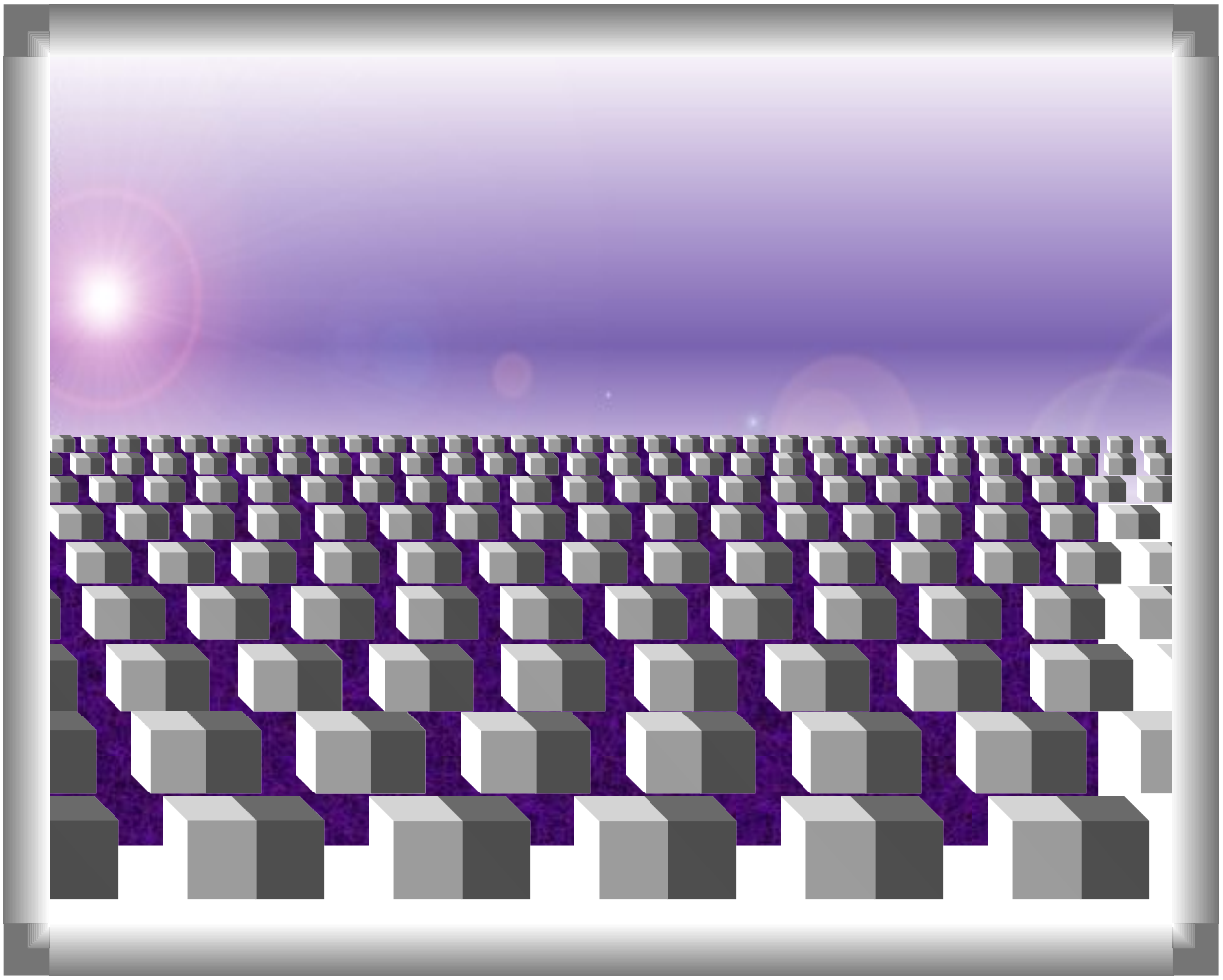

Сибирская школа-семинар по параллельным вычислениям

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИБИРСКАЯ ШКОЛА-СЕМИНАР ПО ПАРАЛЛЕЛЬНЫМ ВЫЧИСЛЕНИЯМ

Томск, 17-20 декабря 2001 года

Под редакцией доктора физико-математических наук
профессора А.В.Старченко

ИЗДАТЕЛЬСТВО ТОМСКОГО УНИВЕРСИТЕТА
2002

УДК 519.6
ББК 22.19
С 34

Сибирская школа-семинар по параллельным вычислениям
С 34 /Под ред. проф. А.В.Старченко. – Томск: Изд-во Том. ун-та, 2002. –
129 с.
ISBN 5 – 7511 – 1508 - 2

В сборнике содержатся материалы, представленные во время работы Сибирской школы-семинара по параллельным вычислениям, проходившей 17-20 декабря 2001 года в Томском государственном университете при поддержке Министерства образования РФ.

В докладах участников школы-семинара рассмотрены актуальные проблемы организации параллельных вычислений на многопроцессорных системах, современное состояние и перспективы развития методов параллельных вычислений, методические основы подготовки и повышения квалификации специалистов в области высокопроизводительных вычислений.

Для студентов, аспирантов, преподавателей, научных работников, желающих изучить и практически использовать в научной работе высокопроизводительные вычислительные ресурсы.

УДК 519.6
ББК 22.19

ISBN 5 – 7511 – 1508 - 2 © Томский государственный университет, 2002

К ПРОБЛЕМЕ ПОДГОТОВКИ СПЕЦИАЛИСТОВ ПО ВЫСОКОПРОИЗВОДИТЕЛЬНЫМ ВЫЧИСЛЕНИЯМ

А.Н.Тихонов, А.К.Скуратов, В.Г.Домрачев, И.В.Ретинская

ГНИИ ИТТ «Информика»

Современная наука и техника не могут обойтись без суперкомпьютерных технологий. Только с их использованием можно оставаться в ряду развитых в технологическом отношении стран, поскольку без параллельных вычислений нет альтернативы в возможности получения значимых для человечества новых научных и технических результатов. Лишь на их основе возможны серьезные достижения в космонавтике и авиации, биологии и медицине, обеспечение экологической безопасности, создание новых материалов и изделий микроэлектроники и многое другое.

Поэтому не случайно США, Япония и другие индустриальные страны мира уделяют развитию параллельных вычислений приоритетное внимание, считая это важнейшей государственной проблемой. Ее решение ведется в нескольких аспектах – создание усилиями многих фирм развитого парка высокопроизводительных суперкомпьютеров и программного обеспечения, организация инфраструктуры удаленного быстродействующего доступа. Каждые полтора года происходит утраивание производительности суперкомпьютеров, среднее время жизни одного суперкомпьютера составляет всего четыре года.

В нашей стране развитию параллельных вычислений также уделяется большое внимание. Однако необходимость привлечения для этого огромных финансовых ресурсов заставляет специалистов идти несколько иными путями в решении этой проблемы. Наряду с традиционным вариантом, когда ставка делается на суперкомпьютеры, в том числе и отечественной разработки, например МВС 1000, создаются кластерные системы на основе персональных компьютеров и рабочих станций. Это направление становится особенно привлекательным из-за меньших финансовых затрат, но имеет большие особенности в плане ограничения универсальности создаваемых вычислительных ресурсов. Здесь важное значение приобретает необходимость обеспечения удаленного доступа к ресурсам, поскольку это не только увеличивает число пользователей, но и создает большие возможности для кооперации.

В связи с этим, наряду с организацией Межведомственного суперкомпьютерного центра и развитием базы суперкомпьютеров в академических научно-исследовательских институтах, огромное значение приобрело созда-

ние сети центров высокопроизводительных вычислений в рамках ФЦП «Интеграция». Каждый центр строится на принципе кооперации университетов и институтов РАН в каком-либо регионе страны. На первых этапах организации центров основное внимание было уделено созданию вычислительных мощностей и сетевых связей. Это процесс сложный и долголетний. Фактически закладывается лишь основа таких центров. Далее они должны постоянно совершенствоваться, поскольку как вычислительные среды, так и средства телекоммуникаций развиваются энергичными темпами. Более того, необходимо и расширение географии центров.

В сложившейся ситуации, когда существует огромное число требующих решения фундаментальных и прикладных научных задач и идет весьма энергичное (по нашим меркам и при наших масштабах финансирования) создание вычислительных и телекоммуникационных ресурсов, остро встала проблема подготовки кадров для проведения параллельных вычислений. Прежде чем перейти к сути проблемы, рассмотрим кратко этапы ее постановки и решения.

В явном виде проблема была наиболее зримо инициирована на Международной научно-методической конференции «Телематика-2000» 29 мая – 1 июня 2000 года. Здесь были учтены предложения ряда вузов и НИИ РАН, опыт передовых университетов, уже занимающихся подготовкой таких специалистов в рамках различных специальностей, решение Координационного совета по использованию высокопроизводительных вычислений и информационных ресурсов в сфере образования, предложения Государственного НИИ информационных технологий и телекоммуникаций «Информика».

Новый импульс необходимости решения проблемы был дан на рабочем совещании «Использование супер-ЭВМ в процессе интеграции высшего образования и научных исследований», проведенном 12-18 июня 2000 года в г.Ростове-на-Дону ГНИИ ИТТ «Информика» и Ростовским государственным университетом.

В июле 2000 года распоряжением заместителя министра образования РФ Б.А.Виноградова была создана экспертная комиссия по подготовке специалистов в области высокопроизводительных вычислений под председательством директора ГНИИ ИТТ «Информика» А.Н.Тихонова.

Для конструктивной работы экспертной комиссии была организована рабочая группа, включающая ведущих специалистов по проблеме из разных городов страны, под председательством академика А.С.Бугаева. Работа комиссии и рабочей группы проходила в ГНИИ ИТТ «Информика».

Принципом работы комиссии была гласность и стремление привлечь к участию все заинтересованные вузы и НИИ РАН. Был составлен перечень вопросов по существу проблемы подготовки специалистов по высокопроизво-

дительным вычислениям, который был разослан всем участникам работ в области параллельных вычислений по ФЦП «Интеграция», а также по адресам проявляющих в этом направлении активность университетов и НИИ. Одновременно информация была выставлена на сервере ГНИИ ИТТ «Информика» с приглашением проявлять инициативу всем желающим.

Поступившие предложения были проанализированы, обобщены и в сконцентрированном виде доложены научно-техническому совещанию «Создание телекоммуникационной среды высокопроизводительных технологий в регионах России: состояние, проблемы» (и одновременно Координационному совету по использованию высокопроизводительных вычислений и информационных ресурсов в сфере образования под председательством заместителя министра образования РФ Б.А.Виноградова), которое состоялось 24-26 сентября 2000 года в Уфимском государственном авиационном техническом университете (докладчик В.Г.Домрачев). В принятом решении рекомендовано следовать стратегии подготовки специалистов по параллельным вычислениям в рамках новых специализаций пока без открытия дополнительных специальностей.

На заключительном этапе работы как интегральный результат рассмотрения всех представленных материалов и высказанных предложений была конкретизирована стратегия подготовки специалистов по высокопроизводительным вычислениям и определены конкретные специализации, в рамках которых целесообразно готовить специалистов. Результаты конкретизации были еще раз апробированы при проведении круглого стола на тему: «Подготовка специалистов по высокопроизводительным вычислениям» (ведущие А.К.Скуратов и И.В.Ретинская) на Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения» в Черноголовке 30 октября 2000 года.

На последнем этапе работы необходимо особо отметить конструктивное участие академиков В.П.Мясникова, А.С.Бугаева, специалистов Московского, Санкт-Петербургского, Пензенского, Уральского государственных университетов, Санкт-Петербургского института высокопроизводительных вычислений и баз данных, Уфимского государственного авиационного технического университета, Московского государственного института радиотехники, электроники и автоматики (при участии кафедры в Межведомственном суперкомпьютерном центре) и Государственного научно-исследовательского института информационных технологий и телекоммуникаций «Информика».

Основные положения стратегии подготовки специалистов в области высокопроизводительных вычислений сводятся к следующему:

- отдавая дань уважения тем университетам, которые уже ведут подготовку вышеназванных специалистов в соответствии с потребностями

своего региона и возможностями преподавательского корпуса в рамках удобных им специальностей, необходимо введение в такую подготовку организационных начал, построенных на коллективном мнении ведущих специалистов страны;

- учитывая, что в настоящее время потребности в специалистах по высокопроизводительным вычислениям не носят массового характера, но одновременно прослеживается тенденция значительного расширения объема таких вычислений, например за счет построения кластеров, организационные начала на текущий момент должны носить рекомендательный характер; после апробации предложений на практике и прояснения ситуации с потребностью в специалистах их статус должен быть повышен;
- подготовка специалистов должна быть организована в следующих направлениях – вузовское (университетское) образование, обучение в аспирантуре, повышение квалификации дипломированных и недипломированных специалистов, то есть охватывать общеизвестные этапы появления и функционирования специалиста;
- подготовку специалистов по высокопроизводительным вычислениям – выпускников вузов целесообразно осуществлять поэтапно; в настоящее время – в рамках существующих специальностей с введением новых специализаций; на втором этапе – как в рамках новых введенных специальностей, так и в рамках специализаций;
- должна быть организована подготовка и переподготовка с обязательной аттестацией преподавателей по дисциплинам высокопроизводительных вычислений с тем, чтобы исключить обучение на непрофессиональном уровне;
- необходимо иметь в стране координационно-аналитический центр по основным аспектам подготовки специалистов в области высокопроизводительных вычислений, обладающий всей информацией по проблеме, разрабатывающий предложения по ее решению и выполняющий различные текущие функции, например по сбору и предоставлению учебно-методических материалов; центр должен оптимизировать интересы НИИ РАН и университетов и не должен быть в структуре какого-либо конкретного вуза;
- целесообразно открытие кафедр по высокопроизводительным вычислениям в ограниченном числе ведущих университетов страны;
- проблема подготовки специалистов по высокопроизводительным вычислениям должна регулярно обсуждаться на конференциях и совещаниях в силу динамичности развития ситуации; необходимы изучение и учет зарубежного опыта подготовки таких специалистов.

Ниже излагаются практические пути подготовки специалистов по высокопроизводительным вычислениям в настоящее время.

В рамках вузовского обучения целесообразно вести подготовку специалистов по трем специализациям:

- специализация «Высокопроизводительные вычислительные технологии и системы» в специальности 220100 «Вычислительные машины, комплексы, системы и сети» направления 654600 «Информатика и вычислительная техника»; готовящиеся специалисты ориентируются на знания вычислительных средств и телекоммуникаций для параллельных вычислений;
- специализация «Математическое и программное обеспечение высокопроизводительных вычислительных систем» специальности 010200 «Прикладная математика и информатика»; будущие специалисты ориентируются на особенности математического и программного обеспечения параллельных вычислений;
- специализация «Вычислительные высокопроизводительные системы и параллельные вычисления»; эта специализация может находиться в рамках различных специальностей и ориентирована на подготовку специалистов по этим специальностям со знаниями в области алгоритмизации и программирования с акцентом на параллельные вычисления.

По первой специализации «Высокопроизводительные вычислительные технологии и системы» образовательным стандартом и учебным планом на специальность отведено 510 часов. Предлагается следующий учебный план специализации .

№ п/п	Наименование дисциплин	Всего часов			ПРИМЕРНОЕ распределение по семестрам (указать крестиком)										Форма итогового контроля (экзамен / зачет)
		Трудоемкость	Из них												
			Аудиторные	Самостоятельная работа											
					1	2	3	4	5	6	7	8	9	10	
1	Аппаратные средства высокопроизводительных вычислительных систем	170	136	34									х		Экзамен
2	Параллельное программирование в многопроцессорных вычислительных системах	170	136	34									х		Экзамен
3	Технологии и языки параллельного программирования.	170	136	34									х		Экзамен

Ниже приводятся предлагаемые рабочие программы дисциплин специализации.

Аппаратные средства высокопроизводительных вычислительных систем

Аудиторные занятия – 136 часов, самостоятельная работа – 34 часа.
Форма итогового контроля – экзамен.

Введение. Способы оценки производительности ВС.

Архитектура вычислительных систем. Создание многопроцессорных вычислительных систем (МВС) как основной путь повышения производительности ВС.

Проблемы выполнения сети связи процессоров в МВС. Организация когерентности многоуровневой иерархической памяти. Принципы построения коммуникационных сред. Примеры построения коммуникационных сред на основе масштабируемого когерентного интерфейса SCI, коммуникационные среды MYRINET, Raceway, коннектор шин PCI, коммуникационные среды на базе транспьютероподобных процессоров, телекоммуникационные технологии.

Математические основы, способы организации и особенности проектирования высокопроизводительных процессоров: ассоциативные, конвейерные, матричные, клеточные и комбинаторные процессоры, однородные и неоднородные процессорные матрицы, коммуникационные процессоры, процессоры баз данных, потоковые процессоры, нейронные процессоры и процессоры с многозначной (нечеткой) логикой.

Коммутаторы для МВС. Простые коммутаторы: алгоритмы арбитража, особенности реализации шин. Составные коммутаторы: коммутатор Клоза, Баньян-сети. Распределенные составные коммутаторы, графы межмодульных связей Convex Exemplar SPP1000, МВС-100 и МВС-1000.

Кластеры и массово-параллельные системы различных производителей. Требования к компонентам МВС. Примеры реализации МВС:

- SMP Power Challenge фирмы Silicon Graphics, семейство SUN Ultra Enterprise фирмы SUN, семейство массово-параллельных ВС МВС-100 и МВС-1000;

- ВС с распределенной памятью фирм SEQUENT и DATA GENERAL, Современные кластеры DIGITAL. Современные суперкомпьютеры – Cray T3E-900. Надежность и отказоустойчивость МВС.

Параллельное программирование в многопроцессорных вычислительных системах

Аудиторные занятия – 136 часов, самостоятельная работа – 34 часа.
Форма итогового контроля – экзамен.

Развитие параллельного программирования.

Организация параллельных вычислений. Масштабирование и переносимость: проблемы перехода от последовательных программ к параллельным. Проблемы организации параллельных вычислений.

Параллельные и распределенные базы данных. Мотивация создания параллельных и распределенных баз данных. Виды параллельной обработки в базах данных. Параллельное выполнение операций.

Стандарт интерфейса передачи сообщений MPI. Базовые понятия, управление группой. Процедуры парных межпроцессорных обменов. Коллективные взаимодействия процессов.

