\phi(x) : x \in [a,b]\} = \min\{\Phi(z) : z \in D\}.$$

Therefore, solving the multidimensional problem (1), (2) is reduced to solving the one-dimensional problem

$$\phi(x^*) = \min\{\phi(x) : x \in [a,b]\}, \tag{4}$$

In addition, if $\Phi(z)$ is Lipschitzean with a constant $K > 0$ then $\phi(x)$ satisfies the Hölder condition

$$|\phi(x') - \phi(x'')| \le H\,|x' - x''|^{1/N},\ x', x'' \in [a,b], \tag{5}$$

with a constant $H \ge 0$ (Hölder constant). For $N = 1$ we will also consider the problem (4) where the first derivative $\phi(x')$ satisfies the Lipschitz condition

$$|\phi'(x') - \phi'(x'')| \le L\,|x' - x''|,\ x', x'' \in [a,b], \tag{6}$$

For solving the problems (4)–(5), (4)–(6) it is proposed to use *parallel characteristical algorithms* generalizing sequential characteristical algorithms. A global optimization algorithm is called a parallel characteristical algorithm if trial points (i.e. that ones where we evaluate $\phi(x)$ and for the problem (4)–(6) also $\phi'(x)$) are chosen according to the following rules.

Trials of the first $n \ge 1$ iterations are performed in arbitrary

$$K = k(n) = p(1) + p(2) + \ldots + p(n)$$

points of the interval $[a, b]$, where $p(i)$, $i \ge 1$, denotes the number of trials of the i-th iteration. Trial points corresponding to any next Q-th iteration, $Q > n$, are chosen according to the rule:

Step 1. Points of the set

$$X_k = \{x^1, \ldots, x^k\} \cup \{a\} \cup \{b\} \tag{7}$$

including the boundaries of the interval $[a, b]$ and coordinates x^j, $1 \le j \le k$, of previous trials, where

$$k = k(Q-1) = p(1) + \ldots + p(Q-1)$$

are renumbered (by subscripts) in the order of increasing the coordinates

$$a = x_0 < x_i < \ldots < x_\tau = b$$

where $\tau+1 = \tau(Q)+1$ is quantity of different elements of the set X_k from (7);

Step 2. A real number $R(i)$ is assigned to each subinterval (x_{i-1}, x_i), $1 \le i \le \tau$, where $R(i)$ is called the *characteristic* of this subinterval;

Step 3. Characteristics $R(i)$ of the subintervals (x_{i-1}, x_i), $1 \le i \le \tau$, are ordered by decreasing

$$R(i_1) \ge R(i_2) \ge \ldots \ge R(i_\tau); \qquad (8)$$

Step 4. The next p trials of the Q-th iteration are performed in parallel on p processors at the points of the set

$$T(Q) = \{x^{k+1}, \ldots, x^{k+p}\}$$

where

$$x^{k+q} = S(i_q)$$

and i_q, $1 \le q \le p$, are the first p indices from the series (8) and the function S is such that

$$x^{k+q} \in (x_{i_q}, x_{i_q}) .$$

In this case it is supposed that

$$p = p(Q) \le \tau, \ Q > n.$$

Many sequential methods proposed for solving the one-dimensional problems (4)–(5) and (4)–(6) can be generalized to the parallel case by using the scheme introduced above

Let us consider a parallel characteristical algorithm (PA) and its sequential analog (SA) obtained by setting $p = 1$. Efficiency of PA in comparison with SA strongly depends on the number $p > 1$ of parallel processors used for parallelization. By increasing p we try to obtain speed up in p times. It is possible only if the number of trials executed by PA is equal to the number of trials done by SA. But these numbers can be different because of the following reason.

In SA the decision about the choice of a new trial point x^{k+1} is based on the complete information

$$w^k = \{x^1,\ldots, x^k; z^1,\ldots, z^k; z'^1,\ldots, z'^k\}$$

available after executing a trial at the point x^k, where $z^i = \phi(x^i)$, $z'^i = \phi'(x^i)$, $1 \le i \le k$. At the same time in PA p points are chosen on the base of the same information w^k and the data

$$\{x^{k+1},\ldots, x^{k+j-1}; z^{k+1},\ldots, z^{k+j-1}; z'^{k+1},\ldots, z'^{k+j-1}\}$$

are absent at the moment of the choice of x^{k+j}, $2 \le j \le p$. Thus, the choice of the points $x^{k+2},\ldots, x^{k+p}$ by PA is not optimal (from the informative viewpoint) in comparison with SA. This fact can cause appearance of redundant (with respect to the sequential search) points in the sequence $\{y^u\}$ of the trial points produced by PA and to slow down the search. To describe the quantity of such points we introduce the *redundancy coefficient*

$$K(u, n) = K'(u, n)/(u - n),$$

where $u > n$ and

$$K'(u, n) = card(\{y^{n+1},\ldots, y^n\} \setminus \{x^k\}).$$

Here, $\{x^k\}$ is the sequence of trial points produced by SA. $K'(u, n)$ is the number of redundant points from $\{y^u\}$ starting from the $(n + 1)$-th to the u-th trial. This definition requires fulfillment of the inclusion $\{x^k\} \subset \{y^u\}$. The case $\{x^k\} = \{y^u\}$ is the best situation with $K(u, n) = 0$: the redundant points have not been produced. This fact implies speed up equal to the number of processors used for parallelization.

On the basis of some additional information about structure of the objective function it is possible to obtain theoretical estimates of $K(u, n)$ and to establish cases where $K(u, n)$ is bounded ensuring high levels of speed up. Wide numerical experiments executed on the parallel computers illustrate performance of the parallel characteristical methods and confirm theoretical results.

