Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Санкт-Петербургский государственный университет

ВЫСОКОПРОЗВОДИТЕЛЬНЫЕ ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА КЛАСТЕРНЫХ СИСТЕМАХ

Материалы шестого Международного
научно-практического семинара

Том 2

12–17 декабря 2006 г.

Издательство Санкт-Петербургского госуниверситета
Санкт-Петербург
2007

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

В93 **Высокопроизводительные параллельные вычисления на кластерных системах.** Материалы шестого Международного научно-практического семинара. Том 2. / Под ред. проф. **Р.Г. Стронгина.** Санкт-Петербург: Изд-во Санкт-Петербургского госуниверситета, 2007. 255 с.

Сборник сформирован по итогам научного семинара, посвященного теоретической и практической проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа. Семинар проходил в Санкт-Петербурге 12–17 декабря 2006 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. **И.М. Григорьев**

ББК 32.973.26–018.2:22

Поддержка семинара

Компания Intel Technologies

СИСТЕМА МОНИТОРИНГА ДЛЯ ГРИД

Лабутин Д. Ю., Боголепов Д. К., Алехин А. А.
Нижегородский государственный университет им. Н.И. Лобачевского

В данной статье рассматривается необходимость мониторинга распределенных вычислительных ресурсов, требования к системам мониторинга, потенциальные проблемы при разработке подобных систем. Кроме того, рассматривается система мониторинга Grid Performance Monitoring (GPM), разрабатываемая в учебно-исследовательской лаборатории информационных технологий (ITLab) Нижегородского государственного университета им. Н.И. Лобачевского.

Введение

Для успешного развития грид-технологогий необходимо решить множество проблем. Одной из таких проблем является проблема мониторинга и обнаружения ресурсов. Особенно решение просто необходимо в ситуациях, когда отдельные части системы географически распределены, и, следовательно, ни один человек не в состоянии знать все детали функционирования отдельных компонент. Функция мониторинга позволяет определять и проводить диагностику множества проблем, возникающих в подобных ситуациях. Функция обнаружения позволяет находить в распределенной системе ресурсы или сервисы с заданными свойствами. Оба механизма требуют возможности сбора информации с множества распределенных ресурсов.

В настоящее время существует множество систем, осуществляющих мониторинг вычислительных ресурсов кластера или распределенных систем. Среди них можно отметить такие популярные инструменты, как Ganglia, Supermon, Clumon, Gridmon и другие. Тем не менее, ни одно из существующих решений не решает всех проблем, которые возникают при мониторинге распределенных систем. Проблема создания универсальной, расширяемой и масштабируемой системы мониторинга по-прежнему остается актуальной.

Материал статьи можно условно разбить следующие разделы:

- Назначение системы мониторинга GPM распределенных систем.
- Архитектура системы.
- Текущие исследования и планы на будущее.

Назначение системы мониторинга GPM

Grid Performance Monitoring представляет собой масштабируемую систему мониторинга распределенных ресурсов. Система GPM способна обеспечить пользователей и администраторов довольно богатым набором информации. Информацию, собираемую системой мониторинга, можно условно разделить на две группы. К первой группе относится информация об аппаратном и программном обеспечении кластера или распределенной системы. В эту группу входит информация об используемых процессорах, оперативной памяти, носителях информации, сетевом соединении, установленной операционной системе и дополнительном программном обеспечении. Ко второй группе относится информация о запускаемых задачах. Система мониторинга позволяет собирать о них исчерпывающую информацию. Доступна такая информация, как список узлов, используемых задачей, список процессов на каждом узле, подробная информация о каждом процессе. Таким образом, система мониторинга GPM является законченным решением в вопросе мониторинга ресурсов и запускаемых задач.

Остановимся подробнее на требованиях, которым должна удовлетворять система мониторинга.

- Гибкость собираемой информации

  Для обеспечения гибкости вводится понятие метрики. Метрика – это некая “элементарная”, “неделимая” характеристика ресурса или запускаемой задачи. Примерами метрик служат: количество доступных байт памяти, процент загруженности процессора, версия операционной системы, объем свободного места на диске и т. д. Таким образом, механизм метрик позволяет четко структурировать собираемую информацию.
- Универсальность относительно используемого программного окружения.

  В первую очередь, речь идет о поддержке различных операционных систем, таких как:
  - ОС семейства Microsoft Windows NT;
  - ОС семейства Unix (Linux, Solaris, Mac OS, …).

  Заметим, что данное требование является абсолютно необходимым, поскольку инфраструктура грид может быть

крайне разнородной. Разные ресурсы могут работать не только под различными ОС, но также различаться архитектурой.

- Универсальность относительно программных клиентов
  Система мониторинга не должна быть ориентирована на какое-то конкретное клиентское программное обеспечение. Иными словами, не должно быть жесткой зависимости от конкретных внешних интерфейсов.
- Универсальность относительно систем запуска программ
  Система мониторинга должна предоставлять информацию не только о загрузке системы в целом, но и том, насколько система загружена конкретными грид-задачами. На узлах грид задачи запускаются, как правило, с помощью средств параллельного запуска, таких как MPI, PVM, Cluster OpenMP и т.д., или с помощью систем управления кластерами.
- Клиентская часть с возможностью анализа собранной информации
  Собранная информация должна быть доступна пользователю в удобном для него виде.
  Анализ информации проводится с целью получения экспертных рекомендаций, советов, подсказок и т. д.

В настоящее время система мониторинга GPM удовлетворяет большинству перечисленных выше требований.

Компоненты системы мониторинга GPM

Система мониторинга построена на основе компонент, разделенных по функциональному признаку. К основным компонентам системы GPM относятся:

- Агент (Agent)
  Приложение, которое устанавливается на каждый узел кластера, о котором следует собирать информацию. Приложение отслеживает состояние узла, на котором оно запущено. Причем, собирается информация как об аппаратном обеспечении узла, так и о выполняемых на данном узле задачах. Вся собранная информация отправляется Менеджеру для её сохранения и дальнейшей обработки. Реализация данной компоненты существенно зависит от операционной системы. Таким образом, существуют различные варианты Агента под различные операционные системы.
- Менеджер (Manager)

Основная компонента системы, представляющая собой контроллер, управляющий всеми подключенными Агентами. Менеджер берет на себя вопросы организации работы системы на кластерном уровне, обработки поступающей информации и ее хранения, предоставления информации любому клиентскому программному обеспечению. Данная компонента системы является межплатформенной.

- Клиенты (Client)
  Отличительной особенностью системы мониторинга GPM является развитая клиентская часть. Клиенты системы мониторинга предназначены для получения и анализа информации, запрашиваемой от Менеджера. Существует несколько видов клиентов: Клиент в виде традиционного графического приложения, Клиент в виде набора веб-страниц, простейший Клиент в виде утилиты командной строки. С помощью графического клиента пользователь может анализировать самую различную информацию, строить диаграммы, сохранять результаты в файл.
- Утилиты (Utilities)
  В состав инструментария GPM также входит набор различных утилит, позволяющих управлять работой Менеджера в пределах локальной сети.
- Веб-сервис (Web service)
  В состав системы мониторинга будет входить компонента, реализующая объединение на межкластерном уровне. Так как необходимо обеспечение сетевой безопасности и интероперабельности в условиях сети Internet, то данная компонента будет создана с использованием технологий веб-сервисов. Объединение планируется организовать иерархическим способом, путем подключения менеджеров грид-сегментов к веб-сервисам. Основные потенциальные проблемы – это вопросы обеспечения безопасности и разграничение прав доступа пользователей.

Менеджер (Manager)

Менеджер – это основная часть системы мониторинга, которая обеспечивает объединение компонент в единую систему мониторинга на уровне кластера. Организация системы на межкластерном уровне предполагается с использованием веб-сервисов, так как данная

технология является эффективным механизмом обеспечения внешнего доступа к системе, реализующая механизм сетевой безопасности.

Для разработки *Менеджера* был выбран кроссплатформенный язык Java. К функциям Менеджера относятся:

- управление метриками: добавление (регистрация), добавление новых значений, уничтожение метрик;
- хранение значений метрик;
- обработка значений метрик и выдача запрашиваемой информации.

Регистрация метрик

Для добавления новой метрики необходимо осуществить её регистрацию. При этом указывается следующая информация: имя метрики, тип собираемых данных, интервал обновления значений, а также, если необходима, функция объединения (группировки) значений метрики за некоторый промежуток времени: функции суммирования значений, минимума, максимума и среднего значения;

Регистрация метрик инициируется *Агентом*, которому в ответ выдается «ключ доступа» к зарегистрированной метрике. Необходимость добавления ключа метрики вызвана двумя факторами:

- во-первых, имена метрик достаточно длинные и при обновлении значений метрики было бы необходимо передавать это длинное имя;
- во-вторых, большей важностью обладает корректность переданных значений, и поэтому данный ключ является некоторой системой безопасности, которая не позволит записать значения метрики без знания данного ключа.

Сбор и хранение информации

Хранение всех значений метрик требует очень много ресурсов, поэтому важно предложить такой способ хранения, который бы удовлетворял следующим требованиям:

- во-первых, минимизировать требования к ресурсам;
- во-вторых, не потерять важную информацию.

С учетом данных требований в Менеджере реализована многоуровневая схема хранения значений. Интервал между соседними значениями увеличивается в зависимости от времени их хранения. Таким образом, значения с интервалом в одну секунду хранятся за

последний час, с интервалом в 30 секунд – за сутки, 15 минут – за две недели, 6 часов – за год, 1 значение в день – за 10 лет.

Итого на хранение значений одной метрики за 10 лет нужно ~11,5 тысяч значений, что намного меньше числа секунд в этих 10 годах.

К преимуществам данного подхода можно отнести:

- простота реализации;
- фиксированные требования к ресурсам хранения;
- низкие затраты на обработку значений из-за их небольшого количества.

К недостаткам следует отнести невозможность получить изменение данной метрики за небольшой промежуток времени в далеком прошлом: например, как изменялись значения метрики в течение часа год назад.

Запросы к Менеджеру

Система мониторинга должна не только собирать и хранить информацию, но и выдавать её различным клиентам. Здесь возникает проблема выбора формата предоставления данных. Ведь существенные различия среди возможных клиентов не позволяют наложить ограничения на формат выходных данных в целом. Поэтому было принято решение сделать формат динамически формируемым, собирая его из фиксированных базовых блоков, таких как числа, константы, совокупность значений метрики и т.д.

Алгоритм сбора данных описывается с помощью языка запросов, основанном на языке XML. Менеджер, получив запрос, сформирует ответ в соответствии с описанным в запросе форматом.

Агент (Agent)

Сбором значений и их отправкой занимается компонента под названием *Агент*, которая должна быть запущена на всех компьютерах системы. Реализация данной компоненты зависит от конкретной платформы, так как большинство информации берется с использованием функций операционной системы. В качестве языка программирования был выбран язык C++.

Функции *Агента*:

- регистрация отслеживаемых метрик;
- получение значений метрик;
- отправка собранных значений Менеджеру;

- отслеживание запуска задач.

Получение значений метрик

Изучив совокупность возможных конфигураций оборудования и программного обеспечения, был выбран набор основных внутренних метрик, измерением значений которых занимается Агент:

- информация о процессорах, об использовании физической и виртуальной памяти и сетевых адаптеров;
- информация об отслеживаемых процессах: использование процессора, памяти, устройств ввода/вывода.

В данный список не включена информация о загрузке локальных дисков, так как обычно основным устройством хранения в кластерах является специально выделенный сервер, доступ к которому осуществляется по сети. Долговременное хранение информации на отдельных узлах кластера не предусмотрено.

Расширение данного списка метрик можно осуществить двумя способами:

- модификацией исходного кода соответствующего Агента, заключающаяся в реализация определенного интерфейса.
- с помощью внешних модулей в виде скриптов (bat или bash), которые выводят информацию о значениях на стандартный поток вывода (stdout).

Отслеживание запуска задач

Большинство задач, исполняемых в распределенных системах, являются параллельными программами и их выполнение необходимо отслеживать. Поэтому нужно предложить легко выполнимые требования на процедуру запуска задачи, для того чтобы система мониторинга могла выделять те процессы, по которым необходимо собирать подробную информацию. Существует несколько возможных вариантов, например:

- уведомление локальных *Агентов* через сеть. При этом нужно, что бы к агенту был сетевой доступ (напрямую или через прокси), что влечет за собой некоторую потенциальную угрозу безопасности системы;
- задавать некоторую переменную окружения у запущенных процессов (например, JOBID) с идентификатором задания:

данный способ легко реализуем со стороны систем запуска задач.

В настоящее время реализован второй вариант.

Заключение и планы на будущее

Использование систем мониторинга необходимо в условиях эффективного использования вычислительных ресурсов в распределенных сетях. Разработка универсальной, легко расширяемой системы, способной обработать сложные запросы является востребованной задачей.

Система поддерживает операционные системы семейства Microsoft Windows NT и Linux, ведется отслеживание задач, запускаемых пользователями, имеется возможность предоставления информации различными способами и отображение её собственными средствами.

На данный момент система GPM проходит тестирование на кластере Нижегородского государственного университета им. Н.И. Лобачевского. В настоящее время проводится интеграция с системой управления кластером, разрабатываемой в Нижегородском государственном университете им. Н.И. Лобачевского.

Отметим некоторые планы на будущее.

- Исследование возможности мониторинга веб-сервисов. Данная возможность интересна, в первую очередь, для использования системы GPM в грид.
- Работа над веб-сервисом для интеграции *Менеджеров* на различных кластерах.
- Расширение клиентской части для поддержки мониторинга на межкластерном уровне.

Со всеми результатами работы проекта вы можете ознакомиться на сайте http://gpm-system.sourceforge.net.

Литература

1. Information Services (MDS): Key Concepts (http://www.globus.org/toolkit/docs/4.0/info/key-index.html)
2. GridClub – Интернет-портал по грид-технологиям (http://www.gridclub.ru/)
3. Ganglia - Distributed monitoring system (http://ganglia.sourceforge.net/)

4. Condor Hawkeye (http://www.cs.wisc.edu/condor/hawkeye/)

Контакты
Россия, 603950, г.Нижний Новгород, пр.Гагарина, 23;
Тел.(+7 8312) 65-79-23 Факс (+7 8312) 34-50-56
Dmitry.Labutin@cs.vmk.unn.ru

ЛАБОРАТОРНЫЙ ПАРКТИКУМ ПО ПАРАЛЛЕЛЬНЫМ ВЫЧИСЛИТЕЛЬНЫМ АЛГОРИТМАМ МОНТЕ-КАРЛО

Ларченко А., Абрамова А., Пименов И., Комалева О., Бухановский А.
СПбГУ ИТМО, Санкт-Петербург

Переход на многоядерные архитектуры не может быть оставлен без внимания специалистами различных предметных областей, которые до этого не встречались с подобными системами, но ощущают насущную потребность в повышении производительности вычислений. Авторами разработан лабораторный практикум по параллельным вычислительным алгоритмам Монте-Карло. В нем основной упор сделан не на конкретные (компьютерные) технологии использования многоядерных архитектур как таковые, а на реализацию параллельных вычисленный в расчетных алгоритмах, которые широко применяются в самых различных областях научной и практической деятельности.

В настоящее время существует достаточно обширный набор учебных курсов и монографий, посвященных параллельным алгоритмам, высокопроизводительным вычислениям и пр. Однако реально используемых лабораторных практикумов с программной реализацией конкретных алгоритмов в ведущих Университетах РФ весьма немного, и область их распространения не велика. Поэтому целью авторов стало заполнить тот пробел, который образовался в процессе обучения существующих специалистов-предметников, применяющих описанные в практике методы, но использующих лишь традиционные подходы к программированию (без использования параллельных вычислений).

Лабораторный практикум в основном ориентирован на спектр задач, решаемых посредством метода Монте-Карло (статистических испытаний). Многие из реализаций, описанных в практикуме,

являются сопутствующим продуктом предшествующих научных исследований авторов в рамках различных проектов. Практикум включает лабораторные работы в следующих направлениях: векторно-матричные операции, решение СЛАУ, вычисление кратных интегралов, генетические алгоритмы, оптимизация случайным поиском, генерация случайных графов и др.

Каждая работа состоит из теоретического описания с кратким обзором рассматриваемой области. Оно дается из того расчета, что курс лекционных занятий по данной теме уже был прочитан. Далее слушателю напоминаются основные идеи, которые могут быть приняты на рассмотрение при реализации параллельных версий алгоритмов, а также описываются конкретные приемы их программной реализации. В заключительной части слушателям предлагается провести ряд тестов, включающих запуск разработанного авторами специально для этих целей модельного программного обеспечения, иллюстрирующего работу алгоритмов, и оценить прирост производительности при использовании преимуществ многоядерной архитектуры. Также для сравнения слушателю предоставляются результаты, полученные авторами в результате аналогичных тестов; они приводятся в виде наглядных графиков и диаграмм.

Слушатели, таким образом, смогут оценить преимущества нового подхода непосредственно в применении к конкретным вычислительным задачам, что может позволить преодолеть барьер, с которым сталкиваются опытные специалисты при встрече с новой компьютерной технологией или, более широко, парадигмой. Включение в состав лабораторных работ готовых программ поможет слушателям осознать как специфику работы самих алгоритмов, что немаловажно, так и с аспекты их параллельного исполнения.

ПО практикума разработано с использованием языка C++ в среде Microsoft Visual Studio 2005. Для демонстрации возможностей параллельного исполнения алгоритмов используются такие технологии, как MPI и OpenMP, что позволяет также подойти к проблеме параллельных вычислений с точки зрения разных архитектур вычислительных систем.

Предлагаемый лабораторный практикум в состоянии помочь обучить современных ученых-исследователей и инженеров, работающих в прикладной сфере, новым технологиям, и тем самым поднять на качественно новый уровень проведение вычислений,

требовательных к мощностям машинного парка. Также, это, возможно, укажет руководителям предприятий, работающих в сфере ресурсоемких вычислений, на перспективы данного пути развития аппаратной инфраструктуры с привлечением дополнительных средств в развитие высокопроизводительных технологий.

Исследования поддержаны Образовательным грантом

МЕТОДЫ ФУНКЦИОНАЛЬНО-ПОТОКОВОГО ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ, УЧИТЫВАЮЩИЕ АСИНХРОННОЕ ПОСТУПЛЕНИЕ ДАННЫХ

Легалов А.И.

Красноярский государственный технический университет, Красноярск

Введение

Функционально-потоковое параллельное программирование позволяет создавать программы, непосредственно задающие информационные графы, проводимых вычислений [1]. Это позволяет реализовывать функции, описывающие максимальный параллелизм решаемых задач, который впоследствии можно сжимать путем изменения стратегии управления вычислениями [2]. Вместе с тем следует отметить, что существующие подходы к построению функциональных и потоковых параллельных программ в основном рассматривают обрабатываемые данные в качестве аргументов, окончательно сформированных к моменту запуска функций. Однако такое видение ведет задержке в проведении вычислений из-за присутствия дополнительной синхронизации данных. В работе [3], были предложены механизмы, учитывающие асинхронное поступление данных. Ниже рассматривается, каким образом новые методы управления данными влияют на организацию функционально-потоковых параллельных программ.

Специфика асинхронного управления данными

Опираясь на результаты работы Хоара [4], можно считать, что события, возникающие при выполнении параллельных процессов, порождаются последовательно. Учитывая то, что при управлении по готовности данных события связаны с моментами получения значений,

будем считать последовательными моменты порождения этих значений во время вычислений. Интервал времени между порождаемыми событиями в асинхронных моделях не считается существенным фактором. То есть, он может быть большим или близким к нулю. Более важную роль в данном случае играют соотношения интервалов между моментами порождения данных и длительностью вычислительных операций. Эти времена могут быть сравнимыми или соотноситься как очень большие и очень малые величины. В последнем случае очень малыми величинами можно пренебречь, приравняв их к нулю.

Представленные рассуждения позволяют, в качестве примера, предложить следующую схему вычисления суммы элементов одномерного массива на основе готовности данных. Элементы массива, асинхронно формируемые в ходе вычислений, образуют очередь, в которую заносятся в порядке появления. При наличии в этой очереди двух или более элементов проводится суммирование образуемых пар с последующей засылкой результата в эту же очередь. Единственный элемент, оставшийся в ходе суммирования, является окончательным результатом. Схема подобной организации вычислений представлена на Рис. 1.

Для реализации управления на основе асинхронно поступающих данных в функциональную модель потоковых вычислений предлагается добавить еще один способ структурирования данных – асинхронный список. Его особенностью является упорядочение элементов в соответствии с последовательностью их порождения. По сравнению с обычным списком данных, формирующим сигнал готовности после поступления всех его элементов, асинхронный список сигнализирует только о появлении первого значения. Это накладывает определенную специфику на его интерпретацию. Можно считать, что об асинхронном списке достоверно известно следующее: или он не готов, или содержит как минимум один готовый элемент (а об остальных данных ничего не известно), или является пустым.

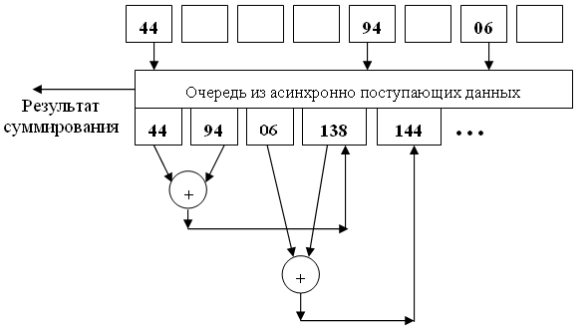


Рис. 1. Суммирование данных, асинхронно поступающих в общий поток

Неготовность асинхронного списка определяется отсутствием сигналов, информирующих о наличии сформированных внутри него данных. В этом случае, в соответствии с принципами потокового управления по готовности данных, такой список не может быть обработан. Выдача сигнала подтверждает готовность асинхронного списка к вычислениям, информируя при этом об одной из двух ситуаций: список содержит хотя бы один элемент или он пуст. Тогда определение внутреннего состояния возможно с использованием функции, определяющей длину асинхронного списка. Длина пустого списка равна нулю.

Длина непустого асинхронного списка всегда равна единице. Это объясняется спецификой его интерпретации. Как только внутри списка появляется хотя бы один вычисленный элемент, происходит выдача сигнала, подтверждающего готовность списка к последующей обработке. Даже при одновременном порождении внутри асинхронного списка нескольких элементов будет выдаваться только один сигнал. В этой ситуации первый элемент списка определяется внутренним арбитром, стратегия работы которого не имеет существенного значения. В данной ситуации о наличии одного элемента достоверно известно. Для того чтобы определить, есть ли в асинхронном списке другие данные, необходимы дополнительные операции.

В принципе, ничто не мешает использовать произвольное число поступивших элементов. Однако, учет их конкретного количества затрудняет программирование, делая его привязанным к размерности данных. Программы в этом случае становятся более громоздкими. При

этом гибкость программирования повышается незначительно, так как все необходимые манипуляции можно делать путем аналогичной обработки только первых элементов асинхронных списков.

Наряду с определением длины, над асинхронным списком допустимы операции чтения первого элемента и формирования нового асинхронного списка, не содержащего первый элемент. Это, в соответствии с концепциями событийного управления, позволяет описывать асинхронный параллелизм как процесс обработки последовательно наступающих событий.

$$A^0 = (.), \qquad \text{если } N = 0,$$
$$A^N = (d, D^{N-1}), \qquad \text{если } N > 0,$$

где d – головной элемент порождаемого списка, D^{N-1} – N-1 элементов списка, оставшиеся еще не обработанными, A^0 – пустой асинхронный список, эквивалентный обычному пустому списку данных, который обозначается в языке «Пифагор» как «(.)» [3-4]. Чтобы отличить асинхронный список от обычного списка данных, используемого в языке «Пифагор», введем для него следующее обозначение:

$$\text{asynch}(d_1, d_2, \ldots d_N) \equiv A^N \ ,$$

где d_1, d_2, … d_N – список элементов, порождаемых в ходе вычислений, проистекающих внутри него, упорядоченный по времени формирования значений. Для выделения и дальнейшей обработки этих значений допускается использовать операции выборки первого элемента и создания списка без первого элемента, эквивалентные по синтаксису и семантике аналогичным операциям списка данных. Таким образом, выделения из списка первого элемента задается следующим образом:

$$\text{asynch}(d_1, d_2, \ldots d_N):1 \Rightarrow d_1 \ .$$

Для выделения прочих элементов в отдельный асинхронный список необходимо воспользоваться следующей операцией:

$$\text{asynch}(d_1, d_2, \ldots d_N):\text{-}1 \Rightarrow \text{asynch}(d_2, \ldots d_N) \ .$$

Одновременное использование этих двух операций позволяет выделить и обработать все элементы асинхронного списка. Выполнение этих операций может продолжиться до получения вместо асинхронного списка пустого списка данных:

$$\text{asynch}(d_1, d_2, d_3, \ldots d_N):[1, \text{-}1] \Rightarrow$$
$$\Rightarrow d_1, \text{asynch}(d_2, d_3, \ldots d_N,) :[1, \text{-}1] \Rightarrow$$
$$\Rightarrow d_1, d_2, \text{asynch}(d_3, \ldots d_N,) :[1, \text{-}1] \Rightarrow$$
$$\Rightarrow d_1, d_2, d_3, \text{asynch}(d_4, \ldots d_N) :[1, \text{-}1] \Rightarrow \ldots \Rightarrow$$

$\Rightarrow$ d_1, d_2, d_3, … d_{N-1}, asynch(d_N):[1, -1] $\Rightarrow$
$\Rightarrow d_1, d_2, d_3, \ldots d_N, (.)$.

Таким образом, прямой доступ к произвольному элементу асинхронного списка невозможен. Если же данные были сформированы заранее, то порядок их следования в асинхронном списке определяется размещением в исходном списке данных. Для заранее заданного одномерного массива X = (44,55,12,42) асинхронный список, порождаемый из исходного списка данных, будет трансформироваться обрабатываться по этим же правилам:

asynch(44,55,12,42):[1, -1] $\Rightarrow$ 44, asynch(55,12,42) :[1, -1] $\Rightarrow$
$\Rightarrow$ 44, 55, asynch(12,42) :[1, -1] $\Rightarrow$ 44, 55, 12, asynch (42) :[1, -1] $\Rightarrow$
$\Rightarrow$ 44, 55, 12, 42, (.) .

Очередной элемент данных, выделенный из списка, может использоваться в выполняемых операциях, а оставшиеся элементы могут выделяться с целью получения очередного первого элемента.

Асинхронный список во многом аналогичен очереди, состоящей из одного элемента, в которую в произвольном порядке поступают данные. Вместе с тем следует отметить, что рекурсивно-параллельная структура модели и языка программирования «Пифагор» накладывают на это восприятие дополнительные особенности.

Использование асинхронных списков

Приведенные выше расширения функционально потоковой модели вычислений можно реализовать в языке программирования «Пифагор» [5]. Это позволит писать параллельные программы в несколько ином стиле, при котором учитывается асинхронное поступление данных. В частности, обобщенная функция асинхронной бинарной свертки одномерного массива может быть реализована кодом:

```
/*Обобщенная асинхронная бинарная свертка
 Формат аргумента: D=(f,asynch(x1, x2, ... , xn))
 где: f - подставляемая бинарная функция, x1, x2, ... , xn -
числа.
*/
ABinTreeReduction << funcdef D {
  f<< D:1;
  X<< D:2;
  x1<< X:1;       // Поступивший элемент данных
  tail_1<< X:-1;  // Хвост асинхронного списка

  // Проверка на пустоту остатка списка
  [((tail_1:|, 0):[=, !=]):?]^
  (
     x1, // В списке, только один элемент - результат
     {
        // Выделение второго аргумента с выполнением
        // бинарной функции
        block {
           x2<< tail_1:1;      // второй аргумент
           s<< (x1,x2):f;      // функция для двух элементов
           tail_2<< tail_1:-1; // "хвост" от "хвоста"
           // Рекурсивная обработка оставшихся элементов
           [((tail_2:|, 0):[=, !=]):?]^
           (
              s,
              { (f, asynch(tail_2:[], s)):ABinTreeReduction }
           ):. >>break
        }
     }
  ):. >>return;
}
```

Использование данной функции для суммирования элементов вектора осуществляется следующим образом:

```
(+,(44,55,12,42,94,18,06,67)):AbinTreeReduction
```

Следует отметить, что при поступлении на вход данной функции обычного списка данных вместо асинхронного списка, он будет корректно обработан. Это обуславливается эквивалентной интерпретацией операций взятия первого элемента и выделения подсписка без первого элемента для обоих списков.

Аналогичным образом можно написать и функцию, осуществляющую попарное перемножение элементов двух асинхронных списков, которую можно использовать и для перемножения двух векторов, заданных обычными списками данных.

Однако, для того, чтобы эта функция могла в дальнейшем использоваться в более сложных асинхронных взаимодействиях, необходимо, в качестве результата, возвращать либо асинхронных, либо параллельный список. Использование последнего позволяет сохранить информацию о порядке следования элементов списка, что позволяет использовать данную функцию в разнообразных ситуациях. Исходный текст рассматриваемой функции выглядит следующим образом:

```
// (f,(asynch(x1,x2,...,xn),asynch(y1,y2,...,yn)))
AVecPairOperation<< funcdef D {
  f<< D:1;
  VecPair<< D:2;
  len1<< VecPair:1:|;
  [((len1,0):[=,!=]):?]^
  (
    {.},
    {(VecPair:[]:1):f, (f, (VecPair:[]:-1)):AVecPairOperation}
  ):. >>return;
}
```

Использование этой обобщенной функции с обычными списками данных позволяет выбирать их элементы в асинхронном режиме. При этом выполнение произведения будет выглядеть следующим образом:

```
(*,((44,55,12,42),(94,18,06,67))):AvecPairOperation
```

Ниже приведено совместное применение двух асинхронных функций для выполнения перемножения вектора-строки на вектор-столбец:

```
AVecProd << funcdef D {
  X<< D:1;  Y<< (D:2:[]:[]);
  (+,asynch((*,(X,Y)):AVecPairOperation)):ABinTreeReduction
>>return;
}
```

В данной ситуации асинхронность позволяет начать суммировать элементы еще до того, как будут получены произведения всех элементов результирующего вектора.

Заключение

Таким образом, использование асинхронных списков позволяет реализовать новый класс алгоритмов, параллелизм которых зависит от временных соотношений между выполняемыми основными и

подготовительными операциями. Можно говорить о специфическом подходе к разработке параллельных функционально-потоковых алгоритмов, позволяющем создавать и исследовать новые методы организации переносимых параллельных программ. В отличие от эквивалентных преобразований обобщенных функций, предлагаемых в [6] для адаптации параллельного алгоритма к конкретной вычислительной системе, использование асинхронных списков позволяет применять один и тот же алгоритм, на который накладываются разные временные отношения между выполняемыми операциями.

Предложенный в работе механизм описания асинхронных вычислений предоставляет в распоряжения программиста дополнительные возможности. Его инструментальная поддержка на уровне языков обеспечивает использование новых методов разработки функционально-потоковых параллельных программ, характеризующихся динамическим изменением параллелизма в зависимости от временных соотношений между выполняемыми операциями. Это повышает гибкость разрабатываемых алгоритмов, и позволяет использовать один и тот же алгоритм для динамического задания различных уровней параллелизма. Формируемые при этом программы позволяют описывать максимальный параллелизм решаемой задачи и не зависят от архитектур используемых вычислительных систем.

Литература

1. Легалов А.И., Казаков Ф.А., Кузьмин Д.А. Водяхо А.И. Модель параллельных вычислений функционального языка // Известия ГЭТУ, Сборник научных трудов. Выпуск 500. Структуры и математическое обеспечение специализированных средств. СПб.: 1996. С. 56-63.
2. Легалов А.И. Использование асинхронно поступающих данных в потоковой модели вычислений. / Третья сибирская школа-семинар по параллельным вычислениям. / Томск. Изд-во Томского ун-та, 2006. С 113-120.
3. Легалов А.И. Об управлении вычислениями в параллельных системах и языках программирования – Научный вестник НГТУ, № 3 (18), 2004. С. 63-72.
4. Хоар Ч. Взаимодействующие последовательные процессы. М.: Мир, 1989. 264 с.

5. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1. С. 71-89.
6. Легалов А.И., Казаков Ф.А. Эквивалентные преобразования функционально-параллельных программ. // Распределенные и кластерные вычисления. Избранные материалы Третьей школы-семинара. Красноярск: Институт вычислительного моделирования СО РАН. 2004. С. 134-141.

ПРОГРАММНЫЙ КОМПЛЕКС РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ НА ОСНОВЕ .NET REMOTING

Логвиненко О.О., Аксак Н.Г.
Харьковский национальный университет радиоэлектроники

Введение

Задачи распределения разнообразных вычислений на данный момент являются очень актуальными. Приложения, системы приложений и программные комплексы, использующие возможности распределения вычислений предоставляют абсолютно новые решения в проектировании и развитии всевозможных областей, повышая эффективность и срок службы хорошо знакомых и отработанных алгоритмов. Некоторые приложения являются распределенными изначально: всевозможные многопользовательские игры, приложения для общения и обмена информацией, для телеконференций - все это примеры подобных приложений. Также, существует много приложений, которые являются распределенными в том смысле, что они имеют как минимум два компонента, работающие на различных машинах сети. Но, в основном такие приложения сильно ограничены в масштабируемости и в гибкости перераспределения, что не позволяет их назвать полноценно распределенными. Распределенные же приложение предоставляют пользователю преимущества динамической масштабируемости и оптимизации используемых сетевых и компьютерных ресурсов, кроме того приложения, которые изначально разрабатывались как распределенные, могут совмещать при взаимодействии клиентов с различными вычислительными мощностями. Но, если вся логика взаимодействия и вычислений комплексного приложения сосредоточена в едином модуле, то

повысить скорость работы без перенастройки приложения можно только одним способом - установкой более быстродействующего, а соответственно и более дорогостоящего аппаратного обеспечения. Современные серверы и операционные системы зачастую подлежат необходимым модификациям, но чаще всего бывает дешевле приобрести еще несколько машин похожей конфигурации, которые бы в сумме давали необходимую вычислительную мощность, чем обновлять текущий сервер до необходимого уровня производительности.

Постановка задачи

Целью данной работы является разработка программного комплекса для выполнения распределенных вычислений в локальной сети. Комплекс должен обладать возможностью расширения количества алгоритмов вычислений и возможностями ввода начальных данных нескольких типов. Немаловажным требованием является реализация двух режимов работы комплекса. В первом режиме комплекс должен визуально отображать ход выполнения вычислений и их управления, таким образом, чтобы пользователь мог понять ход работы алгоритма на любом узле сети, который принимает участие в вычислении. Второй режим будет использоваться как экспериментальный, что позволит собирать реальные результаты работы параллельных алгоритмов, которые выполняются на вычислительных узлах в условиях реального времени без задержек на визуализацию процесса вычисления и обмена данными. Эти требования придадут проекту универсальность использования как в обучающих, так и в экспериментальных целях, что довольно сильно расширит сферу его применения.

Выбор метода и технологии параллельной обработки

Важным моментом данной задачи является выбор модели параллельной обработки данных. Для реализации была выбрана модель делегирования, суть которой состоит в том, что в системах, основанных на данной модели, объекты разделяются на рабочие и управляющие. Рабочие объекты производят вычисления; управляющие - управляют рабочими обьектами. Выбор обусловлен тем, что в модели с равноправными узлами синхронизация потоков выполнения будет производиться средствами этих же потоков без постороннего вмешательства или вынесения данной функции в отдельный модуль.

Это удобно при небольшом количестве потоков или при реализации параллельных алгоритмов, в которых явно можно выделить точки синхронизации. Таким образом, для реализации комплекса в образовательных целях была выбрана модель делегирования, которая позволяет вынести функцию синхронизации вычислительных потоков в отдельный модуль.

Рис. 1 Структурная модель работы программного комплекса

Вторым важным моментом является вопрос об использовании технологии распределенных вычислений. Сокеты, например, предоставляют наилучшие возможности управления процессом взаимодействия, но для построения на их основе сложных, полномасштабных распределенных приложений требуется слишком много усилий. В таких условиях задача создания системы сообщений и формирования потоковых данных полностью возлагается на программиста. Выходом из этой ситуации является применение технологий распределения вычислений. Распределенные объектные технологии позволяют обращаться к объектам из приложений, исполняющихся на других компьютерах. К таким технологиям, например, относятся DCOM и .NET Remoting. DCOM является развитием многокомпонентной модели COM. Эта модель определяет, каким образом компоненты взаимодействуют со своими клиентами. Это взаимодействие осуществляется таким образом, чтобы клиент и компонент могли соединяться без необходимости использования промежуточных компонент системы, и у клиента была возможность вызывать методы компонента. .NET Remoting предоставляет согласованную модель объектов, имеющую точки расширения для поддержки систем, которые до настоящего времени создавались на основе DCOM и, кроме того, проектировщики .NET Remoting имели возможность учесть самые последние требования технологий распределенных вычислений, и в связи с этим выбор был остановлен именно на этой технологии.

Реализация проекта

Реализация проекта сводится к написанию двух приложений, использующих технологию для построения распределенных приложений .Net Remoting. Такая реализация визуально идентична классическому методу клиент-серверного взаимодействия. На рисунке 2 отображена функциональная схема работы комплекса.

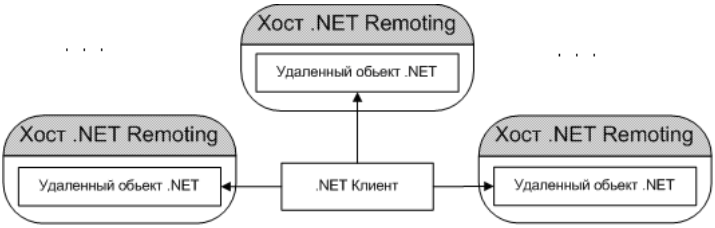


Рис. 2 Функциональная схема комплекса

Из нее видно, что один клиент может взаимодействовать с несколькими удаленными вычислительными объектами. Для поддержания необходимой для проекта структуры, необходима клиентская активизация удаленных объектов, так как она позволяет объектам оставаться активными в интервалах между вызовами методов. Кроме того, немаловажным является то, что такой подход позволяет обрабатывать все клиентские запросы с помощью отдельного экземпляра дистанцируемого типа.

Серверное приложение должно поддерживать следующие функции:

- предоставление возможности клиентам выбирать задания для исполнения;
- возможность корректной обработки завершения задания по требованию клиента;
- отслеживание заданий, взятых на обработку, и выполненных каждым клиентом.

Основная задача клиентского приложения - ввод данных. Поэтому ему необходим современный, высокофункциональный пользовательский интерфейс, способный визуально отображать ход выполнения алгоритма, давать возможность выбора алгоритмов и данных, иметь широкие возможности по настройке параметров

эксперимента с возможностью отслеживания доступных на данный момент серверов. От клиента дополнительно требуется:

- возможность остановки эксперимента до его завершения;
- уведомлять пользователя о возникших проблемах сети и о сбоях в работе алгоритмов;
- собирать статистику по эксперименту с возможностью ее визуального отображения после завершения эксперимента.

Литература

1. Маклин С., Нафтел Дж,, Уильяме К. Microsoft .NET Remoting- Пер. с англ. — М.: Издательско-торговый дом Русская Редакция, 2003. — 384 с.

Контакты

61166, Харьков, пр. Ленина,14, каф. ЭВМ, тел. (0572) 702-13-*54*

ИНТЕРПРЕТАЦИЯ ЯРУСНО-ПАРАЛЛЕЛЬНОЙ И КОНВЕЙЕРНОЙ ФОРМ АЛГОРИТМОВ КОНЕЧНЫМИ АВТОМАТАМИ

Любченко В.С.

«ООО Креол ЛТД», г. Александров

Введение

Ярусно-параллельная (ЯПФ) и конвейерная формы алгоритмов (программ) рассматриваются часто как самостоятельные алгоритмические модели. Цель работы показать, как эти модели можно сделать разными структурными формами одной модели вычислений. В такой модели отражаются состав, связи и последовательность выполнения процессов, соответствующих элементам графа ЯПФ и конвейера.

Далее речь пойдет о сетевой конечно-автоматной параллельной модели вычислений. Будет показано, как в рамках этой модели можно реализовать, казалось бы, разные формы параллельных алгоритмов. А так как конвейер - это фактически ЯПФ, в которой каждому элементу конвейера соответствует отдельный ярус ЯПФ, то ниже нам, не ограничивая общности, достаточно будет рассмотреть лишь ярусно-параллельную форму алгоритмов.

Пример ярусно-параллельной формы алгоритма

Рассмотрим задачу распараллеливания выражения:

$$y = ((ab) \times (c + d)) + ((e \times f) + (g + h)) \qquad (1)$$

Условимся, что операция сложения выполняется один дискретный такт, а операция умножения – пять тактов.

Граф ЯПФ для вычисления заданного выражения представлен на рис. 1:

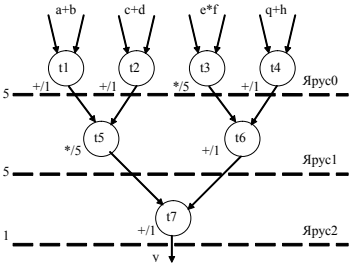


Рис. 1 Ярусно-параллельная форма алгоритма

На нем рядом с операторными вершинами поставлен символ операции и число дискретных тактов для ее исполнения. Слева проставлено время исполнения в дискретных тактах каждого яруса. Легко видеть, что время получения конечного результата, измеряемое числом дискретных тактов, равно одиннадцати.

На рис.2 приведена измененная ЯПФ алгоритма на рис.1. Здесь вершина t3 перемещена на ярус ниже и введен еще один ярус для вершины t6. И, несмотря на то, что ярусов больше, время исполнения алгоритма меньше и равно восьми тактам.

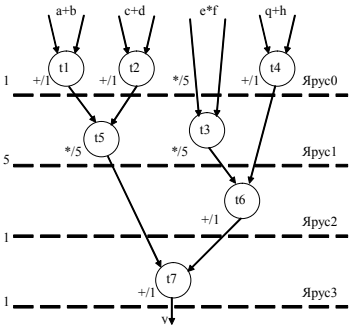


Рис. 2. Ярусно-параллельная форма алгоритма

Таким образом, из анализа приведенных графов ЯПФ видно, что данная форма алгоритма в общем случае не гарантирует минимальное время исполнения алгоритма.

Мы уже отметили эквивалентность ЯПФ и конвейера. Дополнительно заметим лишь, что т.к. элементы конвейера функционируют постоянно, что это нужно учитывать, разрабатывая и реализуя алгоритмы вершин ярусов ЯПФ.

Структурная модель системы

Мы можем подойти к вычислению выражения (1) иначе, если представим, что каждый из операторов функционирует параллельно, как асинхронный процесс. В этом случае ярусов нет, а операторы, получив информацию о готовности данных, тут же выполняют преобразование данных. Результат операции может передаваться дальше или его может брать связанный с текущим процессом следующий процесс-операция. Далее процесс-операция либо ждет готовности очередной порции данных (как в конвейере), либо завершает свою работу.

Структурную модель подобной системы для вычисления выражения (1) можно получить, если граф на рис.1 повернуть на 90º против часовой стрелки и убрать линии, соответствующие делению на ярусы. Теперь каждая вершина графа будет представлять собой «черный ящик» (структурный блок) с двумя входами и одним выходом.

Связи между вершинами выполняют двоякую функцию. Во-первых, они отражают статические связи - связи между структурными

блоками системы. Во-вторых, они определяют и динамические связи - модель взаимодействия блоков, когда по каждой связи передается информация о готовности данных.

Один из вариантов обмена данными между структурными блоками может быть следующим. Каждый блок имеет три элемента данных (например, три переменных, если говорить о программной реализации), соответствующих операндам и результату операции. Значение переменных обновляется либо извне, т.е. блоком/процессом, вычислившим результат той или иной операции, либо самим процессом, который, получив информацию о готовности данных, устанавливает истинное значение переменной. В этом случае можно считать, что и входные данные для выражения тоже готовятся некими внешними процессами.

Сказанное о процедуре обмена данными должно присутствовать и в ЯПФ. Только там ее можно представить как синхронную процедуру, когда данные передаются на следующий ярус по истечении времени отведенного для яруса. Но, кстати, подобным образом можно организовать работу и рассматриваемых структурных блоков, введя дополнительный процесс для синхронизации в соответствии с временем работы ярусов. Но пока мы выбираем более гибкую и независимую организацию взаимодействия блоков.

Алгоритмическая модель

Рассматривая модель функционирования отдельных блоков и системы в целом, остановим свой выбор на модели автоматного типа, когда функционирование отдельного блока описывается конечным автоматом, а всей системы – сетью взаимодействующих автоматов. Для такой сети является выбор «системного времени»: нужно ли считать дискретное время единым для всех автоматов или у каждого автомата оно должно быть своим? Строго оно должно быть единым, но в данном случае ответ на этот вопрос не столь существенен, т.к. взаимодействие между блоками организовано по асинхронному принципу.

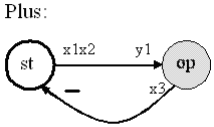


Рис.3. Модель операции сложения

Автоматная модель операции сложения представлена на рис.3. В таблицах 1,2 в терминах объектно-ориентированного программирования (ООП) перечислены свойства и методы активного автоматного объекта, представляющего процесс. Методы объекта, имеющие отношение к автоматной модели сведены в табл.2.

Таблица 1

Свойства		
dOp1	Первый операнд	Pub
dOp2	Второй операнд	Pub
dY	Результат операции	Pub
bIfY	Выходные данные взяты	Pub
Методы		
GetY()	Получить выходные данные	Pub

Таблица 2

Предикаты	
x1	Первый операнд готов?
x2	Второй операнд готов?
x3	Данные взяты?
Действия	
y1	Выполнение операции сложения

Автомат находится в начальном состоянии «st» и ждет готовности входных данных – истинности предикатов x1 и x2. Дождавшись, автомат переходит в состояние «op», выполняя действие y1. В этом состоянии автомат находится до тех пор, пока не будет взят результат операции, о чем будет сигнализировать истинное значение предиката x3. Как только это произойдет, автомат перейдет в начальное состояние, где опять будет ждать готовности очередной порции входных данных.

Автомат, моделирующий операцию умножения, аналогичен. Только в него нужно ввести промежуточное состояние, в котором будут отсчитываться такты (см. далее на рис.4 автоматную модель вершины ЯПФ - t5), на число которых будет задержана выдача результата.

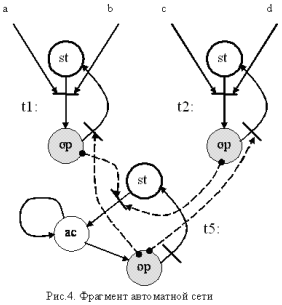


Рис.4. Фрагмент автоматной сети

Фрагмент автоматной сети, содержащей автоматные модели вершин t1, t2, t5 ЯПФ, показан на рис.4. При этом на графе информационные связи между автоматами, которые соответствуют связям структурной модели (соответственно и ЯПФ) показаны в виде штрихпунктирных стрелок.

Быстродействие автоматной модели

Быстродействие автоматной модели можно оценить, если представить ее в форме ЯПФ, которая показана на рис.5. Это фактически две ЯПФ, функционирующих параллельно. Видно, что время работы каждой ЯПФ, как и время их совместного функционирования равно семи тактам, что меньше, чем на любой ранее рассмотренных ЯПФ.

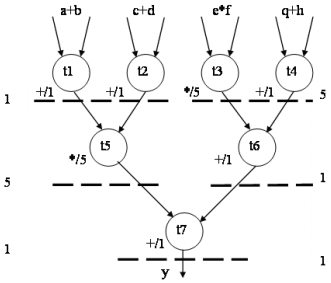


Рис. 5 Параллельная ярусно-параллельная форма

Еще раз напомним, что разбиение на несколько ЯПФ, показанное на рис.5, условно, т.к. каждая из вершин это асинхронный автомат, функционирующий независимо от других автоматов системы. Приведенное разбиение на ярусы, это лишь отражение временной согласованной работы вершин.

Выводы

Из сравнения моделей видно, что сетевая автоматная модель гибче ярусно-параллельной формы алгоритмов. Автоматы сети, работая параллельно и взаимодействуя между собой, самостоятельно решают, кому и когда выдать тот или иной результат. В рамках этой модели не нужно специально организовывать ярусы, чтобы распараллелить работу. Не нужен поиск ЯПФ, что достичь оптимального быстродействия.

Как уже было отмечено, автоматную сеть легко заставить функционировать строго в соответствии с эквивалентной ярусно-параллельной формой алгоритма. Это позволяет упростить протоколы взаимодействия объектов, что может достаточно заметно уменьшить реальное время работы алгоритма.

Литература

1. Основы вычислительных систем. Курс лекций. http://256bit.ru/education/infor2/lecture1-1.htm
2. Форум сайта SoftCraft. Мифы асинхронного параллелизма. http://www.softcraft.ru/forum/viewtopic.php?p=1810#1810

НОВЫЕ МЕТОДЫ СЖАТИЯ ИНФОРМАЦИИ.

Макаров А.А.
Санкт-Петербургский государственный университет,
Санкт-Петербург

Введение

Вейвлеты (всплески) широко применяются при составлении эффективных алгоритмов обработки больших потоков цифровой информации (см. [1]). Идея такой обработки состоит в прореживании "по времени" или "по частоте", что приводит к сжатию поступающего цифрового сигнала. Под эффективностью в данном случае понимают экономное (с точки зрения экономии ресурсов компьютера: памяти и времени обработки) разложение потока информации на составляющие, так чтобы можно было выделить основной информационный поток, уточняющий инфомационный поток и информационный поток с несущественной информацией. Как правило, основной информационный поток значительно менее плотный, чем исходный поток информации; поэтому его можно передать быстро, и при этом не требуется использовать линии связи большой ширины. Уточняющий информационный поток не во всех случаях необходим, его можно передавать фрагментарно в зависимости от потребностей. Наконец, поток с несущественной информацией вообще может быть отброшен. Конечно, вопрос о том, какая информация является основной, какая уточняющей, а какая - несущественной, выходит за рамки математических исследований и должен решаться в каждом отдельном случае специалистом предметной области.

Роль теории вейвлетов (всплесков) состоит в том, что она дает предметному специалисту достаточно широкий арсенал средств, из которых он может выбрать то средство, которое ему подходит для обработки (для разложения на составляющие) интересующего его потока информации. Такими средствами в теории вейвлетов являются наборы вложенных (основных) пространств функций и их представлений в виде прямой (а иногда и ортогональной) суммы вейвлетных пространств. Весьма важным являются базисы основных пространств, а также базисы вейвлетов (всплесков); построению и изучению свойств таких базисов посвящено большое количество работ.

В виду того, что информационные потоки стремительно нарастают, резко увеличиваются объемы цифровой обработки этих

потоков, и поскольку чаще всего требуется обработка их в реальном масштабе времени, то становится актуальным использование высокопроизводительных вычислительных систем с параллельными процессорами. К этому классу систем относятся и многоядерные процессоры. Примером является процессор Intel Pentium D, являющийся двухъядерным процессором. Поэтому решение вопросов распараллеливания на многоядерных процессорах приобретает особую актуальность.

Многоядерный процессор идеально подходит для многократного вейвлетно-сплайнового расщепления цифровых потоков: обработку каждого потока можно поручить одному из ядер процессора; даже два ядра существенно ускоряют обработку и при сжатии, и при восстановлении потоков цифровых сигналов. Многие типы известных вейвлетов обеспечивают быстрое, но весьма не точное сжатие. Здесь предполагается использовать вейвлетно-сплайновые системы (см. [2],[3]) с гарантированно высокой точностью приближения гладких цифровых потоков. Она приводит к эффективному сжатию и к достаточно точному результату, ибо учитывает "гладкость" обрабатываемого потока цифровой информации.

Более того, применение неравномерной сетки позволяет улучшить приближение функций без усложнения вычислений. Однако для дальнейшего улучшения приближения могут понадобиться различные степени измельчения сетки в разных частях рассматриваемого промежутка: для этого двукратное измельчение недостаточно. Особую заботу представляет вейвлетное разложение в случае неравномерной сетки, поскольку обычно применяемое на равномерной сетке преобразование Фурье в условиях неравномерной сетки использовать затруднительно. Оказалось, что использование биортогональной системы функционалов позволяет построить вейвлетные разложения и при произвольном измельчении сетки (это ведет к упрощениям и в случае равномерной сетки).

В работе:

- построена биортогональная система функционалов
- получено сплайн-вейвлетное разложение в случае неравномерной сетки
- выведены формулы реконструкции и декомпозиции
- для процессов реконструкции и декомпозиции предложена эффективная методика распараллеливания вычислений

Литература

1. И.Добеши. Десять лекций по вейвлетам. 2004. 464 с.
2. Ю.К. Демьянович. Всплесковые разложения в пространствах сплайнов на неравномерной сетке. Доклады РАН. 2002. Т. 382. №3. С. 313-316
3. Ю.К. Демьянович. Всплесковое (вейвлетное) разложение пространств минимальных сплайнов и универсальные фильтры. Сб. Методы вычислений. 2003. Вып. 20. С. 78-101.

СТАТИЧЕСКОЕ ПЛАНИРОВАНИЕ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ ГРАФ-СХЕМНЫХ ПРОГРАММ НА КЛАСТЕРАХ

Макарьевский С.Н.

МЭИ, Кафедра прикладной математики , Москва

В докладе рассматривается задача статического планирования параллельных процессов на кластерах. Она предполагает такое начальное распределение фрагментов параллельной программы по компьютерам кластера, при котором достигается баланс между суммарными временами выполнения размещенных фрагментов на компьютеры и минимизируется время межкомпьютерных обменов.

В отличие от статического планирования процессов динамическое планирование позволяет достигать большего эффекта за счет использования более «тонких» механизмов перераспределения процессов параллельной программы во время ее выполнения. Как показано в [1], использование эффективных алгоритмов динамического управления процессами может на 70-80% сокращать время выполнения параллельной программы и существенно повысить эффективность использования ресурсов, в частности, загруженность компьютеров кластера.

Современные программные среды параллельного программирования, такие как MPI, PVM, не имеют средств для динамического планирования процессов. Программист сам должен решать задачу оптимального статического распределения процессов на компьютеры кластера.

Как нам представляется, только комплексное решение проблемы планирования процессов и управления загруженностью кластера с возможностью статического и динамического планирования может

дать необходимый эффект. Именно такой подход используется для решения этой проблемы в созданной на кафедре прикладной математики МЭИ системе граф-схемного потокового параллельного программирования для кластеров [2,3,4].

Преимуществом граф-схемных (ГС) программ является визуальное графическое их представление, строго определяющее связи по данным, которые передаются между подпрограммами модулей программы при ее выполнении. Поскольку эти связи при выполнении граф-схемной параллельной программы выполняют роль буферов данных, в качестве модельного ее представления, используемого для статического планирования, предлагается использовать взвешенные мультиграфы, в которых вершины соответствуют подпрограммам граф-схемной программы, а дуги – передаваемым данным. При этом веса вершин – это вычислительная сложность соответствующих подпрограмм, а веса дуг – интенсивность передаваемых данных от выхода одной подпрограммы ко входу другой (возможно, этой же) подпрограммы.