Стандарт Open MP. Основные положения стандарта, управляющие структуры, синхронизация, библиотека времени исполнения.

Операционная система КОСМОС как пример системы для параллельного программирования.

Технологии и языки параллельного программирования

Аудиторные занятия – 136 часов, самостоятельная работа – 34 часа.

Форма итогового контроля – экзамен.

Проблема распараллеливания вычислительных алгоритмов.

Распределение ресурсов многопроцессорной ВС различным задачам.

Программные средства порождения процессов. Процессы и критические секции. Синхронизация процессов и совместное использование ресурсов.

Синхронизирующие примитивы. Синхронизация процессов посредством семафоров. Условные критические интервалы.

Мониторы. Работа в режиме удаленного доступа.

Дедлоки и защита от них.

По второй специализации «Математическое и программное обеспечение высокопроизводительных вычислительных систем» образовательным стандартом и учебным планом отведено 1428 часов. Предлагается следующий учебный план специализации.

№ п/п	Наименование дисциплин	Всего часов			П Р И М Е Р Н О Е распределение по семестрам (указать крестиком)										Форма итогового контроля (экзамен / зачет)
		Трудоемкость	Из них												
			Аудиторные	Самостоятельная работа											
					1	2	3	4	5	6	7	8	9	10	
1	Архитектуры высокопроизводительных компьютерных систем	204	120	84					x	x					Экзамен
2	Алгоритмы и языки параллельного и векторного программирования	204	120	84					x	x					Экзамен, курсовая работа
3	Операционные системы и системное программирование для суперкомпьютеров	204	120	84						x	x				Экзамен, курсовая работа
4	Высокопроизводительные технологии в интегрированных интеллектуальных системах	204	120	84							x				Зачет, курсовая работа
5	Средства телекоммуникации	124	72	52									x		Зачет
6	Высокопроизводительные системы управления базами данных	90	52	38									x		Зачет, курсовая работа
7	Кластерные системы	72	44	28								x			Зачет

№ п/п	Наименование дисциплин	Всего часов			ПРИМЕРНОЕ распределение по семестрам (указать крестиком)										Форма итогового контроля (экзамен / зачет)
		Трудоемкость	Из них												
			Аудиторные	Самостоятельная работа											
					1	2	3	4	5	6	7	8	9	10	
1	Архитектуры высокопроизводительных компьютерных систем	204	120	84					x	x					Экзамен
2	Алгоритмы и языки параллельного и векторного программирования	204	120	84					x	x					Экзамен, курсовая работа
3	Операционные системы и системное программирование для суперкомпьютеров	204	120	84						x	x				Экзамен, курсовая работа
8	Средства мультимедиа, научной визуализации и виртуальной реальности	102	60	42								x			Зачет, курсовая работа
9	Прикладное программирование в высокопроизводительных вычислительных системах	156	90	66								x	x		Экзамен, курсовая работа
10	Планирование вычислительного эксперимента на суперкомпьютерах	68	42	26									x		Зачет, курсовая работа

№ п/п	Наименование дисциплин	Всего часов			ПРИМЕРНОЕ распределение по семестрам (указать крестиком)										Форма итогового контроля (экзамен / зачет)
		Трудоемкость	Из них												
			Аудиторные	Самостоятельная работа											
					1	2	3	4	5	6	7	8	9	10	
1	Архитектуры высокопроизводительных компьютерных систем	204	120	84					x	x					Экзамен
2	Алгоритмы и языки параллельного и векторного программирования	204	120	84					x	x					Экзамен, курсовая работа
3	Операционные системы и системное программирование для суперкомпьютеров	204	120	84						x	x				Экзамен, курсовая работа
		1428													

Предлагаемые рабочие программы дисциплин этой специализации приводятся ниже.

Архитектуры высокопроизводительных компьютерных систем

Аудиторные занятия – 120 часов, самостоятельная работа – 84 часа.

Модель компьютера фон Неймана. Ограничения на производительность вычислительных систем, необходимость параллелизма.

Классификации компьютерных архитектур, признаки классификации. Архитектуры NUMA, cc-NUMA, Numa flex, SMP, MPP, PVP и кластерных систем. Топологии MPP систем. Устройство компьютера. Взаимодействие с периферией.

Коммуникационные среды (SCI, Myrinet, Raceway, MC и др.).

Общие принципы построения параллельных систем. Параллельность внутри процессора: конвейерный способ обработки. Ускорение при параллельных вычислениях, учет времени коммуникаций, закон Амдаля.

Архитектуры процессоров: CISC и RISC процессоры, суперскалярные, конвейерные, векторные и др.

Коммутаторы вычислительных систем. SMP, массивно-параллельные, кластерные, векторные и пр. Системы различных производителей (SGI, Cray, SUN, HP/Convex, Compaq, Hitachi, Fujitsu, NEC и др.).

Надежность и производительность суперкомпьютерных систем. Система и общие принципы измерения производительности, единицы измерения производительности. Стандартные тестовые пакеты.

Компьютерные сети: принципы построения, аппаратные средства. Типы сетевых соединений, протоколы сетей. Интернет: протоколы, структура, службы. Организация единого компьютерного ресурса.

Альтернативные модели компьютеров. Нечеткие и нейропроцессоры. Нейросетевые вычислительные системы.

Алгоритмы и языки параллельного и векторного программирования

Аудиторные занятия – 120 часов, самостоятельная работа – 84 часа.

Развитие параллельного программирования. Организация эффективных параллельных вычислений. Векторизация. Парадигма векторных операций (SIMD). Эффективность векторных вычислений. Параллельные и распределенные базы данных.

Программные средства организации параллельных приложений. Стандарт интерфейса передачи сообщений MPI. Стандарт OpenMP. Парадигма программирования методом обмена сообщениями (PVM, MPI/MPICH). Системы с общей памятью, расширение MPI в системах с общей памятью, модель взаимодействующих процессов UNIX IPC. Сравнение различных программных средств.

Операционные системы и системное программирование для суперкомпьютеров

Аудиторные занятия – 120 часов, самостоятельная работа – 84 часа.

ОС как интерфейс между пользователем и аппаратными средствами. Командный и графический интерфейс.

Задачи, которые должна решать ОС (доступ к периферийным устройствам, управление одновременными процессами и т.п.).

Различие операционных систем. Однозадачные системы. Системы разделения времени. Многозадачные системы. Многопользовательские системы.

Квантование времени, понятие приоритетности процесса. Временные характеристики процесса (время CPU, пользовательское и системное время).

Реальное время. Обзор рынка операционных систем (MS-Windows, OS/2, UNIX, VAX/VMS и т.п.). Операционные системы реального времени. Требования к ОСРВ. Различие между понятиями ОС и операционной среды. Типы операционных сред (Windows 3.11, Parix, MPI, PVM и т.п.).

Операционная система UNIX (файловая система, программы-оболочки, основные команды, редакторы, X-Window и т.д.)

Системное администрирование в UNIX (работа с пользователями, резервное хранение данных, организация безопасной работы).

Администрирование сети (TCP/IP, DNS, SLIP/PPP). Использование Интернет.

Высокопроизводительные технологии в интегрированных интеллектуальных системах

Аудиторные занятия – 120 часов, самостоятельная работа – 84 часа.

Концептуальные основы разработки интегрированных интеллектуальных систем (ИС). Архитектуры и классификация. Вычислительные платформы.

Интегрированные системы знаний в ИС параллельного действия. Приобретение и формализация знаний. Модели представления знаний и механизмы вывода. Стратегия и методы поиска решений. Анализ альтернатив и принятие решений.

Принципы организации и функционирования ИС реального времени (РВ). Измерительная компонента, интеллектуальный датчик. Интеллектуальный интерфейс, когнитивная графика, средства мультимедиа и виртуальной реальности в ИС РВ.

Концепция "мягких вычислений" в ИС РВ. Нечеткий логический базис, искусственные нейронные сети и генетические алгоритмы в моделях формирования знаний и механизме логического вывода. Распознавание и прогноз поведения динамических систем.

Распознавание образов на базе высокопроизводительных платформ.

Общая концепция многоагентных систем (МАС). Реализация принципа распределенного интеллекта в МАС. Архитектура и классификация МАС. Технология мобильных агентов. Накопление и формирование знаний. Сценарии знаний, логический вывод и принятие решений. Организация кооперативной работы интеллектуальных агентов. Обучение МАС. Интеллектуальные агенты РВ.

Средства телекоммуникации

Аудиторные занятия – 72 часа, самостоятельная работа – 52 часа.

Необходимость компьютерных сетей. Коммуникационные среды (Ethernet, Apple Talk, FDDI и т.п.). Сетевые протоколы. Модель OSI. TCP/IP (четырехуровневая модель, IP-адресация, подсети, маршрутизация, DNS), NetBIOS (WINS, DHCP и т.п.).

Безопасность в компьютерных сетях (брандмауэры и т.п.).

Коммуникационные среды высокопроизводительных систем (SCI, Myrinet, ServerNet, Quadrics и др.).

Интернет. Топология сети. Программные средства. Поисковые системы.

Интеграция информационной и телекоммуникационной технологии. Парадигма программирования многоагентных систем (МАС). Принципы распределенности, открытости и кооперативной работы. Языки программирования МАС.

Организация кооперативной работы интеллектуальных агентов в среде с распределенным интеллектом. Применение МАС в работе глобальной сети Интернет, организации дистанционного обучения и т.п.

Высокопроизводительные системы управления базами данных

Аудиторные занятия – 52 часа, самостоятельная работа – 38 часов.

Высокопроизводительные системы управления базами данных – определение и характеристики. Характеристики СУБД и их значения, позволяющие отнести систему к категории высокопроизводительных систем.

Технологии, применяемые для достижения высокой производительности.

Классификация конкретных СУБД относительно принадлежности к высокопроизводительным (DB2, Oracle, ADABAS, Sybase, Informix, Raima Database Manager++, Microsoft SQL Server 95, Advantage Database Server, Adaptive Server Enterprise, Pervasive, SQL 2000, Cache' (InterSystems), Informix XPS и Oracle Parallel Server).

Необходимая аппаратная реализация сервера.

Кластерные системы

Аудиторные занятия – 44 часа, самостоятельная работа – 28 часов.

История кластерных систем. Проект Beowulf. Современный уровень развития кластерных архитектур.

Кластеры и фермы процессоров.

Принципы программирования в кластерных системах. Системы передачи сообщений, общая память.

Системы пакетной обработки (DQS, Codine, PBS, Condor, Dynamite и др.). Параллельные языки и средства программирования.

Средства мультимедиа, научной визуализации и виртуальной реальности

Аудиторные занятия – 60 часов, самостоятельная работа – 42 часа.

Парадигма мультимедийных технологий.

Основы компьютерного дизайна. Цветоделение. Визуальные эффекты компьютерной графики.

Основы компьютерной анимации. 3D MAX, Maya и другие программные средства. Стандарт OpenGL.

Роль визуализации в научных исследованиях. AVS, TechPlot. Когнитивные визуальные образы.

Аппаратные графические средства для персонального компьютера. Акселераторы, средства ввода и обработки графической информации.

Высокопроизводительные графические системы и средства виртуальной реальности. Аппаратные и программные средства (SGI Onyx2, SGI Octane, SGI O2 и др.).

WEB-технологии. Программирование Интернет-приложений. Язык Java как платформонезависимое средство создания приложений.

Прикладное программирование в высокопроизводительных вычислительных системах

Аудиторные занятия – 90 часов, самостоятельная работа – 66 часов.

Программное обеспечение для векторно-параллельных систем. Компиляторы, отладчики, профайлеры. Пакеты прикладных программ. NAG, Linpack, Eispack, IMSL и др.

Оптимизация прикладных программ. Скалярная, векторная, параллельная оптимизация. Методика оптимизации программ, программирование эффективных конструкций. Ограничения, накладываемые на оптимизацию.

Особенности применения численных методов на векторно-параллельных системах. Различные методы построения конечно-разностных схем. Проблемы устойчивости. Способы построения итерационных алгоритмов для решения существенно нелинейных дифференциальных уравнений. Методы построения расчетных сеток.

Многосеточные методы (ММ). Компоненты ММ. Применение ММ к решению нелинейных задач. Использование ММ для решения нестационарных задач. Анизотропные и параболические проблемы. Параллельные аспекты программной реализации, параллельный ММ и схемы релаксации, проблемы на грубых сетках.

Решения задач математической физики на векторно-параллельных системах. Особенности численных реализаций сложных задач математической физики.

CAD/CAM/CAE и геоинформационные системы на суперкомпьютерах. Обзор промышленных компьютерных пакетов, реализованных на векторно-параллельных системах.

Методы и средства программирования высокопроизводительных интеллектуальных систем (ИС). ИС реального времени. Инструментальные средства и языки программирования ИС. Моделирование в ИС реального времени. Прикладные ИС с параллельной архитектурой.

Планирование вычислительного эксперимента на суперкомпьютерах

Аудиторные занятия – 42 часа, самостоятельная работа – 26 часов.

Теория планирования эксперимента. Планы эксперимента, оптимальные планы, критерии оптимальности. Разные типы моделей (регрессионные, факторные и др.) и соответствующих им планов. Планирование экстремальных, отсеивающих и дискриминирующих экспериментов.

Планирование вычислительного эксперимента. Практические приемы составления оптимальных планов. Выбор наилучших условий вычислительного эксперимента, учет особенностей архитектуры вычислительной платформы. Постановка вычислительного эксперимента на гетерогенном вычислительном кластере. Распределенные вычисления. Способы и программные средства комплексирования удаленных вычислительных систем для решения конкретной задачи.

Планирование вычислительного эксперимента в интеллектуальных системах реального времени. Проверка адекватности моделей.

По третьей специализации «Вычислительные высокопроизводительные системы и параллельные вычисления» однозначный учебный план и тем более рабочие программы представить невозможно. Это связано с тем, что таких специализаций может быть много, поскольку много предметных областей и, соответственно, специальностей, и кроме того, у заказчиков специалистов разные интересы. Это достаточно емкий вопрос, требующий отдельной проработки в будущем. Видимо, нужно разные специальности объединить в группы и для каждой группы составлять свой учебный план.

В качестве удачного примера реализации такой специализации может служить приводимый ниже учебный план, действующий в Уральском государственном университете. Он соответствует взаимным интересам ИММ УрО РАН и УрГУ и отражает следующую тематику научных исследований, на которую ориентируются будущие специалисты:

- разработка теории вычислительных алгоритмов, системных и программных средств параллельных вычислительных архитектур,
- построение распараллеливающихся алгоритмов для задач конфликтного управления при дефиците информации, построение областей достижимости управляемых систем и множеств разрешимости,
- построение параллельных алгоритмов наилучшей аппроксимации функций многих переменных, разработка алгоритмов сжатия информации,
- разработка высокопроизводительных алгоритмов решения задач линейной алгебры и математического программирования,
- разработка параллельных алгоритмов и программных комплексов для решения задач высокоточной навигации и управления с использованием методов обработки изображений и анализа пространственных сцен, обработка космической информации высокого разрешения,
- анализ больших массивов данных о глобальных изменениях окружающей среды для экологического и других прогнозов,
- разработка и реализация на многопроцессорных системах программных комплексов для моделирования глобальных процессов, динамики катастрофических природных явлений.

Еще раз подчеркнем, что рекомендуемые названия, учебные планы и программы дисциплин специализаций являются рекомендательными, поскольку принятие всех решений по специализациям является компетенцией конкретного университета при условии непротиворечивости образовательному стандарту действующей специальности. Вместе с тем отметим, что представленные здесь рекомендации являются коллективным мнением ведущих специалистов страны в рассматриваемой области.

Профессиональная переподготовка научных и педагогических кадров в области высокопроизводительных вычислений характеризуется в настоящее время следующими особенностями:

- Составить один или даже несколько вариантов учебных планов переподготовки специалистов с ориентацией на параллельные вычисления и рассчитанных на весьма ограниченное время обучения практически невозможно, так как существует большое число предметных областей, обилие научных задач даже в одной предметной области, разный уровень подготовки абитуриентов, различные возможности аппаратных и программных средств и т.п.
- Возможностями профессиональной переподготовки кадров при значительном числе НИИ и вузов, имеющих отношение к параллельным вычислениям, обладают лишь немногие из них из-за ограниченной оснащенности и трудностей с педагогическими кадрами.

- Не всегда есть возможность у обучаемых на значительный срок оторваться от текущей работы, а методы и средства удаленного обучения зачастую недоступны или мало эффективны.
- Недостаточно информации о предложениях и потребностях при проведении переподготовки кадров.

В связи с этим организацию профессиональной переподготовки научных и педагогических кадров в настоящее время можно осуществить лишь с оговорками на приведенные выше обстоятельства и силами небольшого числа ведущих университетов. Для серьезного решения этой достаточно емкой научно-организационной задачи необходимы значительные усилия. При этом важной составной частью решения этой проблемной задачи может стать проработка вопроса использования технологий удаленного доступа.

В качестве иллюстрации реального осуществления профессиональной переподготовки научных и педагогических кадров в области методов высокопроизводительных вычислений и их применения в практике научных исследований может служить опыт Санкт-Петербургского государственного университета. Ниже приведены два варианта действующих учебных планов.

ВАРИАНТ 1

Категория слушателей: специалисты естественнонаучных направлений, не имеющие специальной подготовки в области программирования; преподаватели программирования и информатики для специалистов в области физики, химии, биологии.

Срок обучения: 512 часов аудиторных занятий.

Режим занятий: 2-3 раза в неделю, 12-20 учебных часов в неделю.