HIGH PERFORMANCE COMPUTING : TOOLS AND APPLICATIONS

G. Spezzano and D. Talia

ISI-CNR, Via P. Bucci cubo 41C, 87036 Rende, Italy

Abstract

Computers are more and more fast and powerful, however they are used to solve larger and more challenging problems, thus available computing power is never sufficient. For this reason, high performance computing architectures have been designed in the past twenty years from vector supercomputers to massively parallel parallel computers and cluster computing systems. Both in scientific and commercial computing, high performance computers are required to manage complex tasks and store and analyze the flood of data and information today available. This paper discusses research activities in the area of high performance computing at ISI-CNR. In particular, tools and applications in the areas of parallel cellular automata, parallel data mining, evolutionary algorithms, and grid computing are introduced and discussed.

1. Introduction

Computers are ever more fast and powerful, but their computing power is never considered sufficient because when a new machine faster than the previous ones is developed it is used to solve larger and more challenging problems. For this reason, high performance computing architectures have been designed in the past twenty years starting from vector supercomputers to massively parallel parallel computers and cluster computing systems. Both in scientific and commercial computing, high performance computers are effectively used to manage complex tasks and store and analyze the flood of data and information today available.

The demand for computing both in terms of volume of information and the variety of processing is ever increasing. Many previously infeasible systems are expected naturally today. Basic physical limitations on electronic circuits have started to force high performance computations to be targeted toward the exploitation of parallelism. Just as the fastest cycle times are approaching the fundamental barriers imposed by physical device limitations, different kinds of parallel and distributed systems are emerging. It's

only a matter of time before this trend will affect general purpose computing. A step in that direction can be represented by cluster computers built using off-the-shelf components based on PC technology.

Parallel computing systems more and more demonstrate their effectiveness as tools that deliver high-performance in solving complex problems. However, often, parallel application developers are concerned with low-level issues, such as synchronization, process mapping, and architecture details, when they use parallel computers. These issues mainly arise from the different architectural features that characterize parallel computers. These architectures are a class of systems including many different architectures, from single-instruction-multiple-data (SIMD) machines to distributed-memory, multiple-instruction-multiple-data (MIMD) computers and workstation clusters. This architectural heterogeneity complicates the task of designers of parallel applications and leads to the development of parallel software that lacks portability. Parallelism, either at the architecture or algorithm level, is a significant departure from traditional sequential computation. The sequential approach attempts to reduce processing time by carrying out each operation faster and by eliminating redundancies in computation. In parallel computing, we save time by identifying concurrently executable subtasks in a given task and executing them on multiple processors simultaneously. The first step in the development of a parallel program is the identification of sections that can be executed in parallel and how these should be parallelized. But, unlike the von Neumann model for sequential computing, there is no general purpose model for parallel computation thus different approaches must be followed depending on the parallel model and target architecture.

Languages, tools, and environments for parallel computing are very critical for the development of high-performance applications on parallel machines. They allows users to exploit the computational power of scalable computer systems in solving very demanding applications that have large computation and memory requirements. Among several critical research areas in high-performance computing, we discuss here some of those that are today very interesting and promising and in a near future will play a major role in scientific and commercial applications of parallel computers. These areas concern with

high-performance simulation of complex systems based on cellular automata, knowledge discovery on large data sets using parallel data mining

algorithms, parallel genetic programming and grid computing. These are some research topics that are investigated at ISI-CNR in the area of high performance computing.

2. Cellular Automata and Computational Simulation

Cellular Automata (CA) are discrete dynamical systems and, at the same time, are parallel computing abstract models. For this reason CA can be efficiently implemented on parallel computers. When CA are executed on a parallel computer, they allow a user to exploit their inherent, parallelism to support the efficient simulation of complex systems that, can be modeled by a very large number of simple elements (cells) with local interaction only. In the last decade, several years after their definition, with rapid advances in development of high performance architectures, CA have increasingly utilized for more general computer simulation and modeling. The CA model can be effectively used both as a realistic approach to define abstract parallel machines and as a programming methodology for computational science on parallel computers.

From the marriage of cellular automata and MIMD parallel computing was born CAMELot (Cellular Automata environment for systems modeling open technology), a software environment based on the cellular automata model which has been originally implemented on a transputer-based multicomputer and now is available on a large set of parallel architectures (from massively parallel machine to PC clusters). CAMELot is a tool designed to support the development of high-performance applications in science and engineering. It offers the computing power of a parallel computer although hiding the architecture issues to a user. It provides an interface which allows a user to dynamically change the parameters of the application during its running, and a graphical interface which shows the application output. CAMELot has been successfully used for solving many problems such as the simulation of the lava flows and other fluid flow models, for image processing, freeway traffic simulation, and soil bioremediation. The results achieved in terms of thoroughness of simulation and execution speedup show that CAMELot might be used for simulation and modeling of real systems in many areas in a simple way and achieving high performance.

The system provides a high-level programming language, called CARPET, for programming cellular algorithms. The main features of CARPET are the possibility to describe the state of a cell as a set of typed

substates and the definition of complex neighborhoods, that can be also time dependent, in a discrete cartesian space. The portable version of CAMELot has been developed in the COLOMBO (Parallel COmputers improve cLean up of sOils by Modelling BiOremediation) project within the ESPRIT framework [26].

The main goal of CAMELot is to integrate computation, visualization and control into a single environment that allows interactive steering of scientific applications. CAMEL consists of :

- a parallel run-time support for the CARPET language;
- a graphical user interface (GUI) for editing, compiling, configuring, executing and steering the computation ;
- a tool for the interactive visualization of the results.

The run-time support is implemented as a SPMD (Single Program, Multiple Data) program. The latest implementation is based on the C language plus the standard MPI library and can be executed on different parallel machines such as the Meiko CS-2, the CRAY T3E and a cluster of workstations. The parallel program that implements the architecture of the system is composed by set. of macrocell processes, a controller process and a GUI process. Each macrocell process, that operates on a strip of cells of the CA, runs on a single processing element of the parallel machine and executes the updating of the state of cells belonging to its partition. The synchronization of the automaton and the execution of the commands, provided by a user through the GUI interface, are carried out by the controller process. MPI primitives handle all the communications among the processes using MPI communicators.