Пример на рис.1 иллюстрирует это представление.

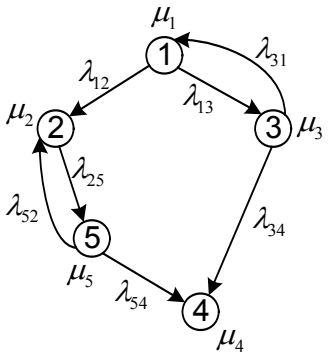


Рис.1 Взвешенный мультиграф граф-схемной программы

На этом рисунке μ_i – приведенные к быстродействию процессора интенсивности выполнения i-ой программы на процессоре компьютера (в измерении оп/сек), а λ_{ij} – приведенные к быстродействию коммуникаций интенсивности передаваемых между i-ой и j-ой подпрограммами данных в процессе выполнения параллельной программы. Эти интенсивности либо программист определяет сам, либо они являются результатом многократных прогонов программы.

Пользуясь введенным представлением, по существу, являющимся моделью линейной стохастической сети, можно сформулировать ряд постановок задачи статического планирования процессов параллельного выполнения граф-схемных программ на кластерах:
Разрезать ГС на *n* фрагментов (*n* – количество компьютеров кластера) таким образом, чтобы при условии минимального среднеквадратичного отклонения по количеству подпрограмм во фрагментах достигался минимум межфрагментных связей.
«Наоборот» - условие минимальности среднеквадратичного отклонения по количеству связей между фрагментами так, чтобы достигался минимум среднеквадратичного отклонения по количеству подпрограмм во фрагментах.
1 и 2 при дополнительном условии, чтобы циклические участки не разрезались.
То же, что в 1, 2 и 3, но в предположении знания сложности подпрограмм t_i (время выполнения i-ой подпрограммы) и интенсивности обменов между i-ой и j-ой подпрограммами λ_{ij} при выполнении ГС-программы.

В настоящее время предложены эвристические алгоритмы решения указанных выше задач статического планирования и ведется их исследование.

Работа поддержана РФФИ, проект 06-01-00817-а

Литература

1. Кутепов В.П. Интеллектуальное управление параллельными процессами и загруженностью в больших компьютерных системах. Изв. РАН, Теория и системы управления (в печати).
2. Котляров Д.В., Кутепов В.П., Осипов М.А. Граф-схемное потоковое параллельное программирование и его реализация на кластерных системах. М: Изв. РАН, Теория и системы управления, 2005, №1.
3. Кутепов В.П., Котляров Д.В., Маркин В.Я. Управление процессами и загруженностью многоядерных кластеров.
4. Кутепов В.П., Котляров Д.В., Маркин В.Я., Маланин В.Н., Панков Н.А. Архитектура и реализация среды объектно-ориентированного параллельного программирования для многоядерных кластеров.

РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО МЕТОДА ГЕНЕРАЦИИ СЛУЧАЙНЫХ БОЛЬШИХ ПРОСТЫХ ЧИСЕЛ ДЛЯ КРИПТОГРАФИЧЕСКИХ СИСТЕМ

Марьин С.В., научный руководитель – Нестеренко М.Ю.
ГОУ ВПО «Оренбургский государственный университет», г. Оренбург

Введение

Сегодня криптография играет важнейшую роль в нашем мире. Без криптографии нельзя обойтись при защите данных, передаваемых по открытым электронным каналам связи, а также там, где необходимо подтверждать целостность электронной информации или доказывать ее авторство. Она широко используется в нашей насыщенной информационными технологиями жизни: в пластиковых картах, в электронной почте, в системах банковских платежей, при электронной торговле через Интернет, в системах электронного голосования и во многом другом.

Главнейшим аспектом использования криптографии является скорость: никому не нужна криптосистема, шифрующая данные со скоростью, сравнимой со скоростью ее взлома. В свете бурного развития технологий распределенных вычислений в последнее время, появилась возможность использовать параллельные программы даже в домашних условиях. А это, в свою очередь, позволяет значительно сократить время шифрования, либо увеличить его криптостойкость без потери в скорости (путем увеличения длины ключа).

Одним из важнейших аспектов современной криптографии является генерация больших простых чисел. Они используются, например, в криптосистеме RSA. Общую задачу генерирования таких чисел можно разбить на три независимых подзадачи:

- Генерация псевдослучайных чисел. Необходимо создавать криптографически сильные последовательности чисел, проходящих тесты на случайность.
- Проверка чисел на простоту. Необходимо за небольшое время проверять числа на простоту.
- Действия с большими числами. Необходимо реализовать арифметические действия с числами порядка 2^{1024}.

Последняя подзадача в рамках данной работы подробно рассматриваться не будет по нескольким причинам. Во-первых,

существует множество библиотек, выполняющих данную работу очень эффективно. Во-вторых, параллельная реализация решения такой задачи не будет давать существенный прирост скорости по сравнению с последовательной версией (о чем будет сказано далее). И, в-третьих, криптостойкость системы не зависит от конкретной реализации этой подзадачи.

Анализ существующих методов генерации случайных чисел

Стойкость криптографической системы зависит от стойкости ее самого слабого звена. Поэтому и генератор случайных чисел, используемый в криптосистеме, должен выдавать непредсказуемые для злоумышленника числа. То есть должна генерироваться такая числовая последовательность, члены которой могут быть вычислены злоумышленником лишь после приложения значительных усилий, сопоставимых с усилиями, которые должны быть потрачены на вскрытие любой другой части криптосистемы.

Основная проблема генерации случайных чисел на компьютере состоит в том, что компьютер может находиться только в ограниченном числе состояний. Поэтому выдаваемый им результат всегда будет строго определяться исходными данными и текущим состоянием компьютера. А это значит, что любой программный генератор случайных чисел будет периодичен, а, следовательно, и не случаен.

Таким образом, лучшее, что может делать компьютер – генерировать некую псевдослучайную последовательность, то есть такую последовательность чисел, которую мы могли бы принять за случайную. Определим требования к таким последовательностям, учитывая их необходимость использования в криптографии:

- Последовательность должна выглядеть случайно, то есть должна проходить все известные тесты на случайность (равномерность распределения, несжимаемость и так далее).
- Последовательность должна быть непредсказуемой. То есть должно быть очень трудно предсказать, каким будет следующее число, даже если известны все предыдущие.
- Создаваемая последовательность не должна быть уверенно воспроизводимой. То есть при запуске генератора с одним и тем же входом, должны выдаваться различные последовательности.

Рассмотрим в качестве примера линейный конгруэнтный генератор, который используется для генерации случайных чисел во многих компиляторах. Члены его генерируемой числовой последовательности имеют следующий вид:

$X_{i+1} = (a * X_i + b) \bmod c$

Для данного генератора является критичным выбор чисел a, b и c для обеспечения максимального периода. Кроме того, данный генератор не является стойким, а потому нежелателен в применении в криптографических целях.

Значительно большую криптостойкость обеспечивает BBS-генератор, названный в честь своих создателей – Леоноры Блюм и Мануэля Блюма и Майкла Шуба. Возьмем произвольно число X и вычислим X_i по формулам:

$X_0 = X^2 \bmod n$
$X_{i+1} = X_i^2 \bmod n$, где $i \geq 0$

Где n – число Блюма, то есть такое, что:

$n = p * q$
$p = 3 \pmod 4$
$q = 3 \pmod 4$
p, q – простые.

Обозначим за b_i первый значащий бит X_i. Тогда генерируемой последовательностью будет последовательность битов b_i.

Кроме того, каждое число X_i может быть получено без знания предыдущих по следующей формуле:

$$X_i = (X_0^{2^i \bmod ((p-1)(q-1))}) \bmod n$$

Несмотря на хорошую криптографическую стойкость (равную стойкости алгоритма RSA), у этого генератора имеется один недостаток – он медленный. В самом деле, если в линейном конгруэнтном генераторе для вычисления следующего числа требовалось всего одно умножение и одно деление (взятие остатка), то в BBS-генераторе эти вычисления используются для получения лишь

одного бита. Чтобы ускорить процесс генерации, можно брать (log log n) битов из каждого X_i. Это не снижает криптостойкости алгоритма.

Знание числа n (без знания X_0) не поможет злоумышленнику определить биты последовательности, поэтому для генерации p и q можно использовать и криптографически небезопасные алгоритмы, например, линейный конгруэнтный генератор.

Для старта генератора можно использовать некоторое «достаточно случайное» число. Для этого может использоваться, например, интервалы между нажатиями пользователем на кнопки клавиатуры, либо даже время доступа к жесткому диску (которое, на самом деле, не постоянно). Таким образом, из одного числа получаем достаточно непредсказуемую, хорошо распределенную длинную последовательность чисел. (Этого бы не получилось, если мы попытались бы постоянно брать в качестве случайного числа, например, значения некоторого таймера; очевидно, что такие числа совершенно не случайны – каждое следующее число всегда больше предыдущего – и относительно легко предсказуемы).

Анализ существующих методов генерации простых чисел

Для проверки чисел на простоту можно выделить два типа алгоритмов: детерминированные и вероятностные. Детерминированные алгоритмы однозначно определяют, является ли заданное число простым или составным. Вероятностные же алгоритмы, как следует из их названия, способны лишь выдать, что с некоторой вероятностью данное число может быть простым.

Плюсы детерминированных алгоритмов очевидны – они дают верный ответ о простоте числа. В то же время, исходя из практических соображений, применение вероятностных алгоритмов (именно для шифрования) более оправданно. Вероятностные алгоритмы работают значительно быстрее, а точность их ответов очень высока.

Самым быстрым детерминированным алгоритмом проверки простоты числа является алгоритм Ленстры-Коена. Самым эффективным вероятностным алгоритмом является тест Миллера-Рабина, который базируется на следующей теореме:

Пусть n – нечетное составное число. Пусть n не делится на 3, $n=2^r t+1$, где $r \geq 1$, t – нечетно. Тогда количество таких чисел a, $0<a\leq n-1$, что либо $a^t \equiv 1 \pmod n$, либо при некотором j, $1 \leq j \leq r$,

$$a^{(n-1)/2^j} \equiv -1 \pmod n,$$

не превосходит n/4.

То есть если выпадает такое a, которое не удовлетворяет требованиям вышеизложенной теоремы, то число n – точно составное. Иначе – возможно, что простое.

Отсюда следует, что если для k случайных значений a мы не обнаружим, что n – составное, то будем считать, что n – простое с вероятностью, не меньшей чем $1-\frac{1}{4^k}$.

Параллельная реализация генератора простых случайных больших чисел

Можно выделить три глобальных подхода к распараллеливанию генератора больших случайных простых чисел – по одному на каждую из решаемых подзадач.

Несмотря на повсеместное использование в задаче больших чисел (и в генераторе случайных чисел, и при проверке на простоту), распараллеливание арифметических действий над большими числами не будет достаточно эффективным по ряду причин. Во-первых, все действия, выполняемые при операциях с большими числами, являются тесно связанными. Это потребует больших затрат на синхронизацию. Во-вторых, существуют эффективные библиотеки для работы с большими числами. Поэтому параллельная реализация не даст ощутимого выигрыша над эффективной последовательной реализацией (по крайней мере, при малом количестве процессоров).

Второй подход – параллельная генерация случайных чисел. В самом деле, случайные числа нужны на протяжении всего процесса решения задачи, а каждое значение X_i в BBS-генераторе может быть вычислено независимо от других. Однако при таком подходе возможны большие простои в работе генерирующих процессов – пока текущее случайное число используется для проверки в алгоритме Миллера-Рабина, следующее число не нужно.

Наиболее эффективным является распараллеливание по тестам Миллера-Рабина – каждый процесс генерирует и проверяет свое число. Это позволяет им действовать практически независимо друг от друга и обеспечивает хорошее масштабирование задачи по количеству процессоров.

В самом деле, в диапазоне от 1 до n находится примерно n/ln(n) простых чисел. Соответственно, в худшем случае необходимо сделать примерно ln(n) проверок чисел на простоту. Для $n = 2^{1024}$ это число

примерно равно 700. Следовательно, если в нашем распоряжении, скажем, имеется не менее 700 процессоров, то нужное число, скорее всего, сгенерируется за один шаг алгоритма. При большем числе процессоров можно просто увеличить размер ключа, что приведет к уменьшению вероятности появления простого числа и увеличению занятости процессоров. Следовательно, потенциал масштабирования данной параллельной реализации неограничен.

При таком подходе от взаимодействующих процессов требуется минимальная синхронизация – только для определения, найдено ли уже каким-либо процессом требуемое число – что позволяет эффективно реализовать такое решение как в системах с общей памятью, так и в системах с распределенной памятью.

Данный подход был реализован и протестирован на двухпроцессорной платформе с общей памятью. Ускорение составило 1,83 раза, соответствующая эффективность – 0,92. Близость последнего параметра к 1 говорит о высокой эффективности распараллеливания программы.

Заключение

В рамках данной работы были разработаны и проанализированы различные методы распараллеливания криптостойкого генератора больших случайных простых чисел. Лучший из подходов был реализован и протестирован. Эффективность параллельной реализации оказалась близка к 1.

Разработанный генератор может использоваться в различных криптосистемах, например, RSA.

В дальнейшем планируется сделать реализацию данного алгоритма для систем с распределенной памятью и протестировать его эффективность на кластере. Кроме того, планируется сделать и сравнить с существующей еще две параллельных реализации генератора: через распараллеливание генерации случайных чисел и через параллельную версию детерминированной проверки чисел на простоту.

Литература

1. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии – М.: МЦНМО, 2003. – 328 с.
2. Шнайер Б. «Прикладная криптография: Протоколы, алгоритмы, исходные тексты на языке Си» – «Триумф», 2002.

3. Кнут Д. Искусство программирования. Т. 2. Получисленные алгоритмы. Вильямс: М.— СПб.—Киев, 2000. 3-е издание.
4. Lenore Blum, Manuel Blum, and Michael Shub. “Comparison of two pseudo-random number generators”, Advances in Cryptology: Proceedings of Crypto '82
5. Koblitz N. Algebraic aspects of cryptography. Springer-Verlag, 1998.
6. Брассар Ж. Современная криптология. – М.: ПОЛИМЕД, 1999
7. Martin Geisler, Mikkel Kroigard, and Andreas Danielsen. “About Random Bits”, 2004
8. Donald Eastlake, Jeffrey Schiller, Steve Crocker. “Randomness Requirements for Security” - Request for Comments: RFC 4086, 2005.

ПРОЕКТ НОВОЙ МАСШТАБИРУЕМОЙ СИСТЕМЫ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ В ЗАДАЧАХ МЕХАНИКИ

Мельникова Н.Б., Орлов С.Г., Шабров Н.Н.
СПбГПУ, Санкт-Петербург

Введение

В настоящее время известно множество программных систем, предназначенных для численного моделирования механических систем. В первой части доклада рассматриваются причины, по которым авторы считают необходимым создание еще одной системы, отчасти аналогичной существующим, но, с другой стороны, лишенной их недостатков. Одна из таких причин состоит в необходимости эффективного использования новых вычислительных архитектур, таких как SMP-машины с многоядерными процессорами и кластеры, составленные из них. Описаны основные черты новой программной системы. Во второй части доклада сделан краткий обзор методов декомпозиции конечно-элементных сеток. Применение этих методов делает возможным распараллеливание сложных вычислительных задач на MPP и SMP-системах. В третьей части проанализированы технические возможности повышения эффективности распараллеливания программ.

Причины создания новой системы численного моделирования

На кафедре «Компьютерные технологии в машиностроении» механико-машиностроительного факультета СПбГПУ разработан ряд моделей механических систем (например, модели тонких тел – стержней, пластин и оболочек) и численных алгоритмов, обладающих преимуществами перед моделями и алгоритмами, используемыми в коммерческих CAE-системах. С другой стороны, становятся доступными параллельные аппаратные архитектуры, на которых многие CAE-системы пока не работают. Что-либо серьезно изменить в этих системах под силу только их разработчикам, круг которых весьма ограничен. Для сторонних разработчиков (каковыми мы являемся по отношению к коммерческим CAE-системам) доступны лишь небольшие изменения и дополнения, например, встраивание в систему пользовательского кода конечного элемента. Едва ли сторонний разработчик в состоянии распараллелить решатель или портировать его на другую операционную систему: для этого нужен исходный код, который зачастую недоступен. Маловероятна и возможность заменить решатель на другой, так как для этого требуется документация на интерфейс решателя. Иными словами, существующие системы слишком закрыты для сторонних разработчиков, и поэтому они нас не устраивают.

Ключевые особенности новой системы численного моделирования

При разработке новой системы мы уделяем большое внимание нескольким ключевым вопросам, желая сделать её гибкой, эффективной и масштабируемой в широком смысле.

Система должна состоять из компонент (модулей), по возможности как можно менее связанных между собой. Ту или иную конфигурацию системы можно набрать из подходящего набора компонент. Сам по себе компонентный подход к разработке ПО не нов, поэтому не будем останавливаться на его преимуществах.

Система должна быть открытой для сторонних разработчиков. Для этого, прежде всего, требуется документация разработчика на интерфейсы, реализуемые компонентами. В качестве технического средства для генерации документации разработчика была выбрана система Doxygen. Для открытости системы также важна доступность имеющихся программных компонент, то есть возможность создавать

их экземпляры и управлять ими. Это автоматически обеспечивается объектной моделью.

Система должна быть масштабируемой в том смысле, что любой сторонний разработчик может добавить новый компонент. В частности, должна быть возможность заменять один компонент на другой, поддерживающий те же интерфейсы. Такое масштабирование обеспечивается, во-первых, компонентным подходом и, во-вторых, открытостью системы.

Система должна быть портируемой на различные операционные системы (ОС). С одной стороны, это традиционно достигается разделением кода, независимого от ОС и кода, специфичного для конкретной ОС. При этом ОС-специфичный код помещается в отдельные модули, образующие промежуточный уровень между ОС и другими модулями. Портирование в этом случае сводится в основном к работе над ОС-специфичными модулями. С другой стороны, портирование некоторых компонент нежелательно по причине большой трудоемкости, а иногда и бессмысленно: например, это относится к GUI. В таких случаях желательно собрать код, не подлежащий портированию, в отдельные модули. Отчасти в этом помогаем предусмотренная в системе возможность создания «составных объектов». Такие объекты могут состоять из частей, реализованных в разных модулях. Концепция «составных объектов» позволяет делить систему «по горизонтали», что делает возможным исключать целые «пласты» при портировании.

Система должна быть масштабируемой в том смысле, что она должна эффективно работать на различных аппаратных архитектурах, в том числе на высокопроизводительных SMP и MPP системах. Этого можно достичь, создавая подходящие (для каждой конкретной архитектуры) реализации решателей – объектов, реализующих численные алгоритмы, но не знающих об объектах, реализующих те или иные модели. Грамотный дизайн интерфейсов, связанных с численным моделированием, позволит избежать изменения множества объектов при портировании на новую аппаратную архитектуру – достаточно будет поменять некоторое фиксированное количество объектов.

Система должна поддерживать удаленный доступ. Это требование отчасти связано с трудностью портирования некоторых компонентов, а отчасти оно полезно само по себе. Например, удаленный доступ позволяет конечному пользователю производить расчеты на

удаленном высокопроизводительном сервере, работая исключительно в GUI-программе на клиентской машине, и иметь почти такой же доступ к результатам расчета, как если бы они находились на клиентской машине. При этом файлы результатов расчета могут быть столь большими, что их скачивание заняло бы довольно много времени. Удаленный доступ в разрабатываемой системе во многом обеспечивается языковой средой, предоставляющей интерпретатор команд. Любое действие над моделью может быть представлено последовательностью команд, поэтому не представляет труда их журнализация и отправка на удаленный сервер. Таким образом, обеспечивается синхронизация исходных данных на клиенте и сервере. Языковая среда также предоставляет возможность организовывать каналы данных между соответствующими друг другу объектами на клиентской и серверной машинах.

Проблемы балансировки нагрузки процессоров на MPP-системах

Плохая масштабируемость параллельного приложения на MPP-системе может быть вызвана а) ростом затрат на коммуникации при увеличении числа используемых процессоров; б) несовершенством схемы распределения вычислительной нагрузки между процессорами.

При увеличении количества процессоров с сохранением размерности задачи увеличивается общее количество вызовов функций MPI в программе. При этом накладные расходы на формирование и отправку сообщений растут, а объем вычислений, приходящихся на один процессор, падает, что и вызывает уменьшение эффективности параллелизации. Все большее негативное влияние в условиях возросшего количества сообщений будет оказывать латентность сети. Для кластеров, узлы которых являются симметричными мультипроцессорами, можно попытаться снизить стоимость коммуникаций, заменив внутри каждого узла многопроцессорную обработку на многопоточную.

Методы распределения нагрузки в современных CAE-пакетах несовершенны – они довольно примитивные, статические и не позволяют корректировать загрузку процессоров в процессе работы приложения. Далее рассматриваются возможные шаги в этом направлении.

Как известно, производительность параллельного приложения, решающего краевую задачу каким-либо сеточным методом, осень

сильно зависит от качества декомпозиции сетки на домены. При декомпозиции необходимо удовлетворить двум требованиям: сбалансировать нагрузку процессоров (для этого количество узлов в доменах должно быть примерно равным) и минимизировать объем коммуникаций (длина границ доменов должна быть минимальной). Для нерегулярных сеток проблема поиска оптимальной декомпозиции нетривиальна; для ее решения разработано уже достаточно большое количество алгоритмов. К ним относятся как координатные методы, так и методы теории графов (любой симметричной матрице может быть единственным образом сопоставлен ненаправленный граф с узлами сетки в качестве вершин, при этом ребро (i,j) графа существует, если узлы с номерами i и j входят в один конечный элемент). Современные CAE-пакеты используют обычно простейшие из методов декомпозиции. Например, применяется алгоритм Катхилл-Макки – узлы сетки перенумеровываются так, чтобы минимизировать ширину ленты матрицы системы, а затем переупорядоченный вектор узловых переменных нарезается равными порциями, соответственно количеству процессоров. Полученный граф системы представляет собой систему уровней смежности, ширина которой (то есть длина границ доменов) минимизирована, что позволяет надеяться на хорошее качество полученных доменов. Однако, на сегодняшний день разработаны и более эффективные процедуры декомпозиции [1]:

1. геометрический метод инерции (основан на том, что все узлы сетки рассматриваются как материальные точки массой, равной валентности узла сетки; сетка рассекается плоскостью, ортогональной главной оси инерции, соответствующей минимальному моменту инерции);
2. спектральный метод (основан на определении второго собственного вектора лапласовой матрицы графа системы с помощью метода Ланцоша);
3. многоуровневый метод (основан на построении последовательности все более грубых графов – аппроксимаций исходного графа, до получения графа малой размерности, разбиении этого графа каким-либо из вышеописанных методов, а потом последовательном применении этого разбиения к графам верхних уровней)
4. локальный алгоритм Кернигана – Лина (служит для улучшения качества уже имеющихся доменов, полученных предыдущими методами);

Численные эксперименты для больших задач показывают, что метод инерции в сочетании с локальным алгоритмом Кернигана – Лина дает достаточно неплохое качество декомпозиции, но спектральный метод и многоуровневый метод, дополненные локальным, приводят к еще лучшей декомпозиции. При этом метод инерции является самым быстрым; почти так же быстро работает многоуровневый метод, а спектральный метод, требующий применения процедуры Ланцоша, довольно неуклюж.

Все эти процедуры предполагается встроить в конечно-элементную MPP-программу и исследовать их работу для различных задач инжиниринга. Также мы предполагаем создать возможность повторной декомпозиции сетки в процессе решения задачи. Такая необходимость возникает, например, в контактных задачах с переменными поверхностями контакта, а также в тех задачах, где необходимо динамическое уточнение сетки в процессе решения (например, в задачах динамики жидкости и газа: представим, что твердое тело движется в газе, разбитом на конечные объемы; тогда нам необходимо иметь самую мелкую сетку вблизи границ движущегося тела).

Программирование в стиле task parallel (параллелизм задач) на кластерных архитектурах

Под кластерными архитектурами будем понимать сейчас любые системы с неоднородным строением памяти, то есть NUMA и MPP-архитектуры. В таких системах для каждого узла память можно разделить на “свою” и “чужую”, где доступ к своей памяти осуществляется значительно быстрее, чем к чужой. При этом адресное пространство может быть единым (как в NUMA), а может быть и своим для каждого узла такой системы, и тогда обмен данными между узлами происходит посредством передачи сообщений. Узел кластерной машины обычно представляет собой симметричную мультипроцессорную систему. Для реализации параллельных алгоритмов на таких архитектурах разумно использовать мультипроцессинг в сочетании с многопоточностью. При этом количество нитей одного процесса соответствует числу процессоров на одном узле системы, а общее количество процессов – числу узлов в системе. В частности, для кластеров рабочих станций и ПК, этот подход можно реализовать с помощью библиотеки MPI (для обмена

между узлами) и библиотеки OpenMP (для порождения нитей и организации их совместной работы внутри одного процесса на каждом узле).

Современные MPP-версии коммерческих CAE-пакетов предназначены для использования на кластерах ПК и написаны с использованием только MPI, в терминах процессов. В результате затраты на коммуникации между процессами внутри одного узла получаются выше, чем могли бы быть при использовании многопоточной обработки.

Существующие реализации MPI для Linux позволяют работать с многопоточными процессами. После вызова функции MPI_Init_threads мы получаем в качестве выходного параметра максимально возможный уровень поддержки нитей в системе. Он может быть, в соответствии со стандартом MPI-2.0, следующим: MPI_THREAD_SINGLE – нет поддержки нитей; процессы могут быть только однопоточными; MPI_THREAD_FUNNELED - процесс может быть многопоточным, но только главная нить (master thread) может вызывать функции MPI (остальные нити могут продолжать работу, пока главная нить выполняет функции MPI); MPI_THREAD_SERIALIZED - процесс может быть многопоточным, и все нити могут делать вызовы MPI, но только по очереди (вызовы MPI не делаются одновременно из двух разных нитей одного и того же процесса); MPI_THREAD_MULTIPLE – все нити процесса могут вызывать функции MPI без ограничений (этот уровень во многих реализациях не поддерживается). Начиная с уровня поддержки MPI_THREAD_FUNNELED, можно организовать работу с нитями. Для этого вызовы функций MPI в главной нити окружаются директивами OpenMP OMP MASTER и OMP END MASTER. Эти директивы не содержат в себе никакой синхронизации значений общих переменных, и для корректной работы программы необходимо явно выполнять синхронизацию нитей, с помощью OMP BARRIER, до входа главной нити в секцию OMP MASTER … OMP END MASTER, либо после выхода главной нити из секции. Обязательным условием эффективной работы такого “гибридного” приложения является одновременная работа всех нитей, даже во время вызова главной нитью функций MPI (отсутствие простаивания нитей). Поэтому нельзя выполнять синхронизацию нитей два раза – перед OMP MASTER и после OMP END MASTER. В дальнейшем мы планируем опробовать

гибридный подход на кластерах ПК, содержащих симметричные мультипроцессоры в узлах.

ПРИМЕНЕНИЕ ПРИКЛАДНЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В НАУЧНЫХ ИССЛЕДОВАНИЯХ

Митрохин Ю.С., Ковнеристый Ю.К., Шудегов В.Е.
Удмуртский государственный университет, г. Ижевск,
Институт металлургии им. А.А.Байкова РАН, г. Москва

Введение

Методы моделирования в физике, химии и в других областях знаний требуют очень больших вычислительных затрат и предъявляют жесткие требования к имеющейся у исследователя вычислительной технике. Раньше такие работы можно было выполнять только на мощных вычислительных центрах которые были дороги и труднодоступны. С появлением персональных компьютеров и ростом их производительности положение дел в этой области изменилось радикально. В настоящее время настольный компьютер имеет более высокую производительность чем большой компьютер 20-летней давности. А после появления параллельных кластеров стало возможным проведение серьезных численных экспериментов во многих ВУЗах и институтах. Одновременно с развитием вычислительной техники развивалось и программное обеспечение для выполнения параллельных вычислений. Основным языком программирования в задачах моделирования был и остается язык Fortran, хотя язык C тоже получает распространение в современных программах. Основным средством создания параллельных программ для распределенных кластеров является MPI. Для компьютеров с общей памятью можно использовать OpenMP. Почти все современные трансляторы с языков Fortran 90 и C имеют опции для работы с OpenMP. То есть современные трансляторы позволяют автоматизировать процесс написания параллельных программ.

В связи с этим, во многих ведущих научных центрах стали создаваться большие и сложные комплексы прикладных программ для моделирования реальных физических и химических процессов. Часть их этих программ являются коммерческими а часть свободно-распространяемыми на основании GNU лицензии. Многие пакеты программ хорошо документированы и имеют много примеров

входных и выходных файлов, что значительно облегчает их освоение. Кроме того, имеется еще возможность подписаться на mailing list. Эта возможность позволяет пользователям задавать по электронной почте вопросы разработчикам программ и получать на них квалифицированные ответы. При инсталляции программ, обычно, выдается список всех платформ на которых может быть установлен данный пакет. Наибольшее распространение в настоящее время получила платформа Intel и ОС Linux. Даже там, где есть более мощная техника, обучение пользователей проводится на платформе Intel. Это связано не только с массовым распространением этой платформы, но и с тем, что трансляторы с языка Fortran 90 для персональных компьютеров хорошо отлажены и создают работоспособные загрузочные модули. На других платформах во время инсталляции и последующей работы программ часто возникают проблемы. Хуже всего с этим обстоят дела на платформе Silicon Graphics.

Методы моделирования, теория

Наибольшее распространение при моделировании физических и химических систем получил метод молекулярной динамики (МД). Он появился вместе с первым языком высокого уровня Fortran. В следующем году мы будем отмечать 50-летний юбилей этого метода [1]. В классическом методе МД моделирование проводится на атомном уровне, но из-за большой массы атомных ядер, движение атомов можно описывать классическим уравнением Ньютона:

$$\vec{F} = m\vec{a} \,. \qquad (1)$$

При этом, основной проблемой классической МД было и остается – выбор модели межатомного взаимодействия, т.е. выбор параметров и аналитического вида потенциала межатомного взаимодействия атомов. Например, из соотношений Коши для упругих констант в кубическом кристалле:

$$c_{12} = c_{44} \qquad (2)$$

следует, что парный межатомный потенциал не может правильно описать межатомное взаимодействие атомов в переходных металлах,

т.к. там это соотношение, в основном, не выполняется. Для кремния и углерода пришлось вводить специальный более сложный 3-х частичный потенциал [2]. Для металлических систем в работе [3] (Sutton-Chen) был предложен многочастичный потенциал:

$$E_{tot} = \sum_{i=1}^{N} V_i = \sum_{i=1}^{N} \left[\frac{1}{2} \sum_{j \neq i}^{N} \varepsilon_{ij} \left(\frac{a}{r_{ij}} \right)^n - c_i \varepsilon_{ii} \sqrt{\rho_i} \right], \qquad (3)$$

где

$$\rho_i = \sum_{j \neq i}^{N} \left(\frac{a}{r_{ij}} \right)^m .$$

Потенциал Саттона-Чена (3) является простейшей реализацией метода погруженного атома (EAM) [4], и в настоящее время наиболее часто применяется для моделирования металлических систем.

В пионерской работе Car Parinello [5], был предложен новый подход моделирования методом молекулярной динамики из первых принципов (ab initio МД). Основное отличие состояло в том, что силы межатомного взаимодействия находились здесь, на основании хорошо известной теоремы Геллманна-Фейнманна, как градиент от полной энергии моделируемой системы:

$$\vec{F} = -\frac{\partial E_{tot}(\rho)}{\partial \vec{R}},$$

где полная энергия E_{tot} зависит от расположения атомов в суперячейке, а следовательно, и от распределения электронной плотности ρ. Для этого нужно было решить квантово-механическую задачу для заданного распределения атомов в расчетной области, т.е. решить уравнение Кона-Шема [6] (аналог уравнения Шредингера для кристалла). Однако, после того как найдены силы, дальнейшая схема моделирования была такой же как и раньше, т.е. движение атомов по-прежнему описывалось с помощью классической механики (1). В этом методе нет подгоночных параметров, а следовательно нет неконтролируемых приближений. Предложенный новый подход получил название метода Car Parinello (CPMD). Таким именем был

назван один из известных пакетов программ разработанных в филиале фирмы IBM в Швейцарии, где работает один из авторов этого метода (M.Parinello). Хотя теорема Геллманна-Фейнманна была известна еще с 30-х годов прошлого века, ее практическое применение стало возможно только после появления мощных современных суперкомпьютеров. Дело в том, что электронная часть задачи, т.е. вычисление распределения электронной плотности в суперячейке большого размера (50-100 атомов), сама по себе, требует очень больших вычислительных затрат. В методе CPMD она решается многократно и на нее уходит больше 99% машинного времени.

Среди других пакетов ab initio МД можно назвать следующие: VASP, CASTEP (коммерческие), ABINIT, PWSCF, SIESTA. В этих пакетах число атомов в моделируемой системе, обычно, не превышает 100. Очевидно, этого не достаточно для получения хорошей статистики. Также мал и временной интервал моделирования, так как обычно, число временных шагов не превышает 2000-3000, а величина шага составляет 1–3 fs (10^{-15} сек.). Но даже для такого малого числа атомов и временных шагов, компьютерное время моделирования на современных параллельных кластерных системах измеряется сотнями часов. Для сравнения можно сказать, что в классической МД число атомов в расчетной области составляет 10^3-10^6, а число временных шагов часто превышает 10^6.

Таким образом, можно сделать вывод, что с появлением ab initio МД, классическая молекулярная механика не утратила своих позиций. Для сложных органических молекул типа ДНК, она по-прежнему остается единственным инструментом для моделирования. Эти два подхода взаимно дополняют друг друга. Если раньше параметры потенциала межатомного взаимодействия можно было получить только из эксперимента методами многомерной аппроксимации, то теперь это можно сделать на основании первопринципных расчетов. Такой подход был продемонстрирован в работе [7]. Автор этой работы использовал метод погруженного атома (EAM). Параметры EAM он получил на основании первопринципных методов моделирования для той же системы. При этом полученные им результаты очень хорошо совпадали с экспериментом. Среди пакетов реализующих классическую МД можно назвать MOLDY и DL_POLY [8].

Моделирование системы Ni_3Al, результаты

На 12-процессорном кластере PARC в УдГУ были выполнены работы по моделированию двойных и тройных сплавов на основе Ni_3Al в твердом и жидком состоянии методами классической и первопринципной молекулярной динамики. Использовались как классические пакеты МД (DL_POLY) [8], так и первопринципные VASP [9-11]. Кроме того, электронная структура этих сплавов рассчитывалась зонными методами TB-LMTO-ASA [12] FP-LMTO[13] При моделировании пакетом VASP, число атомов в суперячейке было равно 64. Исходная конфигурация атомов соответствовала гране-центрированной кубической решетке (ГЦК). Шаг интегрирования по времени был выбран равным 1.5 fs (10^{-15} сек), число временных шагов составляло 3000. Полное время моделирования одного варианта при фиксированной температуре T, составляло более 100 часов. Максимальная производительность достигалась на 6 процессорах.

Такой же расчет был выполнен на 2-х процессорном кластере Opteron в ИМЕТ. Там полное расчетное время для одной точки по температуре было равно 16 часов. Для сравнения, этот же расчет был повторен на 2-х процессорном кластере Itanium на ВЦ РАН (г.Москва). Время расчета уменьшилось в 3 раза и составило немногим более 5 часов. Если исходить из сравнения цена/стоимость, то можно сделать вывод, что хотя Itanium считает в 3 раза быстрее, стоимость его тоже примерно в 3 раза больше. По этой причине многие пользователи выбирают Opteron. Мы тоже сделали такой выбор. Здесь следует отметить один важный для практики ньюанс. На Opteron'е у нас было 4Гб оперативной памяти, а на Itanium – 16Гб. При работе в параллельном режиме почти все параллельные программы завершаются аварийно при превышении оперативной памяти. Поэтому важно иметь не только мощный компьютер, но и соответствующую ему оперативную память. На Itanium все расчеты прошли без сбоев, а на Opteron'е пришлось уменьшить потребности а ОП до 4Гб.

При моделировании классической МД (пакет DL_POLY) использовались следующие параметры: число атомов в суперячейке было равно 1372 (5-кратное размножение элементарной ячейки по каждой из осей x,y,z), временной интервал был тот же 1.5 fs, чисо шагов на стадии релаксации – 300000, число шагов на стадии сбора статистики – 1000000. Время моделирования одного варианта по температуре на кластере PARC составляло 25 часов. Для получения

загрузочных модулей использовался транслятор IFC-9.0 фирмы Intel совместно с библиотекой MKL-8. Библиотека MKL дает выигрыш на процессорах Intel до 20-30 %. Кроме того, она полностью соответствует транслятору IFC, что снимает многие проблемы при сборке загрузочных модулей. Все расчеты проводились в OS Linux (SuSE 9.0).

В данной работе моделировался процесс плавления сплава Ni_3Al. Для этого использовался ансамбль NPT в пакете DL_POLY и ансамбль NVT в пакете VASP. В обоих случаях использовалось одно и то же уравнение состояния, которое бралось из результатов моделирования в NPT ансамбле. Температурный интервал составлял 300 К – 2000 К с шагом 100 К. При каждой температуре, начиная с 300 К система сначала подвергалась релаксации ($3 \bullet 10^5$ МД шагов), а затем производилось усреднение и расчет физических характеристик системы. Основные характеристики – это полная энергия системы, коэффициент диффузии и радиальная функции распределения атомов g(r). Точка плавления определялась по скачку на графике зависимости полной энергии системы от температуры E(T), по резкому увеличению коэффициента диффузии D и по изменению формы кривой функции радиального распределения атомов g(r).

На рис. 1 приведены радиальные функции распределения g(r) для пары атомов Ni-Ni для четырех состояний моделируемой системы с температурами T = 300 К, T = 1300 К, T = 1600 К и T = 1700 К, слева направо и сверху вниз, соответственно. Температура T = 300 К соответствует комнатной температуре и g(r) здесь имеет форму характерную для кристаллического состояния. Следующая кривая g(r) для T = 1300 К соответствует сильно нагретому образцу. Здесь мы видим, что начинается тепловое размытие пиков g(r), т.е. происходит частичное разрушение кристаллической решетки. Третий график , T =1600 К, соответствует состоянию предплавления. Кристаллическая решетка почти разрушена, но остались еще кластеры с кристаллической структурой. И на последнем графике мы имеем g(r) типичную для жидкого состояния. Таким образом, можно сделать вывод, что в нашем численном эксперименте температура плавления сплава Ni_3Al равна 1700 К. Экспериментальное значение температуры плавления для этого сплава – 1650 К [16]. Температура плавления чистого никеля равна 1726 К.

Здесь нужно отметить, что точность определения температуры в таком численном эксперименте невысокая: $\Delta T = \pm 100$ градусов.

Основное его назначение – проверка модели межатомного взаимодействия и надежности программного обеспечения. Это очень жесткий тест для любой программы молекулярной динамики. Для сравнения нами был выполнен аналогичный расчет этой же системы с парным потенциалом Морзе [13, 14]. В этом случае точка плавления получилась на 300 градусов выше, что говорит о том, что парный потенциал Морзе плохо описывает эту систему. Этого и следовало ожидать, так как соотношения Коши (2) хуже всего выполняется для никеля в ряду 3d- переходных металлов. Если же не учитывать уравнение состояния (V = cont), или неправильно его учитывать, то плавление может не быть, даже при значительном перегреве системы.

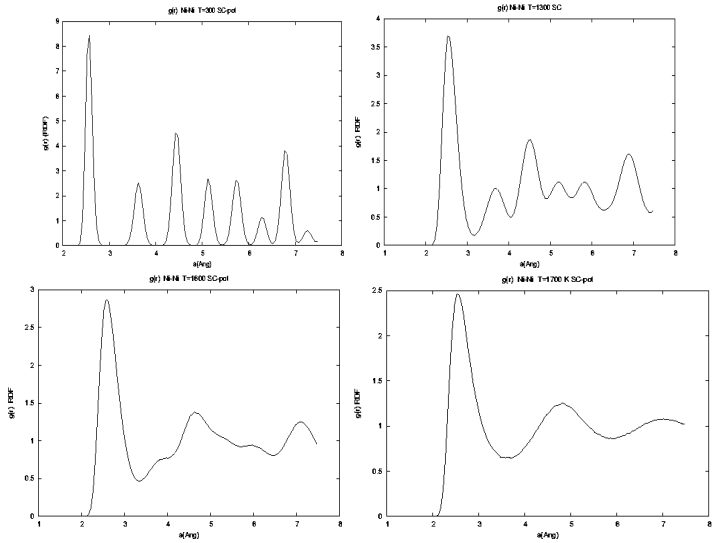


Рис. 1

Литература

1. B.J.Alder, T.E.Wainwright, J. Chem. Phys., 27, 1208 (1957).
2. J.Tersoff, Phys. Rev. B, 37, 6991 (1988).
3. A.P.Sutton, J.Chen, Phil. Mag., 61, 139 (1990).
4. M.S.Daw, M.I.Baskes, Phys. Rev. B, 29, 6443 (1984).
5. R.Car, M.Parinello, Phys. Rev. Let., 55, 2471 (1985).
6. W.Kohn, L.J.Sham, Phys. Rev. 140, A1133 (1965).
7. F.Ercolessi, Europhys. Lett., 26, 583 (1994).
8. W.Smith, T.Forester, J.Molec. Graphics, 14, 136 (1996).

9. G.Kresse, Hafner, Phys. Rev. B, 47, RC558 (1993).
10. G.Kresse, J.Futhmuller, Comp. Mat. Sci., 6, 15 (1996).
11. G.Kresse, J.Furthmuller, Phys. Rev. B, 54, 11169 (1996).
12. O.K.Andersen, Phys. Rev. B, 12, 3060 (1975).
13. S.Y.Savrasov, Phys. Rev. B, 54, 16470 (1996).
14. P.M.Morse, Phys. Rev., 34, 57 (1929).
15. L.A.Girifalko, V.G.Weizer, Phys. Rev., 114, 687 (1959).
16. Н.Н.Степанова и др., ФММ, 1999, Т. 88, Вып. 4, С. 69-75.

Контакты
Тел. +7 (3412) 916-089, 916-090
e-mail: mit@uni.udm.ru

ТРЁХУРОВНЕВАЯ СИСТЕМА РАЗДЕЛЕНИЯ КОДА ДЛЯ РАЗРАБОТКИ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ПАРАЛЛЕЛИЗУЕМЫХ ПРОГРАММ

Монахов В.В., Керницкий И.Б., Евстигнеев Л.А.
СПбГУ, Санкт-Петербург

Введение

Одной из важных особенностей современного программного обеспечения является сочетание использования высокой производительности компьютеров с развитыми средствами взаимодействия с пользователем. При этом возникает несколько проблем. Во-первых, большинство библиотек для высокопроизводительных вычислений в области физики написано на языке FORTRAN, что не даёт возможности пользователям использовать другие языки программирования, а также создавать программы с развитым пользовательским интерфейсом и обеспечивать поддержку системного программирования. Во-вторых, “распараллеливание” программы, первоначально рассчитанной на работу в обычном непараллельном режиме, требует больших затрат. В-третьих, программы, разработанные на персональных компьютерах, часто плохо переносимы в распределённые среды. Нами предлагается стратегия решения данных проблем путём разделения исходного кода программы на три уровня.

Трёхуровневая система разделения кода

Использование DLL – наиболее оптимальный способ вызова функций из программ на различных языках программирования. Он позволяет использовать исходный код без изменений. Это особенно важно при вызове функций из математических библиотек, так как сохраняется стабильность и быстродействие алгоритмов. Другим важным достоинством является то, что функции из DLL можно вызывать из интерпретируемых языков программирования, в частности - BARSIC.

Для оптимизации доступа к высокопроизводительным библиотечным подпрограммам, а также для преодоления ряда возникающих при этом проблем нами была разработана трёхуровневая система разделения исходного кода программ.

- Нижний уровень состоит из стандартных подпрограмм пакетов высокопроизводительных вычислений, написанных на языке FORTRAN, которые компилируются в библиотеки DLL/SO. Ни специалисты из предметной области, ни профессиональные программисты не модифицируют исходный код пакетов, что гарантирует его работоспособность и надёжность.
- Средний (промежуточный, оболочечный) уровень обеспечивает сокрытие особенностей реализации библиотек нижнего уровня, а также необходимые преобразования данных при передачи их с верхнего уровня на нижний и с нижнего уровня на верхний. В результате получаются оболочечные библиотеки DLL/SO, обеспечивающие доступ к библиотекам нижнего уровня, а также реализацию необходимых дополнительных алгоритмов. Кроме того, на этом уровне обеспечивается реализация дополнительных алгоритмов и библиотек системного характера. Программное обеспечение этого уровня пишут профессиональные программисты.
- Верхний уровень включает в себя программный код из предметной области, написанный научными работниками. Из него осуществляется доступ к библиотекам среднего (промежуточного) уровня.

Данная трёхуровневая система успешно использована нами для добавления в создаваемый нами программный комплекс BARSIC [1,2]

возможностей в области высокопроизводительных вычислений. Мы использовали алгоритмы, представленные в открытых библиотеках из репозитария www.netlib.org, а также сайтов http://www.math.niu.edu/SL2/, http://www.fftw.org/ и других. Существенная часть библиотек алгоритмов представлена в исходных кодах на языке FORTRAN (LAPACK, ODEPACK, FFTPACK, SLEIGN2). Для использования этих алгоритмов на основе трёхуровневой системы нами был решён ряд проблем.

Проблемы написания программного обеспечения промежуточного уровня

При вызове из других языков программирования алгоритмов из библиотеки DLL, скомпилированной на FORTRAN, могут возникнуть проблемы технического характера, решение которых в вызывающем коде приводит к потере производительности или невозможно вовсе. Точно так же могут возникнуть проблемы сопряжения с программным обеспечением верхнего уровня. Перечислим проблемы обоих типов:

1. Необходимость выравнивания массивов исходных данных при передаче их в библиотеки высокопроизводительных вычислений. В частности, данные типа double (real*8 по IEEE754) должны быть выровнены на границу 8 байт. По результатам наших исследований отсутствие выравнивания в ряде случаев приводило к падению производительности в 2-3 раза (алгоритмы сингулярного разложения матриц и решения СЛАУ при вызове их из библиотеки LAPACK).
2. Некоторые алгоритмы могут требовать переключения состояния процессора FPU, например, временного отключения обработки исключений в случае переполнения (вариант DSYEVR алгоритма поиска собственных чисел в LAPACK).
3. Необходимость эффективного преобразования типов исходных данных при передаче их на более нижний уровень. Например, из extended (real*10), используемого в Object PASCAL и BARSIC, в double, используемого в C++ и FORTRAN.
4. Повышение удобства в использовании алгоритма нижнего уровня при написании программного обеспечения высокого уровня. Часто функции широко распространенных библиотек, реализующие различные численные алгоритмы, имеют большое количество параметров (до 20-25), позволяющее

осуществлять тонкую настройку алгоритма или получать дополнительные результаты. Но в большинстве случаев при использовании таких функций существенными оказываются всего 5-7 параметров. Оптимальным является предоставление оболочкой промежуточного уровня различных интерфейсов доступа к такой функции – простого (где требуются только наиболее существенные параметры, а остальные задаются стандартными, определяемыми из общий соображений значениями) и сложного, позволяющего задавать все возможные параметры.

5. Согласование на промежуточном уровне соглашений о вызове в случаях, когда язык, используемый на высоком уровне, не позволяет задать модификаторы вызова функций из DLL (например, cdecl или stdcall).

Для решения указанных проблем мы считаем целесообразным использование языка C++.

Таким образом, подключение к программному обеспечению верхнего уровня функций из математической библиотеки осуществляется с помощью промежуточной оболочечной DLL, написанной на C++, которая, в свою очередь, вызывает подпрограммы из основной DLL, скомпилированной из исходных кодов на FORTRAN. Для осуществления связывания между библиотеками удобно использовать неявное связывание (implicit linking).

Такой способ был нами использован для подключения к BARSIC ряда подпрограмм из библиотек LAPACK, FFTPACK, ODEPACK, SLEIGN2. Использование DLL также дает дополнительную возможность подключения различных версий алгоритмов на этапе запуска и выполнения программы. Например, в зависимости от среды выполнения, можно использовать версии DLL, имеющие одинаковый интерфейс, но скомпилированные с оптимизацией под соответствующую платформу. Причем такая замена происходит прозрачно с точки зрения программиста, пишущего программу в предметной области (верхний уровень).

Данная схема нами опробована и показала высокую эффективность и гибкость. Основным преимуществом предложенной схемы является чёткое разграничение ответственности программного обеспечения каждого уровня, обеспечение гибкости без потери производительности, а также возможность избавить программистов от

вникания в особенности предметной области, а специалистов-предметников – от написания системного программного обеспечения. Кроме того, на каждом уровне используется язык программирования, наиболее адекватный задачам, решаемым на этом уровне.

Ещё одна возможность, которая в потенциально может быть обеспечена заменой DLL промежуточного и нижнего уровня без изменения программного обеспечения верхнего уровня – это переход к использованию версий алгоритмов, предназначенных для параллельных вычислений. Существенным достоинством такого подхода является возможность отладить в тестовом режиме программный код, решающий некоторую задачу на отдельном компьютере в непараллельном режиме, после чего запустить вычисления на кластере или многопроцессорной системе.

Особенности использования пакета ScaLAPACK

Многие алгоритмы линейной алгебры эффективно выполнять в параллельных системах. Чаще всего физики используют их реализацию из пакета LAPACK (Linear Algebra PACKage). Для выполнения простейших операций (умножение, скалярное произведение, векторное произведение) над объектами линейной алгебры (вектор, матрица) LAPACK использует пакет BLAS (Basic Linear Algebra Set). Фактически, от конкретной реализации BLAS зависит производительность и модель функционирования алгоритмов пакета LAPACK.

Для обеспечения параллелизации вычислений в кластерных системах и поддержки однородности самой системы используются специальные библиотеки, осуществляющие передачу данных и команд между элементами кластера. Наиболее распространенными являются библиотеки MPI (Message Passing Interface) и PVM (Parallel Virtual Machine). Они обе имеют множество реализаций, как открытых, так и коммерческих, предназначенных для различных платформ - Windows, Unix, суперкомпьютеры.

Существует свободно распространяемая версия LAPACK, реализующая параллельные вычисления поверх MPI и PVM – ScaLAPACK. Пакет ScaLAPACK в работе с матрицами опирается на набор библиотек, осуществляющих примитивные операции и распределенные вычисления:

- BLAS: Basic Linear Algebra Set – та же, что и для LAPACK.

- LAPACK: Библиотека локальных (непараллельных) методов линейной алгебры.
- BLACS: Basic Linear Algebra Communication Subprograms – набор подпрограмм, предназначенных для распределенной работы с объектами линейной алгебры. По сути является оболочкой над конкретной реализацией распределенной виртуальной машины (PVM, MPI).
- PBLAS: Набор подпрограмм, по функциональности схожий с BLAS. Используется в качестве аналога BLAS для распределенных вычислений.

Хотя по функциональности ScaLAPACK схож с LAPACK, порядок вызова его подпрограмм существенно отличается. В первую очередь эти отличия связаны с тем, как ScaLAPACK реализует распределенные вычисления. Для распараллеливания в ScaLAPACK была выбрана схема организации процессов (программ, выполняемых на разных компьютерах или процессорах), в которой они образуют двумерную сетку, т.е. каждому процессу сопоставляется номер строки и номер столбца в сетке. Это позволяет распределять данные между процессами наиболее эффективным образом. Блочно-циклическая схема (block-cyclic distribution) , используемая ScaLAPACK при работе с плотными матрицами, представляет собой такое распределение данных, при котором сначала матрица разбивается на блоки равного размера, которые, в свою очередь, передаются процессам циклически.

Другое важное отличие в использовании подпрограмм ScaLAPACK от LAPACK заключается в порядке вызова самих методов. Необходим следующий порядок:

1. Инициализация сетки процессов
2. Распределение данных среди процессов или задание их таким образом, чтобы локально у каждого процессы находились нужные блоки (с учетом блочно-циклической схемы). Инициализация дескриптора распределенной матрицы.
3. Вызов подпрограммы ScaLAPACK
4. Освобождение сетки процессов

Также необходимо отметить, что процессы, запускаемые на всех узлах сети, полностью идентичны во всём, кроме данных. При этом каждый процесс имеет свой номер строки и столбца в сетке процессов.

Параллелизация программ с использованием трёхуровневой системы разделения кода

Приведенные выше особенности параллельной реализации LAPACK делают сложной, но возможной замену LAPACK на ScaLAPACK без изменения кода верхнего (предметного) уровня. При этом трёхуровневая схема быть дополнена и расширена.

Для определения необходимых дополнений выделим проблемы, связанные со структурой ScaLAPACK:

- Необходимо инициализировать сеть.
- На каждом узле сети должен выполняться одинаковый код.
- Для каждой матрицы, используемой в распределенных вычислениях, необходимо осуществлять распределение данных и создавать дескриптор.
- Не все вычисления необходимо распараллеливать, так как работа с матрицами малых размеров более эффективна в LAPACK на локальном компьютере.

Первая проблема приводит к необходимости добавление в оболочечную DLL функции инициализации модуля ScaLAPACK, так как параметры инициализации могут варьироваться в зависимости от количества компьютеров в виртуальной машине. Эта инициализация может проходить автоматически – при поступлении из программы верхнего уровня в промежуточную DLL команды загрузки DLL. Параметры инициализации должны настраиваться из отдельной программы или просто задаваться в текстовом файле и при необходимости модифицироваться администратором кластерной или многопроцессорной системы.

Вторая проблема не может быть решена напрямую из оболочечной DLL – для этих целей мы предлагаем использовать схему, аналогичную использованию сервисов (демонов) в операционной системе. На каждом узле виртуальной машины запускается процесс (сервис параллельных вычислений для программы верхнего уровня), который ожидает команд со стороны основного процесса. Основным в данном случае является процесс, работающий на узле, где выполняется программа. Связь с этим процессом из программы осуществляется через оболочечную DLL. Такой сервис должен поддерживать, как минимум, следующие операции – инициализация дескрипторов матрицы, распределение матрицы по сети (PxGEMR2D),

вызов необходимой операции параллельной обработки данных, сбор данных на основной узел (PxGEMR2D).

Это также устраняет и третью проблему.

Четвёртая проблема должна решаться на уровне оболочечной DLL.

Таким образом, общая схема использования ScaLAPACK из программы верхнего уровня может выглядеть следующим образом:

- пользователь из программы инициализирует загрузку DLL LAPACK и ScaLAPACK, при этом на виртуальной машине для распределенных вычислений запускается сервис обработки запросов, и инициализируется сеть процессов ScaLAPACK. Сервис устанавливает контакт с оболочечной DLL.