Учебный план

№ п/п	Наименование учебных дисциплин	Всего часов	В том числе		Форма контроля
			Лекции	Практические	
	ОБЩИЕ КУРСЫ				
1	Архитектура параллельных ЭВМ	32	32		Экзамен
2	Параллельные алгоритмы	48	48		Экзамен
3	Основы практической работы в ОС Unix	40	8	32	Зачет
4	Основы программирования	64		64	Зачет
5	Введение в средства параллельного программирования	64	52	12	Экзамен

№ п/п	Наименование учебных дисциплин	Всего часов	В том числе		Форма контроля
			Лекции	Практические	
6	Вспомогательные средства программирования для параллельных систем	32	12	20	Зачет
7	Российские и зарубежные суперкомпьютерные центры (структура и правила доступа к ресурсам)	16	12	4	Зачет
ИТОГО аудиторных занятий		296	164	132	
	КУРСЫ ПО ВЫБОРУ				
1	Программирование с использованием PVM	120	24	96	Зачет
2	Программирование в стандарте MPI	120	24	96	Зачет
3	Программирование в системе PARIX	96	16	80	Зачет
4	Параллельные расширения стандартных языков	96	16	80	Зачет
ИТОГО аудиторных занятий		432	80	352	
ВЫПУСКНАЯ РАБОТА (самостоятельно)		100			Защита на ГАК

ВАРИАНТ 2

Категория слушателей: специалисты естественнонаучных направлений – физики, химики, биологи, не имеющие специальной подготовки в области программирования.

Срок обучения: 110 аудиторных часов.

Режим занятий: 2-3 раза в неделю, 6-8 учебных часов в неделю.

Учебный план

№ п/п	Наименование учебных дисциплин	Всего часов	В том числе		Форма контроля
			Лекции	Практические	
1	Архитектура параллельных ЭВМ	8	8		Зачет
2	Основы практической работы в ОС Unix	10		10	Зачет
3	Основы программирования	16		16	Зачет
4	Программирование с использованием PVM	28	4	24	Зачет
5	Программирование в стандарте MPI	28	4	24	Зачет
6	Программирование на HPF	20	4	16	Зачет
ИТОГО аудиторных занятий		110	20	90	
ВЫПУСКНАЯ РАБОТА (самостоятельно)		20			Защита на ГАК

Важное значение в формировании корпуса высококвалифицированных специалистов в области высокопроизводительных вычислений должна играть аспирантура. Здесь выделим два аспекта.

Первый аспект – необходимо аспирантам многих научных специальностей, способным в своей профессиональной деятельности пользоваться методами параллельных вычислений, и прежде всего аспирантам групп специальностей 05.13, предоставить для изучения общеобразовательную дисциплину по супер-ЭВМ и параллельной обработке данных. Программа такой дисциплины приводится ниже.

Современные супер-ЭВМ и параллельная обработка данных
(30 часов лекций)

1. Введение в предмет – 2 часа

Введение, примеры современных супер-ЭВМ, примеры модельных задач с большой вычислительной сложностью; параллелизм/конвейерность, история появления параллелизма в архитектуре компьютеров, классы современных ВС; обозначение круга проблем в использовании параллельных ВС; закон Амдаля, его следствия, поправки к закону, суперлинейное ускорение программ.

2. Современные параллельные вычислительные системы - 8 часов

Примеры вычислительных систем (особенности архитектуры, как достигается пиковая производительность, что мешает достичь пиковой производительности):

- компьютеры с общей памятью (SMP-архитектура);
- векторно-конвейерные компьютеры;
- компьютеры с массовым параллелизмом, вычислительные кластеры. Топология коммуникационных сетей (решетка, 2D-3D-тор, МВС, гиперкуб и т.д.), типы коммутаторов (полный, бабочка), латентность и пропускная способности сети; масштабируемость компьютеров; балансировка вычислительной нагрузки;
- потоковые, матричные, VLIW и суперскалярные компьютеры;
- иерархия памяти в вычислительных системах.

3. Производительность параллельных компьютеров – 2 часа

Понятия пиковой и реальной производительности, эффективности работы и загруженности ВС; методы измерения производительности параллельных компьютеров: тесты нижнего уровня, SPEC, LINPACK, NPB, приложения.

4. Системы и средства параллельного программирования – 8 часов

Модели параллельных программ, масштабируемость программ, портабельность параллельных программ. Подходы к созданию параллельных программ: автоматическое распараллеливание программ, распараллеливание на основе директив компилятора, расширения существующих языков, языки параллельного программирования, системы с явной передачей сообщений, языки однократного присваивания, библиотеки параллельных программ, предметно-ориентированные пакеты для решения класса задач:

- MPI
- Open_MP
- Linda
- SISAL/HOPMA
- Occam
- High Performance Fortran.

Синхронизация параллельных процессов (тупики, разделяемые ресурсы, недетерминированность поведения параллельных процессов, семафоры, критические секции, барьеры, замки, события, atomic update).

5. Основы теории анализа структуры программ и алгоритмов – 6 часов

Графовые модели программ.

Основы теории графов алгоритмов, возможность статического построения информационной истории для класса программ, разновидности параллелизма: конечный/массовый, координатный/скошенный; эквивалентные преобразования программ, элементарные преобразования циклов; оценивание параллельной сложности программ и алгоритмов.

6. Отображение проблем вычислительной математики на архитектуру параллельных ВС – 2 часа

Влияние выбора метода на производительность, влияние формы записи алгоритма на производительность, влияние выбора структур данных на производительность.

Оптимизация прикладных программ по всей вертикали: анализ конфигурации компьютера, анализ системного и прикладного ПО, анализ структуры программ, анализ алгоритмического подхода.

7. Метакомпьютинг и концепция Grid – 2 часа

Второй аспект – для аспирантов по научным специальностям группы 05.13 необходимо ввести дополнение к программе кандидатского экзамена по специальности. Вариант такого дополнения представляется ниже.

Дополнение к программе кандидатского экзамена по научным специальностям группы 05.13

Параллелизм и конвейерность, параллелизм в архитектуре компьютеров, классы современных ВС; закон Амдала, суперлинейное ускорение программ; компьютеры с общей памятью (SMP-архитектура), векторно-конвейерные компьютеры, компьютеры с массовым параллелизмом, вычислительные кластеры; коммуникационные сети; латентность и пропускная способность сети; масштабируемость компьютеров; потоковые, матричные, VLIW и суперскалярные компьютеры; метакомпьютинг; пиковая и реальная производительность компьютеров; методы измерения производительности параллельных компьютеров.

Модели параллельных программ, масштабируемость программ, портабельность параллельных программ; подходы к созданию параллельных программ: автоматическое распараллеливание программ, распараллеливание на основе директив компилятора, расширения существующих языков, языки параллельного программирования, системы с явной передачей сообщений, языки однократного присваивания, библиотеки параллельных программ, предметно-ориентированные пакеты для решения класса задач.

Синхронизация параллельных процессов, разделяемые ресурсы, тупики, семафоры, критические секции, барьеры, замки, события.

Графовые модели программ; статический и динамический анализ программ; эквивалентные преобразования программ, элементарные преобразования циклов; оценивание параллельной сложности программ и алгоритмов; влияние выбора метода на производительность, влияние формы записи алгоритма на производительность, влияние выбора структур данных на производительность.

Рекомендации по аспирантской подготовке в области параллельных вычислений подготовлены на основе опыта МГУ (В.В.Воеводин).

Изложенное выше можно рассматривать как базовый результат в организации подготовки специалистов по высокопроизводительным вычислениям в стране. Дальнейшие усилия, по мнению экспертной комиссии, целесообразно направить на решение следующих задач:

- мониторинг развертывания в стране подготовки специалистов по высокопроизводительным вычислениям,

• анализ потребности в специалистах в разных регионах, создание системы их трудоустройства по целевым заказам,

• разработка предложений по учебным планам и рабочим программам подготовки бакалавров и магистров,

• выявление мнения институтов РАН и предприятий промышленности по содержанию учебных планов и рабочих программ для подготовки названных специалистов,

• перманентная коррекция разработанных планов и программ, учитывая быстроту развития отрасли,

• постепенный переход от специализаций к специальностям и специализациям,

• создание централизованной базы данных по учебно-методическим материалам в области высокопроизводительных вычислений и разработка принципов кооперации специалистов, занятых этими проблемами, а также защиты их авторских прав,

• разработка электронных средств обучения с удаленным доступом,

• организация сайта по проблеме подготовки рассматриваемых специалистов,

• организация подготовки и аттестации высокопрофессионального, в том числе элитного преподавательского корпуса,

• создание базы преподавателей по результатам сертификации, значительное усиление возможностей удаленного доступа при решении всех аспектов проблемы,

• изучение международного опыта в подготовке специалистов в области высокопроизводительных вычислений,

• создание в структуре ГНИИ ИТТ «Информика» координационно-аналитического центра по основным аспектам подготовки специалистов в области высокопроизводительных вычислений в стране.

Разработанные предложения представлены экспертной комиссией в Минобразования РФ.

Выражаем благодарность всем специалистам, принявшим конструктивное участие в подготовке материалов.

УЧЕБНО-НАУЧНЫЙ ЦЕНТР МГУ ПО ВЫСОКОПРОИЗВОДИТЕЛЬНЫМ ВЫЧИСЛЕНИЯМ*

Вл.В.Воеводин

НИВЦ МГУ

Введение

9 ноября 2001 года была опубликована 18-я редакция списка 500 мощнейших компьютеров мира – Top500. Лидером списка по-прежнему остается корпорация IBM, которой принадлежит 32% установленных систем и 37% от общей производительности. Интересной новостью стало появление Hewlett-Packard на втором месте по количеству систем (30%). При этом поскольку все эти системы относительно небольшие, то их суммарная производительность составила всего 15% от общего показателя. Далее по количеству компьютеров в списке идут SGI, Cray и Sun.

Самым мощным суперкомпьютером мира является по-прежнему система ASCI White в Ливерморской лаборатории (США) с производительностью в 7,2 Тфлопс на тесте LINPACK (58% от пиковой производительности). Этот компьютер содержит 8192 процессора Power3 (512×16), 4 Тбайта оперативной памяти, дисковый массив в 160 Тбайт, пиковая производительность его 12 Тфлопс (flops – floating operations per second, операции с плавающей точкой в секунду, Т(тера) = 10^{12}).

На втором месте теперь стоит система Compaq AlphaServer SC, объединяющая 3024 процессора. Она установлена в Питтсбургском суперкомпьютерном центре и имеет производительность 4 Тфлопс. Замыкает список Top500 116-процессорная вычислительная система Cray T3E с производительностью на LINPACK в 94 Гфлопс.

Список включает уже 16 систем с производительностью более 1 Тфлопс, 8 из которых установлено IBM. Стабильно увеличивается число систем, представляющих собой кластеры из небольших SMP-блоков – сейчас в списке уже 43 такие системы. Однако большинство в списке по-прежнему за массивно-параллельными системами (50%), за которыми идут кластеры, состоящие из больших SMP-систем (29%).

По каким же направлениям идет развитие высокопроизводительной вычислительной техники в настоящее время? Основных направлений четыре.

* Данная работа выполняется при поддержке РФФИ, грант №02-07-90442.

1. Векторно-конвейерные компьютеры. Конвейерные функциональные устройства и набор векторных команд – это две особенности таких машин. В отличие от традиционного подхода, векторные команды оперируют целыми массивами независимых данных, что позволяет эффективно загружать доступные конвейеры. В таких системах команда вида A=B+C может означать сложение двух массивов, а не только двух чисел. Характерным представителем данного направления является семейство векторно-конвейерных компьютеров Cray, куда входят, например, Cray EL, Cray J90, Cray T90. Также сюда относятся компьютеры фирм NEC, Fujitsu и др.

2. Массивно-параллельные компьютеры с распределенной памятью. Идея построения компьютеров этого класса тривиальна: возьмем серийные микропроцессоры, снабдим каждый своей локальной памятью, соединим посредством некоторой коммуникационной среды – вот и все. Достоинств у такой архитектуры масса: если нужна высокая производительность, то можно добавить еще процессоры, если ограничены финансы или заранее известна требуемая вычислительная мощность, то легко подобрать оптимальную конфигурацию, и т.п. Однако есть и решающий "минус", сводящий многие "плюсы" на нет. Дело в том, что межпроцессорное взаимодействие в компьютерах этого класса идет намного медленнее, чем происходит локальная обработка данных самими процессорами. Именно поэтому написать эффективную программу для таких компьютеров очень сложно, а для некоторых алгоритмов иногда просто невозможно. К данному классу можно отнести компьютеры Intel Paragon, IBM SP1, Parsytec, МВС 100/1000. В какой-то степени к этому классу можно причислить IBM SP2 и Cray T3D/T3E, хотя в этих компьютерах влияние указанного фактора ослаблено. К этому же классу можно отнести вычислительные кластеры и сети компьютеров, которые все чаще рассматривают как дешевую альтернативу крайне дорогим суперкомпьютерам.

3. Параллельные компьютеры с общей памятью (SMP-компьютеры). Вся оперативная память таких компьютеров разделяется несколькими одинаковыми процессорами. Это снимает проблемы предыдущего класса, но добавляет новые – число процессоров, имеющих доступ к общей памяти, по чисто техническим причинам нельзя сделать большим. В данное направление входят многие современные многопроцессорные SMP-компьютеры или отдельные узлы компьютеров класса HP V-Class или Sun StarFire.

4. Последнее направление, строго говоря, не является самостоятельным, а скорее представляет собой комбинации предыдущих трех. Из нескольких процессоров (традиционных или векторно-конвейерных) и общей для них памяти сформируем вычислительный узел. Если полученной вычислительной мощности недостаточно, то объединим несколько узлов высокоскоростными

каналами. По такому принципу построены Cray SV1, HP Superdome, Sun StarFire, NEC SX-5, последние модели IBM SP2 и др.

“Кластерное” направление в настоящее время является наиболее перспективным для конструирования компьютеров с рекордными показателями производительности. В последнем выпуске Top500 за кластерами 43 позиции. На сегодня самый «быстрый» Linux-кластер “IBM+NCSA” имеет производительность 2Тфлопс. Кластеры бывают «классическими», бездисковыми, многосетевыми, неоднородными и т.п. Они могут объединять самые разные процессоры: Pentium, Itanium, Alpha и др. Для их соединения используются сети Fast Ethernet, SCI, Myrinet, cLAN, ServerNet, Gigabit Ethernet и т.п. Как правило, кластеры работают под управлением операционной системы Linux, иногда под Windows NT.

1. Кластерные системы в России

В настоящее время в России насчитывается более 40 кластерных систем, среди которых наиболее мощными являются:

- вычислительная система Межведомственного суперкомпьютерного центра: 768CPUs, 2×Alpha 21264 (EV67/667MHz), 2Gb RAM, 15Gb HDD, Myrinet+Fast Ethernet, RedHat Linux;
- вычислительные системы НИВЦ МГУ им.М.В.Ломоносова:

а) 36CPUs, 2×PentiumIII/500MHz, 1Gb RAM, 9Gb HDD, SCI+Fast Ethernet, RedHat Linux;

б) 40CPUs, 2×PentiumIII/850MHz, 1Gb RAM, 2×15Gb HDD, Fast Ethernet, RedHat Linux;

- вычислительная система МФТИ:

24CPUs, 2×PentiumIII/600MHz, 256Mb RAM, 10Gb HDD, SCI+Fast Ethernet, ASP Linux;

- вычислительная система СПбГТУ:

16CPUs, 2×PentiumIII/550MHz, 256Mb RAM, Myrinet+Fast Ethernet, Linux, HDD U2W SCSI 72Gb RAID.

2. Учебно-научный центр МГУ

В 2000-2001 годах при поддержке ФЦП "Интеграция", РФФИ и ректората МГУ в Московском университете создан уникальный и на настоящий день самый мощный среди вузов России вычислительный центр (рис.1). Комбинация вычислительных мощностей с накопленным опытом в области использования высокопроизводительной техники и теоретическими исследованиями в области параллельных вычислений позволяет обеспечить полноцен-

ную среду для поддержки фундаментальных научных исследований и учебного процесса, а также экспериментальный полигон для разработки и апробирования кластерных технологий.

Коллектив лаборатории параллельных информационных технологий НИВЦ МГУ уже много лет работает в области параллельных вычислений и супер-ЭВМ. Мы достаточно хорошо представляем данную предметную область, знакомы со многими коллективами и отдельными учеными, работающими в данной области, участвуем в совместных проектах. На этой основе в мае 1998 года мы создали сервер по параллельным вычислениям в сети Интернет, затем выросший в Информационно-аналитический центр PARALLEL.RU. В 1999 году работа центра была поддержана Российским фондом фундаментальных исследований. Тематика центра охватывает следующие разделы предметной области: обзор новостей мира высокопроизводительных вычислений, описание архитектур компьютеров, технологии параллельного программирования, задачи, решаемые на суперкомпьютерах, обзор источников информации в данной области, списки конференций по данной тематике, списки крупнейших суперкомпьютерных центров, информацию о российских ресурсах в данной области, очерки по истории высокопроизводительных вычислений и многое другое. Задача, которая ставилась нами в самом начале, – структуризация предметной области и построение справочника на базе HTML, практически выполнена.

Задача-максимум для PARALLEL.RU состоит в создании полноценной энциклопедии предметной области на основе веб-технологий, что требует привлечения к проекту гораздо большего числа авторов, специалистов в отдельных областях. Другая задача центра – это создание среды для постоянного общения и взаимодействия российских специалистов по параллельным вычислениям и пользователей высокопроизводительных вычислительных ресурсов. Собирается информация о российских персоналиях и организациях, вовлеченных в данную деятельность, поддерживается "карта российских проектов". Организована рассылка по электронной почте новостей центра и мира высокопроизводительных вычислений, охватывающая все основные коллективы России и ближнего зарубежья, занимающиеся данной тематикой (на октябрь 2001 года более 1600 подписчиков). Функционирует дискуссионный клуб, в котором обсуждаются все интересующие вопросы данной тематики, оказывается информационная поддержка научных конференций, семинаров и других мероприятий вокруг тематики параллельных вычислений. На базе центра реализуются элементы дистанционного обучения. Кроме предоставления учебных материалов и заданий, разработаны также варианты интерактивного взаимодействия с обучаемыми, такие как онлайн-тестирование, сдача зачета через Интернет и др. В 1999 году был проведен пробный кон-

курс по параллельному программированию для студентов ведущих российских вузов.