To improve the performance of applications that have a diffusive behavior such as CFD, CAMELot implements a load balancing strategy for mapping lattice partitions on the processing elements [27]. This load balancing strategy is a domain decomposition technique similar to the scattered decomposition technique. By the CAMELot GUI, a user can assign colors to cell substates values and visualize on the fly, on different windows, the substates values, thus she/he can follow the evolution of the simulation and steer the computation. CAMELot provides the development environment and the runtime system for the parallel execution of CARPET programs.

CARPET (CellulAR Programming EnvironmenT) is a high-level language related to C with some additional constructs to describe the rules of

the transition function of a single cell of a cellular automaton [28]. CARPET supports researchers in the development of models of discrete dynamic systems by a set of constructs that allow to translate a model expressed in the CA formalism into a cellular algorithm. A CARPET programmer can define both standard CA and generalized CA. It describes the rules of the state transition function of a single cell of a cellular automaton. CARPET defines the structure of CA as a lattice in a discrete Cartesian space in which the maximum number of dimensions allowed is equal to 3. Each dimension is wrapped around, e.g. a two-dimensional lattice forms a torus. Border functions can be used to disable the wrap-around. The size of the lattice is defined by the graphical user interface (GUI) of the CAMELot system and is only bounded by the amount of available memory. The radius of the neighborhood of a cell is statically defined and is strictly connected to the dimension of an automaton. It allows a user to define the maximum numbers of cells which can compose the neighborhood of a cell.

To improve the readability of a program and extend the range of the applications that can be coded as cellular algorithms, the state of a cell in CARPET is structured as a record in which the C basic types : char, shorts, integers, floats, doubles and mono-dimensional arrays of these types can be used to define the substates that represent the physical quantities of a system. Furthermore, the substates are also used to store values that specify statistical properties of a variable or to hold a history of a substate. The predefined variable cell refers to the current cell in the d-dimensional space under consideration. A substate can be referred appending to the reserved word *cell* the substate's name by the underscore symbol '_' (i.e, *cell-substate*). CARPET generalizes the concept of neighborhood and allows a user to define a logic neighborhood that describes both regular neighborhood (i.e., von Neumann, Moore) and asymmetrical neighborhoods with different topological properties (i.e., hexagonal) that can be time-dependent. A neighborhood is defined associating it the name of a vector whose elements compose the logic neighborhood and defining a name for each neighboring cell. Both the name of the vector and names of neighbor cells can be used to refer to the value of a cell substate (i.e., identifier[i]_substate, North_substate). Finally, global parameters can be defined in CARPET to model global characteristics (e.g., porosity of a soil, number of Reynolds) of a complex system. As an example, figure 1 shows the declaration part of a CARPET program that defines a two-dimensional automaton. The radius is

equal to 1, the state is composed of two integer substates and one float substate. The Moore neighborhood and two global parameters are defined. These declarations are contained into a *cadef* section at the beginning of a CARPET program.

```
cadef
{
   dimension 2;
   radius 1 ;
   state ( int water, humidity ; float pollution) ;
   neighbor Moore [8] ( [0,–1] North,  [1,–1] NorthEast,  [1,0] East,
                        [1,1] SouthEast,  [0,1] South, [–1,1] SouthWest ,
                        [–1,0] West, [–1,–1] NorthWest);
   parameter (porosity 0.003, permeability 0.4);
}
```

Fig. 1. The CA declaration part

The new value of the substate does take effect from the next iteration. To allow a user to define spatially inhomogenous CA, CARPET defines the *GetX, GetY,* and *GetZ* operations that return the value of the X, Y and Z coordinate of a cell in the automaton. Furthermore, by means of the predefined variable *step,* that contains the number of iterations that have been executed, the language supports the implementation of neighborhoods and transition functions that are time-dependent. The example in figure 2 describes how the CARPET constructs are used to implement the classical life program in a simple and very expressive way. In particular, the *update* statement assigns a new value to the cell state.

CAMELot has been used for the implementation of a large number of applications in different areas. In the scientific and engineering area have been developed simulations of lava flow [29], freeway traffic [30], landslides [31], soil bioremediation [32], gas diffusion, and ising. Applications of artificial life developed by CAMELot are ant colony, flocking and Greenberg-Hastings model [33]. Finally, CAMELot has been used to solve artificial intelligence problems by implementing cellular genetic algorithms. In the [34] is given an outline of a few of these applications.

```
#define alive 1
#define dead 0
cadef {
    dimension 2;       /* bidimensional lattice */
    radius 1;
    state (short life);
    neighbor Moore[8]([0,–1] North, [–1,–1] NorthWest, [–1,0] West,
                      [–1,1]Southwest, [0,1]South, [1,1] SouthEast,
                      [1,0] East, [1,–1] NorthEast );
}
int i; short sum;
{
      /* computing the number of alive neighbors of cell (x,y) */
for (i = 0; i < 8; i++)
  sum = Moore_life[i] + sum;
      /* computing the new state of cell (x,y) */
if ((sum == 2 && cell_life ==1) || sum == 3)
    update (cell_life,alive);
else
    update (cell_life,dead);
}
```

Fig. 2. The game of life algorithm in CARPET.

3. Parallel Genetic Programming

Genetic programming (GP) is a powerful tool to solve problems coming from different application domains such as hard function and combinatorial optimization, neural nets evolution, routing, planning and scheduling, management and economics, machine learning, data mining, robotics and pattern recognition.

GP finds its origin and inspiration from genetic algorithms (GAs) [16] are a class of adaptive general-purpose search techniques inspired by natural evolution. A standard GA works on a population of elements (*chromosomes*), generally encoded as bit strings, representing a randomly chosen sample of candidate solutions to a given problem. A GA is an iterative procedure that maintains a constant-size population of individuals. Each member of the population is evaluated with respect to a fitness function. At each

step a new population is generated by selecting the members that have the best fitness. In order to explore all the search space, the algorithm alters the selected elements by means of genetic operators to form new elements to be evaluated. The most common genetic operators are *crossover* and *mutation.* Crossover combines two strings to produce new strings with bits from both, thereby producing new search points. Through crossover the search is biased towards promising regions of the search space. Mutation flips string bits at random if a probability test is passed; this ensures that, theoretically, every portion of the search space is explored.