- пользователь запрашивает оболочечную DLL о выполнении операции, матрицы проходят предварительную обработку в этой DLL (преобразование типа данных, выравнивание на параграф), и на основании некоторых критериев там принимается решение о непараллельном выполнении операции в LAPACK или параллельном - в ScaLAPACK. В последнем случае матрицы и информация о требуемой операции передаются сервису виртуальной машины, где данные распределяются по сети, вызывается подпрограмма вычислений из ScaLAPACK, информация собирается из сети и передается в оболочечную DLL, после чего возвращается в программу верхнего уровня.
- по окончании работы со ScaLAPACK пользователь вызывает из оболочечной DLL команду, выгружающую из памяти LAPACK и деинициализирующую ScaLAPACK, освобождая ресурсы и останавливая сервисы на виртуальной машине.

Описанная схема накладывает ряд ограничений, которые несколько снижают эффективность ScaLAPACK. В частности, для каждой вызываемой операции необходимо распределять и собирать данные, что при относительно низкой скорости связи между узлами может нивелировать выигрыш в производительности. Эту проблему можно обойти, если в программе верхнего уровня отказаться от использования доступа к отдельным элементам двумерных массивов как основного метода программирования алгоритмов, а выполнять операции с матрицами как единым целым. В этом случае при первом вызове подпрограммы ScaLAPACK матрица может быть распределена между узлами сети, и позволит собирать данные не по завершении

функции, а только в случае необходимости. Например, если пользователь запросит работу с отдельными элементами матрицы. Такого рода работа легко может быть организована в программном комплексе BARSIC, поскольку в языке программирования BARSIC реализована концепция блоков кода матричной алгебры, в которых действия с матрицами осуществляются не поэлементно, а как с цельными объектами. Аналогичные усовершенствования могут быть проделаны в математических пакетах (Maple, Mathematica,MatLab и др.), а также в объектно-ориентированных языках программирования (Java, C++ и др.).

Следует отметить, что предложенная схема распараллеливания должна быть особенно эффективна при выполнении операций линейной алгебры на многоядерных процессорах или в многопроцессорных системах, так как при этом накладные расходы на передачу данных будут минимальны. В ближайшее время мы планируем реализовать изложенную методику параллелизации и проверить её эффективность.

Литература

1. В. В. Монахов и др. BARSIC: программный комплекс, ориентированный на физика-исследователя. Программирование, 2005, N3, с.68-80.
2. V. V. Monakhov *et all*. BARSIC: A Programming System for Physicists. Programming and Computer Software, Vol. 31, No. 3, 2005, pp. 157–165.

CAME&L. – СРЕДСТВО ВЫПОЛНЕНИЯ КЛЕТОЧНЫХ АВТОМАТОВ. ОСНОВЫ И ТЕНДЕНЦИИ РАЗВИТИЯ

Наумов Л.А.

Санкт-Петербургский государственный университет информационных технологий, механики и оптики. Санкт-Петербург
University of Amsterdam. Section Computational Science. Amsterdam

Название обсуждаемого в настоящей работе продукта CAME&L [1–4] представляет собой аббревиатуру и означает «Cellular Automata Modeling Environment & Library», среда выполнения и библиотека разработчика клеточных автоматов.

Как известно, клеточные автоматы [5] представляют собой дискретные динамические системы, поведение которых может быть полностью описано в терминах локальных зависимостей [6]. Область их применения чрезвычайно широка [7–9].

Они являются дискретными эквивалентами понятия «поле». Кроме того, клеточные автоматы образуют общую парадигму параллельных вычислений подобно тому, как машины Тьюринга образуют парадигму последовательных вычислений. Таким образом, они могут быть использованы для реализации параллельных алгоритмов, как модели, обладающие естественным параллелизмом.

Программное обеспечение CAME&L задумывалось, как универсальное, расширяемое средство, предоставляющее широкий спектр инструментов для решения научных и образовательных задач на основе клеточных автоматов. Причём анализ более чем двадцати пяти существующих средств решения задач с помощью клеточных [10] автоматов продемонстрировал адекватность и необходимость разработки нового пакета в силу следующих обстоятельств.

- Большинство существующих программ не удовлетворяют современным требованиям к пользовательскому интерфейсу и не позволяют использовать возможности, которые стали обычными и неотъемлемыми. В настоящее время отсутствуют программные продукты со столь удобным пользовательским интерфейсом, как в предлагаемой среде. В частности, в ней реализованы такие возможности, как
    - поддержка буфера обмена;
    - поддержка отмены/повторения последней операции;
    - поддержка печати;
    - возможность сохранения состояния решётки в виде рисунков для иллюстрирования публикаций.
- Все известные продукты накладывают жесткие ограничения на используемый тип клеточных автоматов. Разрабатываемая среда универсальна и позволяет использовать модели, принадлежащие даже более широкому классу, чем класс «клеточные автоматы».
- Все или почти все средства используют разные языки для описания автомата, причём иногда эти языки весьма сложны. То есть, возникает ещё один навык, которым должен овладеть исследователь прежде, чем приступить к решению задачи.

- Большинство известных сред моделирования клеточных автоматов разработаны для операционных систем семейства Linux. Предлагаемая среда является Windows-приложением. Однако необходимо отметить, что это обстоятельство может быть отнесено, как к достоинствам, так и к недостаткам данного проекта.
- Подавляющее большинство существующих продуктов не «моделируют» клеточные автоматы, а только «имитируют» их, так как не используют естественного параллелизма этих систем. Они не поддерживают никакого инструментария для параллельных или распределённых вычислений. Для решения задач на кластерной платформе программное обеспечение CAME$_{\&}$L использует специализированный сетевой протокол CTP (Commands Transfer Protocol) [11–14], разработанный автором, которой нередко используется сторонними разработчиками безотносительно всего описываемого программного обеспечения в целом. Поддержка многопроцессорной платформы также предусмотрена.

Рассматриваемое в настоящей работе программное обеспечение состоит из трёх основных частей. Во-первых, это – сама среда, приложение с богатым и дружественным пользовательским интерфейсом, имеющее многодокументный интерфейс для того, чтобы управлять несколькими экспериментами одновременно, а также реализовывать межавтоматные взаимодействия в процессе моделирования. Помимо этого она обеспечивает сохранение документов в формате XML (файлы с расширением «cml»). Для уменьшения размера этих файлов информация о состоянии решетки автомата, внутри документа, может быть, по желанию, сжата с помощью алгоритма BZip2.

Второй, важной частью программного обеспечения является библиотека разработчика клеточных автоматов CADLib (Cellular Automata Development Library), представляющая собой набор C++-классов [15], которые являются простым и богатым инструментарием для создания так называемых «компонентов», элементов решения задачи в средстве CAME$_{\&}$L. Библиотека предоставляется для использования и расширения пользователями при разработке собственных решений. Помимо классов, она содержит функции,

макроопределения и константы, делающие разработку компонентов настолько простой, насколько это возможно.

Третьей частью пакета являются «стандартные компоненты» – базовый набор «кирпичиков», из которых могут быть построены решения пользовательских задач.

Важнейшей идеей и новшеством рассматриваемого программного обеспечения является декомпозиция выполняемых автоматов. Каждый автомат представляется набором из четырёх компонентов: решётки (grid), метрики (metrics), хранилища данных (datum) и правил (rules).

Идеология компонентов позволяет «собирать» автоматы с различной функциональностью, переходить от одной задачи к другой (пробовать решать одну и ту же задачу на разных решётках, с различными метриками, на разых вычислительных системах и т.п.), а также распределять разработку программного обеспечения эксперимента между различными исполнителями.

При этом каждый компонент декларирует список собственных параметров, изменение которых обеспечивает его настройку.

Компонент, реализующий правила автомата, помимо этого декларирует также список анализируемых параметров, характеризующих течение эксперимента. Среда позволяет наблюдать изменение их значений в динамике, строить графики, файлы отчётов и т. п.

Для анализа этих параметров, характеризующих систему, предусмотрен пятый тип компонентов – анализаторы (analyzer).

В стандартный набор входит специализированный анализатор для построения графиков динамики производительности вычислений, что позволяет настраивать систему для достижения оптимальных результатов.

Реализация метрики в виде отдельного компонента позволяет использовать нестандартные системы координат, как, например, обобщённые координаты [16], четыре вида которых входят в набор стандартных компонентов.

При постановке эксперимента для решения задачи пользователь может применять компоненты, входящие в набор стандартных, или создать свои собственные. С точки зрения разработчика каждый компонент – потомок некого класса библиотеки CADLib или другого компонента.

Может показаться, что владение языком C++ является слишком сложным навыком, чтобы требовать его от пользователя-

исследователя. Здесь требуется отметить, что, во-первых, знание основ программирования в любом случае необходимо учёному, занимающемуся численным моделированием. Таким образом, этот навык не является чуждым и, скорее всего, был уже получен им на этапе высшего образования. Во-вторых, идеология расширяемой среды в перспективе может избавить пользователя от необходимости владения C++, так как с помощью этого языка может быть разработан компонент правил, реализующий выполнение описания функции переходов на неком языке высокого уровня, который будет сочтён подходящим. При этом исходный код на данном языке будет являться параметром этого компонента правил. В результате, один компонент будет реализовывать не единственную функцию переходов, а целый класс.

Помимо всего вышесказанного, тот факт, что библиотека CADLib и стандартные компоненты доступны с исходным кодом, имеет, как недавно выяснилось, и педагогическое значение, так как они используются, как пример для изучения объектно-ориентированного проектирования и программирования на языке C++.

Проект $CAME_{\&}L$ был неоднократно представлен и описан с разных сторон и на разных языках [1–4, 17, 18], однако в заключение хотелось бы остановиться на том, в какой части ведётся работа над проектом в настоящее время и какие новшества в нём появились с момента предыдущей публикации.

- Появились компоненты (решётки, хранилища данных и метрики) для построения трёхмерных моделей. В частности, решётка, поддерживающая отображение трёхмерного пространства посредством OpenGL.
- Появилась метрика, основанная на кривой Пеано (разработана студентом СПбГУ ИТМО И. Акишевым, выполнявшим курсовую работу посредством программного обеспечения $CAME_{\&}L$).
- Реализованы необходимые компоненты для проведения вычислений на платформе FPGA («Field Programmable Gate Array», программируемая пользователем вентильная матрица) [19]. Вычисления на этой платформе с помощью клеточных автоматов являются многообещающим направлением, в то время как $CAME_{\&}L$ оказывается единственным средством, позволяющим программировать вентильные матрицы на столь высоком уровне, так как среда предоставляет одинаковый

набор базовых абстракций для любых вычислительных платформ, на которых она используется. Данная разработка выполнена в рамках магистерской диссертации студентом Амстердамского университета М. Вуденбергом [20].

- Разрабатываются средства для анализа и моделирования опухолевых процессов, для чего в пакет вводятся инструменты для обработки медицинской информации, клеточных данных и так далее. Данная работа ведётся с университетом Амстердама.
- Произведён рефакторинг библиотеки CADLib, направленный на упрощение программного интерфейса, исключение лишних операций и расширение спектра возможностей разработчика.

Преобразованная библиотека будет включена в следующий релиз пакета.

Воспользоваться плодами и поучаствовать в развитии программного обеспечение CAME&L может каждый, посетив сайт [1].

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту № 05-07-90086 «Разработка среды и библиотеки "CAME&L" для организации параллельных и распределённых вычислений на основе клеточных автоматов».

Литература

1. Сайт «CAME&L Laboratory» – http://camellab.spb.ru.
2. Naumov L. CAME$_{\&}$L – Cellular Automata Modeling Environment & Library // Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. Springer–Verlag. 2004.
3. Наумов Л. CAME$_{\&}$L – Среда моделирования и библиотека разработчика клеточных автоматов // Труды XI Всероссийской научно-методической конференции Телематика'2004. Том 1. СПб.: СПбГУ ИТМО. 2004.
4. Наумов Л. CAME$_{\&}$L – Средство для осуществления параллельных и распределённых вычисления на основе клеточных автоматов // Технологии распределённых вычислений. 2005.
5. фон Нейман Дж. Теория самовоспроизводящихся автоматов. М.: Мир. 1971.

6. Тоффоли Т., Марголус Н. Машины клеточных автоматов. М.: Мир. 1991.
7. Wolfram S. A New Kind of Science. IL: Wolfram Media. 2002.
8. Sloot P.M.A., Chopard B., Hoekstra A. Cellular Automata. Sixth International Conference on Cellular Automata for Research and Industry, ACRI-2004. Springer-Verlag. 2004.
9. Sarkar P. A Brief History of Cellular Automata // ACM Computing Surveys, Vol. 32. № 1. 2000.
10. Наумов Л. Обзор программного обеспечения для решения задач с использованием клеточных автоматов // Телекоммуникации и информатизация образования. 2006. №2
11. Наумов Л. CTP (Commands Transfer Protocol) – Сетевой протокол для высокопроизводительных вычислений // Труды XI Всероссийской научно-методической конференции Телематика'2004. Том 1. СПб.: СПбГУ ИТМО. 2004.
12. Наумов Л. CTP (Commands Transfer Protocol) v. 1.2 – Новая версия сетевого протокола для высокопроизводительных вычислений // Труды XII Всероссийской научно-методической конференции Телематика'2005. Том 1. СПб.: СПбГУ ИТМО. 2005.
13. Наумов Л. Сравнение специализированных сетевых протоколов CTP (Commands Transfer Protocol) и IL (Internet Link) // Труды XIII Всероссийской научно-методической конференции Телематика'2006. Том 1. СПб.: СПбГУ ИТМО.
14. Naumov L. Commands Transfer Protocol (CTP) – New Networking Protocol for Parallel or Distributed Computations // http://www.codeproject.com/internet/ctp.asp. 2004.
15. Страуструп Б. Язык программирования C++. – 3-е изд. СПб: «Невский диалект». 1999.
16. Naumov L. Generalized Coordinates for Cellular Automata Grids // Computational Science – ICCS 2003. Part 2. Springer-Verlag. 2003.
17. Наумов Л. Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME$_{\&}$L. Часть 1. // Информационно-управляющие системы. 2005. №5.
18. Наумов Л. Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME$_{\&}$L. Часть 2. // Информационно-управляющие системы. 2005. №6.

19. Brown S., Rose J. Architecture of FPGAs and CPLDs: A Tutorial // IEEE Design and Test of Computers. 1996. Vol. 13. №2.
20. Woudenberg M. Using FPGAs to Speed Up Cellular Automata Computations // http://camellab.spb.ru. 2006.

Контакты
levnaumov@mail.ru

ОРГАНИЗАЦИЯ (МАКРО)ПОТОКОВЫХ ВЫЧИСЛЕНИЙ В НЕОДНОРОДНЫХ МУЛЬТИЯДЕРНЫХ ПРОЦЕССОРАХ

Недоводеев К.В.
Санкт-Петербургский государственный политехнический университет, Санкт-Петербург

Введение

Одним из направлений развития мультиядерных процессоров является размещение в рамках одного кристалла нескольких разнотипных процессорных ядер, где каждое ядро оптимизировано для выполнения своего круга задач наиболее эффективным образом. Примерами неоднородных мультиядерных процессоров являются отечественные мультиядерные процессоры обработки сигналов семейства «Мультикор» ГУП НПЦ «Элвис», процессор Cell фирмы IBM, а также ряд других.

На сегодняшний день сформировался определенный общий технический облик гетерогенных мультиядерных процессоров, который позволяет рассматривать их как единый класс высокопроизводительных вычислителей с общим кругом проблем, и искать для них общие пути решения задач. В состав гетерогенных мультиядерных процессоров входят [1, 2, 3]:

- высокопроизводительные вычислительные ядра, ориентированные на определенные виды приложений (например, на обработку сигналов – DSP-ядра);
- управляющие процессорные ядра (обычно RISC-ядра);
- ядра управления пересылками данных (например, ядра DMA-контроллеров с расширенными функциональными возможностями).

В гетерогенных мультиядерных процессорах отсутствует общая память, вместо этого, с каждым ядром ассоциирован набор локальных блоков памяти, причем вычислительные ядра ведут обработку только тех данных, которые были предварительно размещены в их локальной памяти, доступ к данным, находящимся во внешней по-отношению к вычислительному ядру памяти осуществляется посредством ядер управления пересылками данных. При этом локальная память вычислительного ядра имеет малый объем. Исходные данные должны быть предварительно загружены в локальную память вычислительного ядра, а результаты работы – выгружены. Такая организация подсистемы памяти получила название трехзвенной архитектуры вида: основная память, локальная память, регистры. Наличие многопортовой локальной памяти и ядер управления пересылками данных позволяет распараллелить вычислительный процесс в ядре с процессом обмена данными, что в целом повышает реальную производительность задачи.

Сущность (макро)потоковых вычислений

Широко известным является тот факт, что получение максимальной производительности тесно сопряжено с детальным учетом специфики архитектуры вычислительной системы, на которой планируется решать поставленные задачи. Это справедливо и для неоднородных мультиядерных процессоров.

Перечислим основные отличительные характеристики такого класса вычислителей:

- наличие у вычислительных ядер локальной памяти малого объема;
- наличие специализированных ядер организации обмена данными;
- наличие управляющих ядер.

Поскольку локальная память вычислительных ядер имеет малый объем, а решение реальных задач сопряжено с обменом большими объемами данных, автор считает целесообразным представление вычислительного процесса в виде (макро)потокового вычисления. В этом случае при решении задачи следует пройти ряд этапов, выполнение которых приведет к организации такого вычисления, перечислим эти этапы вкратце:

- переход к блочному представлению алгоритма решения задачи с выделением отдельных составляющих задачи;

- разбиение данных на блоки;
- организация блочного обмена данными;
- организация согласованной работы всех вычислительных ядер (синхронизация);
- расчет производительности отдельных составляющих задачи, а также времени их решения;
- расчет объема данных, обмен которыми необходимо организовать параллельно с работой вычислительного ядра, для каждого ядра на каждом этапе вычислительного процесса;
- расчет времени обмена данными для каждого ядра, на каждом этапе вычислительного процесса;
- учет накладных расходов, связанных с синхронизацией вычислительных ядер между собой, а также с ядрами организации обмена данными, и времени простоя вычислительных ядер по причине несбалансированности объема данных и объема вычислений отдельных составляющих задачи;
- балансировка вычислений путем варьирования числа вычислительных ядер, задействованных в процессе решения задачи, а также варьирования размерности блока данных.

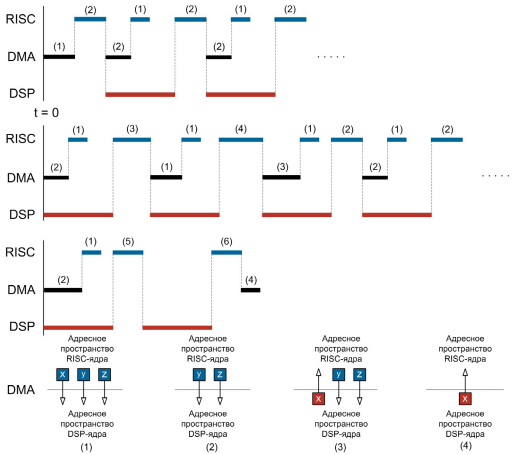


Рис. 1 Схема организации (макро)потокового вычисления

Задачи матричной алгебры дают хорошую основу для разбиения по фрагментам массивов данных, по блокам матриц. Представление

операций над матрицами в блочном виде имеет хорошо развитый математический аппарат [4].

Формирование (макро)потокового вычисления

Автор ранее описывал процесс формирования (макро)потокового вычисления для типовой задачи матричного умножения. На рисунке 1 приведена схема организации (макро)потокового вычисления для задачи матричного умножения в процессоре Мультикор-24 [5]. За более детальным рассмотрением вопросов, связанных с эффективной организацией вычислительного процесса в данной задаче, читателю следует обратиться к более ранним работам автора [5, 6]. Среди основных этапов было дано описание принципов разбиения исходных данных на блоки, организация блочного обмена данными, синхронизация и расчет производительности задачи с учетом накладных расходов. Для данной задачи доля накладных расходов на синхронизацию мала и быстро убывает с ростом размерности блока (рис. 2) [6]. Отношение времени вычислений к времени обмена данными в пределе есть линейная функция от размерности блока.

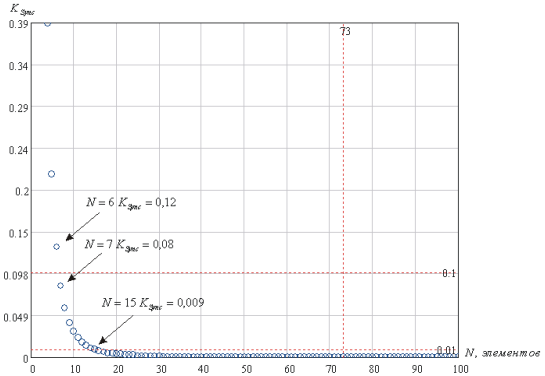


Рис. 2 Доля накладных расходов на синхронизацию

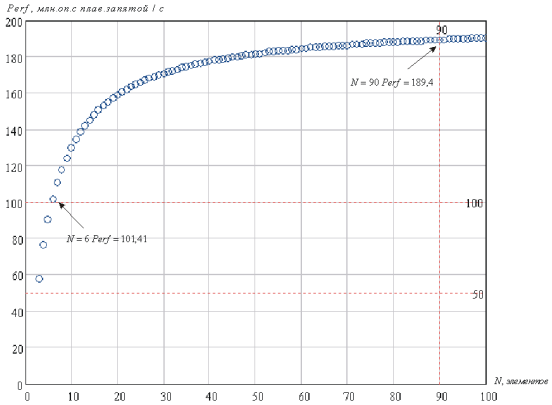


Рис. 3 Реальная производительность подпрограммы матричного умножения

Увеличивая размер блока для матриц одной размерности, мы можем добиться полного перекрытия вычислений и пересылок данных, тем самым, исключив непроизводительные простои вычислительного ядра в ожидании завершения обмена данными (рис. 1). При максимальном размере обрабатываемого за одну итерацию блока данных производительность задачи на процессоре Мультикор-24 без использования SIMD-режима составляет почти 95% пиковой производительности вычислительного ядра (DSP-ядро VLIW-архитектуры с возможностью одновременного исполнения до двух арифметических операций и до двух операций пересылок, частота ядра 100 МГц). На рисунке 3 представлен график реальной производительности подпрограммы матричного умножения в DSP-ядре Мультикор-24 как функции от размерности обрабатываемого блока матрицы [6].

Выводы

Построение процесса решения задач матричной алгебры в виде (макро)потокового вычисления наиболее полно учитывает архитектурную специфику неоднородных мультиядерных процессоров и как следствие позволяет добиться максимальной производительности. Поскольку детальный учет всех факторов влияющих на производительность задачи является достаточно трудоемким процессом основным путем применения данной технологии разработки алгоритмов можно считать создание

прикладных библиотек, при этом, прикладной программист остается «в стороне» от основных трудностей параллельного программирования и может сосредоточиться на решении конкретной задачи.

Литература

1. Солохина Т.В., Александров Ю., Глушков А.В., Беляев А., Петричкович Я.Я. Отечественные трехъядерные сигнальные микроконтроллеры с производительностью 1,5 Gflops // Электронные Компоненты, 2006, №6, с. 73-78.
2. J.A. Kahle, et. al. Introduction to the Cell multiprocessor // IBM Journal of Research & Development №4/5, 2005, pp. 589-604.
3. Moore S.K. Multimedia monster. Cell's nine processors make it a supercomputer on a chip // IEEE Spectrum, January 2006, pp.18-21.
4. Голуб Дж., ВанЛоун Ч. Матричные вычисления. М.: Мир, 1999.
5. Nedovodeev K.V. Multimedia Data Processing On Dual-Core Soc Multicore-24 // Proceedings of the IEEE Tenth International Symposium on Consumer Electronics (ISCE 2006) 28 June – 1 July 2006 St. Petersburg, Russia, pp. 142-147.
6. Недоводеев К.В., Шейнин Ю.Е. Высокопроизводительная обработка больших массивов данных в неоднородных мультиядерных процессорах // Электронные компоненты, 2006, №9, с. 116-122.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ БЛОЧНОГО ШИФРОВАНИЯ DES (DATA ENCRYPTION STANDARD)

Нестеренко М.Ю., Волков М.Н.

ГОУ ОГУ, Оренбург

Введение

В докладе представлен прототип параллельного алгоритма блочного шифрования DES. Приводится анализ вычислительной сложности параллельного и последовательного алгоритмов, на основании которого вычисляется теоретический коэффициент ускорения параллельного алгоритма, также приводятся временные характеристики выполнения параллельного и последовательного

алгоритмов на различных доступных микропроцессорных архитектурах и вычисляются практические коэффициенты ускорения параллельного алгоритма. Осуществляется анализ причин несоответствия теоретического и практического коэффициентов.

Главные производители процессоров сместили акцент с повышения тактовых частот на реализацию параллелизма в самих процессорах за счет использования многоядерной архитектуры. Идея проста: интегрировать в один процессор более одного ядра. Система, включающая процессор с двумя ядрами, по сути, не отличается от двухпроцессорного компьютера, а система с четырехядерным процессором — от четырехпроцессорного. Этот подход позволяет избежать многих технологических проблем, связанных с повышением тактовых частот, и создавать при этом более производительные процессоры. Но если приложение не будет использовать несколько ядер, его быстродействие никак не изменится.

В настоящее время приложения работают с все большими объемами данных и область информационных технологий, связанная с информационной безопасностью не является исключением. В то же время пользователи, осознавая прогресс современных вычислительных систем, порою не могут смириться с длительностью выполнения вычислительных процессов и отдают предпочтение современному, в плане использования технологий и соответственно более быстрому, приложению. Категория программных продуктов, реализующих криптографическую защиту информации, востребована и, следовательно, задача увеличения быстродействия алгоритмов, в частности DES, актуальна и сводится к их параллельной реализации на основе современных технологий.

Технология распараллеливания алгоритма DES

1. Формируется параллельная область, состоящая из M потоков (соответственно числу процессоров в системе).
2. Каждому потоку ставится в соответствие свой блок данных (64 бит), согласно порядку следования в открытом тексте.
3. Каждый поток выполняет последовательный алгоритм DES, т.е. сначала все потоки выполнят первые M блоков (по 64 бит), потом следующие M блоков, и т.д.
4. Количество 64-битовых блоков данных обрабатываемых каждым потоком будет равно:

$$L = \frac{N}{M}$$

где N – количество 64-битовых блоков в открытом тексте;

M – количество потоков созданных параллельным алгоритмом.

Анализ вычислительной сложности алгоритма и коэффициент ускорения

FIPS PUB 81 определяет четыре режима работы: ECB, CBC, OFB и CFB. Банковские стандарты ANSI определяют для шифрования ECB и CBC. Рассматривается режим ECB, так как его алгоритмическая реализация не содержит зависимостей между блоками открытого текста, тогда как алгоритмическая реализация CBC содержит зависимости, например, правая половина блока данных - R_i зависит от идущих перед ним $\overline{0, i-1}$ блоков.

Вычислительная сложность алгоритма зависит от количества и вида используемых внутри него операций. В качестве значимых операций в алгоритме выступают операции сравнения и аддитивные арифметические операции. Оценим количество таких операций в криптоалгоритме DES, согласно шагам алгоритма.

1. *Начальная перестановка* содержит 64 сравнения и 446 сложений.
2. Криптоалгоритм состоит из 16 этапов, поэтому 16 умножим на сумму числа операций в *перестановке с расширением*, *подстановке с помощью S-блоков* и *перестановке с помощью P-блоков*.
3. Перестановка с расширением содержит 286 сложений.
4. *Подстановка с помощью S-блоков* содержит 68 сложений и 8 сравнений в лучшем случае, и 16 в худшем.
5. Перестановка с помощью P-блоков содержит 191 сложение.
6. *Конечная перестановка* содержит 64 сравнения и 446 сложений.

Подготовка ключа не рассматривается, так как эта операция выполняется всего один раз, и несущественна при больших объемах данных.

В таблице 1 представлены оценки количества выполняемых алгоритмом операций в зависимости от объема входных данных (N – количество блоков по 64 бита открытого текста):

Таблица 1. Оценка количества выполняемых алгоритмом операций

Схема	Наилучший случай		Наихудший случай	
	Количество сравнений T1(N)	Количество сложений T2(N)	Количество сравнений T1(N)	Количество сложений T2(N)
Последовательный алгоритм	256*N	9612*N	384*N	9612*N
Параллельный алгоритм для M потоков (данные представлены для 1 потока)	256*N/M	9612*N/M	384*N/M	9612*N/M

Теоретический коэффициент ускорения

Рассчитаем теоретический коэффициент ускорения параллельного алгоритма по отношению к последовательному на 2 (M=2) процессорах:

- наилучший случай

$$S_M(N) = \frac{T_1(N) + T_2(N)}{T_{1par}(N) + T_{2par}(N)} = \frac{256 * N + 9612 * N}{256 * N/2 + 9612 * N/2} = 2$$

- наихудший случай

$$S_M(N) = \frac{T_1(N) + T_2(N)}{T_{1par}(N) + T_{2par}(N)} = \frac{384 * N + 9612 * N}{384 * N/2 + 9612 * N/2} = 2$$

где $S_M(N)$ - теоретический коэффициент ускорения,

$T_1(N), T_2(N)$ - количество значимых операций последовательного алгоритма,

$T_{1par}(N), T_{2par}(N)$ - количество значимых операций параллельного алгоритма.

Практический коэффициент ускорения

Коэффициент ускорения определялся на различных доступных микропроцессорных архитектурах.

Результатом исследования параллельного и последовательного алгоритмов на двуядерном процессоре Intel Pentium D, с тактовой частотой 2800 MHz являются значения коэффициентов ускорения, представленные в таблице 2 (используется программный пакет VTune Performance Analyzer):

Таблица 2: Результаты исследования параллельного и последовательного алгоритмов на двуядерном процессоре Intel PentiumD

Объем данных, Kb	Последовательная программа, мс	Параллельная программа, мс	Коэффициент ускорения	Эффективность
3152	4043	2122	1.905	0.953
6324	8062	4223	1.909	0.955
9289	11832	6258	1.891	0.946
12475	15773	8267	1.908	0.954

Результатом исследования параллельного и последовательного алгоритмов на одноядерном процессоре Intel Pentium 4 с технологией HyperThreading и с тактовой частотой 3000 MHz являются значения коэффициентов ускорения, представленные в таблице 3 (используется программный пакет VTune Performance Analyzer):

Таблица 3: Результаты исследования параллельного и последовательного алгоритмов на одноядерном процессоре Intel Pentium 4 с технологией HyperThreading

Объем данных, Kb	Последовательная программа, мс	Параллельная программа, мс	Коэффициент ускорения	Эффективность
3152	4380	2747	1.594	0.797
6324	7563	5488	1.378	0.689
9289	11074	8033	1.379	0.690
12475	14784	10800	1.369	0.685

Причины возникновения расхождений между теоретическими и практическими показателями

Как можно увидеть из результатов выполнения алгоритма на двуядерной машине, практический коэффициент ниже, чем

теоретический, но достаточно близкий к нему. Это происходит по следующим причинам:

1. При чтении очередного блока данных направляемого на обработку происходит формирование параллельной области. Эта операция требует времени процессора, и суммарное время, затрачиваемое на нее, увеличивается с увеличением размера обрабатываемого файла, но доля от всего времени выполнения приложения остается постоянной.
2. Все инструкции и данные, для каждого потока, находятся в оперативной памяти компьютера. Процессор сначала смотрит в кэш 1-го уровня, если соответствующей инструкции или данных там не оказалось он смотрит в кэш 2-го уровня, если там тоже нет, то в кэш 3-го уровня (если он есть) или в основную память. Эти операции называют промахами кэша, очень часто они происходят вследствие наличия в коде условных или безусловных переходов и циклов, и приводят к существенному уменьшению быстродействия приложения.

Результаты анализа промахов кэша, проведенного с помощью VTune Performance Analyzer, для *последовательного и параллельного* алгоритмов на Intel Pentium 4 с технологией HyperThreading представлены в таблице 4:

Таблица 4: Результаты анализа промахов кэша

	Последовательный алгоритм, %	Параллельный алгоритм, %
Промахи кэша 1-го уровня	40,33	46,72
Промахи кэша 2-го уровня	30,64	46,96

Также на производительность влияет разделение загрузки данных в кэш-строку, т.е. одна часть данных пишется в одну кэш-строку, другая часть в другую. Скорость приложения уменьшается, потому что процессор читает данные отдельно из каждой строки кэш, а потом работает как с одним целым. Для уменьшения влияния разделения загрузки (split loads) необходимо производить меньше манипуляций с указателями на большие объемы данных. В данном алгоритме пришлось бы отказаться от массивов, что практически невозможно.

Результаты анализа событий разделения загрузки данных в кэш-строку, проведенного с помощью VTune Performance Analyzer, для

последовательного и параллельного алгоритмов на Intel Pentium 4 с технологией HyperThreading представлены в таблице 5:

Таблица 5: Результаты анализа событий разделения загрузки данных в кэш-строку

	Последовательный алгоритм, %	Параллельный алгоритм, %
Разделение загрузки	39,71	58,58

Заключение

Прогресс вычислительных систем и растущие требования пользователей стимулируют программистов на разработку параллельных алгоритмов. Но «добавить» параллельные участки в код недостаточно, необходимо формировать параллельные области в местах, обеспечивающих максимальное ускорение. Результаты анализа разработанного прототипа параллельного криптоалгоритма блочного шифрования DES показывают, что ускорение, получаемое этим алгоритмом по отношению к последовательному на двуядерной машине, около 1,9. Такое значение реального ускорения позволяет сказать, что модель распараллеливания алгоритма очень эффективна. Но остается возможность увеличить быстродействие, сократив количество промахов кэша, и количество событий, сопровождаемых разделением загрузки данных в кэш-строку. Именно эта задача требует решения в ближайшее время.

Литература

1. Фергюсон Н., Шнайер Б. Практическая криптография. – М.: Вильямс, 2005. – 424 с.
2. Гергель В.П., Стронгин Р.Г., Основы параллельных вычислительных систем. – Н. Новгород: ННГУ, 2000. – 176 с.
3. Касперски К., Техника оптимизации программ. – С.-Пб.: БХВ-Петербург, 2003. – 560 с.
4. www.parallel.ru.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ GRID-ТЕХНОЛОГИЙ В МАТЕМАТИЧЕСКОМ МОДЕЛИРОВАНИИ РЕГИОНАЛЬНОЙ ЭКОНОМИКИ

Оленёв Н.Н., Стариков А.С., Шатров А.В.
Вычислительный центр им. А.А. Дородницына РАН, Москва
Вятский государственный университет, Киров

Введение

В [1] предложена концепция исследования инновационного потенциала Кировской области на основе имитационной модели ее экономики. Расчеты на модели дают количественную оценку динамики макропоказателей экономики региона, включая оценку показателей инновационного потенциала. Имитационная модель экономики Кировской области является трехсекторной нормативной балансовой динамической математической моделью региональной экономики [2]. Динамика материальных и финансовых балансов в модели задана изменением запасов продуктов, факторов производства и денег. Выделены следующие экономические агенты: Правительство региона G; Производственные сектора X, Y, Z; Банковская система B; Домашние хозяйства H; Внешние агенты O; Торговый посредник T. Производители представлены следующими тремя секторами: (X) лесопромышленный комплекс, включая лесное хозяйство; (Y) комплекс новых отраслей в биотехнологии и химии, включая науку, образование, инновационные отрасли; (Z) комплекс, объединяющий оставшиеся отрасли.

Для оценки огромного числа внешних параметров нормативной модели экономики предложена новая методика их идентификации, основанная на использовании высокоскоростных параллельных вычислений в GRID-инфраструктуре с кластерными узлами [3]. Для идентификации параметров сравнивают полученные при расчетах на модели временные ряды макропоказателей экономики региона с соответствующими статистическими временными рядами. В качестве критериев близости временных рядов использован индекс несовпадения Тейла [4] и специально построенный на основе вейвлет коэффициентов критерий похожести нелинейных временных рядов [5]. При построении и идентификации имитационной модели решаются задачи анализа и структуризации исходных данных в соответствии со структурой модели.

Описание нормативной модели региональной экономики

Производители *X*, *Y*, *Z* используют в производстве труд, капитал и промежуточную продукцию смежных секторов, поставляют продукцию на рынки. Домашние хозяйства *L* предлагают труд и потребляют конечную продукцию. Торговый посредник *T* занят перераспределением материальных и финансовых потоков. Банковская система *B* выдает кредиты производителям с целью извлечения банковской прибыли. Правительство региона *G* собирает налоги с производителей и домашних хозяйств и управляет расходами бюджета. На каждом рынке каждой продукции формируются свои цены и их изменение обратно пропорционально изменению запасов соответствующих продуктов.

Считаем, что произведенный продукт производители делят на легальный и теневой, который не облагается налогами. В результате у производителя оказывается два вида денег – «белые» и «черные». «Черные» деньги могут отмываться, а запас неотмытых денег подвергается штрафным санкциям. Потребитель делит свой доход по заданным нормам потребления легальных и теневых продуктов всех секторов. Производственные сектора $m = X, Y, Z$ платят налог на прибыль n_1, налог на добавленную стоимость n_2, акцизы на валовой выпуск n_3^m, единый социальный налог на фонд заработной платы n_4, таможенные платежи на экспорт n_5. Домашние хозяйства *L* оплачивают таможенные платежи с импорта n_6 и подоходный налог с зарплаты n_7.

Показатели и параметры модели снабжены верхними и нижними индексами, причем верхние индексы используются для агентов, а нижние для благ. Распределение запаса каждого блага производится по нормативу: a_i^{nm} - доля запаса блага *i* агента *n*, идущая к агенту *m*. Распределение денежных средств производится также по некоторому нормативу: b_i^{mn} - доля запаса денег агента *m*, идущая агенту *n* за продукт *i*. Коэффициенты фондоемкости также задаются нормативами: c_i^m – норма затрат продукта *i* на создание единицы фондообразующего продукта агента *m*. Параметры производственных функций секторов заданы константами. Так, выпуск $Y_X(t)$ продукта *X* описывается степенной производственной функцией от используемых факторов производства (запасов *Q*): труда *L*, капитала *K* и промежуточных продуктов из секторов *Y* и *Z*.

$$Y_X = \left(a_L^X Q_L^X\right)^{\delta_L^X} \cdot \left(a_K^X Q_K^X\right)^{\delta_K^X} \cdot (a_Y^X Q_Y^X)^{\delta_Y^X} \cdot (a_Z^X Q_Z^X)^{\delta_Z^X}, \qquad (1)$$

где $\delta_L^X + \delta_K^X + \delta_Y^X + \delta_Z^X = 1$; a_L^X, a_K^X, a_Y^X, a_Z^X, δ_L^X, δ_K^X, δ_Y^X, $\delta_Z^X \in (0, 1)$.

Производство открытого X и теневого V продуктов агент X осуществляет на общих фондах и общих трудовых ресурсах. Доля тени q_X в общем выпуске определяет прирост запасов открытого $Q_X^X(t)$ и теневого $Q_V^X(t)$ продуктов. Изменение запасов открытых («белых») $W^X(t)$ и скрытых («черных») $B^X(t)$ денежных средств у агента X также описывается балансовыми соотношениями. Например, запас «черных» денег у агента X

$$\frac{dB^X}{dt} = \left(p_V^L a_V^{XL} + p_V^Y a_V^{XY} + p_V^Z a_V^{XZ}\right) Q_V^X - \left(b_B^{XL} + b_B^X + b_B^{XG}\right) B^X \cdot \qquad (2)$$

Здесь $w(t)$ – рублевый курс доллара, $p_i^m(t)$ – индексы цен на продукт i на рынке для агента m, $T^{GX}(t)$ - трансферты из бюджета, $T^{XG}(t)$ – налоговые отчисления в консолидированный бюджет, $C^{BX}(t)$ – объем кредитов, $b_B^X B^X(t)$ - поступления из теневого оборота.

Изменение запаса $Q_X^L(t)$ конечного продукта X лесопромышленного комплекса, предназначенного агенту L (домашним хозяйствам), у посредника T определяет изменение индекса потребительских цен p_X^L на продукцию X.

$$\frac{dQ_X^L}{dt} = a_X^{XL} Q_X^X - \frac{b_X^{LX} W^L}{p_X^L},$$
$$\frac{dp_X^L}{dt} = \alpha_X^L \left(\frac{b_X^{LX} W^L}{p_X^L} - a_X^{XL} Q_X^X \right) \cdot \qquad (3)$$

Аналогично определяются изменения запасов других продуктов и цен на их рынках.

Считаем, что рост открытой и теневой ставок заработной платы в секторах может происходить как при нехватке кадров, так и при росте потребительских цен на продукцию сектора. Например, открытая ставка зарплаты $s_L^X(t)$ в секторе X

$$\frac{ds_L^X}{dt} = \left[\alpha_L^X \left(\frac{b_L^{XL} W^X}{s_L^X} - a_L^{LX} Q_L^{LX} \right) + \frac{\beta_L^X s_L^X}{p_X^L} \left(\frac{b_X^{LX} W^L}{p_X^L} - a_X^{XL} Q_X^X \right) \right]_+, \quad \beta_L^X = \delta \alpha_X^L . \qquad (4)$$

Здесь используется обозначение: $A_+ = A$, если $A>0$ и $A_+ = 0$, если $A \le 0$. Считаем, что доля прироста цен, отражающаяся на росте заработной платы, $\delta \in (0,1)$.

Общее число уравнений в модели около 100, число параметров более 80.

Проблема идентификации модели

Параметры модели, которые не удается определить напрямую из статистики, определяются косвенным образом, сравнивая расчетные макропоказатели модели с их статистическими аналогами. При сравнении используются временные ряды тех макропоказателей, для которых имеются соответстсвующие статистические временные ряды. Временные ряды считаются похожими, если они близки как функции времени (другими словами, между значениями временных рядов существует сильная, возможно нелинейная, связь). Мера близости экономических временных рядов определяется индексом несовпадения Тейла $E(X, Y) \in [0, 1]$ [4]. Мера похожести временных рядов рассчитывается на основе спектральных характеристик временного ряда. Используется коэффициент похожести, построенный на основе вейвлет коэффициентов [5].

Пусть $\widetilde{W}_{j,k}^X$ и $\widetilde{W}_{j,k}^Y$ - вектора спектральных характеристик временных рядов одинаковой длины X и Y соответственно. В качестве меры близости между временными рядами X и Y используется косинус угла между векторами характеристик, а именно

$$D(X,Y) = \cos\alpha = \left| \sum_{j,k} \widetilde{W}_{j,k}^X \cdot \widetilde{W}_{j,k}^Y \right| . \qquad (5)$$

Чем ближе коэффициент похожести к единице, тем более схоже поведение этих рядов.

Индекс Тейла *E(X. Y)* измеряет несовпадение временных рядов X_t и Y_t и чем ближе он к нулю, тем ближе сравниваемые ряды. Для удобства рассмотрим коэффициент близости *U(X, Y) = 1 - E(X, Y)*. Чем ближе он к единице, тем более близки ряды.

$$U(X,Y) = 1 - \sqrt{\frac{\sum_{t=t_0}^{T}\left(X_t - Y_t\right)^2}{\sum_{t=t_0}^{T} X_t^2 + \sum_{t=t_0}^{T} Y_t^2}}\,. \qquad (6)$$

Для однозначности выбора оптимального варианта рассмотрим свертку коэффициентов близости и похожести, например, их среднегеометрическую величину.

$$\sqrt[2m]{\prod_{j=1}^{m} D_j\left(\vec{a}\right) U_j\left(\vec{a}\right)} \rightarrow \max_{\vec{a}\in A},$$
$$A = \left\{\vec{a} \in R^N : a_i^- \le a_i \le a_i^+, 1 \le i \le N\right\}. \qquad (7)$$

Здесь m – число макропоказателей; j - номер макропоказателя, $j = 1,\ldots,m$.

Полный перебор по всем неизвестным параметрам, заданных сеткой на интервале, даже на самых мощных суперкомпьютерах невозможен, поскольку сетка из 10 точек на каждый интервал для 80 параметров дает 10^{80} вариантов. Значит надо использовать направленный перебор [6], сокращать число независимых параметров за счет дополнительных предположений, использовать декомпозицию модели на относительно независимые блоки, идентификацию параметров в которых можно производить независимо.

Для сокращения числа независимых параметров параметры производственных функций были определены независимо на основе статистических данных. С помощью системы моделирования экономики ЭКОМОД [7] за счет дополнительного предположения о стационарности начальных данных получены соотношения,

сократившие число независимых параметров до 18. Это число все еще велико для полного перебора на современных компьютерах, значит, для решения задачи идентификации нужно разрабатывать новые методы направленного перебора на многопроцессорных системах и в неоднородных вычислительных средах.

Получен работоспособный вариант параметров модели, в котором параметры модели идентифицированы по порядку величины. Для этого объединялись параметры с одинаковым экономическим смыслом. Расчеты на кластерах МСЦ РАН, ВЦ РАН, ИВМ РАН и в GRID-среде производились с числом независимых параметров девять.

Параллельные вычисления с использованием GRID-технологий

В задаче идентификации параметров модели нехватки имеющихся вычислительных мощностей можно избежать за счет агрегации масштабируемых распределенных кластерных систем, то есть за счет развертывания на их базе GRID-инфраструктуры.

Исходные данные и участники исследовательского коллектива распределены географически (статистические данные региона подготавливают к использованию в расчетах региональные представители коллектива), для географически распределенных баз экономических данных регионов часто ограничен доступ извне к новым статистическим данным. GRID-инфраструктура обеспечивает поддержку стандартных протоколов и форматов обмена данными между отдельными блоками декомпозированной модели и обеспечивает эффективное распределение нагрузки в вычислительной системе.

Для погружения проекта в GRID-среду требуется разместить исполняемые файлы приложения на узлах GRID-сегмента системы. Для размещения информации о проводимом численном эксперименте по идентификации модели экономики региона на портале системы потребуется разработка Web-сервиса-оболочки. Работа через портал облегчит работу с приложением, поскольку позволит следить за состоянием прохождения конкретного запуска и хранить информацию о результатах экспериментов.

Цепочка выполнения приложения: (1) выбор задания, (2) ввод необходимых параметров по форме, сгенерированной по WSDL-описанию сервиса-оболочки; (3) запуск приложения на счет в GRID-среде; (4) отслеживание состояние выполнения задания на

соответствующей странице портала; (5) получение пользователем оповещения о выполнении задания (или аварийном его завершении) и ссылки на результаты; (6) результаты расчета временных рядов макропоказателей визуализируются в Excel-файле.

В настоящее время на основе кластеров ИВМ РАН и ВЦ РАН создана распределенная среда для проведения расчетов на программном комплексе математических моделей экономических систем с использованием GRID-технологий, установлен Globus Toolkit 4.0. Проведено выполнение тестовых параллельных расчетов в распределенной среде с помощью GRID-технологий по идентификации параметров балансовой нормативной динамической модели экономики Кировской области.

В Вятском государственном университете создан портал кафедры математического моделирования в экономике, который в дальнейшем будет одним из узлов предполагаемой GRID-инфраструктуры решения задач моделирования региональной экономики.

Пданируется использовать GRID-инфраструктуру для проведения серии крупномасштабных промышленных расчетов с последующей обработкой результатов распределенных вычислений, формированием базы данных для оценки результатов и принятия решения. Вначале будет обеспечен обмен информацией между географически разделенными компонентами инфраструктуры. В дальнейшем будут решаться задачи идентификации параметров взаимосвязанных блоков модели региональной экономики и обмен информацией во время счета. На основе этих задач будет проверена функциональная полнота и эффективность работы Web-сервисов в рамках построенной GRID-инфраструктуры при решении вычислительно сложных прикладных задач математического моделирования региональной экономики.

Заключение

Численные эксперименты с моделью проводились, чтобы найти работоспособный вариант, качественно верно отражающий процессы, происходящие в экономике Кировской области. Численные эксперименты [2] показали работоспособность полной модели и отдельных ее частей. Это значит, что модель можно использовать в дальнейшей работе. Внешние параметры этого варианта можно взять за основу для более точной идентификации параметров модели в будущем, а сам вариант использовать как базовый при проведении качественных сценарных расчетов

Построенная региональная модель экономики является инновационным продуктом, а полученный опыт ее идентификации тем «ноу-хау», который можно использовать при адаптации модели к условиям других регионов.

Поведение макропоказателей экономики Кировской области существенно зависит от политики, проводимой региональным Правительством. При административном зажимании экономической активности, экономическая деятельность уходит в тень, увеличивается инфляция. При осуществлении задуманной политики повышения производительности факторов производства за счет инновационной деятельности выпуски продукции и доходы всех агентов растут, однако, наличие теневой составляющей в производстве сохраняет высокие темпы инфляции.

Работа выполнена при финансовой поддержке программ фундаментальных исследований (ОМН РАН №3, РАН №14, Президиума РАН №15), гранта Президента РФ по государственной поддержке ведущих научных школ (код проекта НШ-5379.2006.1), государственного контракта 02.449.11.7026, грантов Российского фонда фундаментальных исследований (коды проектов 04-07-90346, 04-01-00606), гранта Российского гуманитарного научного фонда (код проекта 06-02-91821 а/G).

Литература

1. Оленев Н.Н., Шатров А.В. Концепция использования имитационной модели экономики региона для исследования его инновационного потенциала. // Математическое моделирование развивающейся экономики: Сб. тр. летней школы по экономико-математическому моделированию ЭКОМОД-2006. – Киров: Изд-во ВятГУ, 2006. – 152 с. С. 10-24.
2. Оленев Н.Н Параллельные вычисления для оценки параметров динамической многосекторной балансовой модели региональной экономики // *Научный сервис в сети ИНТЕРНЕТ: технологии параллельного программирования*: Тр. Всерос. научн. конф. - М.: Изд-во МГУ, 2006. - 303 с. С. 36-37.
3. Оленев Н.Н., Петров А.А. Создание ГРИД-инфраструктуры для решения вычислительно-сложных задач математического

моделирования экономики // Научный сервис в сети Интернет: технологии распределенных вычислений: Тр. Всерос. научн. конф. - М.: Изд-во МГУ, 2005. - 318 с. С. 20-21.
4. Тейл Г. Экономические прогнозы и принятие решений. М., 1971. 488 с.
5. Бурнаев Е.В., Оленев Н.Н. Меры близости на основе вейвлет коэффициентов для сравнения статистических и расчетных временных рядов // Межвуз. сб. научн. и научно-метод. трудов за 2005 год Вып.10. Киров: Изд-во ВятГУ, 2006. С. 41-51.
6. Гергель В.П., Стронгин Р.Г. Параллельные методы вычисления для поиска глобально-оптимальных решений // Высокопроизвод. параллельные вычисления на кластерных системах. Мат. 4-го межд. науч.-практ. сем. Самара, 2004. С. 54-59.
7. Поспелов И.Г., Поспелова И.И., Хохлов М.А., Шипулина Г.Е. Новые принципы и методы разработки макромоделей экономики и модель современной экономики России. М.: ВЦ РАН. 2006. 240 с.

ПОДХОДЫ К РЕАЛИЗАЦИИ МЕХАНИЗМОВ ДИНАМИЧЕСКОЙ БАЛАНСИРОВКИ ЗАГРУЗКИ В СИСТЕМЕ РАСПРЕДЕЛЁННОГО ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

Осмехин К.А.

Пермский Государственный Университет, Пермь

Введение

Как известно, проблема автоматизации процесса загрузки вычислительных узлов существует в любой многопроцессорной системе. В ходе работы над системой распределённого дискретно-событийного имитационного моделирования Triad.NET, естественным образом появилась потребность решать указанную проблему.

В данной статье под балансировкой загрузки мы будем понимать динамическую балансировку, то есть такую, которая проводится непосредственно в процессе работы системы. Также условимся, что такая балансировка предназначена для работы в кластерной среде.

Задаче балансировки релевантны следующие характеристики вычислительной системы:

- структура межузлового взаимодействия.
- характеристики пропускной способности линий связи.
- характеристики производительности вычислительных узлов.
- структура исполняющейся имитационной модели
- схема взаимодействия вычислительных элементов модели.

Задачу балансировки можно сформулировать как проблему отображения вычислительной активности на вычислительные узлы, таким образом, чтобы общее время выполнения всей модели было минимальным.

Компоненты системы

Существует множество механизмов динамической балансировки. Все они дают какие-либо положительные результаты при определённых требованиях к модели или к вычислительной среде. Более подробно с известными механизмами балансировки можно познакомиться в работах [1, 2, 3, 4, 5, 6].

Так как система Triad.NET является универсальной средой моделирования[7], то было принято решение не выбирать какой-то один алгоритм, а предоставить разработчику возможность выбора.

Если разделить характеристики вычислительной системы на две группы – зависящие от аппаратуры и определяемые моделью, то естественно, что более динамичной является вторая группа. А про эти характеристики больше других знает экспериментатор, создающий модель.

Конечно, человек решающий задачи управления персоналом может не очень хорошо разбираться в программировании. Поэтому в систему встроен компонент, позволяющий эксперту легко общаться с ним, записывая правила балансировки в понятных ему терминах. Этот компонент представляет собой экспертную систему, основанную на правилах. Правила пишутся на специальном языке, который впоследствии может быть переведён на язык Triad. Правила могут быть тривиальными, или сравнимыми по сложности с функциями. В систему изначально заложено несколько известных механизмов балансировки. Пользователь может строить на их основе собственные алгоритмы, каким либо образом улучшая и специализируя стандартные. Множество алгоритмов можно пополнять. Новые алгоритмы балансировки созданные экспертами также могут пополнять этот перечень. Таким образом, в процессе эксплуатации

система накапливает знания. Для описания структуры вычислительной системы используется аналог, применяемый в слое структур языка Triad [7].

Никто не будет спорить, что решение такой сложной задачи, невозможно просто осуществить без удобных средств отслеживания состояния системы. Для этих целей в нашей системе существует компонент визуализации. Механизмы его работы идентичны механизмам работы общего визуализатора модели, о которых можно прочитать более подробно в [8].

В систему встроены удобные средства визуального отображения информации о текущем состоянии вычислительной системы. Можно отображать интересующие сведения на графиках различных типов. Существует возможность в процессе имитационного прогона начать отслеживать какой-либо параметр или отказаться от сбора информации о нём. В конце работы модели эксперт имеет доступ к итоговому отчёту, который содержит историю изменений интересующих пользователя характеристик и агрегированные данные о прошедшей сессии. Такой набор информации позволяет разработчику проводить подробный анализ работы алгоритма балансировки, выявлять узкие места и, в конечном итоге, добиваться лучших результатов всего имитационного моделирования. В основе системы сбора информации лежит механизм информационных процедур, о котором более подробно рассказано в [7].