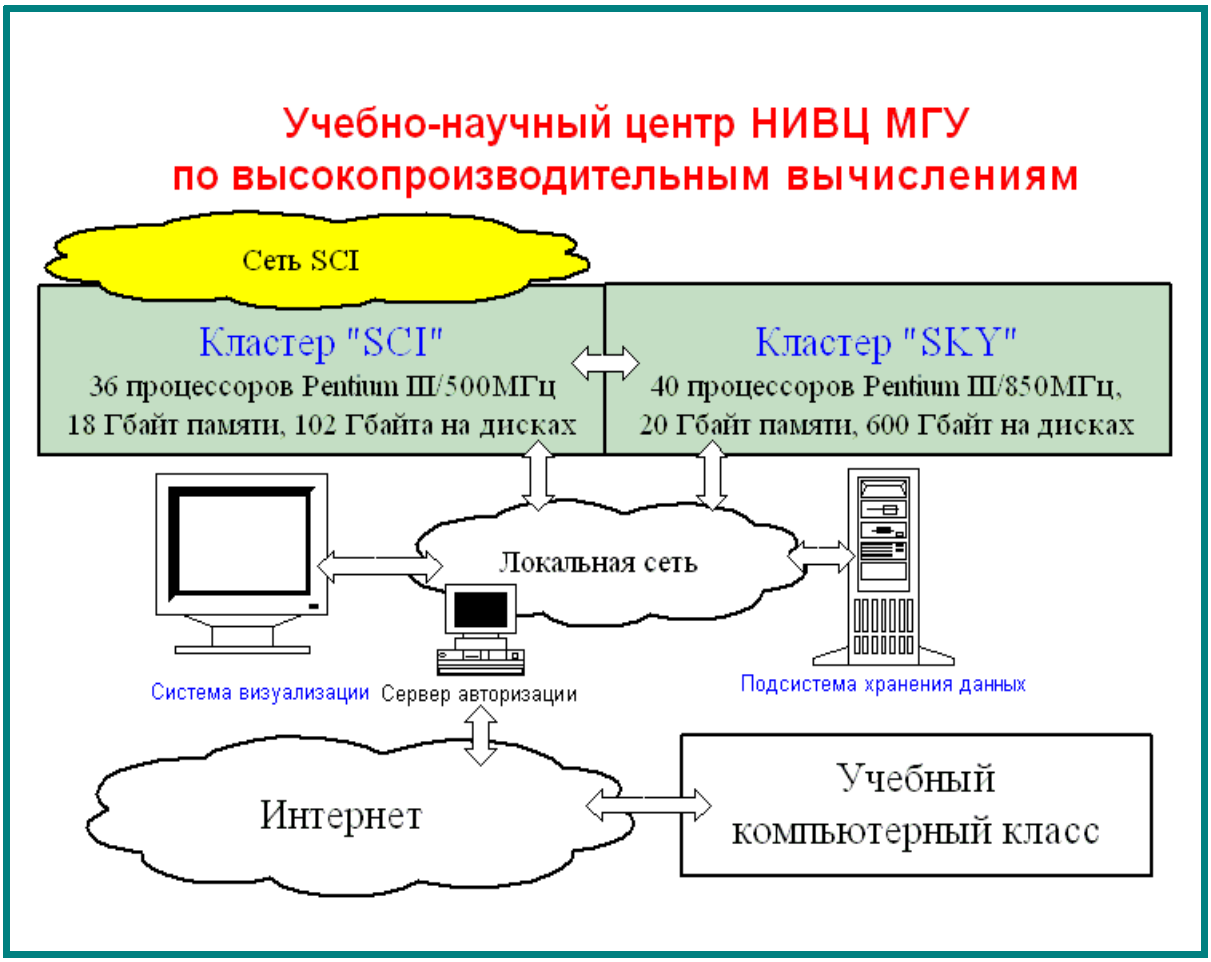

Рис.1

3. Высокопроизводительные вычислительные ресурсы МГУ

В феврале 2000 года в НИВЦ МГУ был установлен первый вычислительный кластер (рис.2), оборудование для которого было закуплено частично за счет ФЦП "Интеграция", частично за счет собственных средств НИВЦ.



Этот кластер состоял из 12 двухпроцессорных ПК-серверов российской сборки (компании "Инел"). Использование двухпроцессорных узлов считается более эффективным по технологическим и экономическим соображениям по сравнению с одно- и четырехпроцессорными узлами. При проектировании кластера с учетом технических требований и имеющихся средств рассматривалось несколько вариантов построения коммуникационной сети, в том числе Fast Ethernet, Gigabit Ethernet, Myrinet и SCI. В результате оценки имеющихся данных по цене, надежности программного обеспечения и производительности этих технологий была выбрана технология SCI (Scalable Coherent Interface), несмотря на то, что стоимость коммуникационного оборудования в расчете на один узел (1700 долларов США) превосходила стоимость самого вычислительного узла (1300 долларов).

Рис.2

Для построения коммуникационной сети были закуплены программно-аппаратные комплекты Wulfkit производства норвежских компаний “Dolphin

Interconnect Solutions” (сетевые адаптеры) и “Scali Computer” (программное обеспечение). Технология SCI отличается от других тем, что не использует специальных коммутаторов, а узлы кластера объединяются в топологию "двухмерного тора", состоящего из однонаправленных колец SCI. При такой организации сети каждый узел имеет по 2 входящих и выходящих канала SCI с очень высокой скоростью передачи и присутствует одновременно в двух кольцах.

При дальнейшем расширении аппаратных ресурсов в начале 2001 года мы сочли необходимым учесть потребности наших пользователей и провели среди них опрос – какие компоненты нужно нарастить в первую очередь? По результатам этого опроса сначала был увеличен объем оперативной памяти на всех узлах до 1 Гбайта, а затем увеличено количество узлов. Таким образом, кластер “SCI” в настоящее время включает 18 двухпроцессорных узлов с процессорами Pentium III 500 МГц (и 550 на новых узлах) и на каждом узле установлено по 1 Гбайту оперативной памяти. Узлы объединены в сеть SCI с топологией двухмерного тора 3×6, а управление кластером производится с отдельной головной машины по сети Fast Ethernet. Кроме того, установлен отдельный файл-сервер с RAID-массивом, на котором хранятся домашние директории пользователей, доступные со всех машин по протоколу NFS. Временные файлы программы пользователей могут создавать на локальных дисках каждого узла. Для поддержки непрерывной работы кластера на всех узлах установлены специальные платы watch-dog, разработанные в ИПС РАН (г. Переславль-Залесский), которые позволяют автоматически перезагружать зависшие узлы.

Весной 2001 года был установлен новый 40-процессорный кластер "SKY" с более мощными процессорами (Pentium III/850 МГц) на базе традиционной сети Fast Ethernet. На этой конфигурации успешно работают многие параллельные приложения, в частности расчет больших квантовохимических систем с помощью PC-GAMESS. Для поддержки больших расчетов на каждом узле кластера установлено по 2 жестких диска, которыми независимо могут пользоваться два процесса параллельной задачи.

4. Характеристики производительности

По результатам измерения характеристик сети скорость обменов данными по сети SCI на уровне MPI-приложений составила более 80 Мбайт/с, а латентность (время задержки сообщений) – примерно 5,6 мкс. Эти результаты практически не зависят от взаимного расположения узлов. При больших размерах сообщения скорость обменов по сети SCI даже превышает скорость обменов внутри одного SMP-узла. При обменах по сети Fast Ethernet пропуск-

ная способность не превышает 10-11 Мбайт/с, а латентность составляет более 130 мкс. Это ограничивает круг параллельных приложений, которые могут эффективно работать на таком кластере.

Традиционным тестом производительности параллельных компьютеров, используемым при составлении списка Top500, является тест решения системы линейных уравнений LINPACK HPC. Лучшие результаты по LINPACK составляют 10 Гфлопс (или 278 Мфлопс на процессор) для кластера “SCI” и 13 Гфлопс (или 325 Мфлопс на процессор) для кластера “SKY” (на задачах размера 44000×44000 и 47000×47000 соответственно). Таким образом, за счет более мощных процессоров кластер “SKY” на тесте LINPACK показывает лучшую производительность. Эти данные говорят не о том, что сеть SCI плохо справляется с нагрузкой, а о том, что тест LINPACK при больших объемах вычислений не предъявляет высоких требований к производительности коммуникационной сети. При фиксированном размере задачи (16000×16000) кластер на базе SCI показывает лучшую масштабируемость при большом числе процессоров.

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

5. Технологии программирования

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

Осуществляется перенос "Библиотеки численного анализа", разработанной в НИВЦ МГУ для машин серии БЭСМ-6, на кластер.

6. Особенности работы центра

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

- необходимо поддерживать множество пользователей с различными задачами и различными требованиями к вычислительным ресурсам;
- большинство пользователей работают в удаленном режиме;
- требуется качественная поддержка учебного процесса;
- задачи пользователей написаны на разных языках с применением различных технологий параллельного программирования;
- нужно поддерживать запуск как параллельных, так и последовательных программ;
- в центре установлены множественные вычислительные ресурсы с различными параллельными средами.

7. Система управления заданиями

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

- поддержка множественных подкластеров с независимыми очередями и с возможностью запуска задач на всем кластере;
- независимые настройки различных очередей;
- гибкая настройка системы на различные параллельные среды;
- поддержка приоритетов задач;
- возможности временной блокировки очередей, задач и узлов кластера;
- настройка различных ограничений и прав пользователей;
- средства оптимального планирования загрузки с учетом предполагаемого времени работы задач;
- внешние модули для поддержки различных стратегий распределения процессоров;
- средства сбора статистики по использованию кластера;
- веб-интерфейс для просмотра состояния кластеров и очередей;
- переносимость кода; система реализована на языке Perl с отдельными элементами на языке Си;
- развитая документация для пользователя и администратора.

8. Поддержка пользователей

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

- подготовку справочной и методической литературы, документации для пользователей; публикацию в рамках Интернет-центра *Parallel.Ru;*
- оперативную поддержку пользователей по электронной почте и по телефону;
- список рассылки оперативной информации для пользователей;
- обязательный учет требований и пожеланий пользователей при расширении аппаратного и программного обеспечения, при настройке параметров системы управления заданиями.

НОРМА – СПЕЦИАЛИЗИРОВАННАЯ СИСТЕМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ[1]

А.Н. Андрианов, А.Б. Бугеря, К.Н. Ефимкин

Институт прикладной математики им. М.В. Келдыша РАН

Введение

Распределенные вычислительные системы, состоящие из объединенных высокоскоростными каналами связи ЭВМ различной архитектуры (включая многопроцессорные ЭВМ с общей и распределенной памятью), в настоящее время следует рассматривать как наиболее доступные и перспективные средства для решения задач, требующих большого объема вычислений.

При создании прикладных программ для подобных систем прикладной специалист скорее всего хотел бы знать как можно меньше об устройстве конкретной системы (но при этом полностью использовать ее возможности); не иметь проблем при переходе с одной системы на другую; избежать детального изучения при программировании моделей параллельности, особенностей компиляторов, организации и использования системных библиотек и т.п.

[1] Работа выполнена при частичной финансовой поддержке РФФИ, грант № 01-01-00411.

Реализация такой комфортной модели разработки требует решения двух основных проблем:

- упрощения программирования и отладки;
- обеспечения мобильности параллельных программ.

В данном проекте предпринимается попытка решить эти проблемы на базе подхода к разработке параллельных программ, основанного на применении специализированного декларативного языка Норма [1,2] (разработка ИПМ РАН).

Специализированный декларативный язык нового поколения Норма позволяет описывать решение широкого класса задач математической физики и при этом использовать привычные для прикладного специалиста понятия. Описание решения задачи в математических терминах имеет очень высокий уровень абстракции и отражает метод решения, а не его реализацию при конкретных условиях. Такое описание не ориентировано на конкретную архитектуру компьютера, поэтому оно предоставляет большие возможности для выявления естественного параллелизма и организации вычислений. Программа на языке Норма фактически является представлением математических расчетных формул, полученных прикладным специалистом. Таким образом, программирование на языке Норма не требует написания программы в смысле традиционных универсальных языков программирования, а выходные программы для параллельных ЭВМ с различной архитектурой строятся автоматически при помощи транслятора-синтезатора.

Язык Норма является языком сверхвысокого уровня и обеспечивает дружественный интерфейс с пользователем: фактически программирование в этом случае не требует написания программ в смысле традиционных универсальных языков программирования [3]. Такой подход к программированию основан на следующем тезисе: «универсальные языки программирования могут быть дружественными только для системного программиста».

Конечно, создание специализированных языков не является новостью в программировании; нельзя также сказать, что исследований в этом направлении нет. Однако авторы считают, что данное направление является весьма перспективным и интенсификация исследований в этой области позволит приблизиться к решению весьма актуальной в настоящий момент, но трудной проблемы разработки прикладных параллельных программ.

1. О подходе к созданию языков нового поколения

Практика показывает, что язык математики позволяет достаточно строго определять предметную область, используя математические понятия. Фак-

тически каждый раз, когда оформлялась какая-либо строгая дисциплина, которая позволяла производить количественные оценки (в широком смысле, например, выбирать элементы, удовлетворяющие определенным условиям), она описывалась в математических обозначениях. В большинстве случаев такие формулировки не носили характер процесса, то есть не предписывали четкой последовательности действий, а лишь задавали те условия, которые должны выполняться в процессе получения результата.

Второе обстоятельство, которое следует подчеркнуть, – в математических описаниях такого рода отсутствовало понятие памяти. В следствие этого значения одного семейства различались значениями индексов и вопрос об использовании одной и той же ячейки памяти для экономии просто не возникал. Теперь очевидно, что такие описания обладают естественным параллелизмом, который обусловлен только функциональными зависимостями по данным. Более того, эти описания упрощают процесс выявления параллельности (распараллеливания), так как в них не используется понятие состояния памяти, зависящего от времени, и нет необходимости решать проблемы, связанные с анализом состояния памяти (поэтому часто говорят о необходимости языков с однократным присваиванием).

Таким образом, в качестве исходного языка описания решения задачи целесообразно взять сложившуюся в данной предметной области математическую нотацию и использовать ее для автоматического построения рабочей программы. Дружественный интерфейс обеспечивается автоматически, поскольку пользователь формулирует задачу в привычных для него терминах.

При таком подходе исчезает проблема доказательства правильности программ, в том смысле, что не требуется доказывать эквивалентность программы, например на языке Фортран, и той математической записи, которая специфицирует правила получения решения. Оказывается, что из этих правил можно, при определенных условиях, автоматически получить конкретный алгоритм, учитывая функцию оптимизации, которая может задаваться по-разному: это может быть экономия памяти, параллельность вычислений и т.п.

В большинстве случаев описание метода решения задачи в математических терминах, применяемых в конкретных областях, без сомнения имеет высокий уровень абстракции и допускает большие возможности выявления естественного параллелизма. Более того, это описание не содержит никакой ориентации на конкретную архитектуру вычислительной системы. Следовательно, этот уровень описания имеет суперустойчивый характер и отражает лишь метод решения, а не его реализацию в конкретных условиях. Подчеркнем, что до появления вычислительных систем математическое описание служило в качестве описания вычислительного решения сложных задач.

Таким образом, применение рассматриваемого подхода связано с решением двух главных задач:

- формализации языка описания решения задач, сложившегося в данной предметной области, до уровня, непосредственно воспринимаемого вычислительной системой;
- разработки методов, позволяющих определять по формальному описанию такой порядок действий, который наиболее соответствует реальному окружению (решение задачи синтеза).

Мы ограничимся в основном обсуждением вопросов, связанных с параллельностью, хотя возникают и другие проблемы, связанные с оптимизацией решения задачи: экономия памяти, удобство отладки и т.п.

2. Конкретизация высказанных соображений на примере

В данном разделе кратко остановимся лишь на некоторых свойствах специализированного декларативного языка для описания решения задач сеточными методами – языка Норма, только в связи с рассматриваемым ниже примером. Более подробные сведения о языке и состоянии разработки даны в следующем разделе.

Приведенный ниже известный пример программы на Фортране взят из статьи [4], в которой автор, Лесли Лэмпорт, использует этот пример для пояснения метода гиперплоскостей, применяемого при автоматическом распараллеливании циклов:

```
DO 99 I=1,L
DO 99 J=2,M
DO 99 K=2,N                                                    (1)
   U(J,K) = (U(J+1,K) + U(J,K+1) + U(J-1,K) + U(J,K-1)) * 0.25
99 CONTINUE
```

Этот фрагмент представляет собой упрощенный вариант разностной схемы для решения задачи Лапласа на прямоугольной сетке (J,K) путем фиксированного числа итераций по I. Ограничения, накладываемые этой конструкцией на порядок вычислений, требуют анализа порядка вычислений (который задается вложенностью циклов), анализа значений переменных (который необходим из-за возникающего при выполнении программы переприсваивания значений переменным), а также анализа индексных зависимостей. Все это необходимо учитывать для того, чтобы параллельный вариант давал тот же результат, что и последовательный.

В результате анализа можно установить, что при вычислении значений в некоторой точке трехмерного индексного пространства с координатами

(I,J,K) (хотя явно указана только плоскость (J,K), переменная I вводит неявно еще одну координату), значения переменных, которые имеют индексные смещения J-1 и K-1, берутся с итерации I, остальные – с итерации I-1. При этом считается, что на всех границах значения переменной U известны и не зависят от I. Доказывается, что можно найти семейство плоскостей таких, что на любом шаге можно подсчитать значения переменной U во всех точках, принадлежащих очередной плоскости, параллельно и независимо (подробности читатель может найти в статье Лэмпорта). Заметим теперь, что этот фрагмент может быть представлен в следующей форме:

$$U_{j,k,i} = \frac{U_{j+1,k,i-1} + U_{j,k+1,i-1} + U_{j-1,k,i} + U_{j,k-1,i}}{4}, \quad (2)$$
$$j = 2,...,M; k = 2,...,N; i = 1,...,L.$$

С точностью до обозначений этот вариант является обычной математической записью, в которой естественным образом учитывается итерационный индекс. Кроме этого, запись не содержит переприсваивания значений переменным (повторного использования памяти). Поэтому при выявлении параллельных действий по записи (2) не нужно учитывать ни это обстоятельство (мы имеем дело с языком с однократным присваиванием), ни порядок вычисления витков цикла. Остается только проблема анализа индексных зависимостей.

Поскольку при таком описании перед математиком возникает геометрическая фигура – параллелепипед, то он вряд ли забудет задать необходимые для счета известные значения на границах (гранях параллелепипеда) или позаботиться о том, чтобы они были вычислены. А если и забудет, то сообщение об ошибке может быть выдано в содержательных терминах, со ссылкой на соответствующую грань.