GP is a variation of genetic algorithms in which the evolving individuals are themselves computer programs instead of fixed length strings from a limited alphabet of symbols [19]. Programs are represented as trees with ordered branches in which the internal nodes are *functions* and the leaves are so-called *terminals* of the problem. Thus, the user of GP has to make an intelligent choice of set of functions and *terminals* for the problem at hand. GP conducts its search in the space of computer programs.

The GP approach evolves a population of trees by using the genetic operators of *reproduction, recombination* and *mutation.* Each tree represents a candidate solution to a given problem and it is associated with a fitness value that reflects how good it is, with respect to the other solutions in the population. The reproduction operator copies individual trees of the current population into the next generation with a probability proportionate to their fitness (this strategy is also called roulette wheel selection scheme). The recombination operator generates two new individuals by crossing two trees randomly chosen nodes and exchanging the subtrees. The two individuals participating in the crossover operation are again selected proportionate to fitness. The mutation operator replaces one of the nodes with a new randomly generate subtree. Koza [19] typically does not use a mutation operator in his applications; instead he uses initial populations that are presumably large enough to contain a sufficient diversity. Thus, the success of GP often depends on the use of a population of sufficient size. The choice of the size is determined by the level of complexity of the problem. When applied to large hard problems GP performance may thus drastically degrade because of the computationally intensive task of fitness evaluation of each individual in the population.

Fortunately, genetic programming is well suited to be implemented on parallel architectures because the population can be distributed across the

nodes of the system [18]. One of the main problems in parallelising GP comes from the global selection of individuals, proportionate to their fitness, both in the reproduction and recombination steps. This kind of selection forces the sharing of the new solutions until the new population can be chosen. Two main approaches to parallel implementations of GP have been proposed to avoid this bottleneck. The *island* model [21] and the *cellular* model [23].

The island model divides the population into smaller subpopulations. A standard genetic programming algorithm works on each partition, and is responsible for initializing, evaluating and evolving its own subpopulation. The standard GP algorithm is augmented with a migration operator that periodically exchanges individuals among the subpopulations.

In the cellular model each individual is associated with a spatial location on a low-dimensional grid. The population is considered as a system of active individuals that interact only with their direct neighbours. Different neighbourhoods can be defined for the cells. The most common neighbourhoods in the two-dimensional case are the 4-neighbour (von Neumann neighbour) consisting of the North, South, East, West and 8-neighbour (Moore neighbourhood) consisting of the same neighbours augmented with the diagonal neighbours. Fitness evaluation is done simultaneously for all the individuals and selection, reproduction and mating take place locally with the neighbourhood. Information slowly diffuses across the grid giving rise to the formation of semi-isolated niches of individuals having similar characteristics. The choice of the individual to mate with the central individual and the replacement of the latter with one of the offspring can be done in several ways.

Folino, Pizzuti and Spezzano developed a tool for parallel genetic applications, called CAGE (CellulAr GEnetic programming tool) [15] , that realizes a fine-grained parallel implementation of genetic programming on distributed-memory parallel computers. Our implementation is based on the cellular model. The cellular model is fully distributed with no need of any global control structure and is naturally suited to be implemented on parallel computers. It introduces fundamental changes in the way GP works.

In the model, the individuals of the population are assigned to a specific position in a large toroidal two-dimensional grid and the selection and mating operations are performed, cell by cell, only over the individual assigned to a cell and its neighbors. This local reproduction has the effect of intro-

ducing an intensive communication among the individuals. The massive communication provides a way to disseminate good solutions across the entire population but influences negatively the performance of the parallel implementation of the GP. Moreover, unlike genetic algorithms, where the size of individuals is fixed, the genetic programs are individuals of varying sizes and shapes. This requires a large amount of local memory and introduces an unbalanced computational load per grid point. Therefore, both an efficient representation of the program trees must be adopted and a load balancing algorithm must be employed to maintain the same computational load among the processing nodes.

The best way to overcome the problems associated with the implementation of the cellular model on a general purpose distributed-memory parallel computer is that to use a partitioning technique based upon domain decomposition in conjunction with the *Single-Program-Multiple-Data* (SPMD) programming model. According to this model, an application on N processing elements (PEs) is composed of N similar processes, each one mapped on a PE that operates on a different set of data. For an effective implementation, data should be partitioned such that communication takes place locally and the computation load is shared among the PEs in a balanced way. This approach increases the granularity of the cellular model transforming it from a fine-grained model to a coarse-grained model. In fact, instead of assigning only one individual to a processor, the individuals are grouped by slicing up the grid and assigning a slice of the population to a node.

CAGE implements the cellular GP model using a one-dimensional domain decomposition (in the x direction) of the grid that contains the population of the genetic programs and by using explicit message passing to exchange information between these domains. This decomposition is more efficient than a two-dimensional decomposition, since in the 2-D case twice the number of messages are sent, which reduces the performance since message startup times are rather long on the current commercially available parallel machines.

The concurrent program which implements the architecture of CAGE is composed by a set of identical *slice* processes. No coordinator process is necessary because the computation model is completely decentralized. Each *slice process,* which contains a strip of elements of the grid, runs on a single processing element of the parallel machine and executes on each grid point

the code, show in figure 3, that updates all the individuals of the sub-population.

```
1. Read from a file the configuration parameters
2. Generate a random sub-population
3. Evaluate the individuals of the sub-population
4. while not numGenerations do
5.     update boundary data
6.     for x =1 to length
7.         for y =1 to height
8.             select an individual k (located at position [x',y']) neighboring with i (located at position [x,y]);
9.             generate offspring from i and k ;
10.            apply the user-defined replacement policy to update i;
11.            mutate i with probability pmut;
12.            evaluate the individual i;
           end for
       end for
   end while
```

Fig. 3. Pseudocode of the slice process.