Балансировка загрузки, предполагает перемещение вычислительных объектов между узлами во время работы всей системы. Поэтому выработка таких механизмов, безусловно, необходима. В системе Triad.NET модель представляет собой множество активных объектов. Каждый такой объект является неделимым элементом вычислительной активности. Под неделимостью здесь понимается отсутствие возможности параллельного выполнения сразу нескольких действий одного объекта. Таким образом, мы осуществляем балансировку загрузки, перемещая активные объекты между вычислительными узлами.

Для перемещения объектов в системе Triad.NET используются механизмы сериализации платформы .NET.

Итак, в подсистеме динамической балансировки загрузки системы Triad.NET можно выделить следующие модули:

- язык описания алгоритмов балансировки
- компилятор алгоритмов балансировки в язык Triad

- множество алгоритмов балансировки
- механизмы сбора информации о вычислительной системе
- модель визуализации
- система исполнения алгоритма балансировки

Следует отметить следующее – подобный подход к гибкой настройке алгоритмов балансировки позволяет добиваться наилучших результатов в каждом конкретном случае, а благодаря стандартным реализованным механизмам даёт возможность на первых этапах построения прототипа модели, оставить в стороне вопросы балансировки, то есть создавать систему можно поэтапно.

Литература

1. Jerrell Watts and Stephen Taylor. A Practical Approach to Dynamic Load Balancing. Scalable Concurrent Programming Laboratory, Syracuse University, 1997
2. Apostolos Gerasoulis and Tao Yang. Sheduling Program Task Graph on MIMD Architectures. Rutgers Univesity, New Brunswick, New Jersey 08903, USA
3. Vipin Kumar and Ananth Y.Grama. Scalable Load Balancing Techniques for Parallel Computers. University of Minnesota, Minneapolis, MN 55455
4. Benedict M. Rafanello and Theodore Johnson. A Simple Approach to Distributed Pools. University of Florida, Department of Computer and information Sciences, 1994
5. Leonid Oliker and Rupak Biswas. PLUM: Parallel Load Balancing for Adaptive Unstructured Meshes. Research Institute of Advanced Computer Science, Nasa Ames Research Center, Moffet Field, California, 1998
6. Yossi Azar. On-line Load Balancing. Dept. of Computer Science, Tel-Aviv Univesity, 1992
7. А.И. Миков, Автоматизация синтеза микропроцессорных управляющих систем, Издательство Иркутского университета, 1987
8. Миков А.И., Замятина Е.Б., Осмехин К.А. Метод динамической балансировки процессов имитационного моделирования. Сборник трудов I-ой международной конференции "Системный анализ и информационные технологии (САИТ-2005)", Пере-славль-Залесский, 2005.

ПОДХОД К СОВЕРШЕНСТВОВАНИЮ IT-ПОДГОТОВКИ СТУДЕНТОВ СПЕЦИАЛЬНОСТИ 010503 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ В ОБЛАСТИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

Петухова Т.П., Нестеренко М.Ю.
Оренбургский государственный университет (ОГУ)

Данный этап развития России характеризуется повышением значимости информационных ресурсов. Знания, представленные в виде информационных ресурсов становятся главным достоянием и важнейшим фактором экономического развития. Процессы информатизации человеческой деятельности, как в производственной, так и непроизводственной сферах столь масштабны и глубоки, что ведут к качественным изменениям самого общества. Основы этих процессов – информационные технологии.

Область ИТ стала обширнейшим полем производственной деятельности с устойчивой динамикой роста, возрастающим спросом на высокопрофессиональное кадровое обеспечение.

За прошедшее время в области ИТ произошли кардинальные изменения: лавинообразное расширение Интернет, развитие технологий мобильной связи и их интеграция с глобальной сетью, значительный прогресс в технологии разработки программного обеспечения, интенсивное развитие аппаратного обеспечения (многоядерные процессоры и многопроцессорные системы).

Система ИТ-образования должна быть самоорганизующейся под современные требования ИТ-индустрии к ИТ-кадрам. Очевидно, учет быстро изменяющихся требований ИТ-индустрии не возможен без ее участия в подготовке специалистов. В настоящее время имеет место ситуация, когда большинство ИТ-компаний являются потребителями ИТ-кадров, никак не участвуя в их подготовке. Однако существуют положительные примеры участия ИТ-компаний в подготовке специалистов (Intel, Лаборатория Касперского и др.). На базе факультетов, ведущих подготовку ИТ-специалистов, совместно с ИТ-компаниями создаются научно-исследовательские лаборатории, ведутся семинары, привлекая к работе над реальными проектами ученых и студентов и обучая необходимым на производстве навыкам

студентов. Такое сотрудничество дает массу положительных моментов – повышается заинтересованность в процессе обучения у студентов, по окончании университета выпускнику не требуется дополнительное обучение узкой специфике работы предприятия, уменьшается утечка высококвалифицированных ИТ-кадров из образования и др. Кроме того, в рекомендациях CC2001 констатируется, что ИТ сформировалась как самостоятельная образовательная дисциплина, которая имеет фундаментальное значение для подготовки кадров по всем образовательным направлениям, выполняя роль базовой дисциплины такой же как, например, математика для естественнонаучных направлений обучения.

Другим важным принципом, сформулированным в CC2001 является концепция комплексного преподавания ИТ-дисциплин с целью подготовки выпускников, готовых к выполнению исследовательской и профессиональной деятельности, а не выпускников университета, прослушавших много курсов. В связи с этим в университете должна быть организована непрерывная подготовка по важнейшим направления, в частности в области параллельного и архитектурно-зависимого программирования.

Интенсивное развитие и ближайшая перспектива широкого распространения многоядерных архитектур и многопроцессорных систем с общей памятью обусловили на рынке труда потребность в специалистах, владеющих технологиями параллельного и архитектурно-зависимого программирования. Однако, проведенный анализ стандартов специальности 010503 Математическое обеспечение и администрирование информационных систем (квалификация: математик-программист) и сопряженных с нею специальностей показал, что содержанию обучения студентов параллельному программированию не уделяется достаточного внимания. Следует также отметить, что преподавание параллельного и архитектурно-зависимого программирования на сегодняшний день в малой степени ориентировано на решение задач в системах с общей памятью и недостаточно осваиваются инструменты разработки, оптимизации и отладки параллельных программ.

В связи с этим в университете должна быть организована непрерывная подготовка по важнейшим направлениям, в частности в области параллельного программирования. Однако, направления развития ИТ-предприятий могут сильно меняться и в настоящее время их множество. Поэтому ИТ-образование должно состоять из

некоторого ядра и вариативной части. В вариативной части возможно изменение тем, задач, методических материалов, технологий и т.п.

В ОГУ предпринята попытка обновления содержания IT-образования студентов специальности 010503 Математическое обеспечение и администрирование информационных систем математического факультета за счет организации непрерывной подготовки в области параллельного программирования и его приложений в рамках существующего стандарта специальности. Данный процесс совершенствования IT-подготовки студентов разбит на два этапа:

- пропедевтический этап - приобретение студентами базовых знаний, умений и навыков в области параллельного программирования за счет обновления содержания дисциплин, читаемых на младших курсах (1 – 5 семестры);
- профессионально-ориентированный этап - использование обучающимися технологий параллельного программирования в прикладных задачах, т.е. обновление содержания дисциплин, изучаемых на старших курсах (6 – 8 семестры), и введение новых курсов по выбору студента (9 семестр).

Список дисциплин, содержание которых подлежит обновлению, приведен в следующей таблице:

№	Дисциплина	Семестр	Вид обновления	Самостоятельная работа студентов
1	Программирование	1,2	Реализация тем «Введение в распределенные вычисления», «Программирование параллельных алгоритмов» с использованием средств автоматизированного распараллеливания	Курсовая работа, связанная с параллельным программированием (по выбору студента)
2	Операционные системы и оболочки	2,3	Добавление темы «Параллелизм в операционных системах, понятия потоков и нитей», расширение раздела «Операционная система Unix»	Индивидуальные задания, связанные с операционной системой UNIX (по выбору студента)

3	Структуры и алгоритмы компьютерной обработки данных	3	Введение тем «Параллельная реализация алгоритмов»; «Выявление и исключение зависимостей в алгоритме»; «Параллельные алгоритмы сортировки»	Курсовая работа, связанная с созданием параллельных алгоритмов (по выбору студента)
4	Архитектура вычислительных систем и компьютерных сетей	3,4	Включение тем: «Архитектура многоядерных процессоров», «Проблемы масштабируемости», «Основы архитектурно-зависимого программирования», «Архитектура протоколов Grid»	Введение тем индивидуальных заданий, связанных с параллельными архитектурами
5	Параллельное программирование	5	Изменение содержания тем: «Основные принципы и парадигмы параллельного программирования»; «Параллельное программирование в системах с общей памятью», «Классические задачи и алгоритмы параллельного программирования», «Языки и библиотеки параллельного и высокопроизводительного программирования» в соответствии с современным состоянием IT-области	
6	Численные методы	6	Использование параллельного программирования при реализации алгоритмов вычислительной математики	Индивидуальные задания, связанные с созданием параллельных алгоритмов реализации численных методов
7	NP-полные задачи	6	Введение темы «Методы организации параллельного перебора»	Расширение тем домашних заданий студентов

8	Системы искусственного интеллекта	7	Добавление раздела «Параллелизм в задачах искусственного интеллекта»	Добавление тем индивидуальных проектов, связанных с разработкой программ, оптимизированных под используемую архитектуру
9	Системы реального времени	8	Введение раздела «Параллелизм при разработке систем реального времени»	
10	Компьютерное моделирование	8	Включение темы «Высокопроизводительные вычисления и библиотеки в компьютерном моделировании»	Курсовая работа, связанная с параллельным программированием (по выбору студента)
11	Параллельное программирование и криптография	9	Курс по выбору студента	Курсовая работа по дисциплине

Необходимость введения специального курса «Параллельное программирование и криптография» продиктована интенсивным развитием информационных технологий и объединением компьютерной техники в информационно-вычислительные сети с потенциальной возможностью доступа ко всем ресурсам для каждого пользователя. Это привело к необходимости использования криптографических средств защиты всеми пользователями, что замедляет и усложняет их работу. С другой стороны, нельзя снижать вычислительную сложность (т.е. длину ключа) криптографического алгоритма, т.к. это приведет к снижению стойкости защиты ко взлому. Отсюда следует необходимость увеличения производительности криптографических программных средств. Большой прорыв в этом направлении может быть достигнут за счет распараллеливания криптографических алгоритмов. Необходимость распараллеливания усиливает тот факт, что в ближайшем будущем фактически все персональные компьютеры будут иметь многоядерные процессоры.

Таким образом, необходимость использования *параллельных вычислений в криптографии* обусловлена следующими факторами:

- защита информации должна быть «невидима» для пользователя, т.е. осуществляться достаточно быстро;
- стойкость ко взлому криптографических методов защиты информации во многом определяется их вычислительной сложностью (или длиной ключа);
- значительное повышение производительности криптографических программных средств может быть достигнуто за счет распараллеливания криптографических алгоритмов, которые в большинстве случаев допускают эффективное распараллеливание в системах с общей памятью;
- развитие персональных ЭВМ с многопроцессорными и многоядерными архитектурами делает возможным применение параллельно выполняемых криптографических программных средств.

Введение специального курса позволит углубить знания студентов о методах параллельного программирования и получить навыки параллельного программирования в системах с общей памятью при решении сложных прикладных задач на примере криптографических алгоритмов.

Опорной точкой методического обеспечения процесса совершенствования IT-подготовки математиков-программистов явился образовательный комплекс «Технологии параллельного программирования в системах с общей памятью», разработанный Нестеренко М.Ю. (математический факультет ОГУ), Калининой А.П. (механико-математический факультет НГУ), Владовой А.Ю. (факультет информационных технологий ОГУ) в основном в рамках проекта Intel «Виртуоз-2005». Данный комплекс состоит из трех разделов:

- Инструменты для разработки параллельных программ в системах с общей памятью
- Методика разработки многопоточных программ в системах с общей памятью с применением инструментов в примерах и задачах
- Оценка масштабируемости многопоточных программ в системах с общей памятью

и включает в себя учебное пособие, презентации к семинарским занятиям, задания для самостоятельной работы, описания

лабораторных работ и совокупность исходных текстов программ, необходимых для выполнения лабораторных работ.

Образовательный комплекс «Технологии параллельного программирования в системах с общей памятью» частично был опробован на ВМК ННГУ (ноябрь-декабрь 2005г.) и на Зимней школе молодых ученых и специалистов по параллельному программированию – 2006, в которой авторы участвовали в качестве преподавателей.

Процесс совершенствования IT-образования будет сопровождаться переработкой рабочих программ дисциплин, приведенных в таблице, подготовкой избранных лекций и их мультимедийного сопровождения, разработкой тематики курсовых работ и индивидуальных проектов, изданием методических указаний к введенным лабораторным работам, а также обновлением методики преподавания отдельных дисциплин в связи с их ориентацией на параллельное программирование.

Литература

1. Computing Curricula 2001. Association for Computing Machinery and Computer Society of IEEE.
2. Государственный образовательный стандарт высшего профессионального образования. Специальность 010503 Математическое обеспечение и администрирование информационных систем. Министерство образования Российской Федерации. 2000.
3. Современные информационные технологии и ИТ-образование. Сборник докладов научно-практической конференции: учебно-методическое пособие. Под ред. проф. В.А. Сухомлина / отв. ред. Е.Н. Никулина. - М.: МАКС пресс, 2005. - 892 стр.
4. Подготовка и переподготовка ИТ-кадров. Проблемы и перспективы. /Под ред. С.В. Коршунова и В.Н. Гузненкова. - М.: Горячая линия-Телеком, 2005. - 262 с.

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА РЕШЕНИЯ СИСТЕМЫ ПОЛУЛИНЕЙНЫХ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА ПРИ МОДЕЛИРОВАНИИ РОСТА ЗЛОКАЧЕСТВЕННОЙ ГЛИОМЫ

Пименова Т.П.

Московский физико-технический институт (государственный университет), Долгопрудный

К многомерным уравнениям параболического типа сводятся математические модели многих природных процессов. В частности, математическая модель роста и пролиферации глиомы (злокачественной опухоли головного мозга) может быть представлена как система полулинейных уравнений параболического типа [1]. Для адекватного описания процесса необходимо решать систему уравнений в области сложной геометрии, учитывающей анатомические особенности головного мозга и распределение в нем крупных кровеносных сосудов. Это требует введения подробных расчетных сеток. Реализация численных методов приводит к большому числу операций и, следовательно, увеличению процессорного времени.

Рассматривается трёхмерная двухкомпонентная модель роста злокачественной опухоли головного мозга, основанная на следующих предположениях. Опухоль растет как трехмерная колония злокачественных клеток, которые могут находиться в состоянии покоя или деления. Скорость изменения количества делящихся клеток определяется равновесием между способностью клеток передвигаться, делиться и переходить из делящегося в покоящееся состояние. Скорость перехода делящихся клеток в покоящееся состояние определяется концентрацией питательных веществ, распространяющихся от некоторого источника и поглощающихся делящимися клетками. Распределение питательных веществ определяется равновесием между этими процессами. Скорость поглощения является функцией концентрации типа Михаэлиса-Ментен. Основными уравнениями системы являются уравнения типа “реакция диффузия” [1].

Для численного решения системы уравнений применяется консервативная явная разностная схема на неоднородной трехмерной сетке. Для построения сетки был использован подход, предложенный в

[2], при котором расчетная область пространственно разбивается на кубические объемы, объединенные в 8-дерево.

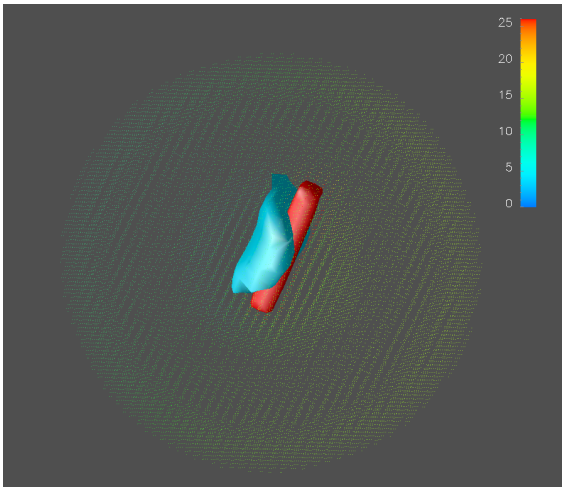


Рис. 1. Захват кровеносного сосуда опухолью (показан фронт опухоли, соответствующий безразмерному значению концентрации раковых клеток 0,2), $t \approx 154$ суток.

В работе иерархическая расчетная сетка строится с использованием восьмидеревной структуры данных. При таком подходе пространство расчетной области вписывается в куб, который может разбиваться на 8 одинаковых вложенных кубов. Эти действия повторяются для тех из них, в которых имеются требующие более тщательного описания области (например, неподвижные границы), до достижения заданной степени подробности.

Для эффективного распараллеливания процесса вычислений требуется обеспечить равномерную загрузку всех вычислительных узлов при одновременном сокращении объема пересылаемых данных. Решение данной задачи осложняется при использовании нерегулярных сеток, поэтому в работе расчетные подобласти выбираются с помощью кривой Гильберта [3]. Данная кривая обладает свойством локальности – близкие в трехмерном пространстве ячейки являются близкими и в порядке обхода вдоль кривой Гильберта.

С использованием предложенного подхода проведены тестовые расчеты роста глиомы в однородной среде с распределенным

источником субстрата. Результат расчета представлен на рис. 1. Показан эффект «захвата» кровеносного сосуда растущей опухолью.

В настоящее время ведется работа по включению в расчетный модуль анатомических особенностей головного мозга.

Результаты численных расчетов находятся в качественном соответствии с клиническими данными.

Литература

1. Астанин С. А., Колобов А. В., Лобанов А. И., Пименова Т. П., Полежаев А. А., Соляник Г. И. Влияние пространственной гетерогенности среды на рост и инвазию опухоли. Анализ методами математического моделирования./ В кн. Медицина в зеркале информатики. М. Наука, 2006.
2. Khokhlov, Fully threaded tree for adaptive refinement fluid dynamics simulations. Journal of Computational Physics, 143:519-543, 1998.
3. G. W. Zumbusch. On the quality of space-filling curve induced partitions. Z. Angew. Math. Mech., 81:25-28, 2001.

Контакты

suntapa@img.ras.ru

РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ВОЗВЕДЕНИЯ ДЛИННЫХ ЧИСЕЛ В СТЕПЕНЬ ПО МОДУЛЮ ДЛЯ КРИПТОСИСТЕМЫ RSA

Полежаев П.Н., научный руководитель – к.т.н. Нестеренко М.Ю.

Оренбургский государственный университет, Оренбург

Введение

В настоящее время большое распространение получила ассиметричная криптосистема RSA, которая является шифром с открытым ключом. Основные ее достоинства: отсутствие необходимости передачи секретных ключей по открытым каналам связи, возможность цифрового подписывания сообщений, простота алгоритма и возможность использования ключей произвольной длины.

Процесс шифрования и дешифрования в криптосистеме RSA осуществляется с помощью возведения длинного числа в степень по модулю. Данная операция является вычислительно трудоемкой и

замедляет работу алгоритма, являясь его узким местом. Возможно повышение эффективности реализации RSA за счет ее распараллеливания для компьютеров с многоядерными процессорами и многопроцессорных рабочих станций.

Анализ известных последовательных алгоритмов возведения в степень по модулю

Пусть необходимо вычислить

$$r = b^p \bmod m, \quad (1)$$

для длинных чисел b, p и m.

Рассмотрим известные последовательные алгоритмы возведения в степень по модулю и проведем анализ на основе оценок времени их работы (см. таблицу 1).

Таблица 1: Временная сложность последовательных алгоритмов возведения в степень по модулю

Название алгоритма	Точная оценка времени	Оценка времени в худшем случае
Последовательное умножение по модулю	$\Theta(2p)$	$O(2^{k+2})$
Индийский алгоритм	$\Theta(2k + 2w(p))$	$O(4k)$
Схема Горнера	$\Theta(2k + 2w(p))$	$O(4k)$

При оценке времени работы мы допустили, что операции «длинного» умножения и вычисления остатка от деления выполняются за $\Theta(1)$, показатель степени состоит из k бит, $w(p)$ - вес Хэмминга показателя степени.

Для реализации параллельной версии алгоритма шифрования и дешифрования криптосистемы RSA была выбрана схема Горнера, т.к. она имеет линейную временную сложность и, в отличие от рекурсивного индийского алгоритма, может быть легко распараллелена.

Схема Горнера возведения длинного числа в степень по модулю использует следующую формулу:

$$b^p \bmod m = (...(((b^{p_0} \bmod m)^{2^0} \bmod m)((b^{p_1} \bmod m)^{2^1} \bmod m)) \bmod m)... \quad (2)$$
$$...((b^{p_{k-1}} \bmod m)^{2^{k-1}} \bmod m)) \bmod m$$

где $p = p_{k-1}...p_1 p_0$ - двоичное представление показателя степени.

Легко вычислить все значения $(b \bmod m)^{2^j} \bmod m$ для всех $j = \overline{0, k-1}$ путем последовательного возведения в квадрат по модулю, а затем умножить по модулю m все полученные числа $(b \bmod m)^{2^j} \bmod m$, для которых $p_j = 1$. Последнюю операцию можно выполнять параллельно, с помощью каскадной схемы умножения.

Каскадная схема умножения набора чисел по модулю

Рассмотрим следующую задачу: имеется k длинных чисел $a_1,...,a_k$, требуется вычислить параллельно произведение этих чисел по модулю m. Данная задача легко решается с помощью каскадной схемы умножения по модулю, принцип работы которой для четырех чисел приведен на рисунке 1.

В общем случае все k чисел разбиваются на пары и перемножаются, затем результаты умножения снова попарно перемножаются и так далее, пока не останется одно число. Очевидно, умножения длинных чисел на одном уровне можно выполнять параллельно.

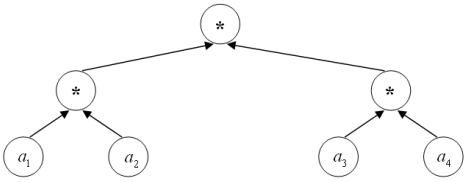


Рис. 1 Принцип работы каскадной схемы

Введем обозначения: пусть s – общее число вершин двоичного дерева каскадной схемы умножения, h – максимальная высота среди всех высот полных двоичных поддеревьев данного двоичного дерева.

Рассмотрим следующую структуру данных – массив $v[0..s-1]$. Элемент $v[i]$ приписан узлу с номером i двоичного дерева каскадной схемы, в $v[k-1], \ldots, v[s-1]$ размещаются исходные операнды $a_1,\ldots,a_k$, а элемент $v[j]$ определяется как результат умножения по модулю элементов $v[2j+1]$ и $v[2j+2]$ при $j=0,\ldots,k-2$. Данная структура данных хорошо подходит для решаемой задачи. Процесс умножения производится по уровням снизу вверх. Умножения на каждом из уровней производятся параллельно.

Приведем параллельный алгоритм на основе каскадной схемы, решающий данную задачу:

Алгоритм 1 – Параллельный алгоритм умножения набора чисел по модулю

Входные значения: количество чисел k чисел, умножаемые числа $a_1,\ldots,a_k$, модуль m

Выходные значения: $r = a_1 \cdot \ldots \cdot a_k$.

```
s := 2*k - 1
h = ⌊log2(s+1)⌋ - 1
shift = k-1
parallel for i := 0 to k-1 do
{
    v[i+shift] := a_{i+1}
}
if 2^{h+1} - 1 < s then
{
     left_num := 2^h - 1
     right_num := shift - 1
     parallel for i := left_num to right_num do
     {
          v[i] := (v[2*i+1]*v[2*i+2]) mod m
     }
}
left_num := 2^{h-1} - 1
right_num := 2^h - 2
```

```
for level:= h – 1 to 0 do
{
     parallel for i := left_num to right_num do
     {
          v[i] := (v[2*i+1]*v[2*i+2]) mod m
     }
     right_num := left_num – 1
     left_num := (left_num + 1) / 2 - 1
}
return v[0]
```

Шаги 1 – 3 представляют собой вычисление основных параметров дерева каскадной схемы, *shift* содержит номер первого листа в дереве. В цикле 4 – 7 в массив *v* записываются исходные умножаемые числа $a_1,...,a_k$, итерации цикла распределяются равномерно между потоками и выполняются параллельно. Условие 8 проверяет, не является ли двоичное дерево каскадной схемы полным, если нет, то отдельно производятся умножения на *h*-м уровне. Цикл 19 – 27 проходит по всем уровням от *h*-1 до 0. Переменные *left_num* и *right_num* содержат номера вершин на текущем уровне, для которых производится умножение по модулю. Умножение по модулю с сохранением результатов осуществляется в циклах 12 – 15 и 21 – 24, выполнение итераций этих циклов распределяется равномерно между потоками и производится параллельно.

$\Theta(\frac{2k}{q})$ - теоретическая оценка времени работы данного алгоритма, где k - количество умножаемых чисел и q - количество вычислительных блоков.

Для оценки ускорения и эффективности параллельной каскадной схемы умножения по модулю была создана ее программная реализация, распараллеливание осуществлялось с помощью OpenMP 2.0. Для различного количества множителей производилось по 10 замеров времени работы с помощью Sampling Wizard инструмента Intel VTune Performance Analyzer, из которых выбиралось минимальное значение.

Все программы запускались на компьютере с двуядерным процессором Intel Pentium D 820, частота каждого ядра 2.8 ГГц, два

кэша L2 по 1Мб на ядро, частота системной шины 800 МГц, RAM 1 Гб. На рисунке 2 приведен график зависимости ускорения от количества умножаемых длинных чисел. Видно, что ускорение асимптотически приближается к 2, что говорит о качественном распараллеливании.

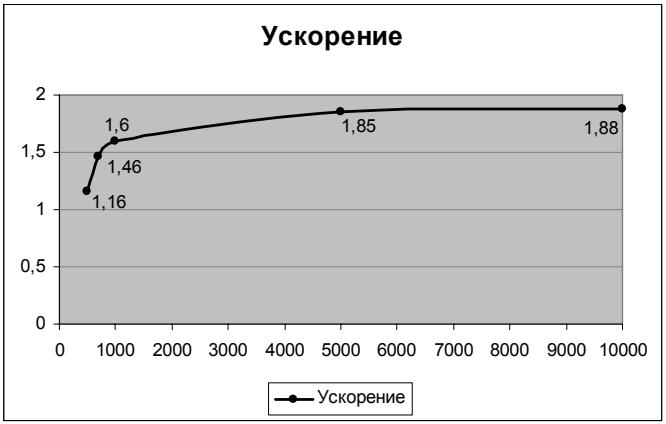


Рис. 2 График зависимости ускорения от количества множителей

Параллельная версия схемы Горнера, использующая параллельную каскадную схему умножения чисел $(b \bmod m)^{2^j} \bmod m$, для которых $p_j = 1$, имеет временную сложность $\Theta(2k + \frac{2w(p)}{q})$. В худшем случае – $O(2k + \frac{2k}{q})$.

Исследование параллельной реализации схемы Горнера

Для различных длин показателя степени было произведено по 10 замеров времени работы программной реализации параллельной схемы Горнера, из которых выбиралось минимальное значение. В результате был построен график зависимости ускорения от длины показателя степени (см. рисунок 3).

Результаты замеров показывают, что параллельный алгоритм на основе схемы Горнера работает быстрее последовательного. При

длине показателя степени от 1024 до 4096 бит ускорение возрастает от 1.28 до 1.31 – это дает результат, близкий к теоретическому, т.к. при $q = 2$ (двуядерный процессор) согласно теоретическим оценкам можно получить ускорение $S_2 = \frac{2k + 2w(p)}{2k + w(p)} \le \frac{4}{3} \approx 1.33$.

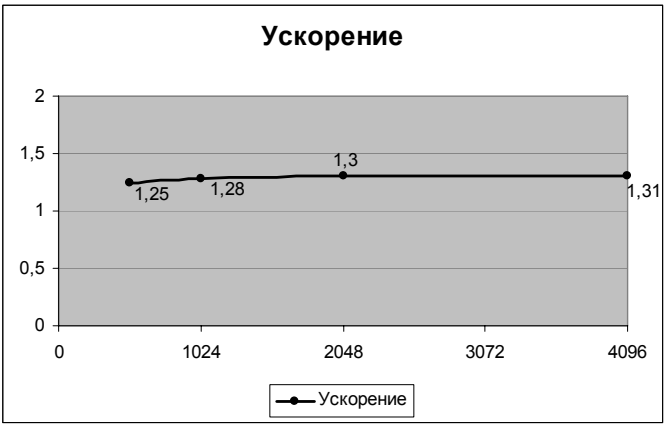


Рис. 3 График зависимости ускорения от длины в битах показателя степени

Заключение

Создан параллельный алгоритм шифрования и дешифрования для криптосистемы RSA, использующий параллельную схему Горнера и модифицированную каскадную схему умножения последовательности чисел по модулю для возведения длинного числа в степень по модулю. Данная параллельная схема может быть также применена при разработке других криптосистем, например, Диффи-Хеллмана, Эль-Гамаля, стандарта цифровой подписи ГОСТ Р 34.10-94, а также для вычислений в теории чисел и вычислительной математике.

Дальнейшие направления продолжения работы:

- оценить масштабируемость разработанного параллельного алгоритма возведения в степень по модулю;
- уменьшить долю последовательных вычислений в параллельной схеме Горнера;
- оценить криптостойкость алгоритма RSA с помощью реализаций алгоритмов факторизации на кластере для ключей

небольшой длины с дальнейшей экстраполяцией результатов на более длинные ключи;
- оценить криптостойкость алгоритма RSA с помощью методов криптоанализа;
- распараллелить алгоритмы других криптографических систем.

Литература

1. Фергюсон Н., Шнайер Б. Практическая криптография. – М.: Вильямс, 2005. – 424 с.
2. Ященко В.В. Введение в криптографию. – М.: МЦНМО, 2000. – 288 с.
3. Гергель В.П., Стронгин, Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. – Н. Новгород: ННГУ, 2000 – 176с.
4. Спецификация OpenMP Application Programming Interface версия 2.5.

ФУНКЦИОНАЛЬНАЯ ВАЛИДАЦИЯ УСТРОЙСТВА УПРАВЛЕНИЯ МИКРОПРОЦЕССОРОВ С EPIC АРХИТЕКТУРОЙ.

Попов И.А.

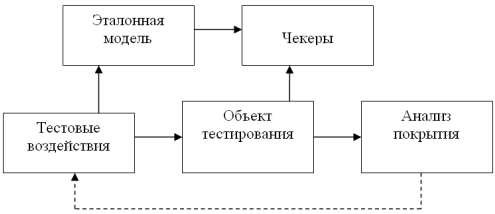
Intel, Москва

Три «кита» валидации

Обычно, команда валидации решает три основных задачи: создание тестовых воздействий, механизмов проверки правильности функционирования Объекта тестирования (создание Чекера) и анализ полноты тестирования. Качество валидации определяется по качеству этих трех компонентов. Недостаточное внимание к любому из них существенно повышает риск выхода продукции, содержащей ошибку.

Критериями качества для Тестовых воздействий являются: скорость моделирования и величина проверяемой функциональности Объекта Тестирования. Учитывая ограниченные вычислительные ресурсы и относительно низкую скорость моделирования, малоэффективные тесты могут стать серьезной проблемой на пути к достижению требуемого качества продукта. Механизмы проверки (Чекеры) проверяют, что поведение Объекта Тестирования соответствует эталонной модели. Анализ покрытия показывает

количественную оценку тестирования. Анализ покрытия подразделяют на 2 части. Анализ покрытия кода, который указывает: какая строчка описания Объекта Тестирования была затронута Тестовым воздействием. Функциональное покрытие собирает информацию о том, была ли модель во время моделирования в описанном функциональном состоянии (например было ли обращение за страницей памяти в спекулятивном режиме при промахе в TLB)



Задача валидации (т.е. нахождение ошибок) реализуется разными способами и на разных уровнях абстракции. Архитектурная валидация использует «черный ящик», который располагает только информацией о поведении входов и выходов Объекта тестирования. Для микропроцессоров - это Система команд, описанная в Руководстве программиста. Микроархитектурная валидация использует «белый ящик», в котором Объект тестирования представляется моделью с известной внутренней структурой. Основой «белого ящика» - описание микроархитектуры и тестплан (плод совместной работы команд валидаторов и RTL разработчиков). Для достижения приемлемого качества валидации необходимо использовать оба типа валидации.

Микроархитектурная валидация в свою очередь делиться на Валидацию на уровне core и Кластерную Валидацию. Объектом валидации на уровне Кластера является – некоторый выделенный по функциональным признакам, фрагмент проекта. Вокруг Кластера создается тестовое окружение, которое включает в себя Генератор тестовых воздействий, Чекер, Имитацию окружения Кластера (т.е. поведение логики вокруг данного кластера).

Преимущества валидации на уровне Кластера заключаются в большей, по сравнению с Валидацией на уровне core, скорости моделирования, доступности логики, простоте выяснения причины и устранения ошибок. Также важным фактором является возможность

начать валидацию на как можно более раннем этапе проекта. Однако, у данного вида валидации существуют явные недостатки, и в первую очередь, к ним нужно отнести большие затраты на создание кластерного окружения. Так количество строк кода, описывающего кластерное окружение, может в десятки раз превышать размер кода Объекта тестирования. Другим недостатком является сильная зависимость от конкретной реализации, и как следствие невозможность вторичного использования (переиспользования) тестов для других проектов или кристалла в целом.

Тестирование на уровне core, основной целью ставит проверку взаимодействий между кластерами.

Далее рассмотрим процесс функциональной верифкации на примере двух микропроцессоров с ILP. Российского «Эльбрус-2000» и Itanium компании Intel.

Эльбрус-2000

«Эльбрус 2000» является представителем процессоров с архитектурой Широкого командного слова. Каждая широкая команда может содержать до 8 – скалярных операции(до 8 целочисленной арифметики, до 4 чтений из памяти и до 2 записей в память), до 3 вычислений предикат, и тд. За компоновку команд в широкую команду отвечает компилятор. Другой отличительной чертой «Эльбруса -2000» стало наличие большого регистрового файла, состоящего из 256 регистров по 64 бит. Регистровый файл имеет 10 портов и возможность совершать операции чтения и записи в одном такте.

Во время цикла разработки были созданы следующие модели**:**. M1- ранний анализ компромиссных решений с упором на прогнозирование производительности; M2 - ISET модель; M3- подробная модель RTL (VERILOG); M4- модель реализации на транзисторном уровне.

Модель M2 стала главным инструментом проектирования для разработки операционной системы и системного программирования. В микропроцесоре с VLIW/EPIC архитектурой большое значение имеет системное программное обеспечение в частности компилятор. VLIW/EPIC компилятор выполняет работу по оптимизации кода, выявлению параллелизма и разрешению конфликтов. Компилятор имеет большую сложность и требует значительных затрат на разработку. Поэтому целесообразно начинать проектирование

компилятора не дожидаясь кремневой реализации микропроцессора. Для этого команде программистов потребовалась дополнительная модель. Скорость моделирования на ISET модели точно имитируюшей систему команд, по сравнению с RTL моделью превышает 1000 раз. Данная модель была создана на C/C++. Все тесты, за исключением тестов на динамику, исполнялись на RTL и ISET моделях.

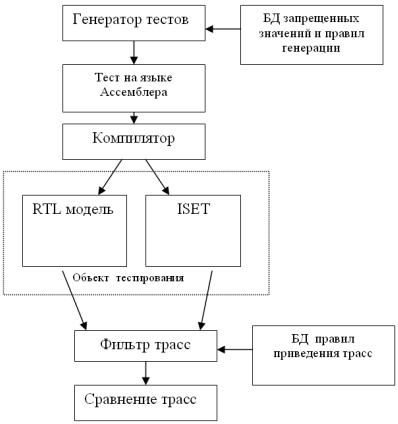
Так как система команд «Эльбрус-2000» была уникальной, то минимальной задачей для команды верификации, стала задача создания полного набора архитектурных тестов, покрывавших все команды. При выборе стратегии верификации основным критерием стало ограничение по ресурсам. Команда сконцентрировала свои усилия на Верификации на уровне core . Тестированием на уровне кластера и устройства в основном занимались инженеры-разработчики RTL. При этом уровень полноты проверки каждый разработчик определял себе сам, единственным требованием было то, чтобы после интеграции модуля в микропроцессор, это не вызвало полной остановки моделирования.

Работа по верификации Модуля Обработки Прерываний «Эльбрус-2000» была сконцентрирована в первую очередь на поиске и проверке «наиболее опасных участков проекта». Тестовые воздействия писались на Ассемблере.Однако оказалось что, создание некоторых интересующих валидаторов ситуации только средствами Ассемблера слишком сложно. В проекте было использовано следующее решение: перед запуском теста, модель микропроцессора с помощью модуля написанного на Verilog , переводилась в заведомо корректное архитектурное состояние (определяемое значениями регистиров управления, состоянием конвейера, и регистрового файла)упрощающее доступ к тестируемому алгоритму.

Когда Ситемма команд была протестирована , команда валидаторов переключила свое внимание на создание стрессовых тестов, используя генераторы. Длинная команда микропроцессора «Эльбрус» способна содержать до 21 простых команд, каждая из которых потенциально может вызвать исключительную ситуацию. Для Модуля Обработки Прерываний и Устройства Управления, особую важность представляла проверка совместного возникновения и обработки нескольких исключительных ситуаций. Для этой цели был создан генератор, создававший различные, не запрещенные комбинации команд и упаковывающий их в длинное командное слово.

Затем эта программа одновременно исполнялась на RTL и ISET моделях. В результате исполнения теста продуцируется протокол-трасса, содержащий архитектурные состояния микропроцессора. Однозначное сравнение трасс полученных с помощью ISET и RTL невозможно, в силу различного уровня абстракции этих моделей. Поэтому возникла необходимость к приведению трасс полученных на различных моделях, к общему виду, позволяющему дальнейшее сравнение, в случаи расхождения, преобразованных таким образом трасс, инициировался процесс выяснения причин расхождения. В результате чего, либо вносились изменения в Генератор тестовых нагрузок, либо накладывались ограничения на модуль отвечающий за приведение трасс, в остальных случаях фиксировалась ошибка либо RTL модели, либо ISET модели.

В завершающей стадии валидации, проводилось тестирование наиболее ответственных модулей, на уровне логического элемента. Так были отвалидированы наиболее сложные сумматоры (использующие методы «быстрой» арифметики). На этой стадии широко применялись Формальные методы верификации.



Itanium 2

Команда разработчиков Itanium 2 избрала несколько иную тактику, и начали акцентированную проверку Модуля Обработки Прерываний, входящего в Устройство Управления, на завершающей стадии Функциональной валидации, когда большинство пунктов тест плана были уже выполнены. Генераторы случайных тестов

перестали выявлять большое количество ошибок. . Инфраструктура для проведения тестирования на уровне устройства состояла из изолированной RTL модели Модуля обработки прерываний, RTL симулятора и тестового окружения написанного на языке Perl и использующего интерфейс с симулятором(API) для управления сигналами объекта тестирования и контроля выходных значений. Тестовое окружение состояло из генератора тестовых воздействий, поведенческой модели(эталонной модели используемой для генерации архитектурно корректных воздействий и в качестве эталонной для проверки функциональной правильности поведения Объекта тестирования), а также Чекера. Стоит отметить что при валидации Itanium, в отличии от «Эльбруса» эталонная модель создавалась специально для Модуля обработки прерываний, и являлась как бы изолированной моделью.

Основным принципом создания тестовых нагрузок является попытка имитации работы реального микропроцессора, с помощью создания случайных воздействий на изолированный объект тестирования. Для достижения этой цели Модуль обработки прерываний представлялся для генератора тестовых нагрузок в качестве «черного ящика». Генератор тестовых нагрузок не имеет информации о внутренней логики работы устройства и имеет доступ только к интерфейсным сигналам. Другим требованием является создание архитектурно верных «интеллектуальных» случайных воздействий.

В проекте Генератор тестовых воздействий имитировал действия которые обычно происходят с аппаратурой, генерируя псевдослучайные события для тестируемого устройства. Генерация происходила до тех пор пока не достигалось ошибочная ситуация. Однако существовала опасность невозможности повторного воспроизведения данной ситуации. Для предотвращения этого, генерация начиналась с заведомо функционально правильного состояния. В случае подозрения на ошибку, должны быть проверены все компоненты имеющие отношение к данной операции, включая анализ корректности тестового воздействия. Зачастую ошибки возникали из за различного понимания микроархитектуры инженерами разработчиками и валидаторами, а также из за недокументированных допущений. После анализа принималось решение либо по исправлению ошибки в RTL модели, либо по исключению данной ситуации из генерируемых, путем наложения

дополнительных ограничений на Генератор тестовых воздействий. Преимущества данного подхода состоят в том, что команда валидаторов, создавая Генератор, руководствовалась исключительно спецификацией, а не предположениями инженеров разработчиков.

Эталонная модель получала те же тестовые воздействия как и RTL модель Объекта тестирования. Это позволяло избежать «мошенничества» со стороны эталонной модели, и заставляло «честно» реализовывать все механизмы Модуля обработки прерываний. Модель включала подробный конвейер, в частности задержки, условия очистки конвейера, различные условия входа в прерывания и возврата из прерываний. Включение динамических характеристик в эталонную модель, позволяло контролировать не только функциональную правильность работы модели, но и выявлять ошибки связанные с планированием и производительностью.

Каждый такт Чекер отслеживал все интерфейсные сигналы между Оъектом тестирования и «внешним миром», а также контролировал некоторые важные внутренние сигналы, определяющие состояние Модуля обработки прерываний. В случаи расхождения в значениях данных сигналов между RTL моделью и Эталонной моделью, Чекер сигнализировал об этом делая соответствующую запись в файл протокола. Ситуации делились на две категории: Запрещенные ситуации – обозначавшие что Генератор тестовых воздействий выдал запрещенную комбинацию, и Ошибка модели – обозначавшая что данные полученные от RTL модели расходятся с Эталонной моделью. В ходе обсуждения эта категория могла перейти либо в Ошибку Модели, либо в Ошибку Аппаратуры, либо Ошибку документации.

. Результатом тестирования на уровне устройства стало обнаружение 44 ошибок не выявленных другими методами, которые могли бы стать серьезной проблемой при выходе продукта.

Выводы

Валидация на уровне Кластера или устройства, позволяет существенно улучшить качество тестирования, а также сократить время валидации, за счет параллельного тестирования Кластеров находящихся в изолированном окружении. Однако, это требует больших затрат на создание и поддержание Тестовового окружения, особенно на ранних этапах проектирования, когда возможны частые изменения в алгоритмах работы модели микропроцессора. Свою

эффективность показало решение по созданию достаточно подробной ISET модели используемой как командой валидаторов, так и командой разработчиков системного программного обеспечения.

Литература

1. Pre-Silicon Validation of IPF Memory Ordering for Multi-Core Processors Soohong Kim Intel Corporation
2. Validating the Itanium 2 Exception Control Unit: A Unit-Level Approach Carl Scafidi Intel J. Douglas Gibson and Rohit Bhatia Hewlett-Packard.
3. Functional design verification for microprocessors by error modeling David Van Campenhout
4. Functional Verification Methodology for the PowerPC 604 Microprocessor J. Monaco, D. Halloway, and R. Raina

КОРРЕКТНОСТЬ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ЛОГИЧЕСКОГО УПРАВЛЕНИЯ.

Попов А.И.

Поморский государственный университет им. М.В. Ломоносова, Архангельск.

Введение

Представлена методика верификации алгоритмов логического управления, включающей проверку корректности алгоритма и его диагностику. Цель проверки корректности – установить, отвечает ли алгоритм выбранным условиям корректности, а диагностика состоит в указании источников нарушений этих условий. Данная методика реализована в программе «Система верификации ПРАЛУ-описаний».

Методика верификации

Исследуется корректность алгоритмов управления, представленных на языке ПРАЛУ и называемых ПРАЛУ-описаниями.

Переменные в ПРАЛУ являются булевыми. Множество условных переменных отображает состояния системы, множество управляющих переменных – сигналы управления.

Главные операции – ожидание и действие. Условия ожидания и выражения в операциях действия представляются элементарными конъюнкциями k_i. Каждой из них соответствует некоторое событие в

системе, наступающее, когда конъюнкция принимает значение 1. Последовательности операций ожидания и действия являются линейными алгоритмами и называются цепочками. Цепочки с совпадающими метками при записи алгоритма группируются вместе и называются предложением. Порядок исполнения цепочек задается с помощью множества меток M. Метки состоят из одного (простые) или нескольких (составные) натуральных чисел, разделенных точками. Исполняются те цепочки, входные метки которых принадлежат множеству запуска $N \subseteq M$. При исполнении цепочки элементы входной метки удаляются из N, а при исполнении операции перехода элементы выходной метки добавляются в N. ПРАЛУ-описание можно представить множеством элементарных цепочек:

μ_i: $-k_i^{/} > k_i^{//} > \nu_i$,

где μ_i, $\nu_i \subseteq M$, μ_i – входная метка, $-k_i^{/}$ – операция ожидания события $k_i^{/}$, $>k_i^{//}$ – операция действия, $>\nu_i$ – операция перехода на выходную метку ν_i. Все цепочки, у которых $\mu_i \subseteq N$, исполняются параллельно и образуют параллельные процессы. Переход на сложную метку запускает несколько параллельных процессов и называется ветвлением И. Составная метка μ_i запускает цепочку только тогда, когда все её элементы находятся в множестве запуска и тем самым образует слияние И. Если предложение содержит несколько цепочек, то все они начинаются с операций ожидания альтернативных событий и образуют ветвление ИЛИ.

Алгоритм называется корректным, если он отвечает некоторому набору формальных условий: непротиворечивость, безызбыточность, настойчивость, самосогласованность, восстанавливаемость, устойчивость и его исполнение соответствует ожиданиям проектировщика системы управления.

С нарушением формальных условий корректности связаны следующие формальные некорректности:

Противоречивость. Предписано одновременное выполнение несовместимых между собой операций. Возникает следующая ситуация:

$$\mu_1, \mu_2 \subseteq N,\ k_1^{/}=1,\ k_2^{/}=1,\ k_1^{/} \wedge k_2^{/}=0$$
$$\mu_1\colon -k_1^{/} > k_1^{//} > \nu_1$$
$$\mu_2\colon -k_2^{/} > k_2^{//} > \nu_2$$

Избыточность. В алгоритме есть операция, которая ни при каких условиях не может быть выполнена.

Ненастойчивость. При параллельном выполнении двух операций одна из них разрушает условия выполнения другой:

$$\mu_1, \mu_2 \subseteq N,\ k_1^{/}=1,\ k_2^{/}=1,\ k_1^{/} \wedge k_2^{//}=0,\ k_1^{//} \wedge k_2^{/}=0,$$
$$\mu_1\colon -k_1^{/} > k_1^{//} > v_1$$
$$\mu_2\colon -k_2^{/} > k_2^{//} > v_2$$

Несамосогласованность. Существует операция, которая повторно инициируется во время ее выполнения.

Невосстанавливаемость. Существует состояние, из которого алгоритм не может вернуться в исходное.

Неустойчивость. Зацикливание алгоритма при неизменных значениях входных переменных.

Проектировщику известно, что некоторые наборы значений переменных недопустимы. Каждому такому набору соответствует некоторая *запрещенная* элементарная коньюнкция. Также можно говорить о запрещенных последовательностях элементарных коньюнкций. Каждая запрещенная последовательность соответствует какому-то нежелательному порядку событий в системе.

Итак, кроме выполнения формальных условий корректности, должно выпоняться требование, что запрещенные наборы и запрещенные последовательности наборов значений перменных не возникают в ходе исполнения алгоритма.

Верификация основана на анализе дерева достижимости алгоритма. Вершины дерева – пары вида (N,V), где $N \subseteq M$, V – множество наборов значений переменных. Пусть (N,V) – произвольная вершина дерева достижимости. $(N_1,V_1),\ldots,(N_n,V_n)$ – все ее дочерние вершины. $\forall i=1,\ldots,n$ (N_i,V_i) получена путем срабатывания одной операции алгоритма, множество запуска и значения переменных которого соответствуют (N,V).

В основе верификации ПРАЛУ-описаний лежит следующий алгоритм.

1. По ПРАЛУ-описанию строится множество S_1 элементов вида $(N_i,\ V_i,\ n_{i1},\ n_{i2})$ таких, что если если $N_i \subseteq N$, $V_i \subseteq V$, то выполнены условия для реализации некорректности. n_{i1} и n_{i2} – номера операций, одновременное срабатывание которых суть реализация противоречивости, ненастойчивости или несамосогласованности. Рассмотрим, например, такой фрагмент ПРАЛУ-описания:

2:-x1.^z1>z3>4

3:-x2>^z3>5

Если 2 $\in$ N, 3 $\in$ N, $x1=1$, $z1=0$, $x2=1$, то выполнены условия одновременного срабатывания операций >z3 и >^z3, что соответствует противоречивости.

2. Строятся множества S_2 запрещенных наборов и S_3 запрещенных последовательностей наборов значений перменных, которые заранее описываются проектировщиком.

3. Для установления избыточности строится массив S_4 из нулей, длина которого равна количеству операций действия в ПРАЛУ-описании.

4. Запускается процедура обхода дерева достижимости алгоритма. Для каждой вершины дерева просматривается S_1. Пусть текущей в дереве является вершина (N_i, V_i), а в множестве S_1 – элемент (N_j, V_j, n_{j1}, n_{j2}). Если $N_i \subseteq N_j$, $V_i \subseteq V_j$, то формируется сообщение о достижимости некорректности, связанной с исполнением n_{j1}-й и n_{j2}-й операций. Далее, просмотр S_1 продолжается. Пусть (N_k, V_k, n_{k1}, n_{k2}) – некоторый его элемент и $n_{k1}=n_{j1}$. Если $N_k \subseteq N_j$ и $V_k \subseteq V_j$, то в сообщение добавляется n_{k2}-я операция.

Когда просмотр S_1 закончен, просматривается S_2. Пусть текущим в S_2 является элемент V_l. Если $V_i \subseteq V_l$, то формируется сообщение о достижимости V_l. i-му элементу массива S_4 присваивается значение 1.

5. Содержание терминальных вершин дерева сравнивается с условием запуска начальной цепочки алгоритма. Делается вывод о восстанавливаемости.

В массиве S_4 ищутся нули. Делается вывод об избыточности.

В пути от начальной вершины до терминальной ищутся участки, соответствующие элементам множества S_3. При нахождении такого участка выводится сообщение о некорректности.

Возможности программы

1. Редактирование ПРАЛУ-описаний.
2. Синтаксический анализ ПРАЛУ-описаний. Программа позволяет выявлять нарушения следующих условий корректности: непротиворечивость, детерменированность, самосогласованность, безызбыточность, настойчивость.
3. Семантический анализ, сведенный к тому, что вручную перечисляются запрещенные элементарные конъюнкции и запрещенные последовательности элементарных конъюнкций, а программа осуществляет их поиск в дереве достижимости.

4. Вывод отчета о проделанном поиске, в котором указываются все пути в дереве достижимости от начальной вершины до терминальных и описываются найденные некорректности.
5. Визуализация поведения алгоритмов. Последовательно подсвечиваются операции, срабатывание которых приводит к возникновению некорректности. При этом показано, как изменяются значения переменных и множество запуска при срабатывании операций.

Описание «Системы верификации»

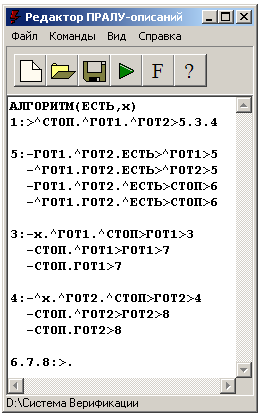


Рис. 1 Редактор ПРАЛУ-описаний.

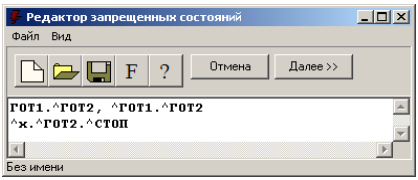


Рис. 2 Окно «Запрещенные состояния» служит для описания запрещенных элементарных конъюнкций и запрещенных последовательностей. Последние описываются как элементарные конъюнкции, разделенные запятыми.

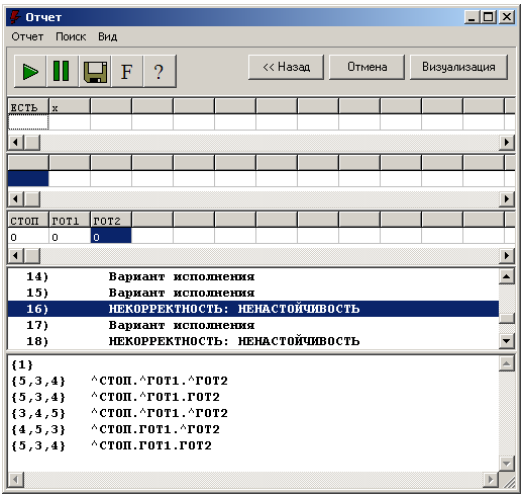


Рис. 3 Окно «Отчет»

Окно «Отчет» служит для запуска процедуры обхода дерева достижимости, поиска некорректностей и вывода результатов поиска. Выводятся пути от начальной вершины дерева достижимости до терминальных. Они обозначаются как «варианты исполнения». Если было выявлено нарушение некоторого условия корректности, то появляется запись, содержащая название некорректности, последовательность операций, ведущих к ее обнаружению и операции одновременное выполнение которых недопустимо. Предусмотрена возможность отложить обход дерева достижимости. Кроме того, в специальной таблице можно задать начальные значения переменных. Переход к следующему окну происходит после выбора интересующей некорректности или варианта исполнения.

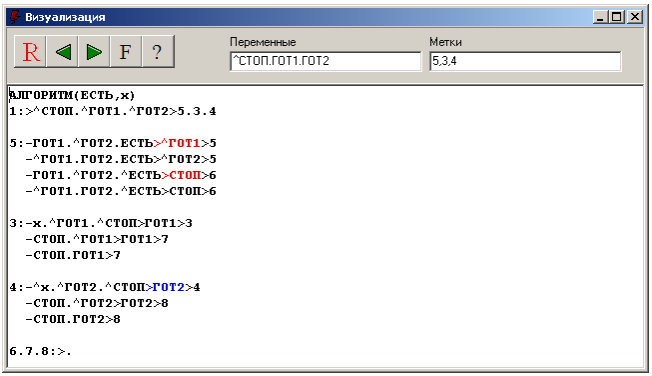


Рис. 4 Окно «Визуализация»

Окно «Визуализация» помогает установить источник интересующей некорректности. Подсвечиваются операции, последовательное срабатывание которых ведет к некорректности, и операции, одновременное срабатывание которых предписано алгоритмом, но недопустимо.