Теперь рассмотрим простейший вариант схемы решения предлагаемой задачи:

$$U_{j,k,i} = \frac{U_{j+1,k,i-1} + U_{j,k+1,i-1} + U_{j-1,k,i-1} + U_{j,k-1,i-1}}{4}, \quad (3)$$
$$j = 2,...,M; k = 2,...,N; i = 1,...,L.$$

Этот вариант получается естественным образом, когда значение на новой итерации находят только по значениям на предыдущей итерации, и допускает широкое и очевидное распараллеливание (можно одновременно считать все точки на новом итерационном слое).

В варианте (1) как сильно усложнено выявление параллельности, так и значительно снижен достижимый уровень распараллеливания. Наличие этого варианта записи может объясняться либо желанием оптимизировать алгоритм за счет экономии памяти, либо желанием уменьшить число итераций, либо и тем и другим. Для последовательной реализации такой подход вполне оправдан. Однако потеря возможности широкого распараллеливания алгоритма может обернуться фатальным обстоятельством в условиях параллельной среды выполнения. Отсюда следует, что идея систематического перехода от последовательной программы к параллельной (как, например, в HPF [5]) не всегда будет приводить к приемлемым результатам. Наоборот, систематический переход от описания, допускающего максимальное распараллеливание, к вариантам, который за счет снижения уровня распараллеливания оптимизируют требуемые параметры, по-видимому, является более перспективным подходом. Отметим также, что алгоритм распараллеливания для формы (2) может дать (в случае оптимизации памяти) тот же уровень распараллеливания и объем используемой памяти, как и метод гиперплоскостей Лэмпорта для фрагмента (1).

В настоящее время типичной является ситуация, когда прикладной специалист разрабатывает (или уже разработал ранее) последовательную программу для решения своей задачи, например на Фортране, и ищет способ перехода от этой программы к параллельной. По мнению авторов, в настоящий момент такой переход требует существенных усилий и рассчитывать на автоматическое эффективное решение этой проблемы (например, при помощи распараллеливающих компиляторов) не приходится.

По нашему мнению, вместо перехода от последовательной версии программы к параллельной либо непосредственной разработки параллельной программы существует более перспективный третий путь, когда первичная формулировка проблемы может реализовываться как в параллельном, так и в последовательном варианте. Этот подход основывается на главном принципе: при формулировке решения задачи не накладывай лишних ограничений – можно столкнуться в дальнейшем с такой средой выполнения, в которой они вызовут неудобства и неэффективность. К счастью, математические описания почти всегда исповедуют этот принцип.

Вернемся теперь к языку Норма, который является практической иллюстрацией рассматриваемого подхода.

3. Язык Норма и содержательные обозначения

Язык Норма изначально был ориентирован на описание методов решения задач математической физики разностными методами. В этой предметной области сложилась определенная система обозначений,

используемая прикладным специалистом при записи расчетных формул, получаемых в процессе разработки метода решения.

Эта система обозначений включает как общие математические понятия (например, целые, вещественные числа, векторы, матрицы, функции, ряды), так и понятия, типичные для конкретной предметной области (например, сетка, индексное пространство, сеточная функция, итерации по времени или пространству).

Типичные понятия используются в конкретной предметной области достаточно часто и отражают ее специфику, однако разделение на общие математические понятия и типичные математические понятия условно. Существенно, чтобы язык, предназначенный для использования в некоторой предметной области, позволял применять как общие, так и типичные обозначения, и этих обозначений было достаточно для описания большинства задач данной предметной области. С другой стороны, возможность применения общих математических понятий определяет возможность использования специализированного языка и в других областях – все зависит от "мощности" набора этих понятий, допустимых в языке.

Проблему определения набора содержательных обозначений (общих и типичных) для конкретной предметной области надо решать, видимо, с учетом двух факторов:

- обеспечения дружественного интерфейса для пользователя (терминов должно быть достаточно, работать с ними просто и удобно, ничего лишнего знать не нужно);

- разрешимости задачи синтеза выходной программы по непроцедурному описанию ("мощные" наборы понятий, близкие к универсальным, часто приводят к трудноразрешимым или неразрешимым проблемам при решении задачи синтеза).

Ниже демонстрируются некоторые возможности языка Норма. Для представления понятия индексного пространства в языке Норма введено понятие области. Область – это совокупность целочисленных наборов $i_1,...,i_n; n > 0$, каждый из которых задает координаты точки n-мерного индексного пространства. С каждым направлением (осью координат) n-мерного пространства задачи связывается уникальное имя – имя индекса (имя оси координат индексного пространства). Частным случаем понятия области является сетка – логически-прямоугольная область. Одномерная область служит для задания диапазона точек на некоторой оси координат индексного пространства. В простейшем случае при описании одномерной области указывается имя одномерной области, имя индекса и диапазон изменения значений индекса:

$AxisK : (k = 1..15).$

Многомерная область строится при помощи операции ";" произведения областей. Пример описания двумерной области, полученной с помощью операции произведения одномерных областей **AxisK** и **AxisL**:

$Square : (AxisK : (k = 1..15); AxisL : (l = 1..5)).$

Для представления простых переменных в языке Норма используются скалярные величины (скаляры), а для представления векторов, матриц – величины, определенные на области. Описание величины задает тип величины – ***REAL, INTEGER*** или ***DOUBLE,*** а величина на области, кроме этого, связывается с указанной в описании областью, то есть значения этой величины могут быть вычислены в каждой точке этой области. Описания

VARIABLE First, Last ***DEFINED ON*** Square

определяют величины First, Last на области Square; это означает, что эти величины могут принимать значения в каждой точке области Square для $k = 1,...,15; l = 1,...,5.$

Расчетные формулы, получаемые прикладным специалистом, обычно записываются в виде соотношений. Например, расчетные формулы для решения системы линейных уравнений методом Гаусса - Жордана имеют вид

$$m0_{i,j} = a_{i,j}, i = 1,...,N; j = 1,...,N;$$

$$r0_i = b_i, i = 1,...,N;$$

$$m_{t,t,j} = \frac{m_{t-1,t,j}}{m_{t-1,t,t}}, j = 1,...,N; t = 1,...,N;$$

$$r_{t,t} = \frac{r_{t-1,t}}{m_{t-1,t,t}}, t = 1,...,N;$$

$$m_{t,i,j} = m_{t-1,i,j} - m_{t-1,i,t} \times m_{t,t,j}, j = 1,...,N; i = 1,...,t-1,t+1,..,N, t = 1,...,N;$$

$$r_{t,i} = r_{t-1,i} - m_{t-1,i,t} \times r_{t,t}, i = 1,...,t-1,t+1,...,N; t = 1,...,N;$$

$$x_i = r_{N,i}, i = 1,...,N.$$

Фрагмент программы на Норме, соответствующий этим расчетным формулам, приведен ниже:

```
Ot:(t=0..n). Oi:(i=1..n). Oj:(j=1..n).
Oij:(Oi;Oj). Otij:(Ot;Oij).
```

Oti:(Ot;Oi). Otij1:Otij/t=1..n.
Oti1:Oti/t=1..n.
DOMAIN PARAMETERS n = 10.
VARIABLE a ***ON*** Oij. ***VARIABLE*** m ***ON*** Otij.
VARIABLE b,x ***ON*** Oi. ***VARIABLE*** r ***ON*** Oti.
INPUT a ***ON*** Oij,b ***ON*** Oi.
OUTPUT x ***ON*** Oi.
FOR Otij/t=0 ***ASSUME*** m = a.
FOR Oti/t=0 ***ASSUME*** r=b.
OtiEQtj1,OtiNEtj1:Otij1/i=t.
OiEQti1,OiNEti1:Oti1/i=t.
FOR OtiEQtj1 ***ASSUME*** m = m[t-1,i=t]/m[t-1,i=t,j=t].
FOR OiEQti1 ***ASSUME*** r = r[t-1,i=t]/m[t-1,i=t,j=t].
FOR OtiNEtj1 ***ASSUME*** m = m[t-1]-m[t-1,j=t]*m[i=t].
FOR OiNEti1 ***ASSUME*** r = r[t-1]-m[t-1,j=t]*r[i=t].
FOR Oi ***ASSUME*** x = r[t=n].

Необходимые вычисления описаны при помощи конструкции ***FOR*** <область> ***ASSUME*** <соотношение>.

Эта конструкция – аналог форм (2) и (3) – является ключевым понятием языка Норма. Соотношение задает правило вычисления значений величины из левой части по значениям величин из правой части и индексные зависимости между переменными. Величина из левой части должна быть вычислена во всех точках области; правило для этого вычисления определено однозначно, однако при этом не требуется немедленное выполнение вычисления в данном месте программы и не задается порядок или способ вычисления. Некоторое внешнее сходство с оператором присваивания не должно вводить в заблуждение.

Индексы без смещений в записи формул могут быть опущены – при анализе программы они автоматически восстанавливаются транслятором.

В рассматриваемом фрагменте использованы также условные области. Так, описание OtiEQtj1,OtiNEtj1:Otij1/i=t определяет две непересекающиеся подобласти OtiEQtj1 и OtiNEtj1. Первая подобласть состоит из точек модифицируемой области Otij1, в которых заданное условие $i = t$ истинно, вторая – из точек, в которых это условие ложно (то есть $i \neq t$). В общем случае условие при описании условных областей представляется логическим выражением.

Отличия в приведенных выше двух способах записи расчетных формул связаны только с формой записи – представлением индексов, линейностью записи, содержательно эти способы эквивалентны.

При решении задач математической физики разностными методами процесс вычислений часто носит итеративный характер. Такие процессы можно задавать с помощью уже описанных средств языка Норма. Для этого необходимо изменить (расширить) описание областей, определенных в задаче, добавив дополнительное направление, соответствующее индексу итерации. Однако такое расширение неоправданно, так как это направление, по существу, является фиктивным – оно отражает способ вычисления, а не пространственно-временную сетку. В языке Норма определена специальная конструкция – итерация, позволяющая описывать итеративный вычислительный процесс. Неформально итерация задает итеративные вычисления с индексом итерации, меняющимся от 1 до некоторого целого положительного значения, которое определяется условием завершения итерации.

Ниже приведена схема некоторого итерационного процесса (номера строк не являются элементами языка и нужны только для пояснений, даваемых ниже). Строка 1 является заголовком итерации. В этой строке указываются имена итерируемых переменных (**r,p,u,e**) и идентификатор итерационного индекса (**nt**).

1. ***ITERATION*** r,p,u,e ***ON*** nt
2. ***BOUNDARY***
3. ***FOR*** gran1 ***ASSUME*** p=0;r=0;u=0;e=1.0.
4. ***END BOUNDARY***
5. ***INITIAL*** nt=0:
6. ***FOR*** gridsol ***ASSUME*** p=p0;r=r0;u=u0.
7. ***FOR*** gridsol ***ASSUME*** e=p/(r*(gamma-1.0))+u*u*0.5.
8. ***END INITIAL***
9. **<тело итерации>**
10. ***EXIT WHEN*** nt=ntend.
11. ***END ITERATION*** nt.

При описании итерации можно использовать возможность задания граничных значений (строки 2-4). Граничные значения итерируемой величины задаются при помощи операторов ***ASSUME***. Эти значения считаются неизменными на протяжении всей итерации и заданными на каждом шаге итерации.

Для задания начальных значений итерируемым переменным используются операторы, приведенные в строках 5-8. Фактически начальные значения – это такие значения, которые присваиваются итерируемым переменным при индексе итерации **nt=0**.

Тело итерации представляет собой часть программы, написанной на языке Норма. В частности, внутри итерации можно определить новую итерацию по другому направлению.

Завершение итерационного процесса происходит в случае, если логическое выражение, заданное в качестве условия выхода (строка 10), принимает значение «истина». Последняя строка (11) является завершением записи итерационного процесса с индексом итерации **nt**.

Знакомство с некоторыми возможностями языка Норма закончим функциями редукции, которые являются аналогами традиционных математических обозначений *sum*, *prod*, *max*, *min*. Обращение к этим функциям имеет следующий вид:

<имя-функции> ((имя-области) арифметическое-выражение).

Область определяет множество точек, в которых необходимо вычислить функцию; арифметическое выражение – множество значений, к которым применяется функция.

Пусть необходимо вычислить $V_i = W_i + \sum_{j=1}^{m} A_{i,j} \times X_j, i = 1,...,n.$

Описание этого вычисления на языке Норма

Grid:(Oi:(I=1..N); Oj:(J=1..M))
VARIABLE A ***DEFINED ON*** Grid.
VARIABLE V,W ***DEFINED ON*** Oi.
VARIABLE X ***DEFINED ON*** Oj.
FOR Oi ***ASSUME*** V = W + ***SUM***((OJ) A*X).

совпадает (с точностью до обозначений) с исходной расчетной формулой. Реализация редукционных функций для различных вычислительных окружений является нетривиальной задачей, однако эта проблема решается транслятором, а не пользователем.

Информация о языке и системе Норма размещена на сайте [15].

4. О результатах исследований по языку Норма

Несколько слов о проблеме разрешимости задачи синтеза выходной программы для языка Норма и результатах практической реализации и применения этого языка.

Язык Норма обладает несколькими характерными свойствами – это язык с однократным присваиванием; такие понятия, как память, побочный эффект, управляющие операторы, отсутствуют по определению. Эти свойства, и некоторые другие ограничения, в первую очередь на вид

индексных выражений и способы описания индексных пространств, позволяют конструктивно обосновать разрешимость задачи синтеза выходной программы [6-8]. Решение задачи синтеза основано на выявлении и анализе естественного параллелизма, который можно извлечь непосредственно из заданного описания, без применения символьных преобразований.

Полученные теоретические результаты использованы при создании транслятора-синтезатора, который позволяет получать по описанию на Норме выходные программы для распределенных параллельных вычислительных систем (параллельные вычисления с передачей сообщений), параллельных вычислительных систем с общей памятью (параллельные вычисления над общей памятью) и последовательных ЭВМ. Выходные программы могут быть построены на языках Фортран MPI, Фортран PVM, Фортран GNS, Фортран Convex, Фортран-77. Определение выходного языка осуществляется заданием соответствующей опции транслятора. Система Норма реализована для ОС UNIX, Windows, DOS. В настоящее время завершается разработка интерфейса, упрощающего работу пользователя с системой Норма в сетевом окружении.

Следует отметить, что ограничения на индексные выражения в Норме аналогичны тем ограничениям, которые накладываются в большинстве известных алгоритмов автоматического распараллеливания циклов. Это в известном смысле объясняет тот факт, что Норма может использоваться не только для описания решения задач разностными методами на статических регулярных сетках, как предполагалось сначала, но и для более широкого класса задач вычислительного характера. Например, на Норме было описано решение задачи [9], которое связано с решением интегральных уравнений, а также решены задачи на нерегулярных и адаптивных сетках.

Отметим интересный факт – в большинстве случаев исходной информацией для разработки Норма-программы была последовательная Фортран-программа, а не расчетные формулы, описывающие метод решения задачи. Это, конечно, затрудняло написание Норма-программы, поскольку требовалось еще понять, а что же делается в Фортран-программе и зачем это делается. С другой стороны, представление метода решения задачи в виде последовательной программы, по-видимому, является весьма распространенной исторически сложившейся формой для прикладного специалиста, который не видит проблем, связанных с извлечением параллелизма из последовательного представления.

Система Норма была использована при разработке параллельных программ для решения ряда задач математической физики на статических структурных сетках: моделирование дозвукового течения вязкого газа на примере обтекания воздухом пластины конечной длины (алгоритм расчета

основан на использовании кинетически-согласованных разностных схем КСРС[10], моделирование ударноволнового запуска сверхзвуковой струи газа из канала прямоугольного сечения (алгоритм расчета основан на трехмерной нестационарной схеме С.К.Годунова [11]), исследование задачи о течении газовой смеси с конденсацией в криовакуумном модуле на основе прямого численного решения кинетического уравнения Больцмана [12] и т.п. Кроме того, авторы выполнили ряд работ, связанных с программированием решения задач математической физики на статических неструктурированных и адаптивных вложенных сетках [13,14].

Список литературы

1. *Андрианов А.Н., Ефимкин К.Н., Задыхайло И.Б.* Непроцедурный язык для решения задач математической физики //Программирование. – 1991. – № 2. – С. 80-94.
2. *Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н., Задыхайло И.Б.* Норма. Описание языка. Рабочий стандарт: Препринт ИПМ им. М.В.Келдыша РАН. – 1995. – № 120. – 50 с.
3. *Задыхайло И.Б., Ефимкин К.Н.* Содержательные обозначения и языки нового поколения //Информационные технологии и вычислительные системы. – 1996. – № 2. – С. 46-58.
4. *Lamport L.* The Parallel execution of DO loops //Communications ACM. – 1974. – Vol.17, N 2. – P. 83-93.
5. *Steele G.L., High Jr.* Performance Fortran: status report. Workshop on Languages, Compilers and Run-Time Environments for Distributed Memory Multiprocessors, Boulder, CO, 30 Sept.-2 Oct. 1992: Reprint in SIGPLAN NOTICES. – 1993. – Vol.28, N 1. – P. 1-4.
6. *Андрианов А.Н.* Организация циклических вычислений в языке Норма: Препринт ИПМ им. М.В. Келдыша АН СССР. – 1986. – № 171. – 28 с.
7. *Андрианов А.Н., Андрианова Е.А.* Организация циклического процесса по непроцедурной записи //Программирование. – 1996. – № 4. – С.62-72.
8. *Андрианов А.Н.* Синтез параллельных и векторных программ по непроцедурной записи в языке Норма: Дис. … докт. физ.-мат. наук. – М., 2001. – 158 с.
9. *Васильев М.М., Ефимкин К.Н., Иванова В.Н.* О применении метода гидродинамических потенциалов к задаче обтекания тела //Математическое моделирование. – 1994. – Т.6, № 10. – С. 58-65.
10. *Андрианов А.Н., Гусева Г.Н., Задыхайло И.Б.* Применение языка Норма для расчета дозвукового течения вязкого газа //Математическое моделирование. – 1999. – Т.11, № 9. – С. 45-53.