Each slice process uses the parameters read from a file to configure the genetic programming algorithm performed on each grid point. The parameters regard the population size, the max depth that the trees can have after the crossover, the parsimony factor, the number of iterations, the number of neighbors of each individual, and replacement policy. CAGE implementes three replacement policies: *direct* (the best of the offspring is always replaced to the current individual), *greedy* (the replacement occurs only if offspring is fitter), *probabilistically* (the replacement happens according to fitness difference between parent and offspring (*simulated annealing*)).

The size of the subpopulation of each slice process is calculated dividing the population for the number of the processors on which CAGE is executed. Each slice process updates sequentially the individuals belonging to its subgrid. At the start, in each process, is generated a random subpopulation and the fitness is evaluated. After, for *numGeneration* iterations the loop for generating the new subpopulation is performed. The variables

length and *height* define the boundaries of the 2D subgrid that is contained in a process. It should be noted that two copies of the data are maintained for calculating the new population. In fact, as each element of the current population is used many times the current population cannot be rewritten.

CAGE uses the standard tool for genetic programming *sgpc* [25] to apply the GP algorithm to each grid point. However, in order to meet the requirements of the cellular GP algorithm, a number of modifications have been introduced.

The same data structure of *sgpc* is used to store a tree in each cell. The structure that stored the population has been transformed from a one-dimensional array to a two-dimensional array and we duplicated this structure in order to store the current and the new generated tree. The selection procedure has been replaced with one that uses only the neighborhood cells and three replacement policies arc added. The crossover operator accepts in input the current tree and the best tree in the neighborhood. Two procedures to pack and unpack the trees that must be sent to the other processes have been added. The pack procedure is used to send in a linearized form the trees of the boundary data to the neighbor processes. Data are transmitted as a single message in order to minimize the message startup and transmission time. The unpack procedure rebuilds the data and stores it in the new processor's private address space.

The execution of a parallel program is composed of two phases: computation and communication. During the computation phase each process of the concurrent program that implements the run-time support executes computations which only manipulate data local to this process. These data can be local variables or boundary data received from neighbor processes. The effective data transmission between processes is done at the end of each local computation. This means that, during computation phase, no data are exchanged.

The parallel implementation has been realized on a multicomputer Meiko CS-2. In [15] we present the convergence results obtained with *CAGE* and compare them with the sequential canonical *GP* and the island model implementation realized by [22] and [24]. *sgpcl.*1 is used as the canonical sequential implementation of genetic programming. Preliminary experimental results shows better performances of tins approach. We are planning an experimental study on a wide number of benchmark problems to substantiate the validity of the cellular implementation.

4. Parallel Data Mining

Research and development work in the area of knowledge discovery and data mining concerns the study and definition of techniques, methods and tools for the extraction of novel, useful and not explicitly available patterns from large volumes of data. In particular, data mining is the main step of the knowledge discovery process (KDD) that consists of several steps from data extraction to models and patterns interpretation (fig. 4) [1]. Data mining techniques originated from the use of statistical analysis and machine learning techniques in the mining of patterns from databases. In the latest few years new techniques and algorithms have been designed.

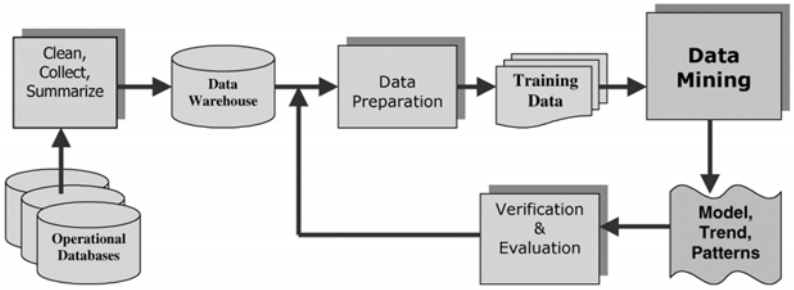


Fig. 4. Description of the KDD process

Today the information overload is a problem like the shortage of information. Knowledge discovery in large data repositories can find what is interesting in them representing it in an understandable way [2]. Mining large data sets requires large computational resources. In fact, data mining algorithms working on very large data sets take very long times on conventional computers to get results. One approach to reduce response time is sampling. But, in some case reducing data might result in inaccurate models, in some other case is not useful (e.g. outliers identification). The other approach is parallel computing. High performance computers and parallel data mining algorithms can offer the best way to mine very large data sets [3] [4]. Is not uncommon to have sequential data mining applications that require several days or weeks to complete their task. Parallel computing systems can bring significant benefits in the implementation of data mining and knowledge discovery applications by means of the exploitation of inherent parallelism of data mining algorithms.

Main goals of the use of parallel computing technologies in the data mining field are:

- performance improvements of existing techniques,
- implementation of new (parallel) techniques and algorithms, and
- concurrent analysis with different data mining techniques and result integration to get a better model (that is more accurate).

There are three main strategies in the exploitation of parallelism in data mining algorithms:

- *independent parallelism,*
- *task parallelism,*
- *SPMD parallelism.*

According to task parallelism (or control parallelism) each process executes different operations on (a different partition of) the data set. In SPMD parallelism a set of processes execute in parallel the same algorithm on different partitions of the data set and processes exchange partial results. Finally, independent, parallelism is exploited when processes are executed in parallel in an independent way; generally each process has access to the whole data set. These three strategies are riot necessarily alternative for parallelizing data mining algorithms. They can be combined to improve both performance and accuracy of results. In combination with strategies for parallelization, different data partition strategies can be used

- *sequential partitioning:* separate partitions are defined without overlapping among them;
- *cover-based partitioning:* some data can be replicated on different partitions;
- *range-based query:* partitions are defined on the basis of some queries that select data according to attribute values.

Parallelization strategies can be applied to different data mining techniques (DM) and algorithms such as classification, association rules, episodes discovery, link analysis and clustering. Among the various DM techniques, one of the most useful is *clustering.* Clustering algorithms try to separate data into several groups so that similar items fall into the same group and similarity between items in different group is low [5]. This partition is performed without any user support so this particular DM task is referred as *unsupervised learning.* Most of the early cluster analysis algorithms have been originally designed for relatively small data sets. In the recent years, clustering algorithms have been extended to efficiently work

on large databases and some of them are able to deal with high-dimensional feature items [6]. When used to classify large data sets, clustering algorithms are very computing demanding (computation may require several days) and require high-performance machines to get results in reasonable time [7] [3].