Применение программы

Предполагается совершествование программы. В настоящем виде она подходит для учебных целей, при изучении теории параллельных процессов, и уже использовалась на математическом факультете ПГУ для проведения ряда лабораторных работ.

Литература

1. Закревский А. Д. Параллельные алгоритмы логического управления. – Минск: Институт технической кибернетики НАН Беларуси, 1999. – 202 с.
2. Попов А.И. Система верификации алгоритмов логического управления // Вестник ТГУ. Приложение. – 2006. – №17. – С. 251–253.
3. Черемисинова Л.Д. Реализация параллельных алгоритмов логического управления. – Минск: Ин-т техн. кибернетики НАН Беларуси, 2002.

МЕТОДЫ ИСПОЛНЕНИЯ ФУНКЦИОНАЛЬНО-ПОТОКОВЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Редькин А.В., Легалов А.И.

Красноярский государственный технический университет, Красноярск

Введение

Функциональный язык параллельного программирования Пифагор [1,2,3] был разработан как изначально предназначенный для использования в параллельных вычислениях. Используя потоковую модель вычислений при срабатывании операторов по готовности данных, разработчикам удалось добиться возможности описания в программе естественного параллелизма присущего задаче.

Несмотря на то, что уже существуют экспериментальные системы для последовательной и крупноблочной кластерной интерпретации потоковой модели, задачу о методе и алгоритмах интерпретации нельзя считать решенной, поскольку вышеупомянутые системы интерпретации работают с грубым приближением модели и не поддерживают асинхронных вычислений.

Актуальной, в настоящее время, задачей является создание методов и алгоритмов интерпретации функционально-потоковой модели параллельных вычислений [3], которые позволят реализовать эффективные системы исполнения функционально-потоковых параллельных программ на различных параллельных вычислительных системах.

Асинхронное расширение модели вычислений [4] позволяет более гибко описывать динамический параллелизм. Поддержка модели асинхронных вычислений является одной из задач при создании событийного интерпретатора.

Модель вычислений функционально-потокового языка параллельного программирования [2] представляет собой информационно-управляющую ресурсную модель, задающую динамику функционирования вычислений. Вычислительный процесс можно представить в виде суперпозиции трех графов: информационного, управляющего и ресурсного.

Программа на языке Пифагор фактически описывает некоторый информационный граф [5]. В целях разделения синтаксиса языка и его семантики было решено использовать промежуточное представление, основанное на информационном графе, для дальнейшего

преобразования в управляющий граф и последующей интерпретации программы (Рис. 1).

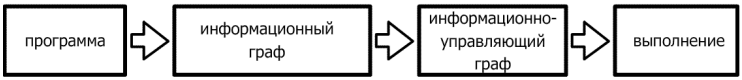


Рис. 1 Этапы преобразования программы на языке Пифагор

Информационный граф определяется как:

$G_i = (V, E)$,

где V – множество вершин определяющих программно-формирующие операторы, а E – множество дуг, задающих пути передачи информации между операторами графа. В рассматриваемой модели вычислений существует 5 типов вершин информационного графа.

Информационный граф отображается во внутреннее представление следующим образом: каждой вершине информационного графа соответствует определенная структура данных, которая содержит информационные связи соответствующей вершины, тип вершины и данные (в случае, если вершина представляет константный оператор).

На основе информационного графа было разработано промежуточное представление языка Пифагор как формата для хранения модулей программы, а также как базовой структуры при интерпретации [5].

Событийная интерпретация

Вычислительный граф программы формируется динамически в ходе вычислений. За динамическое формирование вычислительного графа отвечают параллельные списки, каждый элемент параллельного списка порождает новую ветвь вычислений.

Основными сигналами для событийного интерпретатора являются сигналы-события о готовности данных для вычислений. Для событийного интерпретатора управляющий граф описывает порядок передачи событий готовности данных. Структура события представлена на Рис. 2.

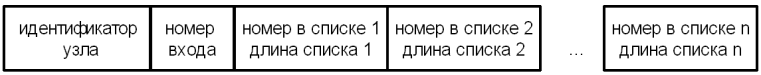


Рис. 2 Структура события

Здесь поля «идентификатор узла» и «номер входа» описывают узел назначения события и номер входа для этого узла, поля «номер в списке 1» ... «номер в списке n» описывают положение события во вложенном параллельном списке.

При обработке события параллельным списком, к нему добавляется информация о номере входа в данном параллельном списке и его длине. В случае вложенных списков произвольной конфигурации, нетрудно заметить, что если сигнал со стороны аргумента имеет вид (X, Px), где Px – информация о вложенных параллельных списках, а сигнал со стороны функции имеет вид (F, Pf), то сигнал результата интерпретации будет иметь вид (Y, Pf, Px). Пример такой обработки представлен на Рис. 3.

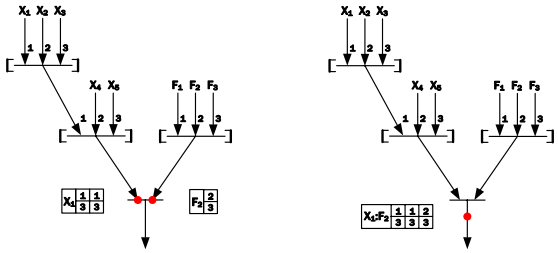


Рис. 3 Пример обработки событий

Таким образом, предложенный способ управления позволяет организовать независимую обработку параллельных списков любой вложенности без потери параллелизма. Также предложенная модель не противоречит правилам преобразования списков языка Пифагор.

Интерпретация на основе очереди событий

Наиболее простой вариант реализации интерпретации на основе событийного подхода – организация очереди событий. Очередь в начале вычислений заполняется событиями, которые порождаются константными узлами информационного графа, поскольку константные узлы изначально готовы к вычислениям. Далее очередь заполняется по мере вычислений и обрабатывается последовательно (Рис.4).

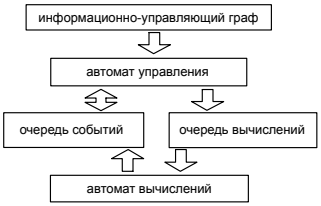


Рис. 4 Общая схема интерпретатора

Вычисления в задержанных узлах, в случае поступления достаточных для начала вычисления сигналов, попадают в из общей очереди в очередь задержанных вычислений для того задержанного подграфа, в котором они находятся (Рис. 5). Очередь задержанных вычислений попадает в очередь вычислений, при раскрытии соответствующего задержанного подграфа (Рис. 6). Очередь вычислений раскрытого задержанного подграфа попадает в конец основной очереди вычислений (на Рис.6 раскрывается задержанный список 2).

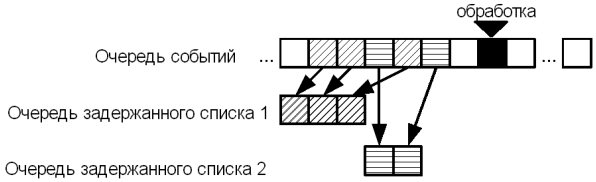


Рис. 5. Преобразования программы на языке Пифагор

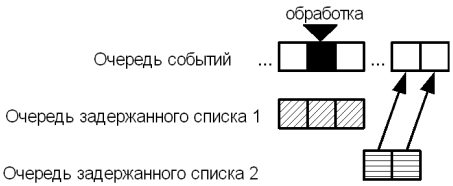


Рис. 6. Преобразования программы на языке Пифагор

Описанный алгоритм работы очереди событий не рассчитан на динамическое параллельное исполнение, но определенным параллелизмом он обладает. В простейшем случае можно, например,

параллельно обрабатывать все события, на данный момент находящиеся в очереди. Более совершенные алгоритмы событийной интерпретации являются предметом дальнейшего исследования в данном направлении.

Заключение

Интерпретация функционально-потоковых параллельных программ может быть реализована на основе событийного подхода путем создания очередей событий. Событийный подход позволит создавать интерпретаторы для различных параллельных систем путем создания нескольких очередей событий и нескольких обработчиков событий для каждой очереди, без серьезных изменений алгоритма.

Литература

1. Легалов А.И., Казаков Ф.А., Кузьмин Д.А. Водяхо А.И. Модель параллельных вычислений функционального языка // Известия ГЭТУ, Сборник научных трудов. Выпуск 500. Структуры и математическое обеспечение специализированных средств. СПб.: 1996. С. 56-63.
2. Легалов А.И. Использование асинхронно поступающих данных в потоковой модели вычислений. / Третья сибирская школа-семинар по параллельным вычислениям. / Томск. Изд-во Томского ун-та, 2006. С 113-120.
3. Легалов А.И. Об управлении вычислениями в параллельных системах и языках программирования – Научный вестник НГТУ, № 3 (18), 2004. С. 63-72.
4. Легалов А.И., Редькин А.В.. Расширение асинхронного управления по готовности данных. Труды III Международной конференции „Параллельные вычисления и задачи управления“ PACO '2006 памяти И.В. Прангишвили. Москва, 2-4 октября 2006 г. Институт проблем управления им. В.А. Трапезникова РАН, 2006. с. 1272-1281.Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. 2005. № 1. С. 71-89.
5. Редькин А.В. Промежуточное представление функционального языка потокового программирования. Третья Сибирская школа-семинар по параллельным

вычислениям/ под ред. проф. А.В. Старченко. Томск: изд-во Том. ун-та, 2006. с. 121-124.

МОДЕЛИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Родионов И.К., Бухановский А.В.
СПбГУ ИТМО, Санкт-Петербург

Оценка производительности распределенных программно-аппаратных комплексов – сложная междисциплинарная задача, которой зачастую не уделяется должного внимания. Как следствие, недооценка реальной производительности выбранного решения приводит к увеличению стоимости владения, а переоценка – к снижению функциональных характеристик, что может привести к пересмотру всего проекта, его стоимости, сроков реализации. Кроме того, при проектировании программно-аппаратной архитектуры необходимо учитывать не только текущие пользовательские требования, но и их изменение в будущем – например, ожидаемый рост нагрузки на систему. Таким образом, недостаточная проработанность рисков, связанных с оценкой производительности может проявиться не сразу, а в течение некоторого времени, когда их устранение может критически сказаться на течении всего проекта.

Большинство существующих методов моделирования производительности сложных распределенных систем основывается либо на экспериментальном измерении производительности с восстановлением зависимости по точкам, либо на аналитическом описании производительности с идентификацией по данным измерений.

Процедуры экспериментального измерения производительности можно схематично разделить на две группы. *Компонентное тестирование*, при котором производится измерение характеристик отдельных частей системы, начиная от процессоров и подсистем памяти и заканчивая тестированием всей аппаратной части (сервера), но без полезной нагрузки. *Интегральный подход*, в свою очередь, характеризуется оценкой производительности комплекса в целом, как его аппаратной, так и программной частей. При этом для измерения производительности может использоваться как само программное обеспечение, на которое ориентировано конечное решение, так и

некоторые модельные приложения, эмулирующие стандартные процессы и нагрузки.

Экспериментальное определение производительности является промежуточным шагом для построения модели поведения системы при различных нагрузках и, как следствие, выполнения расчета характеристик необходимой аппаратной конфигурации для условий работы в реализуемом проекте, или сайзинга (sizing). В основе такого расчета мы рассматриваем аналитическую модель программно-аппаратного комплекса. Она формулируется в виде системы нелинейных уравнений, параметры которых зависят от характеристик нагрузки (числа пользователей, объемов данных и пр.), особенностей аппаратной части и специфики самого программного обеспечения. Параметры модели идентифицируются посредством метода наименьших квадратов на основании специально сформулированных тестов производительности. В настоящее время существует несколько подходов к построению таких моделей, в частности, метод слоев, метод конечных очередей, метод сигнатур и пр.; выбор конкретного метода (или их комбинации) во многом связан со степенью детализации анализируемой системы.

Существенная значимость моделирования производительности проявляется в настоящее время, в условиях массового перехода серверных и десктопных платформ на двуядерные решения. Потому для уже развернутых систем (изначально спроектированных под одноядерные решения), работающих в оперативном режиме, такая модернизация аппаратной базы требует предварительного обоснования. В докладе приводится пример построения модели производительности для реальной экономической информационной системы, реализующей банковские транзакции.

РАЗРАБОТКА ИНТЕГРИРОВАННОЙ СРЕДЫ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ КЛАСТЕРА НИЖЕГОРОДСКОГО УНИВЕРСИТЕТА

Свистунов А.Н., Лабутин Д.Ю., Лопатин И.В., Сенин А.В.
Нижегородский государственный Университет им. Н.И. Лобачевского, Нижний Новгород

В работе рассматриваются проблемы создания интегрированной среды высокопроизводительных вычислений для кластера

Нижегородского университета, который был предоставлен ННГУ в рамках академической программы Интел в 2001 г [3].

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

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

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

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

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

Однако использование готовых систем управления кластером затруднено рядом обстоятельств. Прежде всего, это высокая стоимость подобных систем, достигающая, для кластера подобного кластеру Нижегородского университета, десятков тысяч долларов. Вторым, не менее важным обстоятельством, является закрытость подобных систем, осложняющая проведение некоторых исследовательских работ. Дело в том, что кроме задачи обеспечения функционирования кластерной системы ставилась еще и задача создания испытательного стенда для проведения экспериментов по апробации различных алгоритмов планирования задач на кластерных системах. Для кластера

Нижегородского университета планирование распределения вычислительной нагрузки по узлам кластера является практически важной задачей, в силу его неоднородности.

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

Разрабатываемая программная система в перспективе должна решить еще одну важную задачу – задачу интеграции всех вычислительных ресурсов университета в единую высокопроизводительную среду с единым центром управления.

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

- Реализация по крайней мере минимального набора операций (загрузка задачи, добавление задачи в очередь задач, получение текущего статуса очереди задач и текущей выполняемой задачи, получение результатов вычислений);
- Простой и удобный способ доступа, не требующий установки на рабочих станциях пользователей какого-либо специального программного обеспечения, позволяющий получить доступ к системе из любой точки;
- Собственная система авторизации пользователей не связанная напрямую с системой авторизации операционной системы;
- Наличие системы очередей задач;
- Сохранение задач пользователя после их выполнения и возможность их повторного запуска;
- Автоматическое сохранение результатов работы;
- Возможность расширения системы - включение как новых узлов, так и целых кластеров

Разрабатываемая система, таким образом, может быть представлена двумя уровнями: уровнем локальной системы управления кластером и уровнем интегратора кластеров.

При построении программной системы уровня управления кластером за основу была взята сложившаяся архитектура системы мониторинга и управления[2], включающая, как правило, следующие составные части (Рис. 1)

- компонент, взаимодействующий с пользователем (Менеджер доступа), позволяющий ставить задачи в очередь, удалять

задачи из очереди, просматривать статус задач и т.д., а также ведущий базу данных пользователей и базу данных результатов;

- компонент, оперирующий с очередью заданий (Диспетчер заданий), распределяющий задания по узлам кластера и ставящий их в очередь для выполнения;
- компонент, обеспечивающий мониторинг кластера (Супервизор кластера) и непосредственное выделение ресурсов.

Рис. 1 Архитектура системы

Уровень интегратора кластеров имеет структуру в целом аналогичную представленной, и в свою очередь, имеет собственный менеджер доступа, глобальную базу пользователей, глобальную очередь заданий и супервизор кластеров.

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

- Поддержка минимально - необходимого набора операций по управлению задачами пользователей;
- Возможность доступа к системе с любого компьютера, подключенного к сети Интернет;

- Отсутствие необходимости установки на компьютере пользователя специального программного обеспечения. Использование в качестве клиента web-браузера, telnet-клиента, различных специализированных программ;
- Хранение очереди заданий (как ждущих своей очереди на выполнение, так и уже завершившихся, сформировавших результат) во внешней базе данных, что позволяет обеспечить устойчивость системы в случае сбоя;
- Возможность замены планировщика и изменения стратегии планирования;
- Ведение статистики использования задачами пользователей вычислительных ресурсов кластера;
- Возможность управления несколькими вычислительными кластерами

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

В настоящее время работа над системой продолжается в нескольких направлениях:

1. Возможность одновременного управления несколькими кластерами на базе различных операционных систем: Windows, Linux, FreeBSD;
2. Интеграция с системами управления третьих разработчиков (в случае использования таковых в компьютерных лабораториях и невозможности полного перехода на нашу систему): Microsoft Compute Cluster Server 2003;
3. Простой и удобный способ доступа, не требующий установки на рабочих станциях пользователей какого-либо специального программного обеспечения, позволяющий получить доступ к системе из любой точки (веб-интерфейс);
4. Разработка программ с командным и графическим интерфейсами для предоставления наиболее удобного и быстрого доступа к системе;

5. Административный интерфейс, управляющий ходом работы;
6. Программный интерфейс (веб-сервис, COM-интерфейс и др);
7. Собственная система авторизации пользователей не связанная напрямую с системой авторизации операционных систем;
8. Реализация следующего набора операций над задачами:
    i. Загрузка задачи на сервер;
    ii. Добавление задачи в очередь задач (под очередью задач мы понимаем множество всех задач, ожидающих выполнения на кластерах);
    iii. Получение текущего статуса очереди задач и любой задачи;
    iv. Получение результатов вычислений (перехваченный поток вывода и файлы, созданные программой);

Сохранение задач пользователя после их выполнения и возможность их повторного запуска;
Автоматическое распределение задач по кластерам и далее – по узлам выбранного кластера;
Автоматическое сохранение результатов работы;
Устойчивость к отказам отдельных кластеров и отдельных узлов каждого кластера;
Ведение статистики использования компьютерных ресурсов, автоматическая генерация отчетов.

Разрабатываемая система должна быть достаточно надежна и переносима для использования ее на кластерах широкого типа (при проектировании системы не должно приниматься решений, ограничивающих ее использование кластером ННГУ)

Литература:

1. Rajkumar Buyya. High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., 1999.
2. W. Saphir, L. A. Tanner, B. Traversat. Job Management Requirements for NAS Parallel Systems and Clusters. NAS Technical Report NAS-95-006 February 95.
3. Гергель В.П., Стронгин Р.Г. Высокопроизводительный вычислительный кластер Нижегородского университета. Материалы конференции Relarn. 2002. Н. Новгород.

4. Дрейбанд М.С. Вычислительный кластер ИПФ РАН. Материалы конференции Relarn. 2002. Н. Новгород.

РАЗРАБОТКА ВЫЧИСЛИТЕЛЬНОГО КОМПЛЕКСА ДЛЯ РЕШЕНИЯ ДВУХ И ТРЕХМЕРНЫХ ЗАДАЧ ГАЗОДИНАМИКИ РЕАГИРУЮЩИХ ТЕЧЕНИЙ НА МНОГОПРОЦЕССОРНЫХ ЭВМ

Семенов И. В., Ахмедьянов И. Ф., Уткин П. С.
Институт Автоматизации Проектирования РАН, Москва

Введение

Современные задачи вычислительной газовой динамики в двумерных и, особенно, в трёхмерных областях, требуют значительного количества вычислительных ресурсов. Решение таких задач за приемлемое время невозможно без использования высокопроизводительных параллельных компьютеров. Более того, такие сопутствующие вопросы, как объем оперативной памяти, дискового пространства, или характеристики коммуникационной среды ЭВМ, которые, казалось, давно перестали быть лимитирующими факторами для решения массы вычислительных задач, по-прежнему являются критическими при решении трудоемких задач со сложной геометрией и числом расчетных ячеек, достигающим нескольких десятков миллионов.

Таким образом, проблема разработки эффективных программных комплексов, учитывающих особенности устройства конкретных высокопроизводительных систем, стоит в настоящее время достаточно остро.

Представленная работа посвящена разработке программного комплекса для решения двумерных и трехмерных задач газодинамики реагирующих течений на многопроцессорной вычислительной технике.

Разработка модуля для решения двумерных задач

Хотя подавляющее большинство физических процессов, представляющих интерес для теории и практики, например, распространение детонационных волн или турбулентные течения, являются существенно трехмерными, часто, в силу особенностей

геометрии задачи, достаточно реалистичную картину протекающих процессов дает упрощенное двумерное представление.

При решении двумерных задач в качестве математической модели использовали полную систему нестационарных уравнений Навье-Стокса. Учет протекающих химических реакций производился за счет введения в правые части уравнений источниковых членов. Полученная система решалась методом расщепления по физическим процессам – сначала решалась однородная система уравнений газовой динамики, затем производилась корректировка решения с учетом правых частей. Дискретизация по пространству производилась методом конечных объемов; дискретизация по времени - с помощью явной схемы Эйлера первого порядка. Невязкие потоки определялись из решения задачи Римана о распаде произвольного разрыва на границах ячеек расчетной области по параметрам течения в соседних ячейках (метод типа Годунова). Газодинамические параметры на границах ячеек находились путём одномерной экстраполяции искомых функций от центров ячеек с использованием ограничителя MINMOD. Вязкие потоки через границы ячеек рассчитывались с использованием обобщения схемы центральных разностей на произвольные сетки. Для решения системы кинетических уравнений, описывающих изменение концентраций компонентов смеси, использовался метод Гира, в основу которого положены неявные многошаговые разностные методы высокого порядка точности. Таким образом, для расчетов использовалась явная схема второго порядка по пространственным переменным и первого порядка по времени.

Схема организации модуля для решения двумерных задач представлена на рис. 1.

Рис. 1 Схема организации модуля для решения двумерных задач.

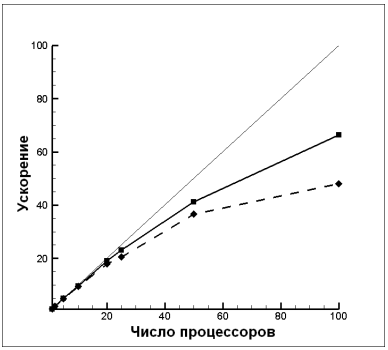


Рис. 2 Зависимость ускорения от числа процессоров. Тонкая линия – идеальное ускорение, жирная сплошная линия – при расчете одного шага по времени, пунктирная – при работе программного комплекса.

Описание структурированной двумерной сетки в виде множества точек, а также начальные и граничные условия задаются через внешние файлы. SOLVER, принципы работы которого были описаны выше, построен по блочному принципу – каждая функциональность реализована в виде отдельного блока, самые крупные из них показаны на рис. 1. Подобная организация комплекса максимально упрощает процесс добавления новых функциональностей (например, новые методы расчета потоков). Результаты работы программы записываются в бинарные файлы, пригодные для визуализации в системе Tecplot – одной из систем, являющейся defacto стандартом визуализации CFD вычислений.

Расчетный алгоритм был адаптирован для проведения вычислений на многопроцессорных ЭВМ. Расчеты проводились на многопроцессорных ЭВМ МСЦ РАН и ИАП РАН. Для распараллеливания использовался одномерный «domain decomposition» метод 1. Проведенные расчеты для тестовой задачи распространения ударной волны в плоском канале, заполненном реагирующей пропано-воздушной смесью, с двумя препятствиями разной формы (область порядка 1000 на 100 ячеек) на кластере МВС-6000IM, установленном в МСЦ РАН, показали высокие характеристики качества распараллеливания. На рис. 2 приведен график зависимости ускорения от количества использованных в расчете процессоров. На рис. 3 приведен график зависимости эффективности (отношения величины ускорения к числу процессоров)

от количества использованных в расчете процессоров. Анализ приведенных графиков показывает, что явная схема расчета одного шага по времени распараллеливается очень хорошо – при использовании 50 процессоров ускорение превышает величину 40, а эффективность 0.8. Даже при использовании 100 процессоров, в силу трудоемкости вычислительных процедур расчета невязких потоков методом типа Годунова на каждой грани всех расчетных ячеек и решения жесткой системы ОДУ химической кинетики, эффективность достигает величины 0.67 (неформальным критерием целесообразности использования того или иного числа процессоров является величина эффективности 0.5). В то же время хорошо видно, что эффективность при работе всего программного комплекса ниже, чем при расчете одного шага по времени. Это связано с тем, что во избежание организации неоднородной схемы работы процессоров, функции по вводу-выводу возложены на один из процессоров, осуществляющих счет, что приводит к разбалансировке нагрузки, приходящейся на один процессор. С другой стороны, тем не менее, эффективность при использовании 100 процессоров при работе всего программного комплекса лишь незначительно ниже 0.5, а при использовании 50 процессоров достигает величины 0.72, что делает неоправданными затраты на организацию сложных схем балансировки нагрузки между процессорами типа master – worker с выделением управляющего процессора.

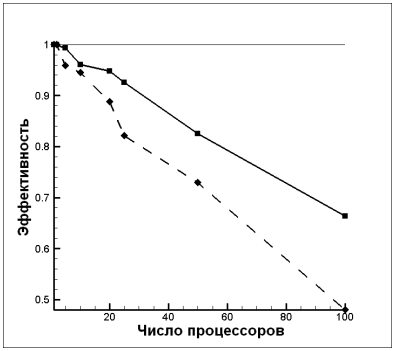


Рис. 3. Зависимость эффективности от числа процессоров. Тонкая линия – идеальная эффективность, жирная сплошная линия – при расчете одного шага по времени, пунктирная – при работе программного комплекса.

Разработка модуля для решения трехмерных задач

Для целого ряда задач вычислительной газовой динамики оправдано применение неявных численных схем. Выигрыш от перехода к счету по неявной схеме заключается в возможности варьирования в широком диапазоне значений числа CFL (Courant-Friedrichs-Levy), характеризующего соотношение между временным и пространственным шагами в процессе вычислений, без потери устойчивости. При расчете по явной схеме условия устойчивости накладывают жесткие ограничения на используемые в расчетах числа CFL. В то же время, переход к неявной схеме вычислений ведет к существенному усложнению алгоритмов распараллеливания. Прежде всего, это связано с тем, что зависимости между газодинамическими параметрами в ячейках разностной сетки становятся гораздо сложнее по сравнению с явными схемами. В действительности, на одной итерации неявного алгоритма значения параметров в одной из ячеек зависят от значений параметров во всех остальных ячейках.

При создании модуля для решения трехмерных задач авторы разработали распределенный вариант LU-SGS алгоритма 2, который реализует неявную численную схему. Распределённый алгоритм условно можно разбить на две составляющие: задание порядка обсчёта ячеек внутри делегированной каждому процессору кластера подобласти, и организация обменов между процессорами кластера внутри одной итерации алгоритма. Всё это осуществляется с целью соблюдения требуемых зависимостей между параметрами в ячейках.

Разработанный модуль реализует данный алгоритм. Основными компонентами в нем являются:

- LU-SGS солвер, производящий вычисления в текущей ячейке, и имеющий доступ только к параметрам в соседних ячейках (основное отличие от явного метода состоит в том, что данные в соседних ячейках уже могут быть изменены солвером в процессе расчёта текущей итерации алгоритма);
- библиотека, реализующая параллельное расширение LU-SGS алгоритма и различного рода сервисные функции.
- Разработанная библиотека характеризуется следующим набором функциональных возможностей:
- эффективная и корректная обработка зависимостей между параметрами в ячейках, как того требует численный алгоритм LU-SGS;
- обработка трёхмерной геометрии;

- качественная разбивка с помощью пакета Metis расчётной области на подобласти, делегированные каждому процессору кластера;
- простой программный интерфейс между солвером и библиотекой;
- поддержка как солверов, работающих по неявной численной схеме, так и солверов, работающих по явной численной схеме;
- поддержка формата CGNS для задания сетки и геометрии расчётной области;
- поддержка выходного формата plt для просмотра результатов в программе Tecplot;
- использование собственного Simple Solution формата, для компактного хранения результатов расчётов без сохранения в каждом файле информации о геометрии расчётной области;
- использование MPI в качестве средства передачи данных;
- кросплатформенность (протестирована на ppc/Suse Linux, ppc/AIX, x86/Debian GNU Linux).

В качестве математической модели для решения трехмерных задач использовались уравнения Эйлера для невязкого сжимаемого газа; дискретизация по пространству, как и в двумерном случае, осуществлялась с помощью метода конечных объемов. Аппроксимация потоков проводилась по методике HLL (Harten – Lax – van Leer) с первым порядком по пространству.

Схема организации модуля для решения трехмерных задач представлена на рис. 4.

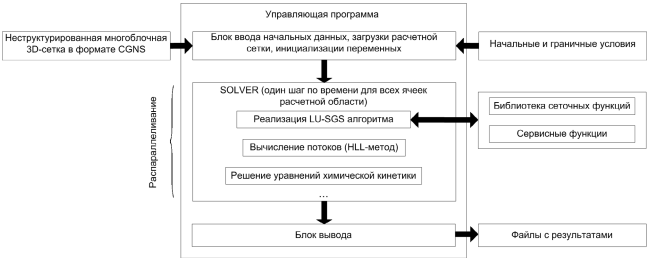


Рис. 4 Схема организации модуля для решения трехмерных задач.

Созданный программный модуль демонстрирует эффективную работу при расчёте области более чем из миллиона ячеек и 32

задействованных процессоров кластера. Это соответствует примерно 40 тысячам ячеек на один процессор.

Применение программного комплекса

Разработанный программный комплекс успешно используется для проведения численных экспериментов по распространению ударных волн и инициированию детонации в областях со сложной геометрией [3, 4, 5]. На рис. 5 приведен пример численной шлирен-визуализации (визуализация модуля градиента плотности газа) течения в плоском канале, заполненном пропано-воздушной смесью, с препятствиями в момент инициирования детонации.

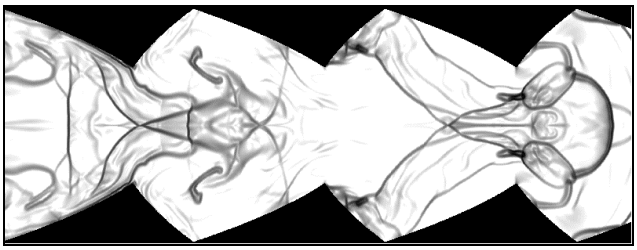


Рис. 5 Численная шлирен-визуализация течения в канале в момент инициирования детонации.

Благодарности

Работа выполнена при поддержке программы № 14 Президиума РАН «Фундаментальные и прикладные проблемы информатики и информационных технологий», гранта РФФИ № 05-08-50115-а, а также гранта «Фонда содействия отечественной науке».

Литература

1. Sourcebook of Parallel Computing/ J. Dongarra [и др.]// Morgan Kaufmann Publishers. – 2003.
2. Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations/ Yoon, Jameson// AIAA J. – 1988 - Vol.26 – No 9 – pp. 1025-1026.
3. On Some Problems of the Electrochemical Pulse Engine Operation/ I.V. Semenov, V.V. Markov, S. Wojcicki// Nonequilibrium Processes. – 2005 – Vol. 1: Combustion and Detonation. G. Roy, S. Frolov, A. M. Starik, Eds. Moscow, Torus Press - pp. 381-387.

4. Shock-to-detonation Transition in Tubes with Shaped Obstacles/ I. Semenov, S. Frolov, V. Markov, and P. Utkin// Pulsed and Continuous Detonations// [Edited by G. Roy, S. Frolov, J. Sinibaldi]. - Moscow: TORUS PRESS Ltd., 2006 – pp. 159 - 169.
5. Численное моделирование реагирующих течений в канале с профилируемыми
препятствиями/ И. Ф. Ахмедьянов, П. С. Уткин/ IX Всероссийский съезд по теоретической и прикладной механике. Аннотации докладов. Т. II// Нижний Новгород, 22 - 28 августа 2006. - Изд-во Нижегородского госуниверситета им. Н. И. Лобачевского, 2006 – стр. 17 - 18.

ЯЗЫК ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ MC# 2.0 И НАПРАВЛЕНИЯ ЕГО РАЗВИТИЯ.

Сердюк Ю.П.

Институт программных систем РАН, г.Переславль-Залесский

В данной статье представляется новый вариант языка MC# - языка параллельного распределенного программирования, базирующегося на платформе .NET.

Мы представляем новые конструкции этого языка – movable-методы, каналы и обработчики; формулируем его особенности и сравниваем MC# с языком X10, одной из последних аналогичных разработок фирмы IBM. В качестве примера программирования на языке MC# 2.0 приводится программа *all2all*, демонстрирующая взаимодействие распределенных процессов между собой по принципу “каждый с каждым”.

Основными направлениями развития языка MC# являются, с теоретической точки зрения, разработка его формального базиса и введение средств описания распределения данных между параллельными процессами, а с практической – интеграция языка MC# в Visual Studio 2005 и реализация метакластерной (Grid-) системы MC# Grid System.

Введение

Широкое распространение вычислительных систем с массовым параллелизмом, таких как многоядерные процессоры, кластеры и Grid-архитектуры, вновь поставило вопрос о разработке высокоуровневых,

мощных и удобных языков программирования, позволивших бы создавать сложное, но, в то же время, и надежное программное обеспечение, эффективно использующее возможности параллельных, распределенных вычислений и легко масштабируемое на заданное количество процессоров, узлов, машин.

Доступные на настоящий момент программные интерфейсы для организации параллельных вычислений, такие как OpenMP (для систем с разделяемой памятью) и MPI (для систем с передачей сообщений), созданы для языков C и Фортран, а потому являются очень низкоуровневыми и не адекватными современным языкам объектно-ориентированного программирования, таким как C++, C# и Java. Новые требования к повышению производительности труда программистов (в частности, через повышение уровня абстракции предоставляемых им языковых средств), а также требования к надежности и безопасности создаваемых ими программ, обусловили тенденцию перехода ключевых понятий наиболее важных API в базовые конструкции языков программирования.

В частности, мы предлагаем сделать еще один шаг в этом направлении – ввести в объектно-ориентированный последовательный язык высокоуровневые конструкции для создания параллельных, распределенных программ, исключив, тем самым, необходимость использования программистом библиотеки System.Remoting (а также, во многих случаях, и System.Threading), необходимой для построения традиционных распределенных приложений.

С практической точки зрения, целью создания языка MC# была разработка языка для промышленного параллельного, распределенного программирования, в которое в ближайшее время будет вовлекаться все больше и больше людей в связи с наступлением эпохи многоядерных вычислений. Это язык призван заменить языки C и Фортран, позволив создавать сложные программные системы, обладающие достаточной эффективностью при исполнении на параллельных архитектурах. Выбор языка C# в качестве базового дает возможность использовать современный объектно-ориентированный язык программирования, снабженный богатыми библиотеками, и, в то же самое время, исключить низкоуровневые и небезопасные средства, такие как C-указатели, которые резко снижают производительность программиста и надежность создаваемых им программ. В этом отношении, наш подход совпадает с подходом, принятым при

разработке языка X10 [1], ориентированным на “неоднородные кластерные вычисления”.

При использовании языка MC#, в отличие от применения интерфейса MPI, нет необходимости явным образом распределять вычислительные процессы по узлам кластера – достаточно только указать какие из функций (методов) могут быть выполнены параллельно. Более того, при исполнении программ на языке MC# новые вычислительные процессы могут порождаться динамически, т.е., уже во время счета, и распределяться по доступным узлам (аналогичная возможность также предусмотрена для “активностей” в языке X10), что невозможно в программах, использующих MPI. Аналогичным образом, в языке MC# нет необходимости вручную кодировать сериализацию данных для передачи их на удаленный узел/машину – Runtime-система языка выполняет сериализацию/десериализацию объектов автоматически.

В действительности, язык MC# есть адаптация базовой идеи языка Polyphonic C# (более точно, join-исчисления [2]) на случай распределенных вычислений. Использование асинхронных методов языка Polyphonic C# предполагалось авторами этого языка только в рамках одной машины, либо на нескольких машинах с зафиксированными на них асинхронными методами с привлечением средств удаленного вызова методов .NET Remoting.

Новые конструкции языка MC#: movable-методы, каналы и обработчики.

Ключевая особенность языка Polyphonic C# заключается в добавлении к обычным, синхронным методам, так называемых асихронных методов. Фактически, эти асинхронные методы предназначены играть в программах две основные роли:

1. автономных методов, исполняемых в отдельных потоках и реализующих параллельные части основного алгоритма программы, и
2. методов, предназначенных для доставки данных (возможно, с предварительной обработкой) обычным синхронным методам.

В языке MC#, эти два вида методов выделены, соответственно, в два специальных синтаксических класса:

1. movable-методов и
2. каналов.

В языке Polyphonic C#, вспомогательные асинхронные методы, служащие для доставки данных, обычно определяются с помощью связок совместно с синхронными методами. Последние в языке MC# оформлены в виде еще одного специального синтаксического класса обработчиков канальных сообщений.

Первая ключевая особенность языка MC# состоит в том, что, в общем случае, при вызове movable-метода, все необходимые для этого метода данные, а именно

1. сам объект, с которым связан данный movable-метод, а также
2. аргументы (как объекты, так и скалярные величины) метода только *копируются* (но не перемещаются) на удаленную машину, а все изменения, которые с ними там впоследствии происходят, не переносятся на исходные объекты.

В частности, если копируемый объект обладает каналами и обработчиками, то они также копируются на удаленную машину, становясь "прокси"-механизмами для оригинальных объектов.

Вторая ключевая особенность языка MC# состоит в том, что каналы и обработчики могут передаваться в качестве аргументов при вызове методов (в том числе, и movable-методов) **отдельно** от объектов, которым они принадлежат.

Третья ключевая особенность языка MC# состоит в том, что если каналы и обработчики скопированы на удаленный сайт (под этим общим названием мы будем подразумевать узел кластера или машину Grid-сети) то ли автономно, то ли в составе некоторых объектов, то на этом сайте они становятся "прокси"-объектами – посредниками с исходными каналами и обработчиками. Причем, для прикладного программиста эта подмена скрыта – на удаленном сайте он может пользоваться переданными каналами и обработчиками (а на самом деле, их прокси-объектами) также, как и оригинальными. При этом, все действия над ними переносятся Runtime-системой на исходные каналы и обработчики. Этим каналы и обработчики отличаются от обычных объектов: действия над последними на удаленных сайтах на исходные объекты не переносятся (см. первую ключевую особенность языка MC#).

Программа, названная *all2all*, демонстрирует, как можно организовать взаимодействие среди множества распределенных процессов по принципу "каждый с каждым". Каждый распределенный

процесс будет экземпляром класса *DistributedProcess*, и будет запускаться на удаленном сайте, выбираемом Runtime-системой, путем вызова movable-метода *Start* этого класса. Каждый распределенный процесс будет создавать на своем сайте экземпляр класса *BDChannel* (Bidirectional channel), состоящий из канала *Send* и обработчика *Receive*. Обменявшись объектами *BDChannel* между собой, распределенные процессы могут посылать и принимать сообщения друг от друга, независимо от их физического расположения. Обмен объектами *BDChannel* осуществляется через главный процесс, исполняющийся на машине, где было запущено приложение. Ниже приведен полный текст программы на языке MC#, в которой количество распределенных процессов N задается как входной параметр.

```
class  All2all   {
 public static void Main (string[] args)   {
  int  i;
  int   N = System.Convert.ToInt32 ( args [ 0 ] );    // N is number of
distributed processes
  All2all a2a = new All2all();
  DistribProcess dproc = new DistribProcess();
  //   Launch distributed processes
  for ( i = 0; I < N; i++ )
    dproc.Start ( i, a2a.sendBDC, a2a,sendStop );
  //   Receive BDChannel objects from processes
  BDChannel[] bdchans = new BDChannel [ N ];
  for ( i = 0; I < N; i++ )
    a2a.getBDC ( bdchans );
  //   Send BDChannel array to each process
  for ( i = 0; i < N; i++ )
     bdchans [ i ].Send ( bdchans );
  //   Receive stop signals from processes
  for ( i = 0; i < N; i++ )
    a2a.getStop();
 }
 CHandler getBDC void(BDChannel[] bdchans)   &
           Channel sendBDC  ( int i, BDChannel bdc ){ bdchans [ i ] =
bdc; }
 CHandler  getStop void() & Channel  sendStop() { return; }
}
class   BDChannel   {
  CHandler   Receive object() & Channel Send (object obj )   {return   (
obj );  }
}
class DistribProcess   {
 movable Start ( int   i, Channel   (int, BDChannel) sendBDC, Channel ()
sendStop ) {
  //  i is a process  proper number
  int   j;
  BDChannel  bdc = new BDChannel();
  sendBDC ( i, bdc );
```

```
  BDChannel[]   bdchans  =  (BDChannel[]) bdc.Receive();
  //   Send messages to other processes
  for  ( j = 0; j < bdchans.Size; j++ )
    if  ( j != i )
        bdchans[j].Send ("Message from process " + i +  " to process "
+ j );
  //   Receive messages from other processes
  for  ( j = 0; j < bdchans.Size - 1; j++ )
    Console.WriteLine ( "Process " + i + " : " + (string) bdc.Receive()
);
  //   Send stop signal to the main program
  sendStop();
 }
}
```

Направления развития.

Как упоминалось выше, идейную основу языка MC# составляет join-исчисление – исчисление процессов с высокоуровневым механизмом обработки сообщений, близким к реально используемым в современных компьютерных системах. Поэтому представляет интерес разработка собственной теоретической модели языка MC# в виде формального исчисления и операционной семантики для него. Учитывая объектно-ориентированную природу языка MC#, нами разработано два исчисления:

- - параллельное (локальное) исчисление объектов, и
- - распределенное исчисление объектов (в которое явно включено понятие места исполнения – "сайта").

В этих исчислениях моделируются только специфические конструкции языка MC# - movable-методы, каналы и обработчики, совместно с такими универсальными конструкциями, как оператор параллельной композиции и объявление объектов. Определена точная операционная семантика для этих исчислений на основе формализма "химической абстрактной машины" [3]. Дальнейшее развитие формальных основ языка MC# будет состоять в разработке системы типов для указанных исчислений, что даст возможность проводить более глубокий статический анализ и оптимизацию MC#-программ на этапе компиляции.

Важное направление усовершенствование работы с данными в языке MC#, также, как и в других параллельных языках – введение средств описания распределения данных между процессами, аналогичных средствам языков ZPL и X10. Эти средства обычно делятся на два класса: операции над массивами типа "вырезания"

заданных подмассивов и назначение отдельных частей массива movable-методам с помощью оператора ***distribution***.

Из практических направлений дальнейшего развития системы следует отметить работы по интеграции языка MC# в Visual Studio 2005, а также включение в Runtime-систему возможности назначения порождаемых потоков на определенные процессоры с использованием конструкций *SetThreadAffinityMask* (Windows) и *sched_setaffinity* (Linux), что позволит более эффективно балансировать нагрузку на многоядерном процессоре. Также уже начата работа над созданием MC# Grid System – системы для проведения метакластерных (Grid-) вычислений на основе языка MC# [4].

Литература

1. Saraswat, V.A., Jagadeesan R. Concurrent Clustered Programming - CONCUR 2005, LNCS 3653, Springer, 2005, pp.353-367.
2. 2. Fournet, C., Gonthier, G. The join calculus: a language for distributed mobile programming. - In Proc. Applied Semantics Summer School, 2000. LNCS, Vol.2395, Springer, pp. 268-332.
3. 3. Berry, G., Boudol G.: The Chemical Abstract Machine. – Theoretical Computer Science, 96 (1992), pp. 217 – 248.
4. 4. В.Б.Гузев, Ю.П. Сердюк MC# Grid System: подход к Grid-вычислениям на основе языка параллельного, распределенного программирования // Труды Всероссийской научной конференции "Научный сервис в сети Интернет: технологии распределенных вычислений", 2005 г., стр. 41-43.

Контакты

Yury@serdyuk.botik.ru

РАЗРАБОТКА УЧЕБНО-МЕТОДИЧЕСКОГО КОМПЛЕКСА “ВЫСОКОУРОВНЕВЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ ДЛЯ ПАРАЛЛЕЛЬНЫХ АРХИТЕКТУР” В УНИВЕРСИТЕТЕ Г. ПЕРЕСЛАВЛЯ-ЗАЛЕССКОГО.

Сердюк Ю.П.

Институт программных систем РАН, г.Переславль-Залесский

Для массового внедрения и эффективного применения многоядерных архитектур, необходимы новые языки параллельного

программирования, которые могли бы быть использованы для регулярного программирования миллионами программистов. С другой стороны, эти языки должны быть достаточно прозрачны и формально “чисты”, чтобы для них могли быть разработаны соответствующие интегрированные инструменты для редактирования, компиляции, отладки, анализа производительности и т.д.

В рамках проекта “Высокоуровневые языки программирования для параллельных архитектур”, осуществляемого в Университете г. Переславля-Залесского и поддержанного в рамках Intel по программе Multicore University Program, предлагаются учебные программы с соответствующими методическими материалами, ориентированные на изучение и практическое применение *современных высокоуровневых языков параллельного программирования*. Использование такого рода языков, *с промышленной точки зрения*, позволяет резко поднять производительность и продуктивность работы программистов, привлечь к параллельному программированию значительно большее число профессиональных программистов, а *с образовательной точки зрения*, дает возможность их успешно изучать и осваивать студентам вузов, начиная уже со 2-го курса.

На настоящий момент, преобладающим подходом как при реализации параллельных приложений, так и при обучении основам параллельного программирования, является использование интерфейсов MPI и OpenMP. По своей природе, эти программные интерфейсы являются очень низкоуровневыми, что

1. затрудняет их изучение и освоение (особенно студентами младших курсов),
2. снижает продуктивность работы программистов и надежность разрабатываемых ими программных систем.

Поэтому основной концепцией, закладываемой в данный проект, является переход к изучению **высокоуровневых** языков программирования в качестве основных инструментов параллельного программирования. Основные причины такого перехода заключаются в том, что

- высокоуровневые языки более легки для изучения (в том числе, первоначального) основ параллельного программирования,
- они позволят привлечь к систематическому параллельному программированию большое количество программистов,

которые не занимались им до появления многоядерных процессоров,

- на их основе разрабатываются современные интегрированные среды для параллельного программирования (см., например, проект PERCS (Productive Easy-to-use Reliable Computer Systems) фирмы IBM [http://domino.research.ibm.com/comm/pr.nsf/pages/news.20030710_darpa.html], объединяющий языки X10, UPC и Co-Array Fortran в рамках оболочки Eclipse.

В данном учебном комплексе найдут свое отражение современные концепции параллельных архитектур, таких как NUMA (Non-Uniform Memory Access), и их отображение в методы мультипотокового программирования и распределения данных в рамках указанных выше языков.

Учебные программы данного комплекса охватывают две группы языков программирования:

1. архитектурно-зависимые языки программирования на основе модели “распределенного глобального адресного пространства” (PGAS – partitioned global address space),
2. архитектурно-независимые языки параллельного, распределенного программирования.

В качестве представителей первой группы языков, студентам предлагается изучить языки UPC (Unified Parallel C), разработанный в Университете Дж. Вашингтона, США [http://upc.gwu.edu/] и X10, разработанный в Исследовательском центре Т.Дж. Уотсона фирмы IBM [http://domino.research.ibm.com/comm/research_projects.nsf/pages/x10.index.html]. Здесь же будут даны краткие сведения об языке Co-Array Fortran [http://www.co-array.org/] и программном интерфейсе OpenMP [http://www.openmp.org/drupal/].

Очень важную роль в повышении продуктивности работы программистов, разрабатывающих параллельные приложения, играют архитектурно-независимые языки, устраняющие необходимость явного управления потоками вычислений, присущую некоторым из вышеупомянутых языков, и, тем более, обязательную в случаях использования интерфейсов MPI и OpenMP. Кроме того, в

архитектурно-независимых языках предлагаются простые методы пересылки данных, не требующие ручного программирования по их сериализации/десериализации.

В качестве представителя этой группы языков, студентам предлагается изучить язык MC#, разработанный в Институте программных систем РАН, г. Переславль-Залесский [http://u-pereslavl.botik.ru/~vadim/MCSharp/index.ru.php]. Поскольку язык MC# основывается, в частности, на механизме передачи сообщений, то здесь же предполагается дать студентам основы низкоуровневого программирования на базе MPI и провести сравнение всех языков, включая языки из первой группы.

Поскольку каждый из вышеупомянутых языков имеет практические реализации, то учебные программы предусматривают практические занятия по программированию на многоядерных системах и кластерах. В качестве практических заданий на параллелизацию будут использоваться хорошо известные задачи, такие как сортировка, задачи линейной алгебры, метод статистических испытаний Монте-Карло, рендеринг изображений на основе трассировки лучей, поиск в Интернет, алгоритм Смита-Уотермена сравнения биологических последовательностей и др.

Для методического обеспечения предлагаемого учебного комплекса, предполагается разработка 3-ех учебных пособий:

1. Параллельное программирование на основе модели “распределенного глобального адресного пространства” (конспект лекций).
2. Введение в параллельное программирование на языке MC#.
3. Методические указания к практическим занятиям по курсу “Высокоуровневые языки программирования для параллельных архитектур”.

Ниже представлена программа лекционного курса “Высокоуровневые языки программирования для параллельных архитектур”. Программа представляет собой семестровый курс (16 учебных недель), ориентированный на студентов 2-3-их курсов, знакомых с основами алгоритмизации и программирования на языках C/C++/C#, Java.

Введение: параллельные архитектуры и языки программирования.

1. Основные типы параллельных архитектур.

Потоки – основное средство организации параллельных вычислений.

- Создание потоков
- Взаимодействие потоков:
  - - на основе глобального адресного пространства,
  - - на основе передачи сообщений.
- Методы синхронизации параллельных потоков.

Мультипотоковое программирование в языках C# и Java.

Программирование на основе модели "распределенного глобального адресного пространства".

1. Язык программирования UPC (Unified Parallel C)

- Модель исполнения программ на языке UPC
- Модель памяти, используемая в языке UPC:
  - Разделяемые массивы и скалярные данные
  - Методы распределения данных между потоками
  - Разделяемые и локальные данные
- Указатели в языке UPC
  - специальные функции и операторы, связанные с потоками и разделяемой памятью: upc_threadof, upc_phaseof, upc_localsizeof, upc_blocksizeof и др.
- Разделение работ между потоками с помощью конструкции upc_forall
- Динамическое размещение памяти в языке UPC
- Коллективные операции в языке UPC
- Примеры программирования на UPC:
  - Перемножение матриц
  - Выделение линий на изображении
- Методы синхронизации в UPC:
  - Барьеры (upc_barrier)
  - Запирания (upc_lock)
  - Ограничители (upc_fence)
  - Контроль непротиворечивости памяти (upc_strict и upc_relaxed)
- Сведения о практической реализации языка UPC

2. Язык программирования X10

- Модель программирования на языке X10: места, активности, разделяемые массивы
- Понятие “место” и “активность” в языке X10:
  - Конструкции работы с местами here, next, prev
  - Порождение и завершение активностей: конструкции async, finish
- Атомарные блоки:
  - - Конструкции atomic и when
- Асинхронные выражения:
  - Конструкции future и force
- Распределение данных по местам:
  - Массивы, регионы и распределения
  - Конструкции for, foreach, ateach
- Синхронизация в языке X10
  - Конструкция clock
- Примеры программирования на языке X10:
  - Метод Монте-Карло
  - Программа RandomAccess (работа с разделяемыми массивами)
- Сведения о практической реализации языка X10

Сведения об языке Co-Array Fortran и программном интерфейсе OpenMP

Архитектурно-независимые языки параллельного, распределенного программирования

1. Язык программирования MC#

- Модель программирования языка MC#: movable-методы, каналы, обработчики, связки
- Movable-методы:
  - Правила определения
  - Особенности использования
- Каналы и обработчики канальных сообщений:
  - - Правила определения каналов и обработчиков
  - - Правила работы Runtime-системы с какналами и обработчиками

- Синхронизация в языке MC#:
  - Определение и использование связок (chords)
- Примеры программирования на языке MC#:
  - Обход двоичного дерева
  - Нахождение простых чисел методом решета Эратосфена
  - Простой рендеринг изображений методом трассировки лучей
  - Вычисление частичных сумм массива
- Сведения о практической реализации языка MC#

Сведения о программном интерфейсе MPI

1. Сравнение высокоуровневых языков параллельного программирования

Контакты

Yury@serdyuk.botik.ru

КОМПЛЕКСНАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА ФАКУЛЬТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ УДГУ

Сивков Д. А. , Исламов Г. Г.

УдГУ, Ижевск

Цель реализации образовательной программы.

Целью реализации образовательной программы «Учебный Центр подготовки специалистов для Удмуртской Республики в области современных информационных технологий» является существенное увеличение эффективности предприятий и повышение доходности за счет использования новейших информационных технологий, обеспечение непрерывных темпов роста, обеспечение потребности научных, производственных, образовательных структур и бизнеса Удмуртской Республики в высококвалифицированных IT-специалистах.

Задачи, реализуемые в рамках образовательной программы проекта.

1. Анализ потребностей и бизнес-ситуации в IT-области Удмуртской Республики;

2. Создание эффективной управляющей инфраструктуры Учебного центра, подготовка учредительных документов;
3. Создание группы анализа и классификации доступной открытой информации;
4. Создание групп внедрения и работы с заказчиками и спонсорами, контактной группы с ведущими мировыми компаниями;
5. Создание групп подготовки и разработки методических материалов;
6. Создание высокотехнологичной площадки для обеспечения деятельности Учебного Центра;
7. Интеграция информационно-вычислительных ресурсов Учебного Центра в международное grid-пространство.

Формы реализации образовательной программы

1. Реорганизация в течение полугода исследовательских групп ВУЗа с целью получения динамичной команды управления прикладными проектами Учебного Центра;
2. В течение полугода подготовить специалистов Учебного Центра в области управления проектами;
3. В течение первого квартала создать группу поиска, мониторинга и классификации доступной открытой информации;
4. Подготовить в течение полугода специалистов в области разработки и внедрения проектов;
5. Создать в течение года группу технического консультирования;
6. В течение первого квартала создать высокотехнологичную площадку обеспечения деятельности Учебного Центра;
7. В течение полугода интегрировать созданную площадку в международное распределенное grid-пространство;
8. Во втором полугодии подготовить образовательный пилотный проект.

Участники реализации образовательной программы

Факультет информационных технологий и вычислительной техники

Ожидаемые результаты реализации образовательной программы

Учебный Центр, обеспечивающий IT-специалистами потребности и нужды Удмуртской Республики. Высококлассная команда управления, позволяющая масштабировать и реплицировать накопленный опыт в другие региона России.

В результате выполнения программы ожидается увеличение эффективности предприятий и повышение доходности за счет использования новейших информационных технологий, обеспечение непрерывных

Контакты
Д. А. Сивков: dimasiv@mail.ru

РЕАЛИЗАЦИЯ СХЕМЫ ОСТАНОВКИ И ВОЗОБНОВЛЕНИЯ СЧЕТА В ПАРАЛЛЕЛЬНОМ ИНДЕКСНОМ МЕТОДЕ ПОИСКА ГЛОБАЛЬНО-ОПТИМАЛЬНЫХ РЕШЕНИЙ

Сидоров С.В., Сысоев А.В.
ННГУ, Нижний Новгород

В настоящей работе рассматривается подход к реализации схемы остановки и продолжения счета в параллельной версии индексного метода [1, 2], реализованного в программной системе “Абсолют Эксперт” [3, 5].