11. *Andrianov A.N., Bazarov S.B., Bugerya A.B., Efimkin K.N.* Solution of three-dimensional problems of gas dynamics on multiprocessors computers //Computational Mathematics and Modeling. – 1999. – Vol.10, N 2. – P. 140-150.
12. *Andrianov A.N., Efimkin K.N., Levashov V.Y., Shishkova I.N.* The Norma Language Application to Solution of Strong Nonequilibrium Transfer Processes Problem with Condensation of Mixtures on the Multiprocessor System //Computational Science - ICCS 2001, Proceedings of International Conference, San Francisco, May 2001, Lecture Notes in Computer Science. – 2001. – Vol.2073. – P. 502-510.
13. *Андрианов А.Н.* Использование языка Норма для решения вычислительных задач на нерегулярных сетках //Тез. Всерос. науч. конф. "Фундаментальные и прикладные аспекты разработки больших распределенных программных комплексов". – Новороссийск, 1998. – С. 120-123.
14. *Андрианов А.Н., Жохова А.В., Четверушкин Б.Н.* Использование параллельных алгоритмов для расчетов газодинамических течений на нерегулярных сетках //Прикладная математика и информатика. – М.: Диалог-МГУ, 2000. – С. 68-76.
15. http: //www.keldysh.ru/norma

О РАСПАРАЛЛЕЛИВАНИИ ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМОВ

В.А. Вшивков

Институт вычислительных технологий СО РАН

Рассмотрим проблемы параллельной реализации численных алгоритмов. В настоящее время существует огромное количество алгоритмов решения задач, которые эффективно реализуются на однопроцессорных ЭВМ. Но по мере развития задачи рано или поздно рамки однопроцессорных машин становятся тесными и возникает необходимость использования более мощной техники. Причин, по которым возникает такая необходимость, по крайней мере две. Первая – мы хотим, чтобы задача, которая успешно решается на однопроцессорной машине, решалась быстрее. И если у нас есть возможность считать на 100-процессорном компьютере, неплохо было бы в 100 раз быстрее получить результат. Это важно, например, в задачах прогноза погоды, в управлении какими-либо технологическими процессами и т.п. Существуют и другие задачи, в которых своевременное получение результатов имеет решающее значение. Это важная причина, но для научных исследований она не является главной. Как правило, нет большой разницы, получится результат через час или через десять часов. И было время, когда на БЭСМ-6 отдельные варианты считались месяцы. Вторая причина заключается в том, что на многопроцессорном компьютере мы можем решить более сложную задачу, и, на мой взгляд, эта причина гораздо важнее, чем первая. Например, многие задачи физики плазмы стали численно решаться только с появлением мультикомпьютеров. Как известно, популярный в физике плазмы метод частиц в ячейках предъявляет особенно большие требования к памяти и быстродействию ЭВМ [1-4].

Некоторые задачи легко поддаются распараллеливанию, т.е. для них легко найти алгоритмы, эффективно реализуемые на мультикомпьютерах, поскольку они требуют выполнения большого числа независимых вычислений. Для решения других же приходится создавать по существу новые алгоритмы. На первый взгляд может показаться, что путь для разработки самого быстрого параллельного алгоритма должен начинаться с самого быстрого существующего последовательного алгоритма. И на практике так чаще всего и бывает. Но это, вообще говоря, неверно. Фактически наилучший параллельный алгоритм нередко не имеет даже отдаленного сходства с наилучшим последовательным алгоритмом. Покажем

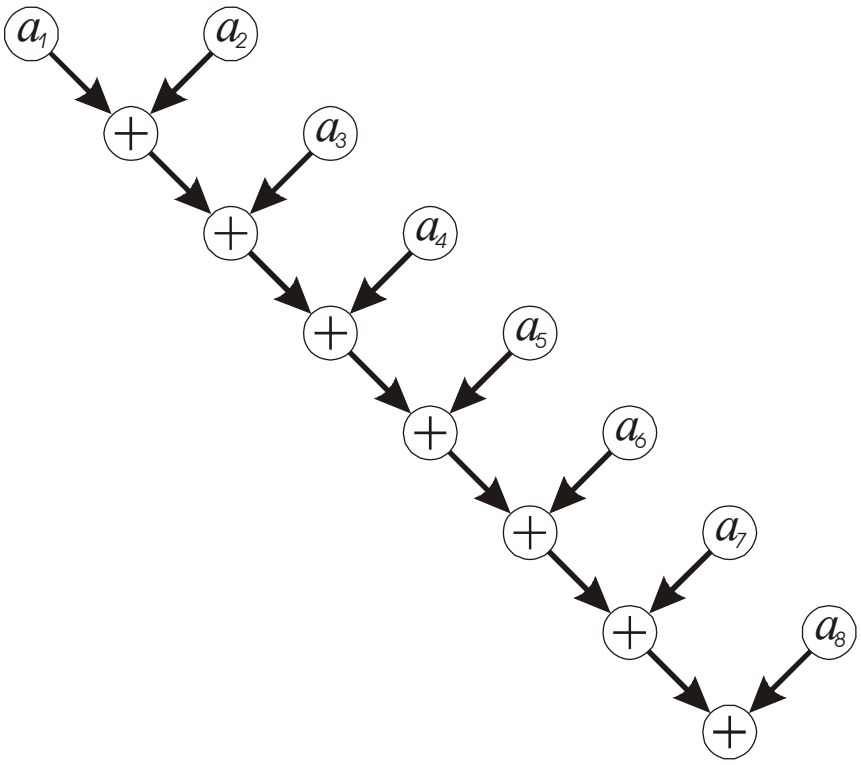

Рис. 1

на простом примере, что существуют как однопроцессорные, так и многопроцессорные алгоритмы, которые не получаются один из другого.

Рассмотрим известный пример сложения 8 чисел. Однопроцессорный алгоритм сложения этих чисел показан на рис. 1. Он эффективно реализуется с помощью оператора цикла и требует 7 операций сложения. Лучшего здесь ничего не придумаешь. Но, имея 8 процессоров и размещая каждое число на одном из процессоров, мы можем сложить эти числа за 3 операции, как показано на рис. 2. Легко видеть, что реализовать этот алгоритм на однопроцессорной машине более сложно, чем первый алгоритм, и очевидно, что он не будет более быстрым. Этот пример наглядно показывает разницу между последовательным и параллельным алгоритмами. В этом же примере можно увидеть и некоторые проблемы, возникающие при распараллеливании алгоритмов. Первая – это то, что возникают обмены данными между процессорами. И очевидно, что скорость передачи данных играет большую роль в оценке эффективности распараллеливания. Понятно, что чем больше происходит вычислений на процессоре и чем меньше обменов данными между процессорами, тем параллельный алгоритм эффективнее. Вторая проблема заключается в том, что хотя мы имеем 8 процессоров, мы не можем

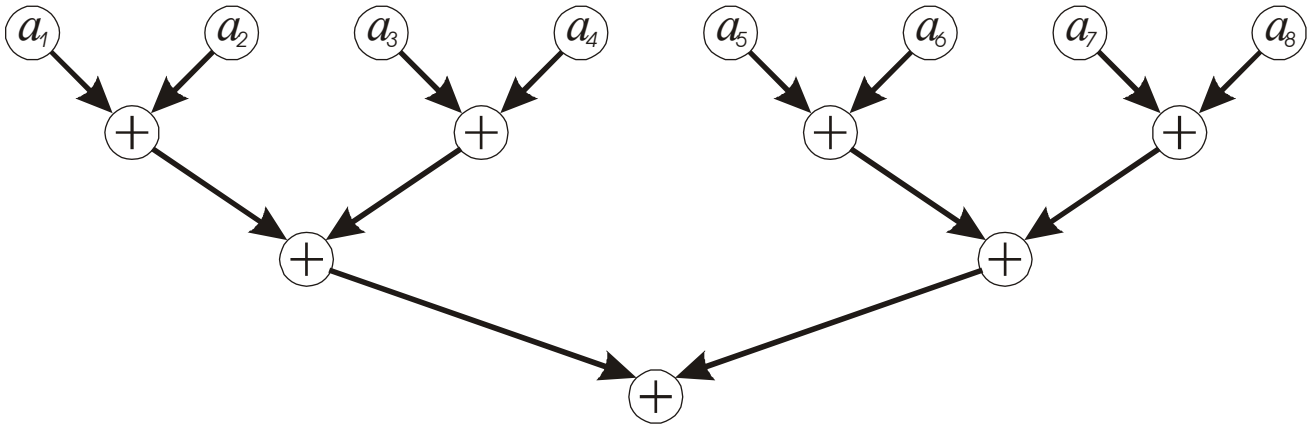

Рис. 2

сложить 8 чисел за одну операцию. В данном случае легко подсчитать, что для сложения N чисел необходимо порядка $\log_2(N)$ операций.

Возьмем более сложный пример. Рассмотрим итерационные методы решения системы линейных алгебраических уравнений:

$$Ax = b.$$

Здесь A – матрица размера $n \times n$; x и b – векторы размерности n. Как известно, решение многих дифференциальных уравнений после дискретизации на сетке сводится к решению системы линейных алгебраических уравнений большой размерности. И чем выше необходимая точность решения, тем больше размерность системы, что приводит к необходимости использования мультикомпьютеров. Итерационных методов решения систем линейных алгебраических уравнений в настоящее время существует довольно много, поэтому остановимся на двух из них, не самых лучших, но самых простых, для того, чтобы показать, какие появляются проблемы при распараллеливании алгоритмов. Первый из них – метод Якоби, который является примером явного алгоритма. В векторной форме он имеет вид

$$x^{m+1} = (I - D^{-1}A)x^m + D^{-1}b.$$

Здесь D – диагональная часть матрицы A; m – номер итерации. Реализация этого метода проста. Из первого уравнения системы находим первую компоненту вектора x^{m+1}, используя все остальные компоненты на итерационном слое m, из второго – вторую и т.д. Компоненты вектора x на слое $m+1$ вычисляются независимо друг от друга, поэтому их распределение между процессорами может быть произвольным. Например, на первый

процессор помещаем компоненты вектора x с номерами $1,...,n/p$, где p – количество используемых процессоров. На второй процессор поместим компоненты вектора с номерами $n/p+1,...,2n/p$ и т.д. Вычисление всех компонент может производиться одновременно на всех процессорах. После каждой итерации вычисленные на каждом процессоре компоненты вектора x рассылаются на все процессоры, а элементы матрицы $(I - D^{-1}A)$ и вектора $D^{-1}b,$ хранящиеся на каждом процессоре, не меняются. После этого цикл повторяется до тех пор, пока решение не будет получено с достаточной степенью точности.

Метод Якоби прост, но требует большого количества итераций. Поэтому возникла идея при вычислении компонент вектора x^{m+1} использовать уже вычисленные к этому моменту предыдущие компоненты вектора x^{m+1}. Эта идея приводит к методу Зейделя:

$$x^{m+1} = -D^{-1}Lx^{m+1} - D^{-1}Ux^{m} + D^{-1}b,$$

где L – нижняя треугольная часть матрицы A, а U – верхняя треугольная. Сходимость метода Зейделя, как правило, лучше, чем у предыдущего метода, но теперь мы вынуждены вычислять компоненты вектора x^{m+1} в строгом порядке, один за другим. Хотя реализация метода Зейделя простая, из вида формулы видно, что он относится к классу неявных алгоритмов. И печальным следствием этого является то, что он не поддается распараллеливанию. Если распределить компоненты вектора x^{m+1} между процессорами, как в предыдущем случае, то никакого ускорения не получится. Пока первый процессор насчитывает первую порцию компонент вектора x^{m+1}, остальные процессоры вынуждены простаивать, дожидаясь окончания его работы. Потом та же история происходит со вторым процессором и т.д. Но теперь обратимся к конкретным задачам.

Рассмотрим двумерное уравнение Пуассона в прямоугольной области S с граничными условиями первого рода на всей границе

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y).$$

В области S введем двумерную равномерную сетку с шагами h по каждому направлению и запишем простейшую разностную схему для решения уравнения Пуассона на заданной сетке. Она имеет вид системы линейных алгебраических уравнений:

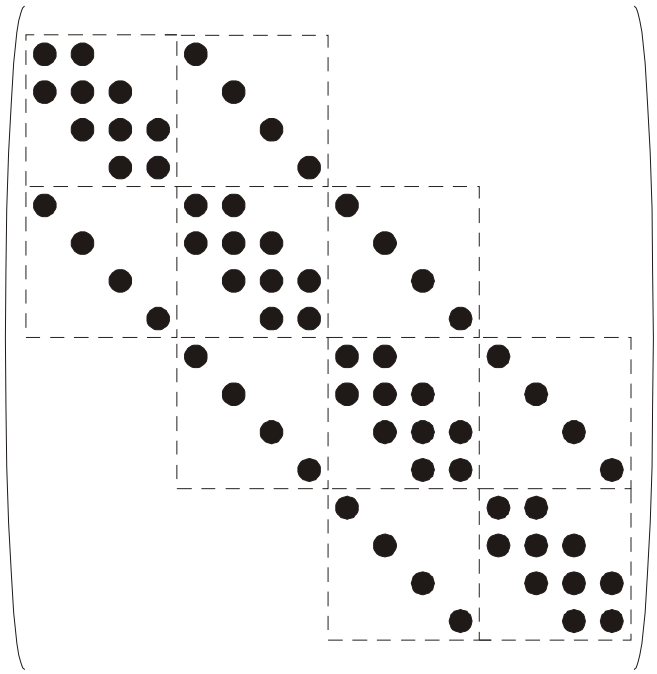

Рис. 3

$$\frac{u_{i+1,k} - 2u_{i,k} + u_{i-1,k}}{h^2} + \frac{u_{i,k+1} - 2u_{i,k} + u_{i,k-1}}{h^2} = f_{i,k},$$
$$i = 1,\ldots,I-1; \quad k = 1,\ldots,K-1,$$

где h – шаг сетки; i, k – индексы, указывающие, в какой точке находятся сеточные функции (i – по оси x, k – по оси y). Если теперь перенумеровать узлы сетки последовательно слева направо и снизу вверх, то матрица A в уравнении $Ax = b$ будет иметь характерный пятидиагональный вид. На рис. 3 приведен вид этой матрицы для $I = K = 5$. Черными кружками обозначены ненулевые элементы, остальные элементы равны нулю.

Метод Якоби для этой системы уравнений легко распараллеливается, а метод Зейделя – нет. Но тем не менее идею метода Зейделя здесь можно использовать. Для этого разобьем все узлы в шахматном порядке на красные и черные (произведем так называемое красно-черное упорядочивание узлов), как на рис. 4 (здесь белыми кружками обозначены красные узлы, черными – черные, а прямоугольниками – известные граничные значения). А сам метод разобьем на 2 этапа. На первом этапе будем вычислять значения сеточной функции $u_{i,k}$ в красных узлах, используя значения в черных узлах на предыдущей итерации:

$$u_{R,i,k}^{m+1} = \frac{1}{4}(u_{B,i+1,k}^{m} + u_{B,i-1,k}^{m} + u_{B,i,k+1}^{m} + u_{B,i,k-1}^{m} - h^2 f_{i,k}).$$

На втором этапе вычисляются значения $u_{i,k}$ в черных узлах, используя значения в красных узлах, но уже новые, вычисленные на первом этапе:

$$u_{B,i,k}^{m+1} = \frac{1}{4}(u_{R,i+1,k}^{m+1} + u_{R,i-1,k}^{m+1} + u_{R,i,k+1}^{m+1} + u_{R,i,k-1}^{m+1} - h^2 f_{i,k}).$$

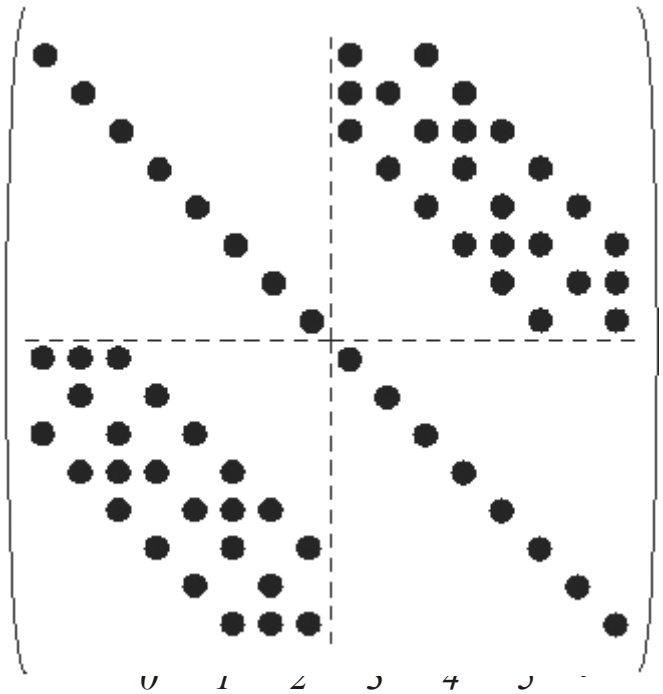

Рис. 4

Полученный метод распараллеливается так же просто, как и метод Якоби. Как легко видеть, компоненты вектора x^{m+1} на первом этапе вычисляются независимо друг от друга; то же самое происходит и на втором этапе. Сходимость метода лучше, чем у метода Якоби. Но, с другой стороны, это уже и не метод Зейделя. Если перенумеровать сначала красные узлы, а

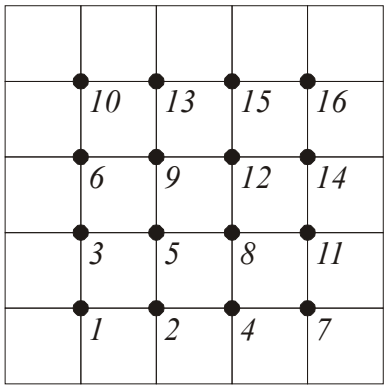

Рис. 6

затем черные последовательно слева направо и снизу вверх, то матрица A будет иметь вид, показанный на рис. 5. Еще следует заметить, что если в методе Якоби, имея n процессоров, мы получаем результат за один этап, то в методе Зейделя при красно-черном упорядочивании – за 2 этапа, и, кроме того, не можем использовать больше чем $n/2$ процессоров. Эта разница, конечно, чисто теоретическая, поскольку на практике, как правило, $n >> p$, тем не менее это иногда надо иметь в виду.