A well-known clustering algorithm is AutoClass [8], designed by Cheeseman and colleagues at NASA Ames Research Center. The execution of the sequential version of AutoClass on large data sets requires high computational times. For instance, the sequential AutoClass runs on a data set of 14K tuples, each one composed of a few hundreds bytes, have taken more the 3 hours on Pentium-based PC. Considering that the execution time increases very fast with the size of data set, more than 1 day is necessary to analyze a data set composed of about 140K tuples, that is not a very large data set. For clustering a satellite image, AutoClass took more than 130 hours and for the analysis of protein sequences its discovery process required from 300 to 400 hours [9].

These considerations and experiences suggested the necessity of a faster version of AutoClass to handle very large data sets in reasonable time. This has been done by Pizzuti and Talia at ISI-CNR by exploiting the inherent parallelism present in the AutoClass algorithm by implementing it in parallel on MIMD multicomputers [10]. The resulting P-AutoClass algorithm is based on the Single Program Multiple Data (SPMD) approach that works by dividing data among the processors and executing the same operations at each processor using a partition of data. At the end of each local computation partial results (weights for each instance and parameters) are exchanged among the processors to obtain the global result. This strategy does not require to replicate the entire data set on each processor. Furthermore, it also does not produces load balancing problems because each processor executes the same program on data of equal size. Finally, the amount of data exchanged among the processors is not so large since most operations are performed locally at each processor.

The main part of the algorithm is devoted to classification, generation and evaluation of generated classification. This loop is composed of a set of sub-steps, among which the new try of classification step is the most computationally intensive. In AutoClass class membership is expressed by weights. This step computes the weights of each item for each class and then it computes the parameters of classification. These operations are exe-

cuted by the function *base_cycle* which calls the three functions *update_wts, update_parameters* and *update_approximations.* The time spent in the *base_cycle* function and it resulted about the 99.5% of the total time. Therefore, this function has been identified as that one where parallelism must be exploited to speed up the AutoClass performance. In particular, if we analyze the time spent in each of the three functions called by *base-cycle,* it appears that the *update_wts* and *update_parameters* functions are the most time consuming functions whereas the time spent in the *update_approximation* is negligible. Therefore, P-AutoClass is based on the parallelization of these two functions using the SPMD approach. To maintain the same semantics of the sequential algorithm of AutoClass, the parallel version is based on partitioning data and local computation on each of *P* processors of a distributed memory MIMD computer and on exchanging among the processors all the local variables that contribute to form the global values of a classification.

P-Autoclass has been implemented using MPI on parallel computers and clusters. Experimental results shown that P-AutoClass is scalable both in terms of speedup and scaleup. This means that for a given data set, the execution times can be reduced as the number of processors increases, and the execution times do not increase if, while increasing the size of data set, more processors are available. The out-of-core technique is more effective when large data blocks are used to transfer data from external to main memory. Finally, the P-AutoClass algorithm is portable to various MIMD distributed-memory parallel computers that are now currently available from a large number of vendors. It allows to perform efficient clustering on very large data sets significantly reducing the computation times on several parallel computing platforms.

Other research activities in the area of high-performance data mining are performed at ISI-CNR in the design and implementation of a grid-based distributed architecture for knowledge discovery on computational grids, called *Knowledge Grid,* discussed in the next section.

5. Grid Computing

The term *Grid* refers to an infrastructure that enables the integrated, collaborative use of high-end computers, networks, databases, and scientific instruments owned and managed by multiple organizations. Grid applications often involve large amounts of data and/or computing and often re-

quire secure resource sharing across organizational boundaries, and are thus not easily handled by today's Internet and Web infrastructures [l1].

Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is. necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. Tools that implement grid services are emerging and some of these, such as Globus and Legion, are used by many research teams in several countries [12].

Grid computing concepts were first, explored in the 1995 I-WAY experiment, in which high-speed networks were used to connect, for a short time, high-end resources at 17 sites across North America. Out of this activity grew a number of Grid research projects that developed the core technologies for "production" Grids in various communities and scientific disciplines. For example, the US National Science Foundation's National Technology Grid and NASA's Information Power Grid are both creating Grid infrastructures to serve university and NASA researchers, respectively. Across Europe and the United States, the closely related European Data Grid, Particle Physics Data Grid and Grid Physics Network (GriPhyN) projects plan to analyze data from frontier physics experiments. And outside the specialized world of physics, the Network for Earthquake Engineering Simulation Grid (NEESgrid) aims to connect US civil engineers with the experimental facilities, data archives and computer simulation systems used to engineer better buildings.

In the grid computing area, at ISI-CNR we are working in two specific areas: grid programming and knowledge discovery on grids. In this section we describe the research achievements in the latter area, and in particular we discuss the *Knowledge Grid* architecture.

The *Knowledge Grid* architecture, designed by Cannataro and Talia [13], is built on top of a computational grid that provides dependable, con-

sistent, and pervasive access to high-end computational resources. The proposed architecture uses the basic grid services (i.e., the Globus services) and defines a set of additional layers to implement the services of distributed knowledge discovery process on world wide connected computers where each node can be a sequential or a parallel machine. The Knowledge Grid enables the collaboration of scientists that must mine data that are stored in different research centers as well as executive managers that must use a knowledge management system that operates on several data warehouses located in the different company establishments.

The *Knowledge Grid* attempts to overcome the difficulties of wide area, multi-site operation by exploiting the underlying grid infrastructure that provides basic services such as communication, authentication, resource management, and information. To this end, the knowledge grid architecture is organized so that more specialized data mining tools are compatible with lower-level grid mechanisms and also with the Data Grid services. This approach benefits from "standard" grid services that are more and more utilized and offers an open Parallel and Distribute Knowledge Discovery (PDKD) architecture that can be configured on top of grid middleware in a simple way.