Необходимость остановки счета

В настоящей работе рассматривается расширении функциональности параллельного индексного метода с одной разверткой, который является частью набора методов системы поиска глобально-оптимальных решений “Абсолют Эксперт”. Указанный алгоритм представляет собой итерационную схему, в которой выбор следующей точки итерации происходит на основе информации накопленной на всех предыдущих итерациях. Таким образом, под остановкой счета понимается остановка итерационного алгоритма с сохранением всей накопленной информации о решении задачи, а так же с сохранением исходных данных о решаемой задаче. Все данные после остановки счета располагаются во внешней памяти и могут быть

перенесены с одного вычислительного узла на другой с возможностью продолжения счета на нем. При возобновлении счета происходит загрузка всех данных о решаемой задаче с возможным некоторым изменением критериев задачи и продолжение выполнения ранее остановленных итераций.

Необходимость остановки счета обусловлена рядом факторов.

Наличие такой возможности влияет на эффективность алгоритма поиска, обеспечивая возможность при решении реальных прикладных задач прекратить в определенный момент счет с получением текущей оценки глобального минимума. В дальнейшем можно использовать полученную оценку, либо сделать послабление в границе области поиска и продолжить нахождение новой оценки.

Возможность остановки и возобновления счета серьезно расширяет функциональность алгоритма, позволяя начать вычисления на одном вычислительном узле, потом при необходимости прекратить вычисления с сохранением всей накопленной информации, и продолжить вычисления на этом или другом узле через любой промежуток времени. Таким образом, в случае окончания квоты на использования вычислительных ресурсов можно прекратить вычисления и продолжить их при появлении таких ресурсов или на других ресурсах.

Описание схем остановки и возобновления

Первоначально рассмотрим стандартный процесс прекращения вычислений алгоритма при выполнении одного из критериев остановки: точность или число выполненных итераций.

В случае прекращения вычислений происходит выход алгоритма из цикла итераций и вывод результатов о текущей найденной оценке глобального минимума.

В случае остановки алгоритма происходит дополнительное сохранение во внешней памяти всех параметров решаемой задачи, а также всей накопленной поисковой информации.

Необходимо отметить, что общая схема сохранения параметров для последовательного и параллельного алгоритмов существенно отличается, так как в зависимости от реализации параллельного алгоритма возможна ситуация распределения всего набора поисковой информации между вычислительными узлами. В этом случае необходимо выполнить первоначальную аккумуляцию поисковой информации на одном узле, а потом выполнить на нем сохранение.

Примером метода с распределенной поисковой информацией является параллельный индексный с множественной разверткой.

В данной работе рассматривается параллельная реализация индексного метода с одной разверткой, которая предполагает дублирование поисковой информации на вычислительных узлах, поэтому после остановки вычислений необходимо только сохранить всю накопленную информацию и параметры решаемой задачи одним из вычислительных узлов.

Сохраняется поисковая информация, включая состояние матрицы поиска и текущую оценку константы Липшица. Также сохраняются следующие параметры решаемой задачи и метода:

- идентификатор объекта оптимизации;
- число задач оптимизации, использующих данный объект;
- параметры всех задач оптимизации, такие как область поиска, список дискретных параметров;
- число ограничений;
- свойства метода: точность, максимальное число итераций, точность развертки, параметр ε - резервирования и т.д.

Необходимо отметить, что возобновление после остановки счета параллельного метода серьезно отличается от возобновления последовательного метода, так как в последовательном методе необходимо только произвести восстановление поисковой информации и информации о решаемой задаче, а дальше запустить цикл итераций.

Особенностью возобновления счета параллельного метода является то, что вся сохраненная информация сосредоточенна только на одном узле, поэтому необходимо после восстановления условия задачи, а также поисковой информации осуществить пересылку всей этой информации всем остальным узлам и произвести восстановление параметров для продолжения счета.

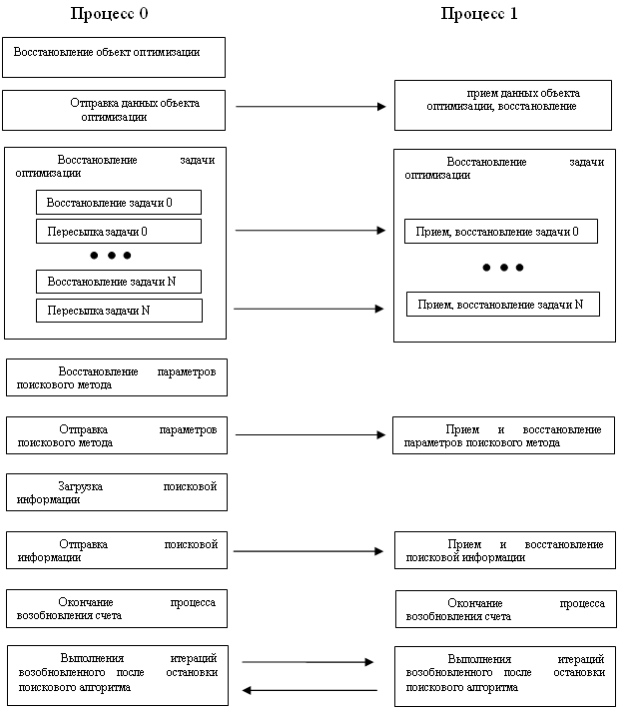
Реализация

Описанная выше схема остановки и возобновления счета на данный момент реализована в параллельном индексном методе с одной разверткой в рамках системы “Абсолют Эксперт”. Сохранение и восстановление отдельных компонент выполнено в качестве методов соответствующих классов.

Реализация сохранения всех параметров решаемой задачи является относительно простым процессом в силу специфики поискового метода, обусловленной дублированием поисковой информации между вычислительными узлами. Таким образом, процесс сохранения представляет последовательный вызов методов по сохранению компонент поискового процесса.

Более высокий интерес представляет реализация возобновления вычисления параллельного поискового алгоритма. Ниже приведена схема реализации восстановления параллельного индексного метода с одной разверткой.

В данной схеме предполагается, что вся сохраненная информация содержится на процессе с номером 0, и возобновляются вычисления для двух процессов.



Результаты экспериментов

Для проверки корректности выполнения процесса остановки и возобновления счета параллельного индексного метода с одной разверткой была проведена серия экспериментов, заключающаяся в сравнении результатов работы алгоритма остановившегося и возобновившего счет после остановки и алгоритмом, который выполнил такое же количество итераций, но не останавливался. Таким образом, считалось, что в ситуации совпадения результатов приостановление и возобновление прошли корректно.

Такого рода эксперименты были проведены на функциях Гришагина 97 и 98. Эксперименты подтвердили корректность работы представленного алгоритма.

Настоящая работа поддержана грантами Netherlands Organization for Scientific Research (NWO) № 047.016.014 и Российского Фонда Фундаментальных Исследований (РФФИ) № 04-01-89002-НВО_а.

Литература

1. Р.Г. Стронгин. “Поиск глобального оптимума”, “Математика и киберненика”, – Знание, 2, 1990.
2. R.G. Strongin, Ya.D. Sergeev. “Global optimization with non-convex constraints: Sequential and parallel algorithms”, – Kluwer Academic Publisher, Dordrecht, 2000.
3. A.V. Sysoyev. “Program system of parallel computations for solving the tasks of global-optimum choice” // VI International Congress on Mathematical Modeling/Book of abstracts, 2004, p. 62.
4. А.В. Сысоев, С.В. Сидоров. “Использование чисел расширенной точности в реализации индексного метода поиска глобально-оптимальных решений” // Материалы пятого Международного научно-практического семинара “Высокопроизводительные параллельные вычисления на кластерных системах”, 2005, стр. 208-215.
5. В.П. Гергель, А.В. Сысоев. “О реализации параллельной версии индексного метода поиска глобально-оптимальных решений в многомерных задачах оптимизации в программном комплексе “Абсолют Эксперт” // Труды всероссийской конференции “Научный сервис в сети Интернет: технологии параллельного программирования”, 2006, с. 115-118.

ТЕСТИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ ТГУ И ИОА СО РАН

Смирнов И.Е.
ТГУ, Томск

Введение

Сегодня можно говорить о том, что кластерные системы успешно применяются для всех задач суперкомпьютинга — от расчетов для науки и промышленности до управления базами данных. Практически любые приложения, требующие высокопроизводительных вычислений, имеют сейчас параллельные версии, которые позволяют разбивать задачу на фрагменты и обсчитывать ее параллельно на многих узлах кластера. Например, для инженерных расчетов (прочностные расчеты, аэромеханика, гидро- и газодинамика) традиционно применяются так называемые сеточные методы, когда область вычислений разбивается на ячейки, каждая из которых становится отдельной единицей вычислений. Эти ячейки обсчитываются независимо на разных узлах кластера, а для получения общей картины на каждом шаге вычислений происходит обмен данными, распространенными в пограничных областях.

Для практических расчетов (3D-анимация, крэш-тесты, разведка нефтяных и газовых месторождений, прогнозирование погоды) обычно используются кластеры из 10-200 узлов. При этом основная задача —эффективная работа кластера с конкретным приложением. Архитектура кластера должна обеспечивать масштабируемость ПО при увеличении количества узлов, т. е. прирост производительности при добавлении новых вычислительных модулей. Для этого важно правильно выбрать конфигурацию кластера в зависимости от профиля обмена данными между экземплярами программы, запущенными на разных узлах. Здесь нужно учитывать общий объем пересылаемых данных, распределение длин сообщений, использование групповых операций и т. п.

Цель работы

Целью работы является тестирование вычислительных кластеров Томского государственного университета и Института оптики и

атмосферы СО РАН на производительность. Показать результаты проверки и сравнить их.

Технология достижения реализации цели

Использованы тестовые программы, разработанные в НИВЦ МГУ [1], которые тестируют эффективность среды (сети) передачи данных между процессорами (узлами кластера):

- **TRANSFER** - тест уровня латентности и скорости пересылок между двумя узлами;
- **NETTEST** - тест пропускной способности сети при сложных обменах по различным логическим топологиям.

Тест производительности межпроцессорных обменов

Назначение теста **TRANSFER** состоит в том, чтобы измерять пиковые характеристики обмена данными между двумя узлами - латентность и пропускную способность.

Понятия пропускной способности и латентности

Предположим, что на двух процессорах вычислительной системы работают два процесса, между которыми с помощью сети пересылаются сообщения. В передаче информации, кроме аппаратных устройств, участвует и ПО, например протокольный стек и реализация интерфейса передачи сообщений MPI.

Основными характеристиками быстродействия сети являются латентность и пропускная способность.

Пропускная способность R сети - количество информации, передаваемой между узлами сети в единицу времени (байт в секунду). Реальная пропускная способность снижается программным обеспечением за счет передачи разного рода служебной информации.

Латентностью (задержкой) называется время, затрачиваемое программным обеспечением и устройствами сети на подготовку к передаче информации по данному каналу. Полная латентность складывается из программной и аппаратной составляющих.

Различают следующие виды пропускной способности сети:

- пропускная способность однонаправленных пересылок ("точка-точка", uni-directional bandwidth), равная максимальной скорости, с которой процесс на одном узле может передавать данные другому процессу на другом узле.

- пропускная способность двунаправленных пересылок (bi-directional bandwidth), равная максимальной скорости, с которой два процесса могут одновременно обмениваться данными по сети.

Методика измерения пропускной способности и латентности

Для измерения пропускной способности "точка-точка" используется следующая методика. Процесс с номером 0 посылает процессу с номером 1 сообщение длины L байт. Процесс 1, приняв сообщение от процесса 0, посылает ему ответное сообщение той же длины. Используются блокирующие вызовы MPI. Эти действия повторяются N раз с целью минимизировать погрешность за счет усреднения. Процесс 0 измеряет время T, затраченное на все эти обмены. Пропускная способность R определяется по формуле R=2NL/T (также определяется пропускная способность двунаправленных обменов).

В этом случае используются неблокирующие вызовы MPI. При этом производится измерение времени, затраченного процессом 0 на передачу сообщения процессу 1 и прием ответа от него, при условии, что процессы начинают передачу информации одновременно после точки синхронизации.

Латентность измеряется как время, необходимое на передачу сигнала или сообщения нулевой длины. Для снижения влияния погрешности и низкого разрешения системного таймера важно повторить операцию посылки сигнала и получения ответа большое число раз. Таким образом, если время на N итераций пересылки сообщений нулевой длины туда и обратно составило T сек., то латентность измеряется как s=T/(2N).

Тест пропускной способности сети при сложных обменах

NETTEST - тест коммуникационной производительности при сложных обменах между несколькими узлами в различных логических топологиях ("звезда", "полный граф", "кольцо"). Тест может использоваться для проверки "выживаемости" сетевого оборудования при пиковых нагрузках и для определения пиковой пропускной способности коммутатора.

Данные методики тестирования обобщаются для случая нескольких процессов. При этом рассматриваются три логические топологии взаимодействия процессов:

- Звезда (Star), т.е. взаимодействие основного процесса с подчиненными;
- Кольцо (Ring), т.е. взаимодействие каждого процесса со следующим;
- Полный граф (Chaos), т.е. взаимодействие каждого процесса с каждым.

Под логическим каналом будем подразумевать неупорядоченную пару узлов (A,B), которые в данной топологии могут обмениваться сообщениями. Логическая топология полностью определяется множеством задействованных логических каналов.

Для каждой топологии рассматриваются однонаправленные и двунаправленные обмены. По каждому логическому каналу между узлами A и B информация в ходе теста передается в обе стороны: от узла A к узлу B и обратно передается по L байт информации. Однако в случае однонаправленных обменов один из узлов, например B, ждет получения сообщения от A, и только тогда может передавать A свое сообщение. Во втором же случае информация может передаваться в обе стороны одновременно.

Предполагается, что из всех, соответствующих по семантике данной топологии и способу обменов будет выбран вариант с наименьшими накладными расходами.

Технические характеристики кластера ТГУ

Кластер состоит из 9 двухпроцессорных копьютеров на базе процессоров Intel Pentium III 650 МГц. Один компьютер является сервером, на котором установлено 512 Мб оперативной памяти и два жестких диска объемом по 18 Гб. На сервере происходит компиляция и запуск программ, а также хранятся домашние каталоги пользователей. Все остальные компьютеры являются узловыми машинами и используются для проведения параллельных вычислений. На них установлено по 256 Мб ОЗУ. Сервер и узловые машины объединены в локальную вычислительную сеть 100Мб Ethernet с помощью свича Cisco Catalyst 2924 [2].

Общие технические характеристики:

- 18 процессоров Pentium III 650
- 2.5 Гб ОЗУ
- 36 Гб дискового пространства

- Реальная производительность на тесте LINPACK - 5.5 Gflops, пиковая – 11 Gflops

Программное обеспечение:
- Операционная система Linux Red Hat
- Кластерный пакет LAM MPI

Технические характеристики кластера ИОА СО РАН

Кластер состоит из сервера, который используется для хранения личных каталогов пользователей, компиляции заданий и удаленной загрузки вычислительных узлов, и 9-ти бездисковых вычислительных узлов. Сервер и вычислительные узлы построены на базе серверных плат Intel STL2 "Tupelo". Коммуникационная подсистема реализована на сетевых адаптерах Intel Pro/1000 server adapter и коммутаторе фирмы 3Com SuperStack 3 Switch 4900 [3].

Технические характеристики кластера:
- 20 процессоров PIII с частотой 1ГГц
- 10 Гб оперативной памяти
- коммуникационная сеть Gigabit Ethernet
- операционная система RedHat Linux
- кластерное программное обеспечение - MPICH 1.2.2 и/или Mosix
- Производительность на тесте LINPACK - 10,5 Gflops

Результаты

В результате запуска тестов на исследуемых кластерах мы получили следующее:

Результаты тестов TRANSFER и NETTEST, полученные на кластере ТГУ:

TRANSFER (MB/sec)

Size(K)	n1	n2	n3
1K	6.63	6.664	6.656
2K	8.813	8.833	8.805
4K	12.89	12.89	12.9
8K	15.67	15.64	15.67
16K	17.88	17.96	18.03
32K	19.59	19.58	19.57
64K	20.33	20.33	20.33
128K	11.4	10.12	10.87
256K	11.57	11.03	11.03
512K	11.66	11.12	11.12
1024K	12.97	11.17	11.17

Латентность 34 микросекунды

NETTEST (MB/sec)

Size,K	Star	Star2	Chaos	Chaos2	Ring	Ring2
1	8.05	15.57	15.61	23.35	14.12	21.06
2	9.99	19.28	18.58	26.72	19.01	25.42
4	12.70	21.78	22.55	31.84	26.77	32.75
8	14.24	21.63	23.99	34.28	32.85	36.65
16	15.27	21.66	24.97	35.80	37.25	39.30

Best network throughput values in MB/sec

Star	Star2	Chaos	Chaos2	Ring	Ring2
15.27	21.79	24.97	35.80	37.25	39.40

Nodes: cluster.tsu.ru node1 node2 node3

Результаты тестов Transfer и Nettest, полученные на кластере ИОА СО РАН:

TRANSFER (MB/sec)

Size(K)	n1	n2	n3
1K	23.07	22.8	22.93
2K	33.34	33.14	33.1
4K	54.12	51.03	54.32
8K	59.87	60.38	59.5
16K	57.25	56.22	56.66
32K	56.24	55.18	56.81
64K	49.75	48.08	36.33
128K	47.59	46.92	47.06
256K	43.52	45.12	44.17
512K	43.58	43.08	42.33
1024K	46.84	46.78	45.42

Латентность 24 микросекунды

NETTEST (MB/sec)

Size,K	Star	Star2	Chaos	Chaos2	Ring	Ring2
1	27.60	40.14	51.94	83.86	45.35	69.93
2	38.96	46.11	73.71	92.47	64.73	89.96
4	54.87	57.31	105.31	117.4	99.35	119.60
8	49.66	50.13	58.11	100.07	88.51	117.16
16	38.48	41.16	68.03	77.76	79.65	91.59

Best network throughput values in MB/sec

Star	Star2	Chaos	Chaos2	Ring	Ring2
54.87	57.31	105.31	117.45	99.35	119.60

Nodes: server ws1 ws2 ws3

Заключение

Тестирование вычислительных кластеров было произведено с помощью тестовых программ. Полученные результаты показывают, что наилучшая пропускная способность сети для вычислительного кластера ТГУ для 4 использованных процессоров достигается на топологии "кольцо" при двунаправленных обменах и составляет примерно 40 Мбайт/сек. Для кластера ИОА СО РАН для 4 процессоров достигается наивысшая скорость передачи данных на той же топологии, также при двунаправленных обменах, но составляет примерно 120 Мбайт/сек. Латентность рассмотренных вычислительных кластеров отличается на десяток микросекунд. Вычислительный кластер ТГУ существенно уступает вычислительному кластеру ИОА СО РАН. Также это неравенство можно проследить по техническим характеристикам суперкомпьютеров. Для дальнейшего обучения студентов и решения более глобальных задач на кластере ТГУ необходимо произвести обновление комплектующих кластера.

Литература

1. PARALLEL.RU. Система тестов производительности для параллельных компьютеров [Электронный ресурс]: Тесты.- Режим доступа: http://parallel.ru/testmpi/, свободный.
2. Старченко А.В., Есаулов А.О. Параллельные вычисления на многопроцессорных вычислительных системах.- Томск: Изд-во Том. ун-та, 2002.-56с.
3. http://www.iao.ru/ru/resources/cluster

РАСПРЕДЕЛЕННЫЙ РЕНДЕРИНГ ФИЛЬМОВ В 3DS MAX

Соколов А.Ю., Гребенщиков А.В.
ВятГУ, Киров

Задача рендеринга фильмов в 3DS MAX

Рендеринг даже небольших фильмов, разрабатываемых с помощью 3DS MAX, является очень длительной операцией. Фильм среднего качества длительностью полминуты может обрабатываться около суток. При этом рендеринг каждого отдельного кадра занимает время порядка нескольких минут и может быть выполнен независимо от остальных кадров. Следовательно, для сокращения времени вычислений возможно распараллеливание данного процесса. С этой целью в пакет 3DS MAX входит специальная программа 3dsmaxcmd.exe, запускаемая с параметрами из командной строки. Используя эти параметры можно задать размеры и формат получаемых изображений и указать номера кадров, изображения которых необходимо получить. Для сокращения трафика при пересылке рисунков пользователю необходимо использовать архиватор.

Для решения данной задачи была использована среда GPE. Был разработан гридбин, обеспечивающий взаимодействие пользователя с распределенной средой.

Используемые приложения:

- непосредственно для рендеринга 3DS MAX 7 с плагином VRay Adv 1.48.03
- для сжатия полученных файлов рисунков PKZIP 2.50

Отзывы о среде GPE

Главный минус - это отсутствие автоматической диспетчеризации. Пользователю приходится самостоятельно обновлять информацию о состоянии своих задач и подбирать свободную целевую систему.

При освоении GPE большой проблемой для нас было недостаточное количество документации.

ТОЧНЫЙ МЕТОД ПОСТРОЕНИЯ СТАТИЧЕСКОГО ОТОБРАЖЕНИЯ СРЕДНЕГРАНУЛЯРНЫХ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА ГЕТЕРОГЕННУЮ ПАРАЛЛЕЛЬНУЮ ПЛАТФОРМУ

Сыщиков А.Ю., Шейнин Ю.Е.
Санкт-Петербургский Университет Аэрокосмического Приборостроения, Санкт-Петербург

The mapping problem

В области параллельных вычислений под «отображением» понимают отображение параллельного вычисления на параллельную вычислительную платформу. В общем случае, в проблеме отображения можно выделить два компонента:

- размещение; размещение компонентов параллельной программы (операторы, объекты-данные) на модули параллельной вычислительной системы (Процессоры, Вычислительные Модули, Вычислительные Машины);
- построение расписаний; построение расписаний выполнения операторов параллельной программы (задач, процессов) на тех модулях вычислительной машины, на которую они размещены.

Решение задачи размещения определяет, как будут размещены процессы параллельной программы на вычислительных модулях. Решение задачи построения расписания определяет, когда эти размещенные процессы должны быть запущены для того, чтобы получить максимально оптимизированное выполнение программы.

Обычно две эти задачи решаются в рамках одной задачи отображения. В частных методах и алгоритмах отображения могут описываться как обе части, так и какая-нибудь одна. При этом подразумевается, что вторая часть задачи уже некоторым образом решена.

Цель задачи отображения – оптимизация выполнения параллельной программы относительно определенных критериев.

В общем виде, существует два вида задачи мапирования: статическая (обычно решаемая на этапе компиляции и компоновки программы) и динамическая (решаемая диспетчером во время работы параллельной программы на вычислительной системе). one. Мы будет рассматривать задачу статического отображения.

При статическом отображении предполагается, что мы знаем такие характеристики параллельной программы [1,2], как: времена выполнения процессов, зависимость по данным между процессами, требования к синхронизации и т.д.

Классические методы решения задач отображения также разделяются на два класса: точные алгоритмы и эвристические алгоритмы.

Точные алгоритмы дают гарантированно оптимальное решение, однако их сложность может быть слишком велика для задач реального размера. Цель не точных, эвристических методов – получение достаточно хорошего решения за разумное время. В данной статье мы выделяем для рассмотрения класс точных алгоритмов.

Точные алгоритмы

Точные алгоритмы могут использоваться, когда пространство конфигураций достаточно мало. Например, несколько процессов должно быть размещено на небольшом количестве вычислительных модулей.

Точные алгоритмы решения задачи мапирования имеют экспоненциальную сложность. Большинство из них вытекают из задач искусственного интеллекта или операций исследований, таких как исследование дерева поиска: алгоритм ветвей и границ, поиск в ширину, просмотр лучших вариантов, алгоритм Дейкстры, A* и т.д. [3,4]. При разработке точных алгоритмов основной сложностью является снижение пространства решений путем отсекания заведомо не оптимальных решений.

Существующие точные алгоритмы мапирования обычно разрабатывались при ряде допущений относительно структуры графа программы или модели вычислительной платформы [3, 4, etc.]. Например, чаще всего существующие алгоритмы имеют следующие допущения:

- Вершины графа параллельной программы имеют одинаковое время выполнения: $\forall i: comp(t_i) = 1$
- Коммуникации между процессами не учитываются: $\forall i,j$ $data(t_i, t_j) = 0$.
- Вычислительная платформа является гомогенной: $\forall p$ $perform(p)=1$.

Кроме того, ряд алгоритмов разработан для специального подкласса графов программ, таких как свободные деревья.

Какие задачи нужно решать нам

Тем не менее, при всей вычислительной сложности, для определенного класса задач точные алгоритмы являются необходимым методом решения задачи отображения.

Точный алгоритм необходим при разработке эвристических алгоритмов для получения «референсных» результатов решения задачи мапирования для оценки эффективности разрабатываемых эвристических алгоритмов. Точные алгоритмы могут применяться при решении задачи мапирования для параллельных программ и вычислительных систем с жесткими требованиями к производительности либо временным характеристикам работы системы в условиях ограниченного набора аппаратных ресурсов (вычислительных модулей, производительности каналов связи и т.д.). Применение точного алгоритма решения задачи отображения, когда затраты на нахождение решения многократно превышают время решения самой задачи оправдано, например, для аппаратной реализации некоторого алгоритма: когда решить задачу отображения можно один раз в условиях практически неограниченных вычислительных ресурсов, в результате чего будут сэкономлены аппаратные средства и уменьшено время работы многократны выполняющегося впоследствии вычисления.

Существующие точные алгоритмы решают задачу отображения лишь для некоторых частных случаев характеристик параллельной программы и вычислительной системы. Для использования в реальных системах необходимо, чтобы алгоритм решения задачи отображения учитывал:

- Различное время выполнения процессов;
- Гетерогенность вычислительной платформы: один и тот же процесс может выполняться на различных вычислительных модулях различное время или даже на каких-то вычислительных модулях не может быть выполнен вообще;
- Взаимосвязь по данным между процессами, объемы передаваемых данных;
- Взаимосвязь между вычислительными модулями вычислительной системы, пропускную способность каналов связи и т.д.

Предлагаемый точный алгоритм

Основной метод работы предлагаемого алгоритма, как всех других точных алгоритмов – полный перебор вариантов. Специфика предлагаемого алгоритма состоит в том, что он принимает во внимание все обозначенные характеристики параллельных программ и вычислительных систем, в отличие от существующих алгоритмов. Кроме того, для данного алгоритма предлагается ряд оптимизаций, позволяющих существенным образом уменьшить количество просматриваемых вариантов, что позволяет существенно увеличить скорость нахождения оптимального решения задачи отображения.

Отображения графа программы осуществляется с помощью следующих ключевых шагов:

1. Построение дерева последовательностей. В программах с параллельными ветвями будет несколько возможных последовательностей рассмотрения процессов. Чтобы гарантировать, что оптимальный результат не будет пропущен, мы должны просмотреть все допустимые последовательности отображения.
2. Размещение процессов и построение расписания. Для каждой возможной последовательности отображения необходимо построить все возможные расписания.
3. Оценка построенных отображений с помощью стоимостной функции относительно заданных критериев задачи отображения. Выбор отображения с минимальной стоимостью: оптимального отображения.

Параллельная программа должна быть представлена в виде направленного ациклического графа параллельных процессов.

Построение дерева последовательностей.

Необходимо построить набор всех возможных последовательностей, в которых мы можем рассматривать процессы при построении размещений и расписаний.

Дерево последовательностей процессов (ДПП) – это дерево, позволяющее просматривать все последовательности. Листьями данного дерева являются наборы последовательностей, в которых процессы могут рассматриваться при построении размещений и расписаний. Последовательность процессов определяется зависимостью по данным между процессами для зависимых процессов

и перебором всех возможных вариантов для процессов, которые могут выполняться параллельно.

Количество ветвлений на каждом этапе соответствует количеству процессов, которых не имеют «предшественников» или все «предшественники» которых уже занесены в последовательность на предыдущем уровне.

Пример дерева последовательностей:

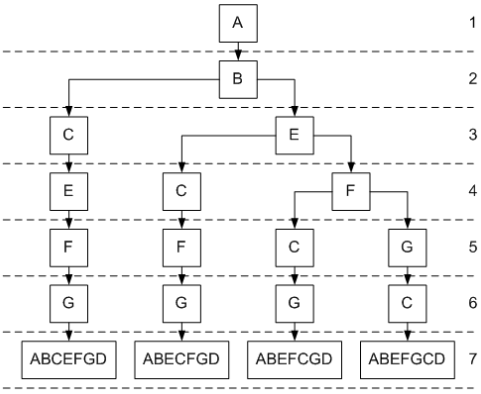


Рис. 1. Дерево последовательностей процессов.

Предполагаемая сложность (количество вариантов):

$$C_{ord} = \frac{(m_1 + m_2 + \ldots + m_n)!}{m_1! m_2! \ldots m_n!}, \text{ где}$$

m*i* – количество процессов в параллельной ветви,

n – количество ветвей в параллельном регионе.

Построение размещения и расписания процессов

Необходимо построить дерево возможных отображений параллельной программы на параллельную платформу для каждого варианта последовательности. Листьями дерева отображения (ДО) являются наборы возможных расписаний выполнения параллельной программы.

Число ветвлений на каждом этапе соответствует количеству вычислительных модулей, где размещаемый процесс может быть выполнен.

Матрица времен выполнений процессов на вычислительных модулях на данном этапе используется лишь для отсечения невозможных вариантов размещения. На данном этапе для отсечения лишних вариантов также может быть использована информация о максимальном количестве процессов, которые могут быть размещены на вычислительном модуле, если такие ограничения имеются.

В расписании время запуска процесса определяется как T_{es}, то есть как максимальное время из:

- Время освобождение вычислительного модуля, на который размещен процесс;
- Максимальное время завершения всех предшественников данного процесса плюс время передачи данных от предшественников, размещенных на других вычислительных модулях.

Формально:

$$T_{es}(t,p) = \max\{T_{ready}(t,p), avail(p)\},$$

где *avail(p)* – минимальное время, когда вычислительный модуль *p* будет готов выполнить процесс,

$$T_{ready}(t,p) = \max_{t_j \in pred(t)} \begin{pmatrix} schedule(t_j) + \\ + T_{exec}(t_j, map(t_j)) + \\ + T_{comm}(t_j, t, map(t_j), p) \end{pmatrix},$$

где *t* – выполняемый процесс, *p* – вычислительный модуль, где процесс *t* должен быть выполнен.

Выбор оптимального отображения относительно заданных критериев

Результаты, полученные по итогам первых двух этапов, позволяют оценивать качества отображений по следующим критериям:

- Общее время выполнения программы;
- Объем коммуникаций (размещение процессов, зависимых по данным на разные вычислительные модули);
- Баланс загрузки вычислительных модулей по количеству размещенных процессов;
- Баланс загрузки вычислительных модулей по времени занятости вычислительного модуля;

Оптимизации общего алгоритма

Оптимизация последовательных регионов

Оптимизация может быть применена при поэтапном построении дерева в случае, если отображаемая программа содержит строго последовательные регионы.

Мы предполагаем, что на некотором шаге (уровне дерева) мы размещаем процесс «C». В каждой ветви он будет размещен на все доступные ему вычислительные модули.

Для каждой ветви (1) мы проверяем, есть ли другая ветвь (2), в которой максимальное время запуска «C» меньше, чем минимальное время запуска «C» в ветви (1). Формально:

$$T_{es}(C)_2 > \max_{p_j \in P}\{T_{es}(C, p_j)\}_1$$

Если такая ветвь (2) существует, то расписание ветви (1) будет заведомо хуже чем ветви (2), таким образом ветвь (1) может быть отброшена и далее не рассматриваться.

Оптимизация при обходе дерева отображений

Оптимизация может быть применена, если дерево отображений строится полностью с последующим обходом ветвей.

В процессе обхода дерева мы сохраняем длину минимально полученного на данный момент расписания. В случае, если в какой-либо ветви длина расписания достигла этого значения – следовательно ветвь будет заведомо менее оптимально и может быть отброшена.

Оптимизация дерева отражений для последовательных регионов

Оптимизация может быть применена, если дерево отображений строится полностью с последующим обходом ветвей и если отображаемая программа содержит строго последовательные регионы.

Утверждается, что если в программе есть строго последовательный регион, то все отображения, начиная с этого региона и до конца алгоритма, будут одинаковы для всех вариантов отображений, которые были от начала программы до этого региона. Разница будет проявляться лишь во времени начала расписания этого региона.

Таким образом, рассчитав расписание для программы начиная с этого региона и ниже, в других ветвях дерева отображений начиная с

этого региона можно использовать уже готовое рассчитанное расписание.

Заключение

Таким образом, представленный точный алгоритм мапирования обладает следующими характеристиками:

- Учитывает необходимые параметры реальных систем, таких как гетерогенность, коммуникации, требования к вычислительным модулям и т.д.
- Позволяет оценивать качество отображения по широкому ряду критериев, таких как общее время выполнения, объем коммуникаций, балансировка загрузки и т.д.
- За счет механизмов оптимизации позволяет получать за разумное время результаты для программ и платформ большей размерности, чем существующий точные алгоритмы.

Литература

1. W.W. Chu, M.-T. Lan, J. Hellerstein. Estimation of intermodule communication (IMC) and its applications in distributed processing systems. IEEE Trans. Comput. C-33, 8 (Aug.) 1984, 691-699.
2. D.D. Gajski, J. Pier. Essential issues in multiprocessors. IEEE Computer 18, 6 (June) 1985.
3. T.L. Adam, K.M. Chandy, J.R. Dickson. A comparison of list scheduling for parallel processing systems. Commun. ACM 17, 12 (Dec.), 685–690. 1974.
4. J. Bruno, E.G. Coffman, R. Sethi. Scheduling independent tasks to reduce mean finishing time. Commun. ACM 17, 7 (July),382–387. 1974.
5. P. Bouvry, J. Chassin, and D. Trystram. Efficient solutions for mapping parallel programs. In Springer-Verlag, editor, Europar 95, volume 966 of LNCS, September

ОПТИМИЗАЦИЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ В КЛАСТЕРНО-МЕТАКОМПЬЮТЕРНЫХ СИСТЕМАХ

Токарев А.Н.

Пензенский государственный университет, Пенза

Распределенные вычислительные системы (РВС) кластерно-метакомпьютерного типа, в которых нет постоянного соединения между вычислительными узлами, и которые могут динамически менять свою конфигурацию, имеют ряд преимуществ перед обычными кластерными системами. Среди них можно выделить возможность оптимального размещения задачи по вычислительным узлам разной архитектуры и различной мощности, возможность динамического подключения и отключения произвольного количества вычислительных узлов. Однако, строятся такие системы чаще всего на имеющейся инфраструктуре средств вычислительной техники, причем узлы РВС не являются выделенными, и выполняют роль вычислительных узлов РВС помимо роли обычных персональных компьютеров.

Поэтому необходимо разработать такую стратегию размещения подзадач, которая будет ориентированная на РВС, в которой каждый узел имеет определенную характеристику надежности.

Будем считать, что для каждого вычислительного узла (в совокупности с относящимся к нему каналом связи) нам известна полученная статистически характеристика надежности $P_w(t)$, представляющая собой вероятность безотказной работы (или функцию надежности). Так как мы рассматриваем восстанавливаемую систему, для которой характерно чередование времени безотказной работы и времени восстановления, поэтому каждый узел будет характеризоваться двумя состояниями – работа и восстановление.

Рисунок 1 иллюстрирует тот факт, что любой узел может отказать на этапе решения своей подзадачи (если время решения подзадачи t_{ij} больше времени безотказной работы узла T_w), и восстановиться лишь через время восстановления T_f.

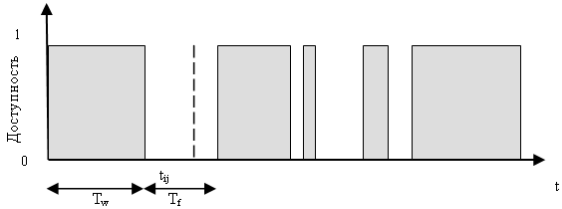


Рис. 1 Работа ненадежного вычислительного узла во времени

Таким образом, подзадача будет решена лишь тогда, когда процесс решения подзадачи уложится в период безотказной работы вычислительного узла. Это достигается тем, что выделенная узлу подзадача разбивается на некоторое количество блоков (т.е. подзадач следующего уровня по отношению к разбиению всей задачи по узлам).

В силу того, что вероятность безотказной работы P_w уменьшается с ростом времени работы t, мы должны определить время T_w, которое будет соответствовать такой вероятности безотказной работы, которая нас устроит. По этой вероятности для каждого узла можно определить время, в течение которого узел будет функционировать, а затем соответственно этому времени определить объем подзадачи S'.

Рассмотрим процесс расчета блока в вычислительном узле 1 (рис. 2)

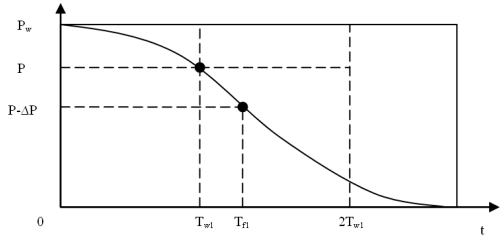


Рис. 2 Процесс расчета блока в одном узле

Допустим, мы определили объем подзадачи S_1', соответствующий времени безотказной работы первого узла $T_{w1}(P)$.

Узел 1 в течение этого времени работает безотказно, и первый блок подзадачи будет решен чуть раньше или к моменту наступления времени $T_{w1}(P)$. После того как узел отправит результат расчета серверу, ему будет выдан следующий блок для расчета $S_1'' = S_1'$. По

прошествии некоторого времени $T_{X1} = \left| T_{f1} - T_{w1}(P) \right|$ узел выходит из строя. С вероятностью $(P - \Delta P)$ это произойдет на отрезке расчета второго блока, т.е. на отрезке $t \in \left[T_{w1}(P); 2T_{w1}(P) \right]$, что задается характером кривой функции надежности и тем, что после начала расчета второго блока планка максимальной вероятности опускается с уровня ранее заданного значения P на допустимое значение уменьшения вероятности безотказной работы ΔP.

Таким образом, на этапе расчета второго блока подзадачи S_1'' в узле 1 произойдет потеря информации, а значит, потеря времени T_{X1}. После этого момента наступает период восстановления узла 1. Так как в нашем распоряжении есть функции восстановления для всех узлов, полученные статистически, по ним можно определить время восстановления первого узла T_{R1}. После восстановления узла надежность опять максимальна, и расчет второго блока S_1'' происходит с нулевой точки отсчета за время $T_{w1}(P)$, как и расчет первого блока S_1'.

Время решения подзадачи первым узлом состоит из времени непосредственного расчета блоков, времени холостого расчета следующих блоков и времени восстановления узла после отказов:

$$T_1 = M_1 \cdot \left[T_{w1}(P) + T_{X1} + T_{R1} \right] - \left[T_{X1} + T_{R1} \right].$$

Условием оптимальности разбиения задачи на подзадачи должно являться условие одновременности получения сервером результатов от всех вычислительных узлов, которое запишется в виде системы линейных уравнений следующим образом:

$$\begin{cases} M_1 \cdot \left[T_{w1}(P) + T_{X1} + T_{R1} \right] - \left[T_{X1} + T_{R1} \right] = M_2 \cdot \left[T_{w2}(P) + T_{X2} + T_{R2} \right] - \left[T_{X2} + T_{R2} \right] \\ M_2 \cdot \left[T_{w2}(P) + T_{X2} + T_{R2} \right] - \left[T_{X2} + T_{R2} \right] = M_3 \cdot \left[T_{w3}(P) + T_{X3} + T_{R3} \right] - \left[T_{X3} + T_{R3} \right] \\ \ldots \\ M_{N-1} \cdot \left[T_{w(N-1)}(P) + T_{X(N-1)} + T_{R(N-1)} \right] - \left[T_{X(N-1)} + T_{R(N-1)} \right] = \\ \qquad = M_N \cdot \left[T_{wN}(P) + T_{XN} + T_{RN} \right] - \left[T_{XN} + T_{RN} \right] \\ \sum_{i=1}^{N} M_i S_i' = S_o \end{cases}$$

В этой системе последнее уравнение является условием полноты задачи.

Предложенная стратегия размещения подзадач позволяет повысить эффективность решения задач в РВС за счет эффективного

распределения подзадач по узлам кластера на основе статистических данных о надежности этих узлов.

Однако, зачастую существует необходимость иметь такую стратегию размещения подзадач, которая не была бы жестко привязана к наличию статистической информации о надежности узлов. Такая стратегия может быть получена с использованием теории марковских случайных процессов.

Процесс функционирования каждого вычислительного узла представляет собой чередование двух состояний (работы и восстановления), причем состояние, в которое узел перейдет в определенный момент, зависит только от того, в каком состоянии он находился до перехода (состояния чередуются), и не зависит от более ранних состояний узла. Таким образом, можно рассматривать процесс функционирования каждого вычислительного узла системы как дискретный марковский процесс с двумя состояниями.

Процесс $\Theta(t)$ в любой момент времени может иметь лишь одно из значений $v_1 = 1$ (работа) и $v_2 = 0$ (восстановление), причем вероятность перехода $v_1 \to v_2$ (отказ узла) за малое время Δt равна $\lambda \Delta t$, а вероятность перехода $v_2 \to v_1$ (возврат узла к работе) равна $\mu \Delta t$. Известны вероятности начального состояния $p_1^0 = 1$, $p_2^0 = 0$.

В соответствии с [3], имея эти исходные данные, можно определить вероятность перехода $\pi_{ij}(t_0, t) = P\{\Theta(t) = v_j | \Theta(t_0) = v_i\}$, где $v_1 = 1$, $v_2 = 0$, $i, j = 1,2$.

В общем случае имеет место система линейных дифференциальных уравнений, полученных из уравнения Колмогорова-Чепмена:

$$\frac{\partial}{\partial t}\pi_{ij}(t_o, t) = \sum_{k=1}^{2} a_{kj}(t)\pi_{ik}(t_0, t), \quad i, j = 1,2$$

где a_{kj} - крутизна изменения вероятности на небольшом отрезке времени.

Для того чтобы применительно к реальным условиям иметь возможность решить эту систему уравнений, необходимо некоторое упрощение, в качестве которого и приняты ранее указанные соотношения $a_{12} = \lambda$, $a_{21} = \mu$, и из условий нормировки $a_{11} = -\lambda$, $a_{22} = -\mu$.

Это значит, что на достаточно небольшом отрезке времени график функции надежности узла можно аппроксимировать линейной функцией.

Поэтому можно воспользоваться методикой расчета переходных вероятностей для дискретного марковского процесса с двумя состояниями и с постоянными коэффициентами a_{ij}. Исходя из этой методики, в результате решения системы уравнений, полученной из уравнений Колмогорова-Чепмена с учетом постоянных коэффициентов, получим следующие выражения для переходных вероятностей:

$$\pi_{11}(\tau) = \frac{\mu}{\lambda+\mu} + (\frac{\lambda}{\lambda+\mu})e^{-(\lambda+\mu)\tau}, \qquad \pi_{12}(\tau) = (\frac{\lambda}{\lambda+\mu})[1-e^{-(\lambda+\mu)\tau}]$$

$$\pi_{22}(\tau) = \frac{\lambda}{\lambda+\mu} + (\frac{\mu}{\lambda+\mu})e^{-(\lambda+\mu)\tau}, \qquad \pi_{22}(\tau) = (\frac{\mu}{\lambda+\mu})[1-e^{-(\lambda+\mu)\tau}]$$

Прежде всего, нас из этих вероятностей интересует вероятность перехода π_{12} узла из состояния 1 в состояние 2, т.е. отказ узла.

Определим значение времени τ из этого выражения:

$$\tau = -\frac{1}{\lambda+\mu}\ln[1-\frac{\pi_{12}(\lambda+\mu)}{\lambda}].$$

Полученное выражение позволит нам, задавшись вероятностью отказа узла π_{12}, получить время этого отказа τ, с учетом поведения узла, описываемого функцией надежности. Исходя из вышесказанного, можно предложить следующую адаптивную стратегию размещения подзадач в распределенной вычислительной системе.

После выдачи стартового набора подзадач вычислительным узлам, сервер переходит в состояние приема результатов. Для каждого узла, приславшего результат, сервер отмечает время его расчета, и выдает следующую подзадачу. Если узел вместо результата решения подзадачи присылает серверу пустой запрос на решение, сервер по этому признаку определяет, что узел отказал при решении ранее выданной ему подзадачи. К этому моменту серверу известны следующие характеристики, относящиеся к этому узлу:

- количество выданных узлу и успешно решенных подзадач K;
- объем каждой подзадачи S_j;
- время решения каждой подзадачи узлом t_j;

- время, прошедшее с момента выдачи последней подзадачи до момента пустого запроса узла на выдачу новой подзадачи t_{K+1}.

Исходя из этих характеристик, можно приблизительно определить время безотказной работы узла на первом этапе и время восстановления после первого отказа

$$t_w^{(1)} \approx \sum_{j=1}^{K} t_j \,, \qquad t_f^{(1)} \approx t_{K+1} - \frac{\sum_{j=1}^{K} t_j}{K} \,.$$

Найденное время безотказной работы узла на первом этапе $t_w^{(1)}$ позволяет приближенно определить коэффициент линейной функции, аппроксимирующей функцию надежности, а время восстановления узла позволяет определить коэффициент при функции восстановления:

$$\lambda^{(1)} = \frac{1}{t_w^{(1)}} \,, \qquad \mu^{(1)} = \frac{1}{t_f^{(1)}}$$

Имея эти коэффициенты, легко найти время безотказной работы τ по приведенной выше формуле, полученной из формулы нахождения переходной вероятности. Дальнейшее распределение подзадач осуществляется с учетом найденного значения времени τ так, чтобы данный узел получил подзадачу такого объема, который будет рассчитан в пределах промежутка времени $(0;\tau)$. После следующего отказа узла процесс перерасчета повторяется, и находятся значения $\lambda^{(2)}$ и $\mu^{(2)}$, и так далее.

Коррекция на каждом шаге значений λ и μ приводит к тому, что по мере протекания процесса вычислений и по мере накопления сведений о длительности промежутков работы и восстановления каждого узла, выдаваемые узлам объемы подзадач все точнее приближаются к оптимальному значению, которое можно выразить как максимальный объем за время безотказной работы узла.

Если по какой-то причине надежность узла начинает падать, коэффициент $\lambda^{(i)}$ увеличивается, соответственно, угол наклона аппроксимирующей прямой уменьшается, и уменьшается прогнозируемое время τ. Однако, это происходит не резко, а плавно, в соответствии с историей накопленной характеристики надежности.

Предложенный адаптивный метод размещения подзадач не требует наличия статистической информации о надежности вычислительных узлов и позволяет изменять нагрузку на вычислительные узлы в соответствии с постоянно обновляемой характеристикой их надежности.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В РЕАЛЬНОМ ВРЕМЕНИ НА КЛАСТЕРНЫХ ВС С РАСПРЕДЕЛЕННОЙ АРХИТЕКТУРОЙ

Улудинцева А. И., Шейнин Ю.Е.
Санкт-Петербургский Государственный Университет Аэрокосмического Приборостроения, Санкт-Петербург

На самом первом этапе проектирования решения задачи имеется лишь описание задачи в терминах ее предметной области. Здесь важно понять: применим ли стохастический метод к ее решению вообще.

Рассмотрим типы систем реального времени и определим типы, к каким из них применим стохастический метод, и классы задач, которые необходимо решать другими способами, отличными от стохастического.

Основная черта системы реального масштаба времени (РМВ) – необходимость получения правильных результатов за определенный крайний срок. Вычислительная правильность системы РМВ зависит от двух составляющих: логической правильности результатов, и правильности выбора времени, то есть способности выполнения вычислений за крайние сроки.

В зависимости от выдвигаемых требований к системе реального времени, она относится: к системам жесткого, мягкого или устойчивого реального времени.

Многие практические задачи в системах обработки информации и управления могут быть переформулированы таким образом, что они попадают в класс систем устойчивого реального времени (Firm Real-Time systems) или мягкого реального времени (Soft Real-Time Systems).

Системы жесткого реального времени налагают детерминированные требования (*deterministic guarantees*) на наступление крайнего времени завершения обработки. Системы устойчивого и мягкого реального времени налагают вероятностные

требования (*probabilistic guarantees*) на наступление крайнего времени завершения обработки.

В классе задач мягкого и устойчивого реального времени допустим стохастический метод организации вычислений РМВ с предсказуемостью требуемых характеристик. В этих системах допустимо превышение времени обработки некоторого процента данных.

Для систем жесткого реального времени стохастический метод не применим. В этих системах критерием правильной работы системы обработки является то, что все данные должны быть обязательно обработаны в однозначно заданных временных ограничениях. Заданные критерии подразумевают стопроцентное выполнение требований ко времени обработки данных, что выходит за рамки вероятностного подхода.

Кроме того, условие к характеристикам подобных задач должно быть не временное ограничение на время обработки данных, а скорость их обработки в заданные границы времени с определенной вероятностью.

Системы жесткого реального времени задают жесткие, детерминированные требования на время решения задачи. Выдержать такие требования во многих параллельных архитектурах, обладающих изрядной недетерминированностью временных характеристик своего функционирования оказывается невозможным. Целые классы параллельных архитектур, например, мультипроцессорные ВС с обменом сообщениями (*message-passing*), кластерные ВС, оказываются непригодны для решения задач РМВ при таком подходе.

Например, в типовых задачах обработки сигналов имеется поток входной информации (цифровых сигналов), поступающих от источника, скорость поступления которых система обработки не может регулировать (например, приостанавливать поступления потока цифровых сигналов). Темп поступления – k блоков информации в секунду. Если ВС не успевает обрабатывать поступающие с заданным темпом цифровые сигналы, то это ошибочная для системы РМВ ситуация. Для таких задач, распространенная формулировка требований РМВ – обработка каждого блока информации до поступления следующего, за 1/k секунд.

Однако, в большинстве случаев содержательное требование звучит значительно мягче – обрабатывать поступающий информационный поток таким образом, чтобы не происходила потеря поступающих

данных. Этим производится переход от жесткого ограничения на латентность обработки индивидуального блока входной информации, $t_{обр} < 1/k$, к более мягкому требованию производительности обработки потока информационных блоков – обработки с производительностью k блоков в секунду.

Анализ содержательных требований в целом ряде задач обработки сигналов показывает, что допустимо и еще некоторое ослабление требований – переход от 100% гарантии отсутствия пропуска поступившего данного, к некоторой, регулируемой постановщиком задачи, вероятности p – вероятности того, что не будет пропущена обработка ни одного поступившего блока из информационного потока.

Таким образом, задача вычислений в РМВ приводится в класс задач, где применимы вероятностные критерии. Это позволяет организовывать параллельные вычисления РМВ на мультипроцессорных ВС с кластерной архитектурой, на системах с обменом сообщениями.

Для построения аналитической модели задач реального времени используется теория массового обслуживания, математический аппарат которой позволит получить критерии производительности и загрузки системы, а также получить закон распределения выходного потока заявок, с помощью которого можно определить вероятностные требования к получению результата обработки за заданное время.

Разным архитектурам ВС будут соответствовать различные типы систем массового обслуживания. Кластеру, состоящему из однотипных ВМ, будет соответствовать система с идентичными устройствами обслуживания – многоканальная система. Конвейеру, по которому данные проходят несколько ступеней, перед тем как покинуть систему, будет соответствовать многофазная система обслуживания или стохастическая сеть.

Рассмотрим методику перехода к системам РМВ с вероятностными критериями, расчета для нее показателей и условий выполнения ограничений РМВ. Пусть ВС состоит из N вычислительных модулей. Вычислительная система имеет кластерную организацию, задача решаемая на данной ВС управляется потоком данных. Таким образом, у нас имеется распределенная ВС, для оценки ее параметров с точки зрения стохастического метода представим ее с помощью схемы взаимодействующих процессов с заданными характеристиками обработки в каждом из ВМ и матрицы переходов потока данных между узлами ВС, рис. 1.

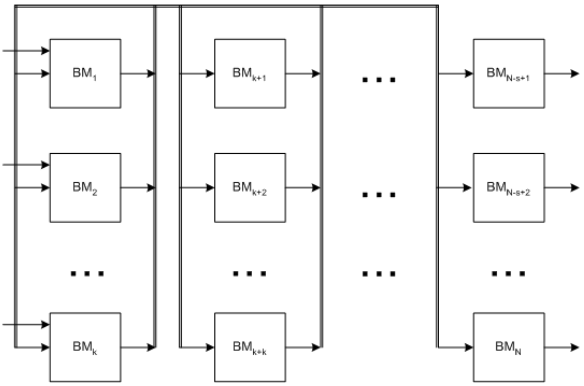


Рис. 1 Представление распределенной ВС схемой взаимодействующих процессов

Каждый переход от одной СМО к другой характеризуется вероятностью перехода P_{ij} ($P_{ij} \leq 1$), где i – номер СМО, от которой идет переход, ($1 \leq i \leq N$);

j – номер СМО, к которой идет переход, ($1 \leq j \leq N$).

Матрица переходов в общем вид имеет вид:

$$P = \begin{bmatrix} P_{00} & P_{01} & \dots & P_{0m} \\ P_{10} & P_{11} & \dots & P_{1m} \\ \dots & \dots & \dots & \dots \\ P_{n0} & P_{n1} & \dots & P_{nm} \end{bmatrix},$$

здесь $m=n$ – количество СМО (вычислительных модулей) в стохастической сети (вычислительной системе).

Каждая вероятность (P_{ij}) зависит от реализации алгоритма и определяется на основе статической информации или логики алгоритма.

Вычислительная система будет характеризоваться следующим набором параметров:

1. Интенсивностью поступления данных на вход системы по каждому из входных каналов - λ1, λ2, …, λk , где k – количество ВМ, на которые поступает входной поток данных. Закон распределения интервалов времени между поступающими данными.

2. Интенсивностью обработки каждой СМО - μ1, μ2, …, μN. Закон распределения времени обработки.
3. Интенсивность и законом распределения интервалов времени между исходящими требованиями из каждой СМО – d1, d2, …, dN.
4. Размер буфера в каждой СМО - l1, l2, …, lN..
5. Загрузкой каждого ВМ - ρ1, ρ2,…, ρN.
6. Требования к параметрам ВС.
7. Вероятностью, которая определяет какой процент от общего количества пакетов, допустимо не успеть обработать вычислительной системе в течение заданного времени.

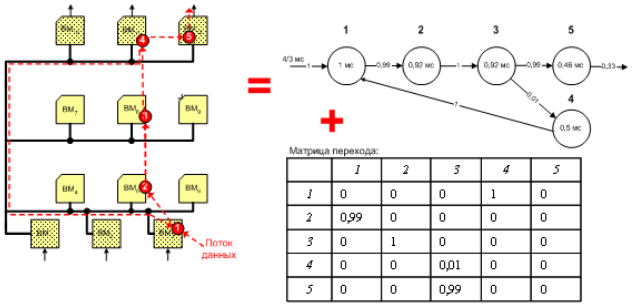
На основании данного описания строиться модель функционирования системы и определяются качественные характеристики ее работы. Моделирование производиться специализированном пакете моделирования, например, GPSS.