К этому можно еще добавить, что в данном конкретном случае можно полностью сохранить исходный метод Зейделя и при этом распараллелить его. Перенумеруем внутренние узлы сетки так, как показано на рис. 6. Тогда все 16 узлов разобьются на неравные по размеру группы, вычисления компонент вектора в которых уже легко распараллеливаются. Матрица A примет вид, изображенный на рис. 7. Как видно из рисунка, получается 7 групп с количеством элементов от 1 до 4. Следовательно, вычисления можно провести за 7 этапов и более 4 процессоров не понадобится. Коэффициент ускорения при этом получается равным $K_4 = t_1/t_4 = 16/7 = 2{,}29$, а эффективность распараллеливания $E_4 = K_4/4 = 0{,}57$.

Важную роль при получении последних методов сыграли большая разреженность матрицы системы линейных алгебраических уравнений и простота шаблона схемы. При более сложных матрицах усложняется и анализ возможности распараллеливания метода. Существуют алгоритмы,

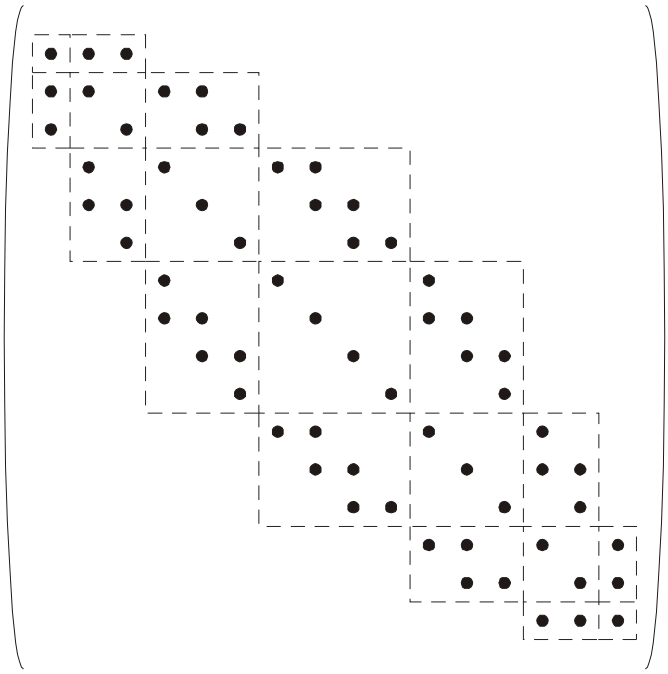

Рис. 7

позволяющие установить, на сколько групп независимых переменных можно разбить вектор решения, но изложение этой теории требует специального рассмотрения. Во всяком случае, учет конкретной задачи, конкретной системы линейных алгебраических уравнений позволяет найти наилучший вариант распараллеливания алгоритма.

Очень важным алгоритмом при решении дифференциальных уравнений является метод прогонки. На его основе разработан целый класс так называемых экономичных разностных схем. Эти алгоритмы неявные, следовательно, более устойчивые, чем явные схемы, иногда абсолютно устойчивые, поэтому они широко применяются при решении различных задач. На основе таких схем разработаны большие пакеты прикладных программ, которые хорошо оттестированы, показали свою надежность и универсальность. Поэтому при распараллеливании таких комплексов нежелательно отказываться от апробированных методов. Но, к сожалению, метод прогонки плохо поддается распараллеливанию.

Рассмотрим, к примеру, двумерное уравнение теплопроводности

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

и выберем для его решения метод продольно-поперечной прогонки. Схема метода двухэтапная и имеет следующий вид:

Первый этап:

$$\frac{u^{m+1/2} - u^m}{\tau/2} = \Lambda_1 u^{m+1/2} + \Lambda_2 u^m.$$

Второй этап:

$$\frac{u^{m+1} - u^{m+1/2}}{\tau/2} = \Lambda_1 u^{m+1/2} + \Lambda_2 u^{m+1}.$$

Здесь τ – временной шаг; Λ_1 – оператор разностной производной второго порядка по x

$$\Lambda_1 u_{i,k} = \frac{u_{i+1,k} - 2u_{i,k} + u_{i-1,k}}{h_x^2},$$

а Λ_2 – по y

$$\Lambda_2 u_{i,k} = \frac{u_{i,k+1} - 2u_{i,k} + u_{i,k-1}}{h_y^2}.$$

Метод абсолютно устойчив, имеет второй порядок точности по времени и пространственным переменным. Метод экономичен, т.е. количество арифметических операций на шаге по времени пропорционально количеству

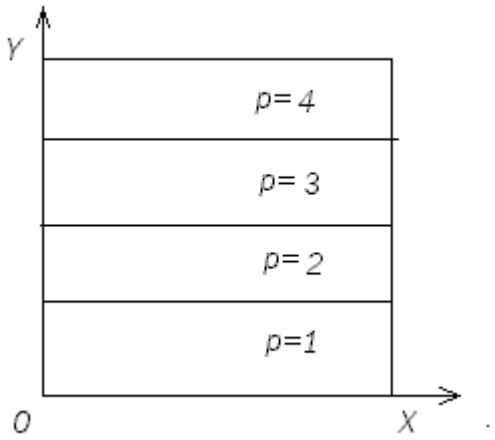

Рис. 8

узлов сетки. Но он требует использования метода прогонки. На первом этапе производится прогонка в направлении оси x, а на втором – в направлении оси

y. Допустим, область решения имеет вид прямоугольника и разбита на одинаковые подобласти вдоль осу *y*. Количество подобластей равно количеству используемых процессоров (рис. 8). Данные распределены по процессорам в соответствии с разбиением на подобласти. Следовательно, все необходимые данные для осуществления прогонки по *x* содержатся в этом процессоре.

Приведем для конкретности обсуждения формулы прогонки в общем виде. Пусть мы имеем следующую систему линейных алгебраических уравнений:

$$A_i y_{i-1} - C_i y_i + B_i y_{i+1} + F_i = 0, \quad i = 1,\ldots,N-1,$$

$$y_0 = \mu_1 y_1 + \nu_1, \quad y_N = \mu_2 y_{N-1} + \nu_2.$$

Коэффициенты $A_i, B_i, C_i, F_i, \mu_1, \mu_2, \nu_1, \nu_2$ считаются заданными. Формулы решения этой системы уравнений с помощью метода правой прогонки имеют следующий вид (номера указывают номера этапов решения):

1) $\alpha_1 = \mu_1, \quad \beta_1 = \nu_1,$

2) $\alpha_{i+1} = \dfrac{B_i}{C_i - A_i\alpha_i}, \quad \beta_{i+1} = \dfrac{A_i\beta_i + F_i}{C_i - A_i\alpha_i}, \quad i = 1,\ldots,N-1,$

3) $y_N = \dfrac{\mu_2\beta_N + \nu_2}{1 - \mu_2\alpha_N},$

4) $y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = N-1,\ldots,0.$

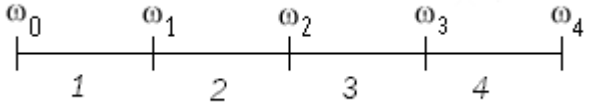

Рис. 9

Здесь α_i, β_i – векторы прогоночных коэффициентов. Легко подсчитать, что количество арифметических операций, требующихся для получения результата, будет порядка *8N*.

Если же необходимо осуществить прогонку в направлении оси *y*, то, поскольку данные для нее находятся на разных процессорах, использование вышеприведенных формул невозможно. В работе [5] предложен алгоритм распараллеливания прогонки, позволяющий получить тот же результат, что и однопроцессорная прогонка. Приведем соответствующие формулы. Разобьем область решения на K частей пропорционально количеству имеющихся

процессоров так, чтобы последний узел предыдущей подобласти совпадал с первым узлом следующей. Значение сеточной функции в этом узле обозначим ω_k, где k – номер левой подобласти (рис. 9). Далее будем считать, что обозначения в каждой подобласти одинаковы и расчеты в каждой из подобластей проводятся на своем процессоре.

Вначале на каждом из *K* процессоров поэтапно выполняем следующие операции:

1) $\alpha_1 = \beta_1 = 0, \quad \gamma_1 = 1,$

2) $\alpha_{i+1} = \dfrac{B_i}{C_i - A_i\alpha_i}, \quad \beta_{i+1} = \dfrac{A_i\beta_i + F_i}{C_i - A_i\alpha_i}, \quad \gamma_{i+1} = \dfrac{A_i\gamma_i}{C_i - A_i\alpha_i},$

$i = 1,\ldots,N-1,$

3) $p_N = r_N = 0, \quad q_N = 1,$

4) $p_i = \alpha_{i+1}p_{i+1} + \gamma_{i+1}, \quad q_i = \alpha_{i+1}q_{i+1},$

$r_i = \alpha_{i+1}r_{i+1} + \beta_{i+1}, \quad i = N-1,\ldots,0.$

По сравнению с предыдущим методом прогонки здесь появляются дополнительные одномерные массивы γ_i, p_i, q_i, r_i. После выполнения вышеприведенных четырех этапов на каждом процессоре мы связываем результаты, полученные на разных процессорах:

5) $A_k p_{N-1}^{(k)}\omega_{k-1} - (C_k - A_k q_{N-1}^{(k)} - B_k p_1^{(k+1)})\omega_k + B_k q_1^{(k+1)}\omega_{k+1} +$

$+ F_k + A_k r_{N-1}^{(k)} + B_k r_1^{(k+1)} = 0, \quad k = 1,\ldots,K-1.$

В этой формуле *k* – номер стыковочного узла. Значения сеточной функции в стыковочных узлах обозначены ω_k. ω_k совпадает с y_N для левого интервала (процессора) с номером *k* и с y_0 для правого интервала (процессора) с номером *k+1* (рис. 10).

Верхний индекс у функций p, q, r (k или $k+1$) показывает, из какого интервала берутся значения функции. Система уравнений для ω_k есть снова система линейных алгебраических уравнений с трехдиагональной матрицей, которая должна решаться методом прогонки. Размер системы уравнений совпадает с количеством используемых процессоров. Она может быть решена на одном процессоре, а затем найденные значения ω_k должны быть разосланы по соответствующим процессорам, либо, предварительно разослав исходные данные по всем процессорам, можно решить данную систему на каждом из процессоров.

После этого на каждом процессоре выполняем последний, шестой этап и получаем значения решения

$$6) \quad y_i = p_i \omega_{k-1} + q_i \omega_k + r_i, \quad i = 0,1,\ldots,N.$$

Если не считать пятый этап, то количество вычислений равно $17N$, где N – полное число узлов. По сравнению с обычной прогонкой количество вычислений выросло чуть больше чем в 2 раза, и на первый взгляд это приемлемая цена за возможность распараллеливания.

В работе [6] экспериментально была исследована эффективность распараллеливания прогонки. Для этого было рассмотрено решение уравнения

$$\frac{\partial^2 \Phi}{\partial y^2} = \rho(x, y)$$

в прямоугольной области с необходимыми граничными условиями. Для решения уравнения использовалась неявная схема, реализация которой

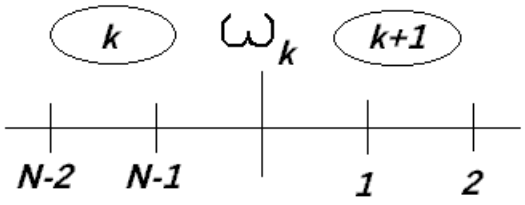

Рис. 10

требовала применения метода прогонки. Были решены 2 задачи, отличающиеся друг от друга только сегментацией полной расчетной области:

а) сегментация в виде P одинаковых вертикальных полос, прогонки ведутся вдоль направления y;

б) сегментация в виде P одинаковых горизонтальных полос, прогонки ведутся по-прежнему вдоль направления y (см. рис. 8).

Результаты расчетов на ЭВМ Siemens RM600 (ИВМиМГ СО РАН, Новосибирск) на сетке (3000,1000) узлов приведены в таблице.

Тип сегмен-тации	Запуск						
	Однопро-цессор-ный	Двухпроцессор-ный		Четырехпроцессорный			
	p=1	p=1	p=2	p=1	p=2	p=3	p=4
а	2,81	2,20	1,87	1,26	0,55	0,57	1,09
б	2,81	15,76	9,26	9,03	7,72	6,83	12,30

В таблице приведены времена (с) выполнения расчетов на каждом из использованных процессоров. Сразу бросается в глаза, что распараллеливание прогонки крайне неэффективно, время ее выполнения не только не уменьшается, а, наоборот, существенно превышает время выполнения на одном процессоре. Кроме того, интересно обратить внимание на то, что одинаковое количество операций на одинаковых процессорах выполняется за разное время.

Так как же быть с прогонкой? Использовать или не использовать? Здесь все зависит от конкретной задачи. Если можно обойтись без нее, то, конечно, ее не надо применять.

Но вернемся к уравнению теплопроводности. Очевидно, явные схемы для нее легко распараллеливаются. Вычисления на каждом шаге по времени на процессорах выполняются независимо друг от друга, обмен граничными значениями между соседними процессорами выполняется в конце каждого шага. А неявная схема требует использования прогонки, количество арифметических операций, приходящихся на каждый узел, больше, чем в явной схеме. Но она абсолютно устойчива, позволяет считать с существенно большим временным шагом, чем явная схема, и поэтому количество пересылаемой между процессорами информации существенно меньше. И здесь вступает в игру фактор, обычно не рассматриваемый в методах вычислений, – архитектура машины. Как связаны между собой процессоры? Какова скорость пересылки данных между процессорами? Каково быстродействие процессоров и количество операций, выполняемых на них между обменами с другими процессорами? И вот эти, казалось бы, чисто технические вопросы, могут играть решающую роль при выборе того или иного алгоритма. И может случиться, что неявные алгоритмы с применением распараллеливания прогонки окажутся более эффективными, чем более простые для распараллеливания явные.

Существует мнение, что при разработке алгоритма решения какой-либо большой задачи надо сразу ориентироваться на решение ее на многопроцессорной машине, т.е. сразу создавать параллельные алгоритмы. На мой взгляд, это чистая фантастика. Ведь алгоритмы появляются при решении каких-либо практических задач, и уже в самой задаче порой лежит возможность или невозможность разделения ее решения на несколько независимых подзадач. И перед специалистом-вычислителем возникает порой небогатый выбор между явными и неявными алгоритмами, о преимуществах и недостатках которых уже сказано выше. Чтобы глубже понять задачу, построить эффективный алгоритм, необходимо провести много предварительных тестовых расчетов, и в этом случае вешать на задачу еще и сложности пропуска ее на многопроцессорных ЭВМ крайне неэффективно. Поэтому решение практической задачи всегда будет начинаться с однопроцессорного алгоритма.

Список литературы

1. *Березин Ю.А., Вшивков В.А.* Метод частиц в динамике разреженной плазмы. – Новосибирск: Наука, 1980. – 94 с.
2. *Хокни Р., Иствуд Дж.* Численное моделирование методом частиц. – М.: Мир, 1987. – 638 с.
3. *Григорьев Ю.Н., Вшивков В.А.* Численные методы "частицы-в-ячейках". – Новосибирск: Наука, 2000. – 184 с.
4. *Вшивков В.А., Краева М.А., Малышкин В.Э.* Параллельная реализация метода частиц //Программирование. – 1997. – № 2. – С. 39-51.
5. *Яненко Н.Н., Коновалов А.Н., Бугров А.Н., Шустов Г.В.* Об организации параллельных вычислений и «распараллеливании прогонки» //Численные методы механики сплошной среды. – 1978. – № 7. – С. 136-139.
6. *Вшивков В.А., Тарнавский Г.А., Неупокоев Е.В.* Параллелизация алгоритмов прогонки: многоцелевые вычислительные эксперименты //Автометрия. – 2002. – Т.4. (в печати).

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ

В.Э. Малышкин

Институт вычислительной математики
и математической геофизики СО РАН

Введение

Основная тема обсуждения – какими свойствами должны обладать алгоритмы и программы численного моделирования для их эффективного выполнения на мультикомпьютерах и какими способами нужные свойства могут быть обеспечены. Иными словами, обсуждаются вопросы параллельной реализации алгоритмов численного моделирования на мультикомпьютерах.

Проблема здесь в том, что нет единой теории параллельных вычислений, а раз нет теории, то нет и не может быть конкретных рецептов на все случаи жизни, как построить хорошо параллельно реализуемый алгоритм. Однако можно говорить о правильной точке зрения на алгоритм и на его параллельную реализацию.

1. Необходимые свойства параллельной программы

Начнем с обсуждения новых свойств параллельной программы (ПП). Эти свойства не требовались (или их отсутствие было не очень критичным) от последовательных программ, и в обеспечении этих свойств состоит одна из главных трудностей распараллеливания последовательных и создания параллельных программ. Любая хорошая ПП должна в большей или меньшей степени обладать рядом необходимых свойств общего характера [1].

- При исполнении параллельной программы на мультикомпьютере в каждый его процессорный элемент (ПЭ) загружается по крайней мере одна программа со своим набором входных данных для обработки и, таким образом, вся *параллельная программа представляется как множество взаимодействующих процессов*[*] .
- *Недетерминизм исполнения*. Последовательная программа тоже может быть представлена системой взаимодействующих процессов, но для нее это ненужная и даже вредная роскошь, так как ухудшается производительность программы. Для последовательного исполнения на множестве всех процессов

[*] *Процесс* – исполняемая программа + данные.

должен быть установлен линейный порядок, для каждого процесса точно указано, за кем он должен исполняться, а именно процессы должны выполняться в заданном линейном порядке. Поэтому последовательная программа, многократно исполняемая с одними и теми же данными, будет всегда вырабатывать один и тот же результат.

Для параллельного исполнения на множестве процессов устанавливается частичный порядок, и процессы исполняются, не нарушая этот порядок. При каждом конкретном исполнении ПП (с одними и теми же входными данными) ее процессы исполняются, вообще говоря, в разном порядке относительно друг друга, и в случае ошибки ПП может вырабатывать разные результаты при разных ее исполнениях. Поэтому локализация ошибок в ПП крайне затруднена. ПП может годами правильно работать, чтобы в самый «нужный» момент преподнести неприятный сюрприз.