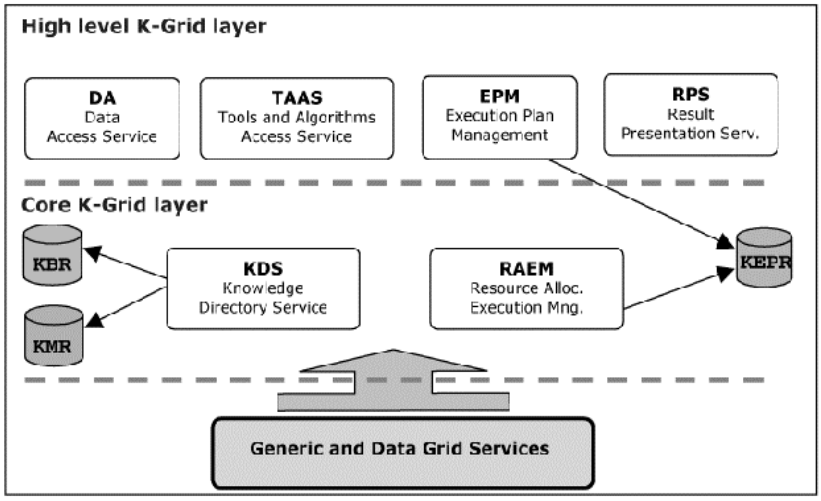


Fig. 5. Layers and components of the Knowledge Grid architecture

The Knowledge Grid services (layers) are organized in two hierarchic levels: *core K-grid layer* and *high level K-grid layer*. The former refers to services directly implemented on the top of generic grid services, the latter are used to describe, develop and execute PDKD computations over the Knowledge Grid (see fig.5).

The core K-grid layer supports the definition, composition and execution of a PDKD computation over the grid. Its main goals are the management of all metadata describing characteristics of data sources, third party data mining tools, data management, and data visualization tools and algorithms. Moreover, this layer has to coordinate the PDKD computation execution, attempting to match the application requirements and the available grid resources. This layer comprises the following basic services:

- Knowledge Directory Service (KDS) responsible for maintaining a description of all the data and tools used in the Knowledge Grid.
- Resource allocation and execution management (RAEM) services used to find a mapping between an execution plan and available resources, with the goal of satisfying requirements (computing power, storage, memory. database, network bandwidth and latency) and constraints.

The high-level K-grid layer comprises the services used to compose, to validate, and to execute a PDKD computation. Moreover, the layer offers services to store and analyze the knowledge discovered by PDKD computations. Main services are:

- Data Access (DA) services that are responsible for the search, selection (Data search services), extraction, transformation and delivery (Data extraction services) of data to be mined.
- Tools and algorithms access (TAAS) services that are responsible for the search, selection, downloading of data mining tools and algorithms
- Execution plan management (EPM) that handles execution plans as an abstract description of a PDKD grid application. An execution plan is a graph describing the interaction and data flows between data sources, extraction tools, DM tools, visualization tools, and storing of knowledge results in the Knowledge Base Repository.
- Results presentation service (RPS) that specifies how to generate, present and visualize the PDKD results (rules, associations, models, classification, etc.). Moreover, it offers the API to store in different formats these results in the Knowledge Base Repository.

This *Knowledge Grid* represents a first step in the process of studying the unification of PDKD and computational grid technologies and defining an integrating architecture for distributed data mining and knowledge discovery based on grid services. We hope that the definition of such an architecture will accelerate progress on very large-scale geographically distributed data mining by enabling the integration of currently disjoint approaches and revealing technology gaps that require further research and development. Currently a first prototype of the system built on top of Globus is available. In particular, Cannataro, Talia and Trunfio have implemented the Knowledge Directory Service and the Knowledge Metadata Repository of the Core K-grid layer, and the Data Access Service of the High level K-grid layer [14].

The metadata describing relevant objects for PDKD computations, such as data sources and data mining software, are represented by XML documents into a local repository (KMR), and their availability is published by entries into the Directory Information Tree maintained by a LDAP server, which is provided by the Grid Information Service (GIS) of the Globus Toolkit. The main attributes of the LDAP entries specify the location of the repositories containing the XML metadata, whereas the XML documents maintain more specific information for the effective use of resources. The basic tools of the DA service have been implemented allowing to find, retrieve and select metadata about PDKD objects on the grid, on the basis of different search parameters and selection filters. Moreover, we are modeling the representation of execution plans as graphs, where nodes represents computational elements (data sources, software programs, results, etc.) and arcs represents basic operations (data movements, data filtering, program execution, etc.). We plan to consider different network parameters, such as topology, bandwidth and latency, for PDKD program execution optimization.

6. Conclusion

Parallel computers are today the more effective tools for the implementation of high performance applications. In scientific, industrial, and commercial computing, parallel computers are critical resources for solving complex problems. Among several significant research areas in high-performance computing, we discussed here some of those that are very promising and in a near future will play a major role in scientific and com-

mercial applications of parallel computers: high-performance simulation of complex systems using cellular automata, knowledge discovery on large data sets using parallel data mining algorithms, parallel genetic programming and grid computing. Algorithms, tools, and applications that have been developed have been discussed and their relevance in the high performance arena have been outlined.