Ниже представлен список определяемых качественных характеристик ВС:

Общие параметры	
1	Вероятность потери пакета Этот параметр задается при описание системы и является критерием качества ее функционирования.
2	Распределение задержки пакета
3	Средняя задержка пакета
4	Распределение времени занятости системы
Параметры каждого ВМ	
6	Распределение задержки пакета
7	Распределение времени ожидания
8	Среднее время ожидания
9	Распределение времени занятости
10	Среднее время занятости
11	Распределение длины очереди

На небольшом примере рассмотрим порядок определения качественных характеристик системы.

Пусть ВС состоит из 5 ВМ. Граф потока данных в ВС, схема взаимодействия процессов и матрица переходов имеет вид, рис. 2:



	1	2	3	4	5
1	0	0	0	1	0
2	0,99	0	0	0	0
3	0	1	0	0	0
4	0	0	0,01	0	0
5	0	0	0,99	0	0

Рис. 2. Граф потока данных в ВС, схема взаимодействия процессов и матрица переходов

ВС характеризуется следующими параметрами:

1. Интенсивность поступления данных: $\gamma_1=3/4 мс^{-1}$.
2. Закон распределения промежутков времени между входными данными: детерминированный.
3. Интенсивность обработки в каждом ВМ: $\mu_1=1$ мс$^{-1}$; $\mu_2=1/0.92$ мс$^{-1}$; $\mu_3=1/0.92$ мс$^{-1}$; $\mu_4=1/0.5$ мс$^{-1}$; $\mu_5=1/0.46$ мс$^{-1}$. Закон распределения времени обработки: экспоненциальный.
4. Требование к задержке данных в системе: T = 8 мс.

После моделирования функционирования ВС в пакете GPSS получаем следующие характеристики ВС, рис 3.:

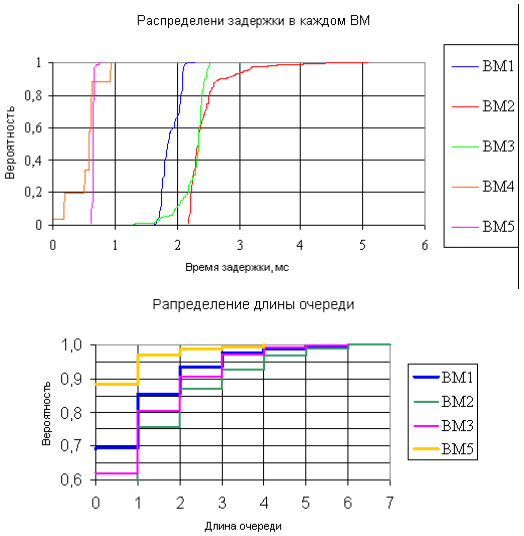


Рис. 3. Характеристики примера ВС

На основании этих показателей системы делается вывод: какими должны быть характеристики системы, чтобы удовлетворять требуемому качеству работы. Или каково будет качество системы при имеющихся характеристиках.

После определения временных характеристик решения задачи на данной ВС можно делать вывод требований к характеристикам задачи и ВС, ее компонентам.

В случае не соответствия задачи временным требованиям производится анализ параметров системы и коррекция начальных данных системы в части скорости и параметров обработки данных.

Например, в случае кластерной архитектуры, если не выполняются временные ограничения на время обработки, можно заложить в модели большее количество вычислительных модулей, или заменить ВМ на более производительные и провести повторный анализ системы.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ О ФИЛАМЕНТАЦИИ МОЩНОГО ФЕМТОСЕКУНДНОГО ЛАЗЕРНОГО ИМПУЛЬСА

Федоров В.Ю.

Московский Государственный Университет им. М.В. Ломоносова, Москва

Введение

Явление филаментации заключается в концентрации значительной части энергии мощного импульса с длительностью $10^{-11} \div 10^{-14}$ с, которая сохраняется на больших расстояниях, намного превосходящих дифракционную длину. В воздухе, для импульсов длительностью 100 фс и длиной волны λ=0.8 мкм характерный диаметр филамента составляет 100 мкм, интенсивность в нем достигает $5 \cdot 10^{13}$ Вт/см^2. Ширина спектра генерируемого суперконтинуума лежит в диапазоне от 0.5 до 3 мкм [1]. Такие характеристики позволяют использовать явление филаментации для создания широкополосных лидаров для экологического мониторинга окружающей среды, систем управления газовым разрядом, для создания элементов микрооптики. Для всех этих приложений важной является возможность управления филаментацией. Эксперименты по филаментации трудоемки и дороги, поэтому для выявления основных закономерностей используется численное моделирование. В данной работе предлагается параллельный алгоритм для решения уравнения описывающего филаментацию мощного лазерного импульса в турбулентной атмосфере.

Динамическая модель стохастической филаментации в турбулентной атмосфере

В модель распространения лазерного импульса в атмосфере включены явления дифракции, дисперсии, керровской самофокусировки, дефокусировки в индуцированных импульсом плазменных каналах, а также турбулентные флуктуации показателя преломления $\tilde{n}$ атмосферы [2,3]. Уравнение для комплексной амплитуды электрического поля $E(x,y,z,t)$ импульса имеет вид:

$$2ik_0 \frac{\partial E}{\partial z} = \Delta_\perp E - k_0 k_\omega'' \frac{\partial^2 E}{\partial t^2} + \frac{2k_0^2}{n_0}\left[\Delta n_k(x,y,z,t) + \Delta n_p(x,y,z,t) + \tilde{n}(x,y,z)\right]E \qquad (1)$$

где k_0 – волновое число, $k''_\omega = \partial^2 k / \partial \omega^2$, ω – центральная частота импульса, n_0 – линейный показатель преломления.

Изменение показателя преломления воздуха, вызванное керровской нелинейностью, имеет вид:

$$\Delta n_k(x,y,z,t) = \frac{1}{2} n_2 |E|^2 + \frac{1}{2} n_2 \int_{-\infty}^{t} H(t-t') |E(x,y,z,t')|^2 dt' , \qquad (2)$$

где n_2 – коэффициент кубичной нелинейности показателя преломления атмосферы, $H(t)$ - функции отклика, описывающая нестационарный вклад комбинационного рассеяния в кубичную нелинейность [4].

Изменение показателя преломления в плазме, индуцированной лазерным импульсом при фотоионизации, выражается следующим образом

$$\Delta n_p(x,y,z,t) = -\frac{\omega_p^2}{2 n_0 \omega^2} , \qquad (3)$$

где $\omega_p^2(x,y,z,t) = 4\pi e^2 N_e(x,y,z,t)/m$ - плазменная частота; e, m – заряд и масса электрона, Ne – концентрация свободных электронов, которая определяется согласно кинетическому уравнению:

$$\frac{\partial N_e}{\partial t} = R(|E|^2)(N_0 - N_e) , \qquad (4)$$

где N_0 – концентрация нейтральных молекул, $R(|E|^2)$ – скорость ионизации, которая рассчитывается в соответствии с моделью Переломова-Попова-Терентьева (ППТ) [5].

Флуктуации показателя преломления в атмосфере $\tilde{n}(x,y,z)$, создавая возмущения светового поля в поперечном сечении импульса, ответственны за случайный характер филаментации. Флуктуации $\tilde{n}$ моделируются цепочкой фазовых экранов.

Считалось, что комплексная амплитуда светового поля $E(x,y,z=0,t)$ на выходе лазерной системы имеет вид:

$$E = E_0 \exp\left(-\frac{x^2 + y^2}{2a_0^2} - \frac{t^2}{2\tau_0^2} \right), \qquad (6)$$

где a_0 – поперечная ширина импульса, τ_0 – его длительность, E_0 – начальная амплитуда.

Проблемы численного моделирования филаментации

Численное моделирование филаментации мощного фемтосекундного лазерного импульса связано с большими вычислительными затратами. Это обусловлено тем, что при нелинейно-оптической трансформации импульса диапазон характерных масштабов лазерного излучения сильно расширяется. При распространении мощного импульса образуется филамент, характерный масштаб которого в воздухе составляет 10^{-2} см. Тонкая структура изменения фазы светового поля в филаменте имеет масштаб порядка 10^{-3} см. Таким образом, в процессе распространения по трассе импульса радиусом несколько сантиметров диапазон пространственных масштабов изменения светового поля расширяется на 3 порядка.

Для разрешения тонкой структуры филамента требуется порядка 10 узлов на характерном масштабе изменения фазы, и шаг расчетной сетки h в плоскости XOY должен быть порядка 10 мкм. Вместе с тем, при моделировании распространения пучка в свободном пространстве область изменения координат в поперечной плоскости должна быть достаточно велика и составлять порядка 10 радиусов исходного пучка. В реальных экспериментах используются импульсы диаметром несколько сантиметров, и расчетная область может составлять 10÷50 см. В результате, по каждой из координат x, y необходимо порядка $10^4 \div 10^5$ узлов, и общее число узлов расчетной сетки в этой плоскости достигает $10^8 \div 10^{10}$. Для сокращения размерности обрабатываемых массивов нами используется специальная расчетная сетка с переменным шагом в плоскости XOY [6]. Вся область поперечного сечения импульса разбивается на две зоны. В первой из них, находящейся у оси импульса, там, где сосредоточена основная часть энергии, шаг сетки h выбирается малым и постоянным для адекватного воспроизведения пространственного изменения фазы светового поля. Во второй зоне, находящейся на периферии импульса, шаг медленно нарастает при удалении от оси. Расчетная сетка с переменным шагом в плоскости поперечного сечения импульса позволяет сократить время, затрачиваемое на выполнение программы и размер обрабатываемых массивов в 4÷5 раз.

Для адекватного воспроизведения изменения огибающей импульса и процесса генерации лазерной плазмы необходим достаточно мелкий шаг Δt расчетной сетки по времени. Шаг Δt должен составлять не более 10^{-2} от длительности импульса, а размер расчетной области по времени - не менее 3÷5 его длительностей. Число узлов расчетной сетки по времени достигает $10^2 \div 10^3$.

В целом филаментация импульса описывается нелинейно-оптической стохастической задачей размерностью (3D+1). Согласно приведенным выше оценкам моделирование такой задачи требует обработки численных массивов общим объемом $10^{10} \div 10^{13}$. Так в случае расчетов с одинарной точностью, комплексная амплитуда поля в одном узле сетки будет занимать 8 байт оперативной памяти, что в соответствии со сделанной оценкой величины массивов приведет к тому, что в оперативной памяти должен хранится массив размером несколько десятков терабайт. Поиск эффективных методов численного исследования этой задачи является весьма важным. Такой поиск возможен по двум взаимодополняющим направлениям. Во-первых, это использование высокопроизводительных вычислительных комплексов и алгоритмов распараллеливания задачи для их оптимальной загрузки. Второе направление состоит в разработке простых физических моделей для исследования конкретных проблем явления филаментации лазерных импульсов.

Параллельный алгоритм

Расчетный код написан на языке Fortran95 с использованием стандарта MPI для обмена сообщениями между вычислительными узлами кластера. При решении уравнения (1) используется метод разделения по физическим факторам [7]. Согласно этому методу, на каждом шаге Δz вдоль оси распространения, решение (1) заменяется последовательным решением цепочки уравнений:

$$\begin{cases} 2ik_0 \dfrac{\partial E}{\partial z} = \dfrac{\partial^2 E}{\partial x^2} E \\ 2ik_0 \dfrac{\partial E}{\partial z} = \dfrac{\partial^2 E}{\partial y^2} E \end{cases}, \tag{7.1}$$

$$2ik_0 \frac{\partial E}{\partial z} = -k_0 k_\omega'' \frac{\partial^2 E}{\partial t^2}. \tag{7.2}$$

$$2ik_0 \frac{\partial E}{\partial z} = \frac{2k_0^2}{n_0} \Delta n_p(x, y, z, t) E \,, \tag{7.3}$$

$$\begin{cases} 2ik_0 \dfrac{\partial E}{\partial z} = \dfrac{2k_0^2}{n_0} \Delta n_p(x, y, z, t) E \\ \dfrac{\partial N_e}{\partial t} = R(|E|^2)(N_0 - N_e) \end{cases}, \tag{7.4}$$

$$2ik_0 \frac{\partial E}{\partial z} = \frac{2k_0^2}{n_0} \tilde{n}(x, y, z) E \,. \tag{7.5}$$

В качестве начального условия для каждой следующего уравнения берется решение предыдущего. Применение данного метода позволяет для каждого уравнения из цепочки (7) использовать наиболее эффективный метод решения.

Комплексная амплитуда поля $E(x,y,z,t)$ в расчетной программе представляет собой трехмерный массив $E(N_x,N_y,N_t)$, где N_x, N_y, N_t – количество узлов расчетной сетки по каждой из координат. Координата z является эволюционной и в процессе счета, при достижении определенного расстояния по z происходит запись необходимой информации на жесткий диск. Между вычислительными узлами кластера массив амплитуды поля разбит вдоль координаты x. На каждом узле кластера содержится одна часть $E(N_{xi},N_y,N_t)$ исходного массива (N_{xi} – количество узлов расчетной сетки содержащихся на вычислительном узле с номером i, $\sum_i N_{xi} = N_x$).

Уравнения (7.3), (7.4) и (7.5) интегрируются аналитически, и каждое из них на текущем шаге Δz имеет аналитическое решение, что позволяет обойтись без передачи данных между вычислительными узлами. Уравнение (7.2) решается с помощью быстрого преобразования Фурье и, поскольку, оперирует только с временным сечением массива комплексной амплитуды поля, необходимость в обмене данными между вычислительными узлами также отпадает.

Основную сложность при решении цепочки уравнений (7) представляют уравнения дифракции (7.1). Как было отмечено выше, для экономии оперативной памяти в плоскости поперечного сечения XOY используется специальная сетка с переменным шагом. Ее

использование делает невозможным применение быстрого преобразование Фурье для решения этих уравнений. Однако матрица системы уравнений (7.1) на используемой неоднородной сетке получается трехдиагональной, что позволяет использовать для решения метод прогонки [8]. Согласно этому методу сначала происходит вычисление «прогоночных» коэффициентов α_i по рекуррентной формуле вида

$$\alpha_i = f(\alpha_{i-1}), \tag{8}$$

где значение первого коэффициента α_1 берется из граничных условий. Таким образом, для определения i-го «прогоночного» коэффициента необходимо знать все i-1 предыдущих коэффициентов. Затем, на основе этих коэффициентов, происходит вычисление значений амплитуды поля в узлах сетки по рекуррентной формуле вида

$$E(x_i) = f(E(x_{i+1})). \tag{9}$$

Аналогичная формула имеет место в случае дифракции по оси *y*. Здесь также для вычисления значения поля в i-ой точке необходимы все i-1 предыдущих значений. В уравнение дифракции по оси *y* оперирует только с сечением массива амлитуды поля вдоль координаты *y*. Поэтому его решение не связано с передачей данных между вычислительными узлами кластера. Основной проблемой при решении цепочки (7), таким образом, является решение методом прогонки уравнения дифракции (7.1) по оси x.

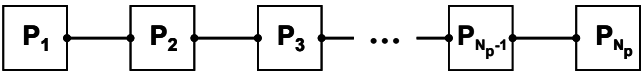


Рис. 1 Топология вычислительных узлов. Np – количество вычислительных узлов в задаче

Параллельный алгоритм решения уравнения дифракции по оси *x* выглядит следующим образом. Все вычислительные узлы объединяются в декартову топологию (рис. 1), так что каждый узел имеет соседа справа и слева (за исключением первого и последнего узла, имеющих по одному соседу).

Каждый вычислительный узел, для каждого из узлов расчетной сетки по координатам *y* и *t* осуществляет следующие действия:

1. Получение значения «прогоночного» коэффициента от соседа слева. При этом самый левый узел вычисляет первый «прогоночный» коэффициент исходя из граничного условия.
2. Расчет «прогоночных» коэффициентов в узлах сетки содержащихся на данном вычислительном узле по рекуррентной формуле (8).
3. Отправка последнего рассчитанного «прогоночного» коэффициента соседу справа. При этом самый правый вычислительный узел ничего не делает.
4. Получение значения комплексной амплитуды поля от соседа справа. При этом самый правый узел рассчитывает первое значение амплитуды исходя из граничного условия и значения последнего «прогоночного» коэффициента.
5. Расчет значений комплексной амплитуды поля в узлах сетки содержащихся на данном вычислительном узле по рекуррентной формуле (9).
6. Отправка последнего рассчитанного значения амплитуды поля соседу слева. При этом самый левый вычислительный узел ничего не делает.

Предложенный алгоритм требует организации двух сеансов приема-передачи сообщений для каждого из расчетных узлов по координатам y, t. Это приводит к тому, что значительное время выполнения программы тратится на обмен сообщениями и это время тем больше, чем больше вычислительных узлов кластера задействовано в задаче. Так, например, если количество расчетных узлов сетки составляет $N_x \cdot N_y \cdot N_t = 512 \cdot 512 \cdot 600$, то вычисление одного шага по оси z требует время 204.28 с при запуске кода на одном вычислительном узле (процессор Intel® Xeon® с тактовой частотой 2.8 ГГц). При запуске кода на четырех вычислительных узлах с тем же процессором (сеть – Gb Ethernet) это время увеличивается до 406.65 с. Однако, несмотря на заметную потерю в скорости, применение параллельного кода делает принципиально возможным решение уравнения (1) с параметрами близкими к реальному эксперименту, так как путем разделения массивов данных между вычислительными узлами удается обойти ограничения связанные с необходимым объемом оперативной памяти.

Один из возможных путей повышения скорости счета – накопление данных перед отправкой. Например, посылка сообщения

соседу после предварительного расчета значений «прогоночных» коэффициентов и значений амплитуды поля для нескольких расчетных узлов сетки. Однако применение такого подхода требует дополнительных исследований и оптимизации, что планируется сделать в будущем.

Литература

1. Kasparian J., Sauerbrey R., Chin S.L., 2000, Applied Physics B, 71, 877.
2. Андрианов К.Ю., Кандидов В.П., Косарева О.Г., Чин С.Л., Талебпур А., Петит С., Луи В., Ивасаки А., Надё М.-К., 2002, Изв. РАН. Сер. Физическая, 66, 1091.
3. Шленов С.А., Кандидов В.П., 2004, Оптика атмосферы и океана, 17, 630.
4. Mlejnek M., Wright E. M., Moloney J. V., 1998, Optics Letters, 23, 382.
5. Переломов А.М., Попов В.С., Терентьев М.В., 1966, ЖЭТФ, 50, 1393.
6. Кандидов В.П., Федоров В.Ю., 2004, Квантовая электроника, 34, 1163.
7. Годунов С. К., Рябенький В. С., Разностные схемы – М.: Наука, 1973.
8. Калиткин Н. Н., Численные методы – М.: Наука, 1978.

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ПРИ РЕШЕНИИ ЖЕСТКИХ ЗАДАЧ КОШИ

Фельдман Л.П., Назарова И.А.

Донецкий национальный технический университет, Донецк, Украина

Широко распространенными способами создания параллельных методов являются: распараллеливание хорошо исследованных и многократно апробированных последовательных численных алгоритмов, а также создание новых параллельных методов, изначально ориентированных на распределенную реализацию. Если ограничиться рассмотрением численных алгоритмов решения задачи Коши, основанных на конечно-разностных схемах, то параллельные свойства таких алгоритмов во многом определяются видом лежащей в

их основе численной схемы. Наиболее изученными являются явные методы, однако присущие этим схемам недостатки, одним из которых является их условная устойчивость, ограничивают область применения таких алгоритмов. В этой связи значительный интерес представляют неявные схемы, которые, не смотря на большую вычислительную сложность, не имеют альтернативы среди одношаговых методов при решении жестких задач.

В докладе представлены параллельные алгоритмы решения нелинейной задачи Коши для СОДУ на базе неявных схем с встроенными альтернативными методами оценки апостериорной локальной погрешности, а также предложен специальный подход, учитывающий особенности решения линейных задач.

Распараллеливание решения задачи Коши для СОДУ на основе неявных одношаговых разностных схем

Численное решение задачи Коши для системы обыкновенных дифференциальных в общем случае нелинейных уравнений (СОДУ) первого порядка размерности m с известными начальными условиями полностью неявным *s*-стадийным методом типа Рунге-Кутта (ПНМРК) можно получить последовательно по шагам с помощью следующей схемы:

$$\begin{cases} \bar{y}_{n+1} = \bar{y}_n + h \cdot \sum_{i=1}^{s} b_i \cdot \bar{k}_i ; \\ \bar{k}_i = \bar{f}(x_n + c_i h, \bar{y}_n + h \sum_{j=1}^{s} a_{ij} \bar{k}_j), \ i = \overline{1,s}, \end{cases}$$

где $\bar{y}_n = \bar{\tilde{y}}(x_0 + nh)$ – приближенное решение на шаге n, *s*-размерные вектора $c = (c_1, c_2, ..., c_s)$, $b = (b_1, b_2, ..., b_s)$ и полностью заполненная $s \times s$ матрица $A = (a_{ij}), i, j = \overline{1,s}$ описывают уникальный вариант метода и выбираются из соображений точности, h – выбранный шаг интегрирования.

К достоинствам полностью неявных, одношаговых методов общего вида (на основе квадратурных формул Радо и Лобатто) следует отнести хорошие характеристики устойчивости и точности, достаточные для решения жестких задач. Так например, *s* – стадийный

метод РадоIA имеет порядок практически в 2 раза больше, чем число стадий и обладает A-устойчивостью [1]. Недостатками этих методов является высокая вычислительная сложность, обусловленная итерационным процессом определения шаговых коэффициентов, которая не позволяет эффективно применять эти методы на последовательных машинах.

При решении систем ОДУ с использованием полностью неявных методов Рунге-Кутта все $m \cdot s$ неизвестных должны определяться одновременно, что существенно усложняет задачу (всего шаговых коэффициентов s и размерность каждого вектора $\bar{k}_i$ равна. Для решения таких систем используется метод функциональной итерации:

$$\begin{cases} \bar{g}_i^{(0)} = 0; \bar{k}_i^{(0)} = \bar{f}(x_n + c_i h, \bar{y}_n); \\ \bar{g}_i^{(l)} = a_{i1}\bar{k}_1^{(l)} + a_{i2}\bar{k}_2^{(l)} + ... + a_{is}\bar{k}_s^{(l)}; \\ \bar{k}_i^{(l)} = \bar{f}(x_n + c_i h, \bar{y}_n + h \cdot g_i^{(l-1)}); \\ i = 1,...,s; l = 1,...,N. \end{cases}$$

Здесь итерационный процесс, повторенный l – раз представляет $\bar{k}_i^{(l)}, i = \overline{1,s},$ как l – тую аппроксимацию для вектора шагового коэффициента $\bar{k}_i$. Скорость сходимости итерационного процесса может быть определена следующим образом: $r^* = min(r, N+1),$ где r – порядок используемого ПНМРК. Как и для всех одношаговых методов, распараллеливание ПНМРК базируется на выполнении одного шага интегрирования. Все множество процессоров разбивается на s групп по числу шаговых векторов, каждый процессор в группе вычисляет всего $\lceil m/p \rceil$ компонент $\bar{k}_i$ и затем передает их всем процессорам группы. Вычисление решения в следующей точке требует реализации операции множественная пересылка данных "*all-to-all*".

Межгрупповой обмен может быть осуществлен двумя способами:

1. каждый первый процессор в группе передает вектор $\bar{k}_i$ в первый элемент каждой другой группы + параллельный групповой обмен в каждой группе;
2. межгрупповая передача по типу "все-всем".

Для неявных схем определение локальной погрешности с целью управления шагом интегрирования производилось на основе локальной экстраполяции Ричардсона и методов вложенных форм. Трудоемкость ПНМРК существенно зависит от сложности функции $\bar{f}$ – правой части СОДУ. В общем случае вложенные явные методы имеют меньшую вычислительную сложность, чем неявные и этот эффект увеличивается с увеличением порядка метода. Для вложенных методов трудоемкость коммуникационных операций зависит от числа этапов метода: *s,* для НМРК и от числа итераций**,** причем с ростом порядка метода число этапов растет быстрее, чем число итераций.

Проведенные теоретические исследования и вычислительный эксперимент, дают возможность говорить о том, что не смотря на большие накладные расходы, алгоритмы решения нелинейной задачи Коши для систем обыкновенных дифференциального уравнений на основе полностью неявных методов Рунге-Кутты при тщательном подборе всех параметров могут быть достаточно эффективно отображены на параллельные структуры с топологиями гиперкуб и тор при асинхронной передаче данных.

Особенности решения задачи Коши для линейных СОДУ на параллельных ВС

Экспоненциальный метод относится к специальным методам численного решения задачи Коши для систем линейных обыкновенных дифференциальных уравнений (СЛОДУ), основан на точном представлении решения в аналитической форме и вычислении матричной экспоненты. Предлагаемый метод особенно эффективен для решения систем с большой константой Липшица, в частности для жестких систем уравнений. Этот способ решения требует меньшего объема вычислений, чем стандартные методы и позволяет построить более простые и эффективные параллельные алгоритмы решения задачи Коши.

Общий вид задачи Коши для однородных СЛОДУ с постоянными коэффициентами:

$$\begin{cases} \bar{y}'(x) = A \cdot \bar{y}(x), \ \bar{y}(x_0) = \bar{y}_0, \\ A = \| a_{ij} \|, i, j = 1..m; A, b = const, \end{cases}$$

где $\bar{y}$ – вектор неизвестных; $\bar{y}_0$ – вектор начальных условий; A – матрица коэффициентов линейной системы.

Приближенное решение СЛОДУ можно построить, аппроксимировав матричную экспоненту отрезком ряда Тейлора при малом h:

$$F(hA) \cong \sum_{k=0}^{r} \frac{(hA)^k}{k!}, \bar{y}_{n+1} = F^r(Ah) \cdot \bar{y}_0,$$

а затем, используя некоторый алгоритм умножения матриц, вычислить численное решение.

Решение задачи Коши для неоднородной системы:

$$\begin{cases} \bar{y}'(x) = A \cdot \bar{y}(x) + b, \quad \bar{y}(x_0) = \bar{y}_0, \\ A = \| a_{ij} \|, i, j = 1..m, \ A, b = const \end{cases}$$

имеет вид:

$$\bar{y}_{n+1} = \bar{y}_n + R(h) \cdot (A \cdot \bar{y}_n + b),$$

$$R(h) = \sum_{i=1}^{\infty} \frac{1}{i!} \cdot A^{i-1} \cdot h^i,$$

где $R(h)$ – матричная функция, которая также может быть аппроксимирована отрезком ряда Тейлора: $R(h) \cong \sum_{i=1}^{r} \frac{1}{i!} \cdot A^{i-1} \cdot h^i$.

Основной особенностью метода является возможность введения подготовительного этапа, который будет выполняться до начала интегрирования и вынести в него наиболее ресурсоемкие операции нахождения матричных форм, а именно матричное умножение, не зависящее от шага интегрирования. Матричная функция $F(h)$ обладает следующим свойством: $F(h) = F(h/2)F(h/2)$, что позволяет достаточно легко встраивать алгоритмы определения локальной погрешности на основе дублирования шага и локальной экстраполяции. Применение экспоненциального метода для вложенных форм делает практически ненужными вычисления по формуле высшего порядка:

$$\begin{cases} \bar{\bar{y}}^{n+1} = [E + hA + \dfrac{h^2}{2}A^2 + \ldots + \dfrac{h^r}{r!}A^r + \dfrac{h^{r+1}}{(r+1!)}A^{r+1}] \cdot \bar{y}^n; \\ \bar{y}^{n+1} = [E + hA + \dfrac{h^2}{2}A^2 + \ldots + \dfrac{h^r}{r!}A^r] \cdot \bar{y}^n; \\ \left\| \bar{\bar{y}}^{n+1} - \bar{y}^{n+1} \right\| = \dfrac{h^{r+1}}{(r+1)!}A^{r+1} \cdot \bar{y}^n. \end{cases}$$

Для плотнозаполненных матриц имеется два принципиально различных класса последовательных алгоритмов умножения матриц: традиционные и рекурсивные методы быстрого умноження. В классическом варианте, как известно, алгоритмы рекурсивного умножения матриц очень чувствительны к ошибкам округления, т.е. плохо обусловлены, что ограничило их область применения. В связи с этим предлагается использовать полиалгоритмы рекурсивного и блочного систолического умножения, что позволяет избежать указанных недостатков для умножения матриц практических размеров.

Для построения и исследования эффективности параллельных алгоритмов решения СЛОДУ на основе экспоненциального метода использовалась декомпозиционная методика и аппарат графов влияния. Проведенные исследования позволяют сделать выводы:

- решение линейной задачи Коши на основе модифицированного экспоненциального метода с использованием локальной экстраполяции и вложенных форм имеет практически одинаковую вычислительную сложность, хотя локальная экстраполяция дает нам целую таблицу решений;
- наиболее эффективной топологией для реализации методов является решетка и ее замкнутый эквивалент тор, поскольку на такой топологической схеме наиболее эффективно выполняются матричные операции.

Параллельные блочные методы решения задачи Коши для СОДУ

До сих пор рассматривались вопросы построения параллельных алгоритмов решения задачи Коши посредством распараллеливания эффективных последовательных методов. К методам изначально

ориентированным на параллельную архитектуру следует отнести блочные одношаговые и многошаговые k - точечные методы. В [2] подробно рассмотрены вопросы построения конкретных вычислительных схем блочных методов, приведено доказательство сходимости, дана оценка погрешности и алгоритм решения нелинейной разностной задачи. Здесь рассмотрим способы оценки локальной апостериорной погрешности для параллельных блочных методов и дадим оценку параллелизма.

Множество M точек равномерной сетки x_m, $m=1,2,...,M$ и $x_M = H$ с шагом h разобьем на блоки, содержащие по k точек, $kN \geq M$. В каждом блоке введем номер точки $i = 0,1,...,k$ и обозначим через $x_{n,i}$ точку n – го блока с номером i. Точку $x_{n,0}$ назовем началом блока n, а $x_{n,k}$ – концом блока. При численном решении задачи Коши одношаговым блочным методом для каждого следующего блока новые k значений приближенного решения вычисляются одновременно с использованием значения только в последней точке предшествующего блока.

Обозначим через $y_{n,i}$ – приближенное значение решения задачи Коши в точке $x_{n,i}$ – обрабатываемого блока. Тогда, для одношаговых блочных методов разностные уравнения имеют вид:

$$y_{n,i} = y_{n,0} + ih\left[b_1 f_{n,0} + \sum_{j=1}^{k} a_{i,j} f_{n,j} \right], \quad i = \overline{1,k}, \quad n = 1,2,...,$$ где

$f_{n,j} = f\left(x_n + jh, y_{n,j}\right)$.

В общем случае уравнения многошаговых разностных методов для блока из k точек при использовании вычисленных значений приближенного решения в m предшествующих блоку узлах, с учетом введенных выше обозначений можно записать в виде:

$$y_{n,i} = y_{n,0} + ih\left[\sum_{j=1}^{m} b_{i,j} f_{n,j-m} + \sum_{j=1}^{k} a_{i,j} f_{n,j} \right], i = \overline{1,k}$$,n=1,2,…

Определить коэффициенты $a_{i,j}$ и $b_{i,j}$ можно интегро-интерполяционным методом, построив интерполяционный многочлен

$L_{m+k-1}(x)$ с узлами интерполяции $x_{n,j-m}$ и соответствующими им значениями правой части СОДУ.

Для оценки локальной погрешности при решении одношаговым многоточечным методом используем идею вложенных форм. Решаются две задачи на одной сетке с одним и тем же шагом h: первая – одношаговым *k*-точечным методом; вторая – одношаговым *(k+1)*-точечным методом. Локальная погрешность приближенного решения одношаговым *k*-точечным методом в *i*-ом узле блока определяется формулой:

$$y_{n,i}^{(k)} - y(x_{n,i}) \approx \varphi^{(k+2)}(x_{n,o}; \tilde{y}_{n,o})h^{k+2}, i = \overline{1,k},$$

для *(k+1)*-точечного метода локальная погрешность в том же узле равна:

$$y_{n,i}^{(k+1)} - y(x_{n,i}) \approx \varphi^{(k+3)}(x_{n,o}; \tilde{y}_{n,o})h^{k+3}, i = \overline{1,k}.$$

Вычитая из верхнего соотношения нижнее, получим представление главного члена погрешности *k*-точечного метода на шаге в виде:

$$\gamma_{n,i}^{(k)} = \varphi_i^{(k+1)}(x_{n,0}, \tilde{y}_{n,0})h^{k+1} \approx y_{n,i}^{(k)} - y_{n,i}^{(k+1)}, i = \overline{1,k},$$

который может быть использован для оценки локальной погрешности.

Для оценки шаговой погрешности при решении многошаговым многоточечным методом используется аналогичный подход.

При численном решении задачи Коши для сравнительной характеристики методов можно рассматривать различные показатели, например: коэффициенты ускорения и эффективности, которые в свою очередь зависят от многих параметров. В случае произвольной правой части уравнения трудоемкость метода напрямую зависит от размерности задачи и числа процессоров, топологии соединения, типа параллельной ВС и машинозависимых констант. При сложных правых частях блочные методы обладают достаточными оценками эффективности: практически линейным ускорением и единичной эффективностью.

В докладе приведен обзор и анализ результатов исследований, посвященных параллельным методам численного решения задачи Коши для СОДУ, и является продолжением ранее опубликованных работ [2-5]. Перспективным направлением дальнейших исследований является разработка масштабируемых алгоритмов решения задачи

Коши с встроенными альтернативными методами оценки апостериорной локальной погрешности для блочных параллельных методов, а также исследование влияния параметров параллельной системы на динамические характеристики полученных параллельных алгоритмов.

Литература

1. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. – М.: Мир,1999. – 685с.
2. Фельдман Л.П. Параллельные алгоритмы моделирования динамических систем, описываемых обыкновенными дифференциальными уравнениями.//Электронное моделирование, 2004, том 26, №1, с.19-30.
3. Назарова И.А., Фельдман Л.П. Масштабируемый параллельный алгоритм численного решения линейных СОДУ для компьютеров с распределенной памятью // Тезисы докладов V Международной конференции по неравновесным процессам в соплах и струях (NPNJ-2004), Самара, 5-10 июля 2004 г. – М.: Вузовская книга, 2004, с. 153–155.
4. Назарова И.А. Параллельные полностью неявные методы численного решения жестких задач для СОДУ. //Научно-теоретический журнал ИПИИ МОН и НАН Украины «Искусственный интеллект», №3, 2005. – Донецк: ИПИИ, 2005. – с. 185–193.
5. Фельдман Л.П., Назарова И.А. Параллельные алгоритмы численного решения задачи Коши для систем обыкновенных дифференциальных уравнений // Математическое моделирование, т.18, № 9, 2006, с. 17-31.

Контакты

nazarova@r5.dgtu.donetsk.ua

ИСПОЛЬЗОВАНИЕ МАРКОВСКИХ МОДЕЛЕЙ ДЛЯОЦЕНКИ ЭФФЕКТИВНОСТИ КЛАСТЕРНЫХ СИСТЕМ

Фельдман Л.П., Михайлова Т.В.
Донецкий национальный технический университет, Донецк

Введение

Одним из способов исследования различных структур является использование непрерывных [1] или дискретных аналитических моделей [2]. Дискретные модели Маркова, отражающие модель вычислительной среды более точно, имеют большую размерность и эффективно распараллеливаются на параллельные вычислительные структуры [8]. Непрерывные модели менее трудоемкие, поэтому более широко применяются для исследований.

В настоящее время широкое распространение получили кластеры различной архитектуры. По критерию совместного использования дискового пространства они классифицируются следующим образом: с совместным использованием дискового пространства и без предоставления доступа к ресурсам [5].

Анализ кластерных систем

На основании методик [3,4] можно построить модели кластеров с совместным использованием дискового пространства (рис.1, рис.3) и без предоставления доступа к ресурсам (рис.5).

В каждом из кластеров *M* рабочих станций пользователей, *N1* серверов, *N2* дисковых массивов.

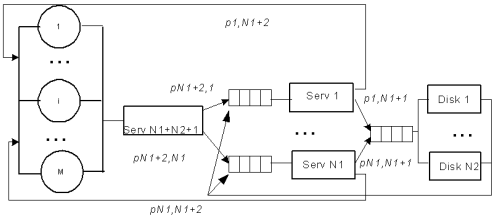


Рис. 1 Структура неоднородного кластера

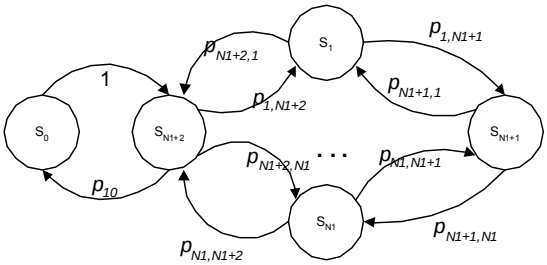


Рис.2 Граф передач неоднородного кластера

Функционирование рассматриваемой системы можно представить замкнутой стохастической сетью, содержащей $N1+2$ системы массового обслуживания (СМО), в которой циркулирует M заявок. Граф передач этой сети изображен на рис.2. По графу передач можно определить коэффициенты посещений каждой СМО.

За состояние системы можно принять распределение заявок по СМО

$\overline{m} = (m_{N1+2}, m_1, \ldots, m_{N1}, m_{N1+1})$,

где $m_{N1+2}+m_{1+}\ldots+m_{N1}+m_{N1+1}=M$, и по теореме Джексона вычисляются стационарные вероятности, которые позволяют вычислить характеристики вычислительной среды: загрузки устройств; среднее количество занятых устройств в s-м узле; среднее количество задач, находящихся в s-м узле; среднее количество задач, находящихся в очереди к s-му узлу; средние времена пребывания и ожидания в s-м узле; средние времена пребывания и ожидания в системе [2,4].

При $N2$=2, $N1$=2. Классы задач, решаемых в вычислительной среде заданы таблицами (табл.1-вероятности обращения к серверам, табл.2-вероятности обращения обращения к дискам). Количество задач M=10.

Таблица 1 Вероятности обращения к серверам

	Номер варианта (вар.$p_{серв.}$)			
	1	2	3	4
p(5,1)	0.83125	0.59375	0.35625	0.11875
p(5,2)	0.11875	0.35625	0.59375	0.83125

Таблица 2 Вероятности обращения к дискам

	Номер варианта (вар.$p_{диска}$)			
	5	6	7	8
p(2,3)	0.125	0.375	0.625	0.875
p(1,3)	0.875	0.625	0.375	0.125

Зависимости времени решения задачи от вероятностей обращения к серверам и дискам приведена рис.3. Эффективными являются классы задач, соответствующие вариантам (1,7) и (4,6), а также вариантам (1,8), (2,7), (4,5), (3,6), у которых время решения задачи наименьшее.

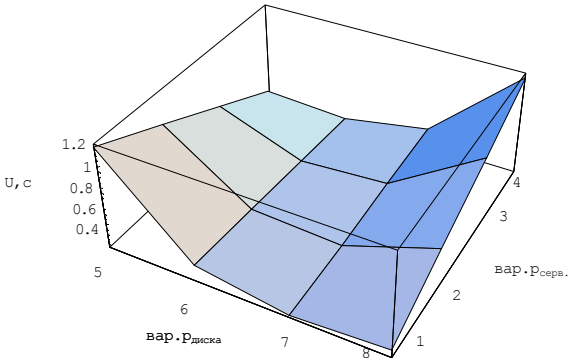


Рис. 3 Зависимость времени отклика от вероятностей обращения к диску и к серверу

Аналогично можно построить модель кластера топологии NxN (рис.4) и с разделяемыми дисками.

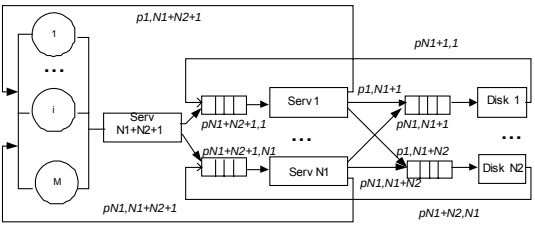


Рис. 4 Структура кластера топологии NxN

Используя полученные при моделировании стационарные вероятности, можно вычислить основные характеристики кластерных систем [2,4]. Если анализируется конкретный вид кластера для решения определенного класса задач, то с помощью этих характеристик можно определить эффективность вычислительной среды (в зависимости от критерия: равномерная загрузка всех узлов, минимальное время отклика и т.д.) [4].

Для подбора оптимального коэффициента мультипрограммирования можно использовать методику [6], в которой предлагается критерий сбалансированности, составляющие которого цена простоя оборудования и штраф за задержку выполнения запроса.

Для выбора вида кластера при решении на нем определенного класса задач можно сравнить получаемые при моделировании характеристики и выбрать подходящую структуру вычислительной среды.

Вероятностные модели можно использовать для задач синтеза вычислительных структур.

Синтез кластерных систем

Для оптимизации состава и структуры вычислительных систем можно использовать методы [7], позволяющие определить структуру вычислительной среды минимальной стоимости при заданном времени отклика или, наоборот, с минимальным временем отклика заданной стоимости, а также модифицированный метод оптимизации состава и структуры высокопроизводительных вычислительных систем с использованием метода средних [9].

Алгоритм с использованием теоремы Джексона имеет комбинаторный порядок, а алгоритмы с использованием теоремы о среднем [9] – полиномиальный, что позволяет решать задачи, которые вообще не решаются аналитическим методом на современных ЭВМ в течение реального времени.

Заключение

Предложенные методики анализа эффективности вычислительных систем позволяют получить характеристики функционирования этих сред при решении в них различных классов задач и, по возможности, контролировать качество функционирования ВС.

Рассмотренные способы оптимизации состава и структур высокопроизводительных ВС с использованием метода средних можно использовать для проектирования ВС.

Таким образом, использование вероятностных моделей при проектировании, эксплуатации и оптимизации вычислительных систем позволяет вырабатывать рекомендации по рациональному использованию ресурсов этой вычислительной среды.

Литература

1. Авен О. И. и др. Оценка качества и оптимизация вычислительных систем. – М.: Наука, 1982, 464с.
2. Клейнрок Л. Вычислительные системы с очередями. – М.:Мир, 1979, 600с. Последовательно - параллельные вычисления: Пер. с англ. - М.: Мир, 1985. - 456 с.
3. Михайлова Т.В. Анализ оценки эффективности кластерных систем с использованием вероятностных моделей //Системный анализ и информационные технологии. Тезисы докладов учасників Международной научно-практической конференции,2003г., г.Киев, 83-85с.
4. Основы теории вычислительных систем/С.А.Майоров, Г.И.Новиков, Т.И.Алиев и др.М.: Высшая школа, 1978, 408с.
5. Спортак М., Франк Ч., Паппас Ч. и др. Высокопроизводительные сети. Энциклопедия пользователя.- К.:"ДиаСофт", 1998.-432с.
6. Фельдман Л.П., Михайлова Т.В. Оценка эффективности кластерных систем с использованием моделей Маркова. //Известия ТРТУ. Тематический выпуск: Материалы Всероссийской научно-технической конференции с международным участием «Компьютерные технологии в инженерной и управленческой деятельности». – Таганрог: ТРТУ, 2002. – №2 (25). – с. 50–53.
7. Фельдман Л.П., Михайлова Т.В. Способы оптимизации состава и структуры высокопроизводительных вычислительных систем //Научные труды Донецкого государственного технического университета. Серия «Информатика, кибернетика и вычислительная техника»(ИКВТ-2001).- Донецк: ДонГТУ.- 2000. С. 80-85.
8. Фельдман Л.П., Михайлова Т.В. Параллельный алгоритм построения дискретной марковской модели

/Высокопроизводительные параллельные вычисления на кластерных системах. Материалы четвертого Международного научно-практического семинара и Всероссийской молодежной школы. /Под редакцией член-корреспондента РАН В.А. Сойфера. – Самара, 2004. – с. 249–255.
9. Фельдман Л.П., Михайлова Т.В. Оптимизация состава и структуры высокопроизводительных вычислительных систем с использованием метода средних // Научные труды Донецкого государственного технического университета. Серия «Информатика, кибернетика и вычислительная техника», випуск 39: – Донецьк, ДонНТУ, 2002. - С. 53–58.

ИСПОЛЬЗОВАНИЕ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ ДЛЯ САПР ЭЛЕКТРОНИКИ

Хаханов В.И., Обризан В.И., Гаврюшенко А.Ю.
Харьковский национальный университет радиоэлектроники, Харьков, Украина

Введение

Размерность современных цифровых систем на кристаллах повышает сложность верификации, синтеза и тестировании при проектировании и производстве. Это делает проблему моделирования неисправностей и автоматической генерации тестов актуальной. Производительность средств моделирования и производительность рабочих станций растет значительно медленнее, чем сложность проектируемых устройств или стоимость верификации [1].

В эру встроенных систем, очень просто создать сложные устройства, используя системный уровень проектирования, но в то же самое время их очень трудно моделировать, верифицировать и тестировать. Ранее инженеры использовали высокопроизводительные рабочие станции, чтобы сократить время моделирования. Но на сегодняшний день тактовые частоты перестали расти и, чтобы решить проблемы моделирования, компьютеры вошли в эру многоядерности. Мультипроцессорные системы пришли в дом и офис, а не только в суперкомпьютерные центры. Таким образом, гигагерцы больше не определяют производительность рабочей станции. Также известно, что однопоточные приложения (даже хорошо оптимизированные для

последовательной обработки) не показывают ожидаемого прироста производительности на мультипроцессорных системах. В настоящее время, каждое приложение должно быть спроектировано таким образом, чтобы получить максимум производительности на многоядерных архитектурах. Это утверждение и есть основой настоящего исследования.

Объект исследования – алгоритмы моделирования в составе автоматизированной системы проектирования электроники. Цель исследования – сокращение времени на разработку микроэлектроники за счет использования параллельных алгоритмов проектирования на многоядерных процессорах.

Эта статья организованна следующим образом. Во втором разделе описан вектор развития многоядерных платформ Intel. В третьем разделе показаны результаты экспериментов типичных приложений САПР электроники и показана проблема их использования. В четвертом разделе рассматриваются источники для параллелизма при моделировании. В завершении мы делаем выводы по исследованию и даём рекомендации по разработке нового программного обеспечения в области САПР электроники.

Развитие многоядерных процессоров

Через несколько лет платформы Intel – рабочие станции, мобильные устройства и серверы – будут в основном использовать многоядерные процессоры. Сейчас мы уже используем двуядерные процессоры на рабочих станциях, серверы поддерживают восемь потоков, а к концу десятилетия Intel предложит потребителям серверы, которые будут одновременно выполнять 32 потока (см. рис. 1) [2]. В настоящее время, Intel работает над многоядерной архитектурой, которая в итоге будет включать в себя сотни вычислительных ядер в одном процессоре.

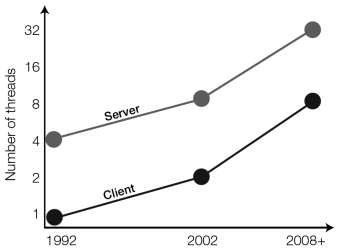


Рис. 1 Количество одновременных потоков (прогноз Intel)

Также следует отметить доступность двуядерных процессоров на рынке. Например, микропроцессор Intel Pentium D 915 с тактовой частотой 2.8 ГГц и общим объемом кэш памяти 4 МБ стоит 145 долларов США, что является приемлемой ценой для компьютеров среднего уровня (если учесть, что фактически одно вычислительное ядро стоит 145/2=78 долларов).

Эксперименты с существующими САПР

Были проведены ряд экспериментов с системой верификации проектов Aldec Active-HDL 7.1 (www.aldec.com) и системой логического синтеза для FPGA/CPLD устройств Synplicity Synplify 8.1 (www.synplicity.com) с целью анализа производительности на однопроцессорных и многоядерных рабочих станциях. Эксперимент состоял из двух тестов: 1) выполнение задачи на процессоре, содержащем одно вычислительное ядро; 2) выполнение задачи на двуядерном процессоре. В первом случае наблюдалась 100% загрузка ресурсов процессора (Task Manager CPU Usage), что символизировало об эффективном использовании компьютера. Во втором же случае, всего лишь 50% ресурсов были использованы, и только один поток был выделен для решения задачи логического моделирования. Тот же самый результат наблюдался для задач компиляции исходных файлов, логического синтеза и размещения. Таким образом, экспериментально подтверждено, что традиционные последовательные программы не показывают эффективного использования ресурсов на мультипроцессорной системе (см. рис. 2). Согласно прогнозам [2], эта ситуация будет только ухудшаться в будущем.

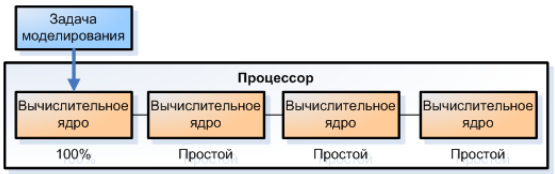


Рис. 2 При четырех вычислительных ядрах однопоточное приложение моделирования или синтеза покажет только 25% загрузки всего процессора

Источники параллелизма

Эффективное использование многоядерных процессоров предполагает использование параллельных алгоритмов в программном обеспечении. При проектировании микроэлектроники инженер

работает с моделью цифровой схемы, которая представляет собой объект физического мира. Все процессы в реальном мире протекают параллельно, поэтому модель тоже должна включать в себя параллелизм. Архитектор программы САПР должен грамотно отобразить модель устройства на параллельную архитектуру или технологию.

Существует несколько источников параллелизма для улучшения производительности программы моделирования [3]. *Алгоритмический параллелизм* (algorithmic parallelism) использует конвейер, чтобы ускорить цикл моделирования. Различные шаги алгоритма моделирования выполняются на различных процессорах, например, управление очередью событий, вычисление значений функций. Масштаб распараллеливания может быть не очень большим, потому что обычно количество шагов в алгоритме ограничено. *Параллелизм данных* (data parallelism) предполагает использование различных процессоров для моделирования схемы на различных входных наборах. Этот подход очень эффективен для моделирования неисправностей, где нужно оценить качество большого количества независимых тестовых последовательностей, а также различных списков неисправностей. Он менее эффективен для верификации проекта, где основная цель – это сократить время моделирования одного вектора. При *модельном параллелизме* (model parallelism) используются различные процессоры для вычислений функций различных логических элементов схемы. Потенциально этот подход удовлетворяет требованию к *масштабируемости* – количество параллельно действующих процессов прямо пропорционально размеру (сложности) модели.

Заключение

Проведя исследование можно сделать несколько важных выводов.

1. Имеется много программного обеспечения, которое используется ежедневно многими инженерами. Это программное обеспечение не показывает ожидаемого прироста производительности на многоядерных процессорах.
2. Одно из важных требований для параллельного приложения – масштабируемость. Количество ядер на кристалле будет возрастать год от года и приложения должны показывать одинаково хорошую эффективность для любых платформ.
3. Новые приложения САПР электроники должны разрабатываться по принципиально новым методам, в которых

упор будет не на алгоритмы, а на отображение параллельных моделей на соответствующие параллельные архитектуры и технологии.

Литература

1. Bergeron Janick. Writing testbenches: functional verification of HDL models. Boston: Kluwer Academic Publishers. 2001. 354 p.
2. Intel® Software Insight. Multi-core Capability. July 2005.
3. Roger D. Chamberlain. Parallel Logic Simulation of VLSI Systems. Proceedings of the 32nd ACM/IEEE conference on Design automation, 1995. P.p. 139-143.

Контакты

{hahanov, obrizan}@kture.kharkov.ua, т./ф.: +380 (57) 702-13-26

АНАЛИЗ ДИНАМИЧЕСКОЙ РЕАКЦИИ КОНСТРУКЦИИ НА ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРАХ

Черников С.К., Баскевич Т.П.

КФТИ КазНЦ РАН , Казань

Введение.

При численном моделировании механических процессов в сложных структурах приходится иметь дело с моделями, насчитывающими миллион и более неизвестных. Трудоемкость задач обуславливается еще и тем, что при использовании явных процедур интегрирования процессы приходится исследовать с достаточно мелким шагом по времени. Это делает естественным решение таких задач на вычислительных кластерах, вызывая необходимость разработки соответствующих параллельных программ. В настоящей работе предполагается, что исследуемая конструкция может состоять из большого количества конечных элементов балки, мембраны, пластины [1] или оболочки. Кроме того в модель могут быть включены различные пружины, демпферы и сосредоточенные массы. В качестве треугольных элементов оболоки использовалась композиция изгибаемой треугольной пластины Зенкевича и треугольного элемента мембраны с постоянными напряжениями. Четырехугольный элемент был реализован в соответствии с работой [2].

Движение ансамбля конечных элементов описывается системой уравнений:

$$\boldsymbol{M}\ddot{\boldsymbol{u}}(t)+\boldsymbol{C}\dot{\boldsymbol{u}}(t)+\boldsymbol{K}\boldsymbol{u}(t)=\boldsymbol{R}(t), \qquad (1)$$

где $\boldsymbol{M}$, $\boldsymbol{C}$ и $\boldsymbol{K}$ соответственно матрицы масс, демпфирования и жесткости; $\boldsymbol{R}$ – вектор внешней узловой нагрузки; $\boldsymbol{u}$, $\dot{\boldsymbol{u}}$ и $\ddot{\boldsymbol{u}}$ – векторы узловых перемещений, скоростей и ускорений ансамбля конечных элементов. Для интегрирования этой системы уравнений воспользуемся схемой прямого явного интегрирования по времени методом центральных разностей. Для дискретного момента времени $\boldsymbol{t}$ система (1) разрешается относительно ускорений:

$$\ddot{\boldsymbol{u}}_t = \boldsymbol{M}^{-1}\left[\boldsymbol{R}_t - \boldsymbol{C}\dot{\boldsymbol{u}}_t - \boldsymbol{K}\boldsymbol{u}_t\right].$$

Скорости и перемещения для следующего шага рассчитываются по формулам:

$$\dot{\boldsymbol{u}}_{t+\Delta t/2} = \dot{\boldsymbol{u}}_{t-\Delta t/2} + \ddot{\boldsymbol{u}}_t \Delta t$$
$$\boldsymbol{u}_{t+\Delta t} = \boldsymbol{u}_t + \dot{\boldsymbol{u}}_{t+\Delta t/2}\Delta t$$

Обновление скоростей в моменты, сдвинутые на половину шага по времени, улучшает точность и сходимость решения. Необходимые для вычисления ускорений $\ddot{\boldsymbol{u}}_t$ скорости $\dot{\boldsymbol{u}}_t$ в момент времени t можно найти либо с использованием разного вида экстраполяций, либо, в простейшем случае, считать, что $\dot{\boldsymbol{u}}_t = \dot{\boldsymbol{u}}_{t-\Delta t/2}$.

Выбор шага интегрирования.

Главным недостатком метода центральных разностей является его условная устойчивость. Шаг интегрирования Δt должен быть меньше критического значения Δt_{cr}, вычисляемого исходя из инерционных и жесткостных свойств всего ансамбля элементов. Далее в качестве Δt_{cr} используется наименьшее значение критического шага

$$\Delta t_{cr}^{e} = \frac{T_{\min}}{\pi},$$

где $T_{\min}$ -минимальный период колебаний элемента, зависящий от его размеров и типа.

Для балочных элементов длиной L с сечением площадью F и моментом инерции J из материала с плотностью ρ и модулем

упругости E значение критического шага Δt_{cr}^{e} может определяться как продольными, так и изгибными колебаниями элемента. Считая элемент защемленным по концам и, имея в виду, что $T_{\min} = 2\pi / p_1$, а собственные значения частот продольных и изгибных колебаний элемента определяются соотношениями $p_1^{прод.} = \frac{\pi}{L}\sqrt{\frac{E}{\rho}}$ и $p_1^{изг.} = \frac{22.37}{L^2}\sqrt{\frac{EJ}{\rho F}}$ [3, 4], найдем соответствующие периоды колебаний по соотношениям $T_{прод.} = \frac{2\pi}{p_1^{прод.}} = 2L\sqrt{\frac{\rho}{E}}$ и $T_{изг.} = \frac{2\pi}{p_1^{изг.}} = 0.28L^2\sqrt{\frac{\rho F}{EJ}}$.

Величиной критического шага Δt_{cr}^{e} для элемента балки, таким образом, будет наименьшее из значений $\Delta t_{прод.} = T_{прод.}/\pi = 0.67L\sqrt{\frac{\rho}{E}}$, $\Delta t_{изг.} = T_{изг.}/\pi = 0.089L^2\sqrt{\frac{\rho F}{EJ}}$.

Для элемента квадратной пластины с размером стороны L и толщиной h из материала с плотностью ρ, модулем упругости E и коэффициентом Пуассона ν значение критического шага Δt_{cr}^{e} определяется изгибными колебаниями. Считая элемент защемленным по контуру, определим частоту колебаний по соотношению $p_1 = \frac{35.999}{L^2}\Big/\sqrt{\frac{\rho h}{D}}$ [3], где $D = \frac{Eh^3}{12(1-\nu^2)}$ - цилиндрическая жесткость. Период колебаний элемента найдем по формуле $T = \frac{2\pi}{p_1} = \frac{\pi\sqrt{3}}{9}\frac{L^2}{h}\Big/\sqrt{\frac{E}{\rho\left(1-\nu^2\right)}}$, а критический шаг для

квадратного элемента пластины определится соотношением $\Delta t^e_{cr} = \frac{\sqrt{3}}{9}\frac{L^2}{h}\Big/\sqrt{\frac{E}{\rho\left(1-\nu^2\right)}} = 0.19\frac{L^2}{h}\Big/\sqrt{\frac{E}{\rho\left(1-\nu^2\right)}}$.

Для элементов произвольной четырехугольной и треугольной пластины можно пользоваться этим же соотношением, используя в качестве L характерный линейный размер элемента. В наших исследованиях для 3-х угольных элементов за L принималась наименьшая высота элемента, для 4-х угольных – отношение площади элемента либо к длине наибольшей стороны, либо к длине наибольшей диагонали.

Для элемента оболочки значение критического шага Δt^e_{cr} определяется как наименьшее из величин, определяемых по соотношениям $\Delta t^e_{cr} = L\Big/\sqrt{\frac{E}{\rho\left(1-\nu^2\right)}}$ и $\Delta t^e_{cr} = 0.19\frac{L^2}{h}\Big/\sqrt{\frac{E}{\rho\left(1-\nu^2\right)}}$. Здесь L - характерный линейный размер элемента, определяемый как в предыдущем случае.

Распараллеливание алгоритма решения.

При распараллеливании алгоритма решения системы (1) учитывалось, что матрицы жесткости $\boldsymbol{K}$ и демпфирования $\boldsymbol{C}$ конструкции представляют собой суммы матриц жесткости и демпфирования элементов $K = \sum_{i=1}^{n_e} K_i^e$, $C = \sum_{i=1}^{n_C} C_i^e$. Если разделить вектора $\ddot{\boldsymbol{u}}$, $\dot{\boldsymbol{u}}$ и $\boldsymbol{u}$ на равные блоки по количеству доступных параллельных процессов и рассортировать матрицы жесткости и матрицы демпфирования элементов в соответствии со степенями свободы, входящими в каждый из блоков, то в каждом процессе параллельно могут выполняться действия

$$\ddot{\boldsymbol{u}}_t^j = \boldsymbol{M}^{-1}\left[\boldsymbol{R}_t^j - \sum_{i=1}^{n_C^j} C_i^e \dot{\boldsymbol{u}}_{t-\Delta t/2} - \sum_{i=1}^{n_e^j} K_i^e \boldsymbol{u}_t\right]$$

$$\dot{\boldsymbol{u}}^{\boldsymbol{j}}_{t+\Delta t/2} = \dot{\boldsymbol{u}}^{j}_{t-\Delta t/2} + \ddot{\boldsymbol{u}}^{\boldsymbol{j}}_{\boldsymbol{t}}\Delta t$$

$$\boldsymbol{u}^{j}_{t+\Delta t} = \boldsymbol{u}^{\boldsymbol{j}}_{\boldsymbol{t}} + \dot{\boldsymbol{u}}^{j}_{t+\Delta t/2}\Delta t$$

После завершения этих вычислений процессы должны обменяться результатами для формирования в каждом из них векторов $\dot{\boldsymbol{u}}_{t+\Delta t/2}$ и $\boldsymbol{u}_{t+\Delta t}$.

При реализации описанного алгоритма использовалась технология MPI. Программа написана на языках FORTRAN и C++. Для обеспечения расширения библиотеки элементов в процессе эксплуатации программы предусмотрена возможность встраивания элементов без компиляции и сборки текстов программы. При этом процедуры, соответствующие добавляемым элементам, помещаются в отдельную динамически присоединяемую библиотеку и описываются в конфигурационном файле.

Оценка точности и сходимости элементов

Для оценки точности и сходимости элемента балки рассмотрим двухопорную балку (Рис. 1) с характеристиками: ***L***=100 см, ***H***=1 см, ***B***=1см, ***E***=2000000 кГ/см^2, $\rho = 0.78\times10^{-5}$ тем/см^3.

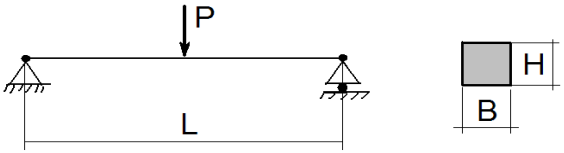


Рис. 1

В центре пролета к балке прикладывалась сосредоточенная сила величиной 1 кГ. Величина прогиба в центре балки в момент времени 0.01 секунды сравнивалась со значением, полученным аналитически по формуле $v\left(L/2, t\right) = \dfrac{2P}{mLp_1^2}\left(1 - Cos\left(p_1 t\right)\right)$, заимствованной из источника [4]. В ней $m = \rho HB$ - погонная масса, $p_1 = \dfrac{\pi^2}{L^2}\sqrt{\dfrac{EJ}{m}}$ - собственная частота первого тона балки. При увеличении числа

элементов с 2 до 100 оценка прогиба в центре находилась в пределах 0.1078 - 0.1074см. Аналитическое решение дает оценку 0.1074см.

Исследование точности и сходимости элементов мембраны проведем на примере, заимствованном из работы [5]. В нем рассматривалась консольная балка с характеристиками: ***L***=10 in, ***H***=1 in, ***B***=1 in, ***E***=12000 psi, $\nu = 0.2$, $\rho = 0.1024 \times 10^{-5}$ lb.sec^2/in^4.

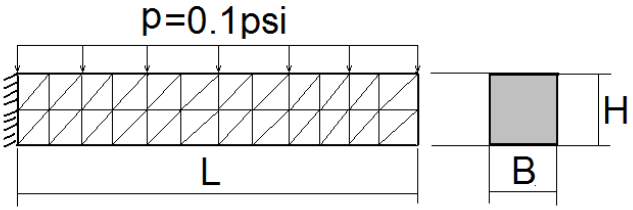


Рис. 2

К балке прикладывалась нагрузка, равномерно распределенная по длине. Расчетная схема балки приведена на рисунке 2. Максимальный прогиб на конце балки, полученный для различных сеток трехугольных элементов мембраны, приведен в Таблице 1:

Сетка	2x10	4x20	8x40	16x80	Аналитическое решение [5]
Прогиб, (x10^{-3}in)	9.31	17.74	23.00	24.86	25.00

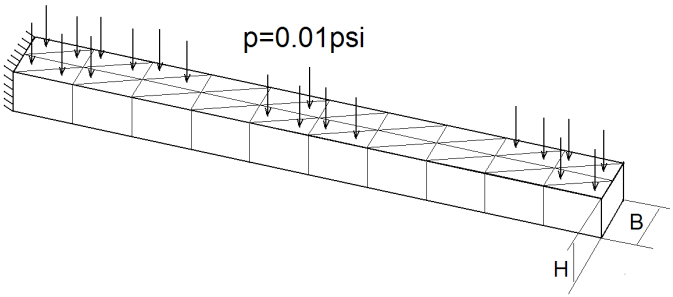


Рис. 3

Исследование точности и сходимости элементов трехугольной и четырехугольной пластины проводилось на примере балки, показанной на рисунке 3, с характеристиками, аналогичными описанным в предыдущем разделе. Максимальные прогибы на конце балки, полученные для различных сеток трех- и четырехугольных элементов пластины и оболочки, приведены в Таблице 2:

Сетка	Прогиб, ($x10^{-3}$in)		
	Треугольные элементы пластины	Четырехугольные элементы пластины	Треугольные элементы оболочки
1x5	25.04	25.21	25.06
2x10	25.05	25.11	25.04
4x20	25.05	25.08	25.03
Аналитическое решение [5]	25.00	25.00	25.00

Для исследования точности и сходимости элементов пружины, линейного демпфера вязкого трения и сосредоточенной массы рассмотрим систему, состоящую из тела массой m, удерживаемого упругой связью жесткостью c, и демпфера вязкого трения с коэффициентом демпфирования α, находящуюся под действием внешней силы $P(t)$ (Рис. 4).

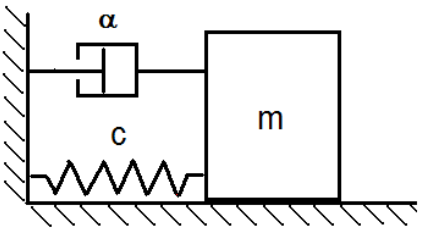


Рис. 4

Если к неподвижной и недеформированной системе приложить нагрузку, изменяющуюся по линейному закону $P(t) = kt$, то без учета демпфирования ($\alpha = 0$) при $t \geq 0$ положение тела определится

выражением $x = \frac{k}{mp^3}(pt - \sin pt)$. Положим $m = 1\,кг$, $c = 16\,н/м$, $P_0 = 10\,н$, и выбрав $k = 1000$, для $t = 0.01\,сек$ получим $x = 1.6499 \times 10^{-4}\,м$. Численное решение дает оценку $x = 1.6665 \times 10^{-4}\,м$.

Для тестирования элемента линейного демпфера вязкого трения рассмотрим свободные колебания системы. Уравнение движения системы описывается уравнением $m\ddot{x} - \alpha\dot{x} + cx = 0$. Аналитическое решение этого уравнения при начальных условиях $x_{t=0} = x_0$, $\dot{x}_{t=0} = \dot{x}_0$ имеет вид $x = e^{-nt}\left[x_0 \cos p_1 t + (\dot{x}_0 + nx_0)\frac{1}{p_1} Sin\, p_1 t\right]$, где $n = \frac{\alpha}{2m}$, $p_1 = \sqrt{p^2 - n^2}$, а $p = \sqrt{\frac{c}{m}}$. Выберем $m = 1\,кг$, $c = 16\,н/м$, $\alpha = 1\,нсек/м$, $x_0 = 0$, $\dot{x}_0 = 1\,м/сек$, тогда для $t = 1.0\,сек$ получим $x = -0.1147\,м$. Численное решение дает оценку $x = -0.11\,м$.

Оценка масштабируемости алгоритма

Оценка времени решения при использовании различного числа узлов кластера проводилось при исследовании динамического отклика консольной балки, свободный конец которой нагружался синусоидально изменяющейся во времени нагрузкой. Балка моделировалась конечными элементами треугольной оболочки. Рассматривались три задачи, отличающиеся количеством конечных элементов. Для сохранения постоянного шага интегрирования по времени размеры конечных элементов при этом не изменялись, а корректировались размеры (длина и высота) балки. Время выполнения 31400 шагов приведено на Рис. 5. На этом же рисунке приведены теоретические кривые (*), соответствующие линейному ускорению.

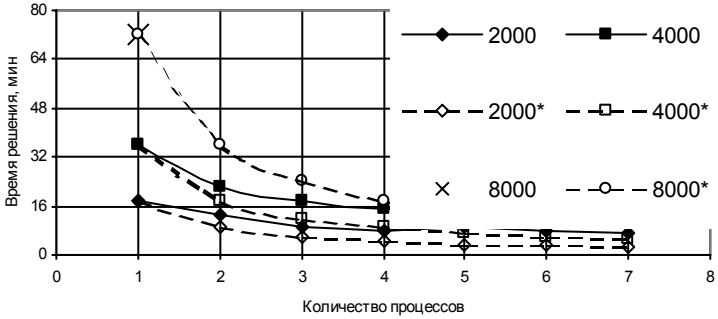


Рис. 5

Литература

1. Н.Н.Шапошников, Н.Д.Тарабасов, В.Б.Петров, В.И.Мяченков. "Расчеты машиностроительных конструкций на прочность и жесткость" - М.: Машиностроение, 1981. 334 с.
2. Черников С.К., Садчиков Ю.В. "Варианты четырехузлового изопараметрического конечного элемента оболочки для расчета автомобильных рам". Труды III международной конференции "Автомобиль и техносфера". Казань, 2003, с.269-278.
3. И.А.Биргер и Я.Г.Пановко. «Прочность, устойчивость, колебания», Справочник, том 3. - М.: Машиностроение, 1968. 567 с.
4. В.Л.Бидерман. "Прикладная теория механических колебаний". - М.: Высшая школа, 1972.- 416 с.
5. T.Belytschko, J.I.Lin. “Explicit algoritms for the nonlinear dynamics of shells”. Comp. Meth.Appl.Mech. and Engrg. 42(1984) 225-251.

РЕШЕНИЕ НЕКОТОРЫХ ЗАДАЧ ФИЗИКИ И МЕХАНИКИ НА МНОГОПРЦЕССОРНЫХ СИСТЕМАХ С РАСПРЕДЕЛЕННОЙ ПАМЯТЬЮ

Черников С.К., Садчиков Ю.В., Ашихмин А.Н.
КФТИ КазНЦ РАН , Казань

В предлагаемой работе приведены примеры использования семейства вычислительных кластеров, построенных на процессорах AMD и недорогой комуникационной аппаратуре Gigabit Ethernet, для решения ряда задач физики и механики. Рассмотренно несколько примеров эффективно работающих параллельных программ.

Симуляция спектров ЭПР в реакционном центре фотосинтеза

Реакционные центры фотосинтеза представляют большой теоретический интерес. Этот интерес вызван желанием создать искусственные системы, которые эффективно преобразуют световую энергию в химическую. Первичной фотохимической стадией фотосинтеза является разделение зарядов в реакционном центре, когда электронно-возбужденный квантом света хлорофилл (донор) отдает электрон акцептору. Восстановленный акцептор участвует в дальнейших химических превращениях. Разделенные заряды в реакционном центре являются парамагнитными частицами, образуя электрон-дырочную пару в неравновесном состоянии, которую можно изучать с использованием методов ЭПР-спектроскопии. Чтобы по наблюдаемому ЭПР-спектру судить о процессах, происходящих в реакционном центре, необходимо иметь математическую модель, описывающую спиновую динамику электрон-дырочной пары. Описание одного из вариантов такой модели можно найти в статье [1].

Вычисляемое значение интенсивности спектра можно определить как некоторую комбинацию элементов матрицы плотности $\rho(t)$, которая, в свою очередь, определяется из системы уравнений $\hat{L}_{eff}\left|\rho(t)\right\rangle = \left|\rho(0)\right\rangle$ (1). Матрица $\hat{L}_{eff}$, имеющая порядок 16x16, формируется из гамильтониана $\hat{H}_{mw}$, описывающего взаимодействие между электроном и дыркой, взаимодействие электрон-дырочной пары с внешним магнитным полем и магнитными ядрами реакционного центра. Взаимодействие электрон-дырочной пары с магнитными ядрами реакционного центра называется сверхтонким

взаимодействием. В отличие от модели, описанной в [1], это взаимодействие в определенной степени учитывается случайными параметрами с известными характеристиками распределения. Для оценки влияния сверхтонкого взаимодействия на наблюдаемое значение интенсивности спектра можно воспользоваться методом Монте-Карло, многократно решая систему (1) с различными значениями характеристик сверхтонкого взаимодействия и вычисляя математическое ожидание наблюдаемого значения интенсивности спектра.

Отметим, что матрица $\hat{L}_{eff}$ существенно зависит от ориентации реакционного центра относительно внешнего магнитного поля, т.е. угловых параметров Ω. Предположив, что при любом эксперименте мы одновременно имеем дело с большим числом произвольно ориентированных относительно внешнего поля реакционных центров, найдем наблюдаемую интенсивность спектра как среднее по направлениям, определяемым согласно [2].

Для исследования реакционных центров методами ЭПР-спектроскопии наиболее важной характеристикой является форма спектра, поэтому интенсивность спектра приходится вычислять многократно для различных значений напряженности внешнего магнитного поля из некоторого заданного диапазона. Таким образом, в процессе вычисления спектра приходится многократно (до 10^{11} раз) формировать и решать систему уравнений (1), что при использовании однопроцессорных компьютеров занимает существенное время (4-5 суток). В то же время эта задача допускает достаточно простое распараллеливание, что позволяет эффективно решать ее на многопроцессорных кластерах. Распараллеливание задачи заключается в разделении области Ω на подобласти Ω_i по числу доступных параллельных процессов. В каждом процессе реализовывался описанный выше алгоритм вычисления математического ожидания наблюдаемого значения интенсивности спектра при фиксированной ориентации реакционного центра относительно внешнего магнитного поля. В последующем вычислялось среднее значение интенсивности спектра по подобласти Ω_i. После этого вызовом процедуры MPI_REDUCE проводилось окончательное вычисление интенсивности спектра. Программа написана на языке FORTRAN.

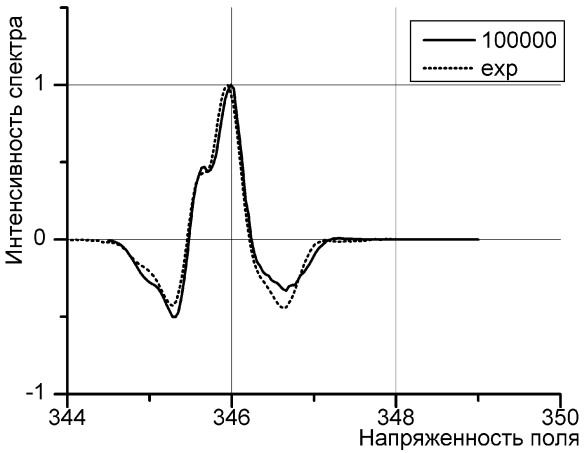


Рис. 1

На рисунке 4 приведен спектр реакционного центра, расчитанный с помощью описанной программы и спектр полученный экспериментально. Время, необходимое для расчета спектра, сопоставимого по точности с экспериментом, на кластере при использовании 16 процессоров составило 3 часа. Рассматривалось 500 различных ориентаций реакционных центров относительно внешнего магнитного поля. При этом по каждому направлению сверхтонкое взаимодействие оценивалось 100000 комбинаций случайных параметров.

Расчет МСМ-изображения ферромагнитной сферической наночастицы

Основной задачей магнитно-силовой микроскопии является восстановление структуры намагниченности образца по его МСМ-изображению. В настоящее время в общем случае решить такую задачу строго математически не представляется возможным. Поэтому при анализе МСМ-изображений решают обратную задачу: рассчитывают МСМ-изображение с учетом реальной геометрии и возможных распределений намагниченности иглы и образца, а затем сравнивают смоделированное МСМ-изображение с

экспериментальными данными. Принцип работы МСМ заключается в регистрации силового взаимодействия между зондом микроскопа и магнитным полем, создаваемым измеряемым образцом на зонде. Получаемое в МСМ изображение отображает распределение градиента магнитного поля вдоль поверхности образца, при перемещении магнитной иглы вдоль поверхности образца на определенной высоте.

При использовании численных методов магнитные области иглы и образца делятся на физически малые объёмы, каждый из которых аппроксимируется точечным магнитным диполем, а градиент $\frac{\partial F_{magz}(\mathbf{r})}{\partial z}$ Z-компоненты силы, действующей на иглу, вычисляется по формуле:

$$\frac{\partial F_{magz}(\mathbf{r})}{\partial z} = \sum_{\substack{i-tip \\ j-sample}} \left(\left(\mathbf{m}_t^i \nabla \right) \frac{\partial}{\partial z} H_z^{ij}(\mathbf{r}) \right),$$

где $\mathbf{m}_t^i$ – i-ый магнитный диполь иглы, $\boldsymbol{H_z^{ij}}$ – z-компонента магнитного поля, создаваемого j-ым магнитным диполем образца ($\mathbf{m}_s^j$) на i-ом магнитном диполе иглы. Компонента $\boldsymbol{H_z^{ij}}$ вычисляется по формуле:

$$H_z^{ij}(\mathbf{r}) = \frac{3\left(z + z_t^i + z_s^j\right)\left(\mathbf{m}_s^j \cdot (\mathbf{r})\right)}{|\mathbf{r}|^5} - \frac{m_{s\,z}^j}{|\mathbf{r}|^3}$$

Магнитное поле вокруг образца определяется структурой его намагниченности. В последнее время для моделирования этой структуры большую популярность приобрел программный комплекс OOMMF (Object Oriented Micromagnetic Framework), широко используемый для расчета структур намагниченности ферромагнитных микро- и наноструктур.

Методы компьютерного моделирования МСМ-изображений позволяют учитывать не только форму и распределение намагниченности в игле и образце, но и сложную траекторию движения иглы над образцом, что позволяет объяснить ряд особенностей в экспериментальных МСМ-изображениях, однако для получения качественного результата необходимо решать задачи с высокой вычислительной трудоемкостью. Использование современных методов параллельных компьютерных вычислений

позволяет существенно сократить время для моделирования МСМ-изображений.

МСМ-изображение представляет собой двухмерную матрицу значений градиента, вычисленных в различных точках траектории движения иглы микроскопа при сканировании. При этом величины градиентов в каждой точке вычисляются независимо друг от друга. Поэтому оказывается естественным алгоритм распараллеливания процедуры получения МСМ-изображения с разделением траектории иглы на число участков, равное числу вычислительных узлов. В результате чего можно рассматривать процедуру построения МСМ-изображения, как построение нескольких независимых изображений с их последующим объединением.

Для проведения численного эксперимента была использована модель ферромагнитной сферической наночастицы диаметром 100 нм, лежащей на немагнитной подложке. В качестве МСМ-зонда был взят немагнитный усеченный конус с закругленным кончиком с нанесенной на него ферромагнитной пленкой постоянной толщины. Деление магнитных объемов зонда и образца на ансамбли магнитных диполей осуществлялось путем разбиения магнитных частей с помощью кубической сетки с размером ячейки 10 нм, при этом наночастица состояла из 552 диполя, а зонд – из 7546.

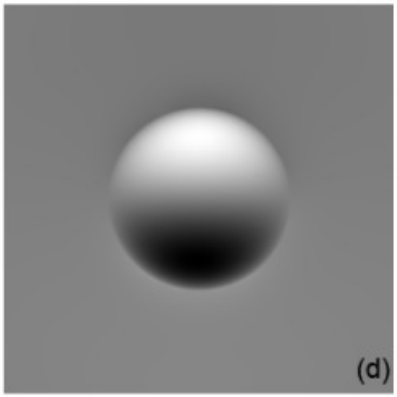


Рис. 2а

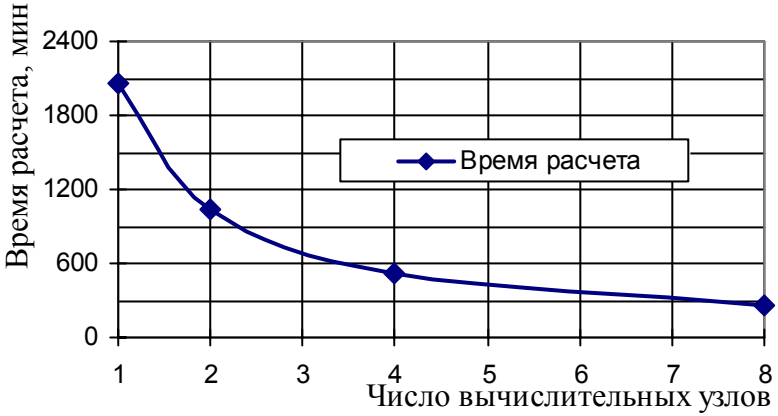


Рис. 2б

МСМ-изображение образца полученое расчетным путем представлено на рисунке 2а. График, представленный на рисунке 2б демонстрирует изменение времени расчета при изменении числа задействованных узлов кластера от 1 до 8.

Определение перемещений и напряжений в конструкции методом конечных элементов.

При численном моделировании механических процессов МКЭ исследуемая структура рассматривается как совокупность большого количества конечных элементов. В сложных объектах число элементов может достигать 1 миллиона и более. При наличии необходимых параллельных программ это делает естественным решение таких задач на вычислительных кластерах. Основные принципы реализации решения задач статики и динамики сложных структур МКЭ на параллельных вычислительных системах с распределенной памятью были описаны в работе [3]. В предлагаемом комплексе программ предусматривается моделирование структур конечными элементами балки, мембраны, пластины и оболочки. Элементы мембраны, пластины и оболочки могут быть как трехугольными, так и четырехугольными. Кроме этого, возможно использование элементов вязкого трения и осевых пружин. Комплекс реализован на языке программирования FORTRAN.

Семейство вычислительных кластеров

Для эффективного решения задач физики и механики в рамках небольшого НИИ или учебного учреждения было разработано семейство кластеров на базе одно- и двухядерных процессоров AMD.

Рис. 4

Все модели семейства представляли собой классический Beowulf-кластер и включали по 8 вычислительных узлов, коммутирующую аппаратуру, переключатель консоли и источник бесперебойного питания, смонтированные в открытой стойке (Рисунок 4). Особенности конструктивного исполнения кластеров и их технические характеристики приведены в таблице.

Модель	9A2000	8A64X2	16O246	16O270
Узел	7 одно- и 1 двухпроцессорный	Однопроцессорный	Двухпроцессорный	Двухпроцессорный
Процессор	Athlon XP/MP 2800	Athlon 64X2	Opteron 246	Opteron 270
Количество ядер	9	16	16	32
Сетевой адаптер	Gigabit Ethernet HardLink HA-32G	Gigabit Ethernet HardLink HS-8G и Marvell 88E1116	2 Gigabit Ethernet Broadcom BCM-5704	2 Gigabit Ethernet Broadcom BCM-5704
Сетевой коммутатор	Gigabit Ethernet HardLink HS-8G	Gigabit Ethernet D-Link DGS-1016D	Gigabit Ethernet D-Link DGS-1016D	Gigabit Ethernet D-Link DGS-1016D
Переключатель консоли	Edimax EK-16RO	Edimax EK-16RO	PCT MPC8701	PCT MPC8701
Источник бесперебойного питания	APC Smart-UPS 3000 RMI 5U	APC Smart-UPS 3000 RMI 5U	APC Smart-UPS 5000 RMI 5U	APC Smart-UPS 5000 RMI 5U
Производительность (пиковая), Гфлопс	7.5 (9.6)	37* (50)	42 (52)	80* (104)
Производительность сети: скорость передачи, Мбит/сек (латентность, мсек)	360 (170)	850 (45)	900 (38)	900 (38)

Литература

1. K.M.Salikhov, S.G.Zech, D.Stehlik “Light induced radical pair intermediates in photosynthetic reaction centres in contact with an

* характеристика определена на основании тестирования 2 узлов.

observer spin label: spin dynamics and effects on transient EPR spectra". Molecular Physics, 2002, Vol. 100, №9, 1311-1321.
2. A.Ponti "Simulation of Magnetic Resonance Static Powder Lineshapes: A Quantitative Assessment of Spherical Codes". Journal of Magnetic Resonance, 138, 288-297 (1999)
3. Черников С.К., "Метод подконструкций - эффективный инструмент распараллеливания алгоритмов в механике". Труды второго международного научно-практического семинара "Высокопроизводительные параллельные вычисления на кластерных системах", сс. 318-326. Н.Новгород: Издательство НГУ. 2002.

ОЦЕНКА ПАРАМЕТРОВ ТЕХНИЧЕСКОГО СОСТОЯНИЯ ТЕПЛОЭНЕРГЕТИЧЕСКИХ ОБЪЕКТОВ

Шагатдинова Э.М., Владова А.Ю.
Оренбургский государственный университет, Оренбург

В настоящее время более 70% теплоэнергетического оборудования исчерпало свой проектный ресурс. Газоконденсатопроводы и теплоэнергетическое оборудование (ТЭО) относятся к категории опасных производственных объектов (ОПО), отказы которых сопряжены со значительным материальным и экологическим ущербом. Увеличивающаяся продолжительность эксплуатации ОПО, износ оборудования выдвигают вопросы оценки технического состояния в ранг наиболее важных научных проблем.

Поэтому оценка параметров технического состояния теплоэнергетических объектов таких объектов является актуальной задачей. Таким образом, в качестве исследуемого объекта будет выступать ТЭО электростанций, представленное результатами измерений поврежденности теплоэнергетического оборудования, полученных в результате диагностирования. Базы данных объектов недавно прошедших диагностический осмотр составляет порядка нескольких тысяч элементов. А с учетом дальнейшего устаревания оборудования соответственно предусматривается увеличение количества проведения осмотров, что приведет в свою очередь к росту затрат на однотипные расчеты агрегированных моделей, в результате чего происходит дублирование операций над статическими массивами данных.

В последнее время все большие требования предъявляются к производительности разрабатываемых программных продуктов. Одним из путей повышения производительности может стать оптимизация кода, второй за счет организации конвейерной обработки или параллелилизма. Таким образом, применение параллельных вычислений в нашей задаче по оценке параметров технического состояния ТЭО является востребованным. 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.

Выходом из создавшейся ситуации является увеличение скорости вычислений за счет введения параллельной обработки массивов данных.

Целью задачи оценки параметров технического состояния теплоэнергетических объектов стало повышение производительности вычислений. А в качестве задач, призванных реализовать поставленную цель, были определены следующие: выбор оптимальной модели распараллеливания; разработка эффективного алгоритма; выбор среды и средств программирования; оптимизация разработанного параллельного кода.

Автоматизация задачи оценки параметров технического состояния теплоэнергетического оборудования обусловливает необходимость анализа повреждений элементов. Основной причиной отказов является превышение рабочей температуры и ускоренное протекание ползучести. Поэтому представляется актуальной оценка эквивалентной температуры эксплуатации поврежденных вследствие перегрева элементов ТЭО. Рассмотрим фрагмент алгоритма программы для определения эквивалентной температуры, вычисляемой по формуле $T=(a+b1*\sigma+c1*\sigma^2)*t+a2+b2*\sigma+c2*\sigma^2$, у которого одна точка входа и одна - выхода. Его можно очевидным способом представить в виде ориентированного графа передачи управления (блок-схемы), в котором вершинами являются операторы, а ребра указывают на очередность выполнения операторов.

Передачу управления сразу нескольким оператором обозначим несколькими ребрами, выходящими из одной вершины.

Для синхронизации параллельно выполняемых участков программы будем использовать оператор ожидания, в который будет

входить несколько ребер графа. Оператор ожидания передает управление следующему только после того, как получит управление от всех входящих в него вершин. Его можно представить как счетчик, изначально хранящий число ноль, после получения управления от одного из входящих в него ребра счетчик увеличивается на единицу. Когда значение счетчика станет равным количеству входящих ребер, он сбрасывается и передает управление следующим за ним операторам. Часто в программах присутствуют переменные для промежуточных вычислений. Одна и та же переменная может использоваться в нескольких выражениях, причем значение, присваиваемое ей в одном выражении, не используется в других. Для распараллеливания вычислений оказывается эффективным вместо одной такой переменной использовать необходимое количество различных. Опишем алгоритм построения схемы передачи управления. Для выявления инструкций, которые могут выполняться параллельно, построим схему потока данных. Для каждого оператора строим списки ресурсов, которые он использует и ресурсов, состояние которых изменяется в результате работы оператора. Создаем множество промежуточных переменных. Просматривая инструкции, начиная с последней, добавляем переменные по следующему правилу: если переменная X изменяется инструкцией, но не используется, то заменить X новой переменной во всех инструкциях, предшествующих данной.

Применяя перечисленные правила для рассматриваемого примера, получаем нужные отношения. Отсутствие зависимостей, означает, что эти инструкции могут быть выполнены одновременно, что сокращает общее время выполнения программы. По полученным отношениям строим граф передачи управления.

Следующим шагом стал выбор метода и языка программирования. С учетом задачи достижения максимальной масштабируемости останавливаемся на методе параллельной работы с данными OpenMP, предназначенном для повышения производительности за счет многозадачности и среде MS Visual C. В процессе написания кода решались вопросы количества используемых процессов, распределения нагрузки между ними и вопросы взаимодействия и синхронизации. Достигнутое ускорение (тестирование на двухпроцессорной машине) составило 1.33.

Разработка параллельных приложений - процесс трудоемкий, требующий специальных навыков. Сама параллельная программа

обычно реализует различные механизмы синхронизации, что усложняет программу и может привести к ошибкам. Масштабируемость параллельной программы достигается за счет аппаратного наращивания, затруднение вызывает распределение нагрузок по ресурсам.

ТЕХНОЛОГИЧЕСКИЕ АСПЕКТЫ СТРАТЕГИИ КОМПАНИИ SUN MICROSYSTEMS В ОБЛАСТИ ПОСТРОЕНИЯ РЕШЕНИЙ ДЛЯ РЫНКА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Шапошник Р.В.
Sun Microsystems, Menlo Park,USA

Введение

Компания Sun microsystems является на сегодняшний день одним из крупнейших поставщиков вертикальных решений, нацеленных на успешное применение в самых различных областях промышленности и научно-технического процесса. Одной из уникальных черт, отличающих компанию от ее конкурентов, является тот факт, что все аспекты конечной вычислительной системы (такие как апаратное обеспечение, програмное обеспечение и прикладные пакеты) производятся внутри компании, что приводит к возможности производства максимально производительной системы за счет детальной проработки интерфейсов всех компонент. По сравнению с прочими компаниями, являющимися во многом всего лишь интеграторами решений и производящих лишь несколько аспектов конечной системы, успех Sun microsystems обеспечивается именно уникальностью подхода.

В докладе будут рассмотрены варианты к оптимизации конечных вычислительных систем для рынка высокопроизводительных вычислений, в том числе параллельных и использующих кластерную архитектуру. Так же будет рассмотрен весь спектр программных продуктов, поставляемых с этими системами, и в заключении будет дан обзор самых успешных примеров, таких как Tsubame Supercomputer, являющийся на сегодняшний день седьмым по счету суперкомпьютером в мире по версии TOP500.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ СРЕДСТВ ОТЛАДКИ В ПАРАЛЛЕЛЬНОМ ПРОГРАММИРОВАНИИ ДЛЯ МНОГОЯДЕРНЫХ ПРОЦЕССОРОВ

Шипунов В.А.

Харьковский национальный университет радиоэлектроники, каф. автоматизации проектирования вычислительной техники, Харьков, Украина.

Введение

Основная концепция параллельного программирования для многоядерных процессоров состоит в проектировании, разработке и применению потоков в пределах приложения, и координации между потоками и их соответствующими операциями. Причины эффективности использования многопоточности в приложениях [1]:

- производительность – простой метод получить преимущества многоядерности.
- улучшение использования ресурсов – ключ к росту производительности вычислительной системы;
- эффективное использование данных – использование данных через память эффективнее передачи сообщений по сети.

Кроме выгод возникают определенные риски при использовании потоков – возрастает сложность приложения; трудности при отладке – возникают новые модели неисправностей, которых не было при последовательном программировании: гонки за данными (data races) и мёртвые хватки (deadlocks).

При разработке программного обеспечения, анализ производительности – это исследование поведения программы, используя информацию, собранную во время работы, как противоположность статическому анализу исходного кода. Цель анализа производительности – это определение узких мест в коде, какую часть программы оптимизировать по скорости или использованию памяти. Анализ выполняется при помощи профайлера – это инструмент анализа производительности, который измеряет характеристики поведения программы, пока она работает. Выходные данные – это поток записанных событий, или статистическая сумма наблюдаемых событий (профиль). Профайлеры используют широкое разнообразие способов сбора данных, включая прерывания

оборудования, исследование кода, методов операционной системы и счётчиков производительности.

Цель исследования – сравнительная характеристика наиболее распространённых средств анализа, профилирования и отладки.

Задачи исследования:

- поиск существующих средств отладки;
- изучение особенностей отдельных представителей;
- сравнительный анализ средств отладки;
- разработка рекомендаций по использованию средств.

Эта работа организована следующим образом: во втором разделе показаны существующие средства и архитектуры, в третьем разделе приведена сравнительная характеристика существующих программ анализа производительности. В заключении нами даны рекомендации по их использованию.

Средства параллельного программирования для многоядерных процессоров

OpenMP – средство для написания параллельных приложений. OpenMP это кроссплатформенное средство, поддерживающее компиляторы C, C++ и Fortran. Модель программирования OpenMP предоставляет набор платформо-независимых указаний транслятору (директив), вызовов функций и переменных среды, которые указывают компилятору места программы, которые нужно распараллелить.

WinAPI и библиотека MFC поддерживают разработку многопоточных приложений для Winodows. Используя эту библиотеку можно создавать потоки, привязывать к ним задания, следить за синхронизацией и уничтожать потоки. По сравнению с OpenMP, этот подход обладает большей гибкостью.

POSIX threads – переносимая библиотека, созданная с целью предоставления интерфейса программирования, независимо от операционной системы. Сейчас это стандартная технология распараллеливания для Linux и широко использующаяся на большинстве платформ UNIX, также доступна версия для Windows. Предоставляет больше контроля и обладает большей гибкостью, по сравнению с OpenMP.

Использование средств отладки при параллельном программировании

Этапы преобразования однопоточного приложения в многопоточное:

1. используя средства анализа производительности (Intel VTune Performance Analyzer [1,2], AMD Code Analyst [3], Rational Rose Rational Quantify, AutomatedQA AQTime, GProf) составить отчёт производительности;
2. рассчитать теоретическое ускорение и эффективность распараллеливания;
3. выполнить ряд тестов на приложении до распараллеливания, для сравнения результатов;
4. по графу вызовов, выбрать место в приложении, которое выполняется максимальное количество времени, которое использует большую часть данных;
5. используя средства распараллеливания (OpenMP, WinAPI, POSIX threads) распределить выполнение по потокам;
6. используя средства диагностики потоков (Intel Thread Checker, Rational Rose Rational Purify) протестировать приложение на предмет ошибок, связанных с обращением потоков к памяти;
7. выполнить анализ синхронизации потоков, при помощи средств анализа (Intel VTune Performance Analyzer, AMD Code Analyst, Rational Rose Rational Quantify), определить время, которое тратится на синхронизацию, ожидание и блокировку потоков;
8. запустить ряд тестов на приложении, проверить с исходными на правильность, а также сравнить время выполнения. Сделать выводы о эффективности распараллеливания.

Здесь важным является тот факт, что шаги поддерживаются различными средствами, и эффективность решения будет тем выше, чем эффективнее используется каждое средство в отдельности.

Сравнительная характеристика доступных средств отладки

Ниже (см. табл. 1-4) представлена сравнительная характеристика основных возможностей средств анализа и отладки параллельных приложений ведущих производителей: Intel (VTune Performance Analyzer, Thread Checker, Thread Profiler), AMD (Code Analyst), Rational (Rose, Quantify, Purify).

Таблица1: Статистика по средствам анализа производительности

	Intel	**AMD**	**Rational Rose**
Средство	VTune Performance Analyzer	Code Analyst	Rational Quantify, Rational Purify
Граф вызовов	Список вызовов, граф вызовов, сводки по функциям	Список вызовов, сводки по функциям	Граф, список вызовов, сводки по функции.
Событийная выборка	В виде списка модулей или функций, исходного кода	Списки системных данных, графики, списки процессов	Отсутствует
Временная выборка	В виде списка потоков и процессов	В виде списков системных данных, графика, списка процессов	Отсутствует
Статический анализ	В виде списка проблем или предупрежде ний	В виде исходного кода, инструкций, построчного моделирования	В виде списка ошибок, модулей, функций и исходного кода

Табл.2. Статистика по средствам диагностики потоков

	Intel	**AMD**	**Rational Rose**
Средство	Thread Checker	Отсутствуют	Rational Purify
Особенности	Адаптирован к OpenMP	Отсутствуют	Упор на анализ затрат памяти
Обнаружени е дефектов	Data races, deadlocks, stalled threads, lost signals, abandoned	Отсутствуют	Утечки памяти

	locks		
Управление памятью	Присутствует, информация о потоках, утечках памяти	Отсутствуют	Присутствует, информация о потоках в виде графа
Глубина анализа	Модули, функции, потоки, исходный код	Отсутствуют	Линии, модули, функции, исходный код

Таблица 3. Статистика по средствам анализа производительности потоков

	Intel	**AMD**	**Rational**
Средство	Thread Profiler	Code Analyst	Rational Quantify
Отображение результатов	Общее распределение, регионы, потоки, редактор данных, с предполагаемыми ускорениями	График с временной шкалой, таблица с информацией о потоках и памяти	Интерактивный график, с информацией о состоянии потоков
Особенности	Временная шкала, возможные решения проблем, предполагаемое ускорение	Информация о потоках/ядрах и обращениях к памяти	Интерактивность графика, анализ части программы, детальный вид каждого потока
Типы проблем	Проблемы синхронизации, времени блокировки	Проблемы синхронизации	Проблемы синхронизации, времени блокировки
Работа с библиотекам и	WinAPI, OpenMP, POSIX	WinAPI, OpenMP, POSIX	WinAPI, POSIX, .NET Threads.

Таблица 4: Общая характеристика средств

	Intel	**AMD**	**Rational Rose**
Микро-архитектуры	IA-32, Itanium®, XScale®	x86, AMD64	x86, IA32
Linux приложения	Да	Да	Да, также Unix
Java приложения	Да	Да	Да
.NET приложения	Да	Да	Да
Win64 приложения	Да	Да	Нет, планируется
Командная строка	Да	Да	Да
Интеграция	Visual Studio .NET	Панель инструментов в MS Visual Studio 2005	MS VS, MS VS.NET, Rational Visual Test, Robot, ClearQuest
Аппаратурные требования	32/64 bit Single/multi core Intel CPU	32/64 bit Single/Dual core AMD CPU	32/64 bit Single/multi core Intel®/AMD CPU
Средства	WinAPI, OMP, POSIX, .NET Threading	WinAPI, OMP, POSIX, .NET Threading	Visual C/C++/Basic, .NET, Java, Microsoft Word/Excel plug-ins.
Условия распростране-ния	Trial, 30 days full version free for registered developers	Free	Trial, 15 days full version free
Стоимость	Full - $699 academic ~$280	Free	Full license - $988

Выводы

Процесс разработки многопоточных приложений не обходится без использования средств анализа производительности, отладки, профилирования и диагностики потоков. Выбор используемого средства всегда должен быть мотивирован. Средства AMD свободнодоступны и имеют открытый исходный код, хотя они предоставляют не такой широкий спектр возможностей. Если при анализе потоков, основной требуемой информацией является утечки/затраты памяти, предпочтительнее использовать Rational Quantify.

Средства Intel предоставляют наиболее богатые возможности по анализу и диагностике потоков, тем самым фирма подтвердила своё технологическое лидерство в области средств разработки и анализа параллельных приложений для многоядерных процессоров.

Литература

1. Shameem Akhter, Jason Roberts. Multi-Core Programming. Intel Press, 2006. P. 336.
2. Intel VTune Performance Analyzer home page http://www.intel.com/cd/software/ products/asmo-na/eng/vtune/index.htm.
3. AMD Code Analyst Tool Help, http://developer.amd.com/articles.jsp?id=2&num=1, http://developer.amd.com/developercenter.jsp.

Контакты

vshipunov@yandex.ru, т./ф.: +380 (57) 702-13-26

ИСПОЛЬЗОВАНИЕ НЕЧЕТКИХ МНОЖЕСТВ В ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧАХ

Шубин А.Ю., Петрова А.В., Шубина Н.Ю.

Sun Microsystems, Санкт-Петербург

Введение

Задач, решаемых на кластерах, очень много. Среди них - задачи из метеорологии, астрономии, финансов, логистики и пр. Любая задача из

этих областей подразумевает интенсивные вычисления с использованием большого числа входных/вычисляемых параметров, измеренных с погрешностью или не всегда известных точно. В данном докладе рассмотрены следующие проблемы такого рода вычислений: погрешности вычислений, связанные с неточным представлением вещественных чисел в машинном виде, и их автоматическая оценка, погрешности входных данных, возникающие при наличии входных параметров, заданных с известной ненулевой ошибкой, и их влияние на конечный результат. Кроме этого рассмотрен вопрос снижения ресурсоемкости отдельных задач.

Проблема анализа погрешности численных результатов независимо от их природы – одна из фундаментальных в проблеме надежности вычислений. В качестве одного из способов решения этой проблемы предложено использовать математический аппарат нечетких множеств и интервальную арифметику.

Используются понятия теории нечетких множеств: «нечеткое множество» и «нечеткое число». Нечеткое множество – множество элементов, каждому из которых сопоставлено значение функции принадлежности с областью значений – [0,1]. Нечеткое число – нечеткое множество с выпуклой кусочно-непрерывной функцией принадлежности с максимумом равным единице.

Основное достоинство нечетких чисел – с ними можно работать как с обычными вещественными числами, оценки погрешности вычислений при этом получаются автоматически. Основной недостаток нечетких чисел – накладные расходы на операции с ними при реализации на ЭВМ: чем больше значений необходимо для представления множества, тем более ресурсоемкими становятся операции над ними.

Самым простым и наименее требовательным к вычислительным ресурсам примером нечеткого числа является диапазон значений, задаваемый двумя вещественными числами, с функией принадлежности строго равной 1 (далее – интервал).

Есть несколько программных продуктов, в которых реализована работа с нечеткими множествами вообще и с интервалами в частности: SunStudio Fortran и SunStudio C++ работают с интервалами как со встроенными типами, отдельные функции работы с интервалами реализованы в Intel Math Kernel Library и Sun Performance Library,

пакет MathLab предоставляет возможность работы с нечеткимимножествами и пр.

На примерах реализации простых численных методов показаны возможности интервалов и нечетких множеств по оценке ошибки вычислений. Также обсуждаются проблемы применения интервалов в решении вычислительных задач.

Приведены примеры оценки качества нескольких используемых в астрономии вычислительных программ с помощью замены вещественного типа на интервальный. Показано, что используя такую замену можно:

1. выбрать более корректную программу из нескольких сходных
2. достаточно быстро найти вычислительную некорретность в программе, например деление на разность близких величин
3. оценить сверху ошибку вычислений.

Также показаны преимущества и недостатки использования нечетких множеств при построении интеллектуальных систем и оценена ресурсоемкость такого подхода в сравнении с подходом, их не использующим.

ОЦЕНКИ ОПТИМАЛЬНОГО ЧИСЛА ПРОЦЕССОРОВ ОВС ДЛЯ НАИСКОРЕЙШЕГО ИСПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Юфрякова О.А.

ИИТ СГМУ,.Архангельск

В настоящее время принято разделять вычислительную сложность параллельного алгоритма (без учета времени на коммуникации) и коммуникативную сложность [2].

При этом постоянно делаются оговорки о том, что количество элементарных машин (ЭМ или узлов кластера или процессоров суперкомпьютера) разумно ограничить некоторым числом, определяемым производительностью сети коммуникаций и законами Амдала [1,2,3].

Но оценки этого «разумного» числа ЭМ не приводятся. Такие оценки вычисленные для различных архитектур были бы несомненно полезными для планирования эффективного использования ресурсов вычислительной системы (ВС).

Пусть t_c – среднее время выполнения локальных вычислительных операций (сложения, умножения, обращения к локальной памяти ЭМ). Характерный масштаб t_c – несколько десятков тактов таймера ЭМ.

Пусть t_s – среднее время пересылки элементарного пакета данных между соседними ЭМ по сети коммуникаций. Оно складывается из времени прохождения сигнала по кабелю, латентности и задержек при передаче данных при превышении пропускной способности.

Например, если тактовая частота процессора ЭМ 3 Ггц, т.е. $3 \cdot 10^9$ тактов в сек, то 1 такт = $0{,}33 \cdot 10^{-9}$ сек. Скорость света 300000 км/сек = $3 \cdot 10^8$ м/с. За один такт свет пройдет 1/10 м = 10 см. Таким образом каждый метр кабеля дает принципиально неустранимую задержку в 10 тактов. С ростом тактовой частоты эта задержка увеличивается.

Например, если латентность в сети коммуникаций ВС, как у Cray T3E-1200E, равна 1 мкс = 10^{-6} сек = 3300 тактов, то характерный время выполнения операций пересылки данных на два порядка превосходит время выполнения вычислительных операций. В сетях Ethernet латентность в 10-100 раз выше чем у Cray.

Мы видим, что именно операции пересылки данных являются наиболее медленными даже для специализированных суперкомпьютеров и являются узким местом всех параллельных ВС. Поэтому распределение данных должно быть по возможности крупноблочным и по вычислениям и по пересылке. Эти операции мы будем называть локальными.

Но в любом параллельном алгоритме присутствуют глобальные операции – создание виртуальной ВС, начальная рассылка данных, сбор результатов вычислений с ЭМ (broadcast, scatter, reduce, gather и т.д.). При оценке сложности о них обычно умалчивается или предполагается, что они уже выполнены. При этом говорится, что выполнена предварительная подготовка данных или, что данные уже распределены по памяти ЭМ.

На самом деле это существенная часть параллельного алгоритма, использующая массовые обмены данными между ЭМ. Поэтому при оценке сложности параллельных алгоритмов следует совместно рассматривать вычислительные и коммуникационные аспекты. Только на этом пути можно получить оценки для оптимального числа ЭМ, минимизирующего время исполнения параллельного алгоритма как функцию от объема исходных данных n.

Рассмотрим идеальный, с точки зрения законов Амдала, параллельный алгоритм A, исполняемый на однородной ВС. Введем следующие обозначения.

$C_A(n)$ – временная вычислительная сложность A, как ее понимают в классической теории сложности вычислений, т.е. асимптотика при $n \to \infty$ суммарного количества элементарных вычислительных операций необходимых для последовательного исполнения алгоритма A на данных объемом n в худшем случае.

$S^l_A(n)$ – локальная коммуникационная сложность A, т.е. асимптотика (при $n \to \infty$) суммарного количества локальных элементарных операций пересылки данных между соседними ЭМ, необходимых для исполнения параллельного алгоритма A при заданной топологии ОВС на задаче объемом n в худшем случае.

$S^g_A(n)$ – коммуникационная сложность глобальных операций алгоритма A – количество элементарных операций пересылки данных на каждую (одну) ЭМ при инициализации виртуальной ВС, начальном распределении данных, глобальной синхронизации и так далее. Общее время исполнения этих операций, очевидно, пропорционально количеству P элементарных машин.

Тогда время исполнения всего параллельного алгоритма A на ОВС из P ЭМ будет равно:

$$T_A(n,p) = \frac{C_A(n)\cdot t_c + S^l_A(n)\cdot t_s}{P} + S^g_A(n)\cdot P\cdot t_s$$

Найдем то количество ЭМ P, при котором время исполнения алгоритма будет минимальным. Как функция от P это выражение имеет ярко выраженный единственный минимум при $P > 0$, поэтому ее минимум найдем методами классического анализа функций.

Утверждение. Время исполнения идеального параллельного алгоритма A на ОВС из P ЭМ будет наименьшим при

$$P \approx \left[\left(\frac{C_A(n)\cdot t_c + S^l_A(n)\cdot t_s}{S^g_A(n)\cdot t_s} \right)^{\frac{1}{2}} \right]$$

Замечание 1. P зависит от графа коммутаций и не может быть произвольным натуральным числом. Например, для гиперкуба $P = 2^N$,

для *2D*-тора *P= MxN* Поэтому *P* надо брать ближайшим возможным числом к приведенной оценке.

Замечание 2. Для увеличения масштабируемости необходимо уменьшать t_s и $S^g_A(n)$, выражая глобальные коммуникационные операции через локальные, тем самым распараллеливая глобальные операции.

При определении эффективности параллельных алгоритмов сравнивают времена исполнения $T_A(n,1)$ и $T_A(n,P)$. На одном процессоре нет операций коммутации, значит $S^g_A(n)=0$ и $S^l_A(n)=0$. Поэтому «полезными» считают только вычислительные операции. Разобьем $T_A(n,P)$ на две части – «полезную» и коммуникационную:

$$T_A(n,p) = \underbrace{\frac{C_A(n)\cdot t_c}{P}}_{\text{«полезная»}} + \underbrace{\frac{S^l_A(n)\cdot t_s}{P} + S^g_A(n)\cdot P\cdot t_s}_{\text{«коммуникационная»}}$$

В силу выше сказанного, естественным кажется требование:

$$\frac{C_A(n)}{P} \geq \frac{S^l_A(n)}{P} + S^g_A(n)\cdot P$$

– «полезные» операции должны составлять не меньшую часть, чем «бесполезные». Это требование приводит также к ограничению для числа ЭМ *P*

$$P \leq \sqrt{\frac{C_A(n)\cdot t_c - S^l_A(n)\cdot t_s}{S^g_A(n)\cdot t_s}},$$

при условии, что $C_A(n)\cdot t_c \geq S^l_A(n)\cdot t_s$.

Последнее неравенство не зависит от *P* и ограничивает разумное количество локальных пересылок данных $S^l_A(n) \leq C_A(n)\frac{t_c}{t_s}$ в «приличных» параллельных алгоритмах.

По сути приведенные ограничения являются еще одним дополнительным коммуникационным аспектом законов типа Амдала.

Было бы интересно проверить эти ограничения на конкретных параллельных алгоритмах в различных вычислительных средах.

Литература

1. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
2. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. – Нижний Новгород: ННГУ, 2000. – 176с.
3. Корнеев В.Д. Параллельное программирование в MPI. – Новосибирск: ИВМиМГ СО РАН, 2002. – 215с.

ОГЛАВЛЕНИЕ