References

1. Fayynd U.M., Piateskv-Shapiro G., Smyth P. From Data Mining to Knowledge Discovery: an Overview. In *Advances in Knowledge Discovery and Data Mining,* pp. 1–34, AAAI MIT Press, 1996.
2. Berry, M. J.A., Linoff G. *Data Mining Techniques for Marketing, Sales, and Customer Support,* Wiley Computer Publishing (1997)
3. Freitas A. A., Lavington S. H. *Mining Very Large Database with Parallel Processing,* Kluwer Academic Publishers (1998).
4. Skillicorn, D. Strategies for Parallel Data Mining, *IEEE Concurrency,* 7, december (1999).
5. Aldenderfer M.S., Blashfield R.K. *Cluster Analisys*. SAGE Publications, Inc., Beverly Hills, California.
6. Chen M.S., Han J., Yu P.S. Data Mining: An Overview from a Database Perspective. In *IEEE Trans. on Knoweldge Discovery and Data Engineering,* Vol. 8, No. 6, pages 866–883, 1996.
7. Stoffel K., Belkoniene A. Parallel K-Means Clustering for Large Data Sets. In Proc. of Euro-Par '99, LNCS 1685, pp. 1451–1454, 1999.
8. Cheeseman P., Stutz J. Bayesian Classification (AutoClass). Theory and Results. In *Advances in Knowledge Discovery and Data Mining,* pages 153–180, 1996.
9. Hunter L., States D.J. Bayesian Classification of Protein Structure. *IEEE Expert,* 7(4):67–75, 1992.
10. Foti D., Lipari D., Pizzuti C., Talia D. Scalable Parallel Clustering for Data Mining on Multicomputer. In *Proc. of the 3rd Int. Workshop on High Performance Data Mining HPDM00-IPDPS,* pages 390–398, May 2000.
11. Foster I., Kesselman C. *The Grid* : *Blueprint for a New Computing Infrastructure,* Morgan Kaufmann Publ. (1999).
12. Foster I., Kesselman C. Globus : a metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications* 11 (1997) pp. 115–128.

13. Cannataro M., Talia D. Parallel and Distributed Knowledge Discovery on the Grid: A Reference Architecture, *Proc. of the 4th International Conference on Algorithms and Architectures for Parallel Computing* (*ICA3PP*)*,* Hong Kong, World Scientific Publ., pp. 662–673, 2000.
14. Cannataro M., Talia D., Trunfio P. The Knowledge Grid: Towards an Architecture for Knowledge Discovery on the Grid, *First EuroGlobus Workshop,* Lecce, 2001.
15. Pizzuti C., Folino G., Spezzano G. *CAGE: A Tool for Parallel Genetic Programming Applications* , Proc. of the 4th European Conf. on Genetic Programming EuroGP2001, Springer-Verlag, LNCS 2038, pp. 64–73, 2001.
16. Michlewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs,* Springer-Verlag, Second Edition, Berlin, 1994.
17. Andre D., Koza J.R. Exploiting the fruits of parallelism: An implementation of parallel genetic programming that achieves super-linear performance. *Information Science Journal,* Elsevier, 1997.
18. Cantú-Paz E. *A summary of research on parallel genetic algorithms,* Technical Report 950076, Illinois Genetic Algorithm Laboratory. University of Illinois at Urbana Champaign, Urbana. July 1995.
19. Koza J. R. *Genetic Programming: On the Programming of Computers by means of Natural Selection,* MIT Press, Cambridge, 1992.
20. Koza J.R., Andre D. Parallel genetic programming on a network of transputers. *Technical Report CS-TR-95–1542, Computer Science Department,* Stanford University, 1995.
21. Martin W.N., Lienig J., Cohoon J.P. Island (migration) models: evolutionary algorithms based on punctuated equilibria, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation.* IOP Publishing and Oxford University Press, 1997.
22. Niwa T., Iba H. Distributed Genetic Programming -Empirical Study and Analyses - *Genetic Programming* 1*996, Proceedings of the First Annual Conference,* MIT Press. Stanford University. July 1996.
23. Pettey C.C. Diffusion (cellular) models, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation,* IOP Publishing and Oxford University Press, 1997.
24. Punch W. F. How Effective are Multiple Populations in Genetic Programming. *Genetic Programming '98, Proceedings of the Third Annual Conference,* MIT Press, University of Winsconsin. July 1998.

25. Tackett W.A., Carmi A. Simple Genetic Programming in C, *Available through the genetic programming archive at ftp::/ftp.io.com/pub/ genetic-programming/code/sgpcl.tar.Z.*
26. Telford S., Smith G., Baracca C., Longo A., Ornelli P., Spezzano G., Talia D. COLOMBO WP3: Design for Portable, Parallel CA Software Environment, *Deliverable D6,* May 1998.
27. Cannataro M., Di Gregorio S., Kongo R., Spataro W., Spezzano G., and Talia D. A Parallel Cellular Automata Environment on Multicomputers for Computational Science, *Parallel Computing ,* 21, 803–824, 1995.
28. Spezzano G., Talia D. A High-Level Cellular Programming Model for Massively Parallel Processing, *Proc.* 2*nd Int. Workshop on High-Level Programming Models and Supportive Environments - HIPS97.* IEEE Computer Society Press, pp. 55–63, 1997.
29. Di Gregorio S., Rongo R., Spataro W., Spezzano G., Talia D. A Parallel Cellular Tool for Interactive Modeling and Simulation, *IEEE Computational Science & Engineering,* 3(3), 33–43, 1996.
30. Di Gregorio S., Festa D.C., Kongo R., Spataro W., Spezzano G., Talia D. A Microscopic Freeway Traffic Simulator on a Highly Parallel System, in *Parallel Computing: State-of-the-Art and Perspectives,* E. D'Hollander et al, eds., pp. 69–76, Elsevier, 1996.
31. Spezzano G. Simulating Parallel Models of Landslides by a Computational Steering Environment, *Proc. of the 18th International Conference on Applied Informatics AI'2000,* IASTED Press, Austria, 2000.
32. Spezzano G., Talia D. Designing Parallel Models of Soil Contamination by the Carpet Language, *Future Generation Computer Systems,* 13(4–5), 291–302, 1998.
33. Talia D. Solving Problems on Parallel Computers by Cellular Programming, *Proc. of the 3rd Int. Workshop on Bio-Inspired Solutions to Parallel Processing Problems BioSP3–IPDPS.* LNCS 1800, Springer-Verlag, 2000.
34. Spezzano G., Talia D. CAMELot: A Parallel Cellular Environment for Modelling Complexity, *AI*IA Notizie,* 2001 (to appear).

СОДЕРЖАНИЕ