Спешить избавляться от недетерминизма исполнения, однако, не следует, так как в ходе выполнения ПП многочисленные ресурсы мультикомпьютера используются разными процессами разных ПП или операционной системы, ресурсы освобождаются в разные непредсказуемые моменты времени. Если не допускать недетерминизма, то свободные ресурсы вычислителя не всегда могут быть немедленно использованы (например, процесс мог бы начать исполняться и загрузить свободный ресурс, но ему предписано ждать завершения некоторого другого процесса, в результате и ресурс простаивает и ПП медленнее исполняется) и, как следствие, производительность ПП может резко упасть. Некоторые другие важные свойства ПП также не могут быть обеспечены без недетерминизма исполнения.

• *Гибкая настройка на все доступные ресурсы мультикомпьютера*. Серьезная проблема – распределение ресурсов, включая время, мультикомпьютера так, чтобы обеспечить их равномерную загрузку (отображение прикладного алгоритма на ресурсы мультикомпьютера). При исполнении ПП не должно быть так, чтобы, например, все процессы, кроме одного, завершились, а этот один ещё долго бы работал. Работа ПП завершается, когда завершились все её процессы. В тех случаях, когда независимо исполняющиеся процессы требуют существенно разное количество ресурсов, хорошего распределения, как правило, нет. Да и количество необходимых ПП ресурсов чаще всего становится известным лишь в ходе исполнения программы, до начала вычислений этой информации обычно нет.

• *Динамическая балансировка загрузки*. В зависимости от входных данных процессы могут потреблять разное количество ресурсов (например, исполняться за разное время в зависимости от числа итераций внутреннего

цикла). В ходе исполнения ПП количество потребляемых процессами ПП ресурсов также может существенно изменяться (подробный пример приведен ниже). Во всех этих случаях нужна динамическая (в ходе исполнения) балансировка загрузки мультикомпьютера.

- *Переносимость в классе мультикомпьютеров*. Параллельная программа должна быть хорошо переносима в классе мультикомпьютеров, т.е. хорошо исполняться на всех мультикомпьютерах некоторого класса. Ведь даже на одном и том же мультикомпьютере число доступных ПЭ меняется от одного запуска ПП на счет до другого. А ПЭ больших мультикомпьютеров вообще разделяются между задачами, и при каждом старте ПП ей выделяется разное количество ПЭ. Конечно, ПП должна хорошо (если можно, оптимально) выполняться на всех тех ресурсах, что ей выделены при текущем исполнении.
- *Динамизм поведения*. При реализации численных моделей большого размера ПП должна уметь учитывать поведение модели и подстраиваться под него для достижения высокой производительности. Детальный пример рассматривается далее.

Все перечисленные свойства очень непросто реализуются, а динамические свойства исключительно сложны в отладке. Достичь перечисленных свойств при создании параллельной программы (распараллеливании последовательной) можно на пути создания мелко фрагментированных программ, чья фрагментированность сохраняется в ходе вычислений. Каждый фрагмент кода определяет отдельный процесс. При этом должны соблюдаться два сильных условия:

1) все процессы должны потреблять примерно одинаковое количество ресурсов;
2) на множестве всех процессов должен существовать такой частичный порядок, что все процессы взаимодействуют только со своими непосредственными соседями, т.е. процесс p_i может взаимодействовать с процессом p_j, если $p_i < p_j$, и не существует процесса p такого, что $p_i < p < p_j$.

Явные численные алгоритмы на прямоугольных сетках – хороший пример класса алгоритмов, допускающих такое представление и нередко хорошо реализуемых на мультикомпьютерах. Примерами фрагментов могут служить вычисления внутри одной ячейки сетки или вычисления внутри всех ячеек строки либо слоя ячеек. Исполнение системы параллельно выполняющихся и взаимодействующих процессов на ПЭ мультикомпьютера можно организовать так, что они будут вести себя в мультикомпьютере подобно жидкости в системе сообщающихся сосудов: если некоторый ПЭ окажется перегруженным, то лишняя работа (процессы) «переливается» в

соседние (связанные физическими линками) процессоры. Таким способом достигаются высокое качество динамического распределения ресурсов и гибкая настройка параллельной программы на все доступные ресурсы, а значит, и ее переносимость в классе мультикомпьютеров. Программа может вообще ничего не знать о структуре коммуникационной сети мультикомпьютера и будет очень устойчива (практически 100%-ная устойчивость) к разным сбоям оборудования.

2. Параллельная реализация PIC

2.1. Вычислительная схема реализации метода частиц-в-ячейках

При моделировании физического явления методом частиц-в-ячейках [2-5] исходное физическое пространство представляется в компьютере *пространством моделирования* (ПМ). В тех задачах, которые мы решали (моделирование разлета облака плазмы в замагниченном фоне, взаимодействие электромагнитного пучка с пластинкой), ПМ имеет форму прямоугольного параллелепипеда.

Плазма представляется множеством модельных частиц, характеризующихся координатами, скоростью и величиной заряда. Частицы распределены в пространстве моделирования в соответствии с некоторым законом, который изменяется во времени.

В отличие от прямого метода частица-частица, в котором рассматриваются все парные взаимодействия между частицами, в PIC методе частицы непосредственно друг с другом не взаимодействуют. Частицы влияют друг на друга, изменяя значения электромагнитных полей.

Для дискретизации значений электрического E и магнитного H полей все пространство моделирования разбивается прямоугольной сеткой на ячейки. В вершинах ячеек определяются значения электрического E и магнитного H полей. Чем меньше шаг сетки, то есть чем больше ячеек, тем точнее представление полей (рис.1).

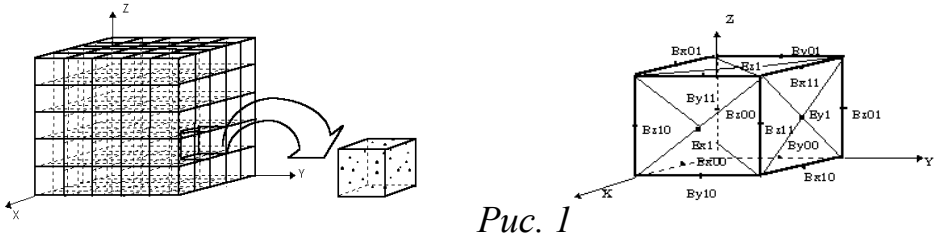

Рис. 1

С каждой ячейкой связаны частицы, находящиеся в этой ячейке в данный момент времени. Электромагнитные поля, заданные в вершинах

ячейки, действуют на частицы внутри нее, изменяя их скорости и координаты.

В свою очередь, “перелетевшие” в новую точку ПМ частицы влияют на значения электрического и магнитного полей в вершинах ячейки. Заряды частиц внутри ячейки интерполируются в вершины ячейки, таким образом вычисляются значения средней плотности тока и заряда в узлах сетки. Эти значения используются для пересчета значений электромагнитных полей. Назовем значения, заданные в узлах сетки, *сеточными значениями*.

Процесс моделирования состоит из серии временных шагов. На каждом временном шаге в каждой ячейке:

1) на основе значений электромагнитных полей вычисляется сила, действующая на частицу;
2) вычисляются новые скорости и координаты частиц;
3) из новых координат и скоростей частиц вычисляются значения средней плотности тока и заряда в узлах сетки;
4) из полученных значений средней плотности тока и заряда в узлах сетки пересчитываются значения электромагнитных полей.

Количество шагов моделирования определяется условиями физического эксперимента. Так как координаты частиц изменяются в процессе моделирования, частицы “перелетают” из ячейки в ячейку. Для устойчивости решения временной шаг выбирается столь малым, чтобы в течение одного шага частица улетала не далее соседней ячейки.

Заметим, что в первых трех стадиях вычисления ведутся в цикле по всем частицам. Время, затрачиваемое на выполнение этого цикла, составляет в зависимости от задачи от 70 до 90% всего счетного времени.

2.2. Особенности параллельной реализации метода частиц

Обычно задачи, решаемые с использованием метода частиц, характеризуются большим объемом данных и вычислений. На каждом временном шаге для вычисления новых скоростей и координат одной частицы требуется порядка ста арифметических операций. До того момента, когда можно наблюдать интересные физические результаты, требуется провести более 1000 временных шагов.

В алгоритме реализации PIC заложен внутренний параллелизм. Так как частицы между собой непосредственно не взаимодействуют, действия, связанные с перевычислением координат и скоростей частиц, можно проводить независимо друг от друга.

При решении задач на мультикомпьютерах возникает вопрос: каким образом распределить данные между ПЭ. Так как в каждый момент времени частица находится в некоторой ячейке, для уменьшения коммуникационных

издержек в процессе моделирования желательно так распределять данные, чтобы сеточные значения, заданные в вершинах ячейки, хранились в том же ПЭ, что и атрибуты частицы.

Чтобы удовлетворить этим условиям, можно

- либо дублировать массивы всех сеточных значений (во всех узлах сетки) в памяти каждого ПЭ. В этом случае частицы равномерно распределяются между ПЭ, вне зависимости от их расположения в ПМ. В этом случае, однако, на каждом временном шаге требуется обмениваться массивами сеточных значений между всеми ПЭ;
- либо применить пространственную декомпозицию – разрезать ПМ на подобласти по числу ПЭ. В этом случае массивы сеточных значений также разрезаются на части (с дублированием граничных строк и столбцов). Тогда в каждый ПЭ помещаются части массивов сеточных значений и те частицы, которые находятся в соответствующей подобласти ПМ. Пространство моделирования разрезается на подобласти таким образом, чтобы в каждом ПЭ находилось примерно одинаковое количество частиц. Таким образом, в начальный момент времени достигается балансировка загрузки ПЭ.

Однако в процессе моделирования частицы изменяют свое местоположение в ПМ, перелетая из ячейки в ячейку. Для того чтобы частицы по-прежнему находились в том же ПЭ, что и соответствующие ячейки, иногда приходится передавать атрибуты частицы из одного ПЭ в другой. Таким образом, в процессе моделирования может накопиться большой дисбаланс загрузки ПЭ – в некоторые ПЭ прилетит много больше частиц, чем в другие. Поэтому время от времени необходимо производить динамическую перебалансировку загрузки ПЭ.

Второй подход используется в наших параллельных реализациях метода частиц. Реализация основана на применении сборочного подхода к конструированию параллельной программы [6, 7], при котором определяется минимальный неделимый фрагмент вычислений, а вся программа собирается из этих атомарных фрагментов. Фрагментарная структура сборочной программы сохраняется и в исполняемом коде, что позволяет при необходимости передавать фрагменты на исполнение из одного ПЭ в другой. В нашем случае таким фрагментом является ячейка (вместе с частицами и сеточными значениями, необходимыми для вычисления силы, которая действует на частицу, а также со всеми вычислениями над этими данными). Поэтому частицы и сеточные переменные, заданные в вершинах ячейки, всегда оказываются в одном ПЭ.

3. Параллельная реализация PIC на линейке ПЭ

При реализации PIC на линейке ПЭ в качестве минимального фрагмента выбирается слой ячеек с тем, чтобы сохранить регулярность структур – хранить сеточные значения по-прежнему в массивах и производить вычисления в цикле над массивами. В начальный момент времени слои распределяются между ПЭ таким образом, чтобы в каждом ПЭ было примерно одинаковое количество частиц. Смежные слои назначаются в один или соседние ПЭ. Смежные слои, размещенные в одном ПЭ, образуют блок (рис.2).

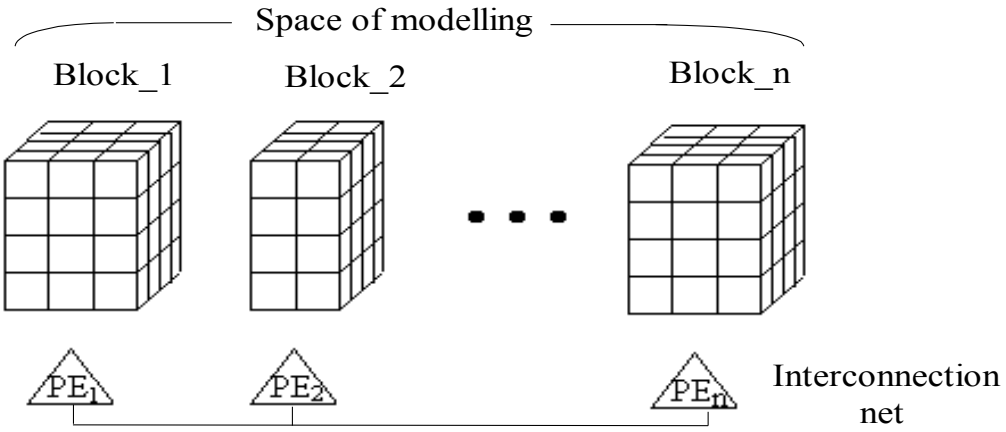

Рис. 2

Такая форма представления PIC называется *линеаризованной*.

4. Параллельная реализация PIC на решетке ПЭ

Линеаризованный PIC хорошо подходит для исполнения на 2-D решетке ПЭ. Отображение линеаризованного PIC на 2-D решетку ПЭ мультикомпьютера выполняется следующим способом.

Пусть дана двумерная решетка $l \times m$ процессорных элементов. Тогда **Block_i** назначается на обработку на i-ю строку ПЭ, $1 \le i \le l$. **Block_i** формируется таким образом, чтобы обеспечить равную нагрузку каждой строки ПЭ мультикомпьютера. Понятно, что это линеаризованное представление PIC. Затем **Block_i** делится на m подблоков **Block_ij**, j=1,...,m, которые назначаются на обработку на m ПЭ строки ПЭ. Каждый **Block_ij** формируется таким образом, чтобы обеспечить равную загрузку всех ПЭ строки (рис.3).

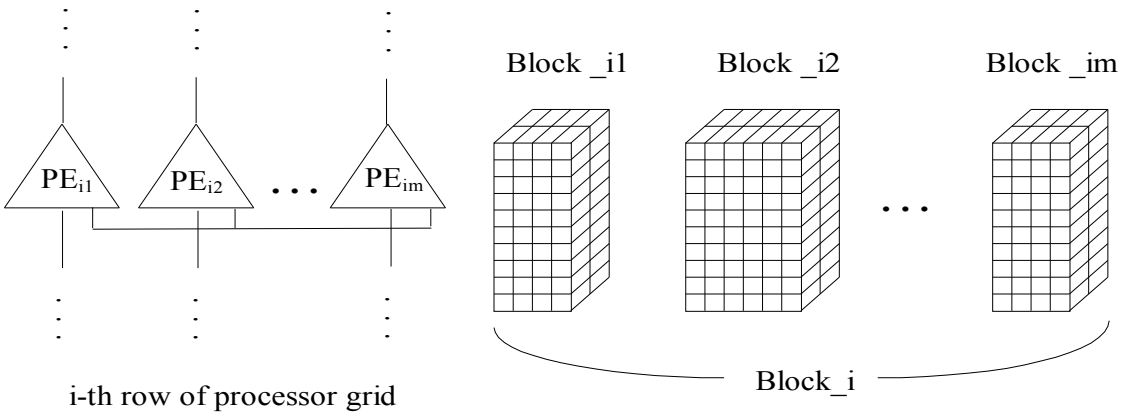

Рис. 3

Понятно, что в качестве минимального неделимого фрагмента вычислений здесь используется колонка ячеек пространства моделирования.

5. Балансировка загрузки ПЭ

Разработан эвристический алгоритм *начальной балансировки загрузки*. В каждом ПЭ строятся два массива:

- *A* содержит данные о количестве частиц в каждом слое ПМ (*A*[i] – количество частиц в *i*-м слое);
- в *S* заносятся данные о распределении слоев между ПЭ (*S*[*i*]) – номер первого слоя блока, размещенного в ПЭ$_i$).

Таким образом, каждый ПЭ "знает", какие слои назначаются в определенный ПЭ, это собственно *карта распределения нагрузки* между ПЭ.

Алгоритм имеет сложность *O(km)*, где *km* – количество слоев ПМ. Обозначим общее количество частиц как *NP*, число ПЭ – как *NPE*, среднее количество частиц в каждом ПЭ *ANP = NP/NPE*. Нулевой слой назначается в ПЭ$_0$ (нумерация ПЭ и слоев начинается с нуля). Далее в цикле по элементам массива *A*, слой *j*+1 назначается в ПЭ$_i$, если при этом количество частиц в первых *i*+1 ПЭ менее отличается от *(i+1)×ANP*, чем без частиц этого слоя. Этот эвристический алгоритм на всех проведенных тестах давал хорошие результаты.

Динамическая балансировка загрузки. Рассмотрим, как работает механизм перебалансировки загрузки. Каким-либо образом определяется порог разрешенного дисбаланса *BP* – максимальное количество частиц свыше среднего количества частиц, которое может обрабатываться одним ПЭ. Например, его можно вначале задать константой. После каждого шага по времени каждый ПЭ подсчитывает количество частиц в нем. Если это количество отличается от среднего количества частиц *ANP* больше, чем на заданный порог, ПЭ сигнализирует о необходимости проведения пере-

балансировки загрузки. Выделенный ПЭ собирает сведения о количестве частиц в каждом слое, записывает данные в уже описанный выше массив *A* и рассылает этот массив всем ПЭ. Далее, алгоритмом начальной балансировки загрузки каждый ПЭ строит массив *S*, в который заносятся данные о новом распределении слоев между ПЭ. После чего производится обмен слоями между соседними ПЭ – сначала (если необходимо) одновременно обмениваются слоями каждый $ПЭ_{2*i}$ и $ПЭ_{2*i+1}$, затем каждый $ПЭ_{2*i+1}$ и $ПЭ_{2*i+2}$.

Определение порога разрешенного дисбаланса BP. Были проведены эксперименты для *BP,* равного 0% (когда расписание перебалансировки строилось на каждом шаге), и 5% от общего числа частиц. При *BP*=5% были проведены 2 серии экспериментов. В первой серии новая карта распределения нагрузки строилась каждый раз, когда уровень дисбаланса превышал заданный порог. Во второй серии экспериментов новая карта строилась только тогда, когда увеличивалось максимальное количество частиц по сравнению с предыдущим шагом.

В двух других сериях экспериментов *BP* вычислялся динамически из времени, затраченного на выполнение перебалансировки, и времени, затрачиваемого на обработку одной частицы. Таким образом, можно было подсчитать, обработке какого количества частиц эквивалентно выполнение одной перебалансировки. Это значение не является точным, так как время, затрачиваемое на проведение одной перебалансировки, может значительно отличаться от времени, затрачиваемого на проведение другой перебалансировки, в зависимости от того, сколько слоев передается. Результаты тестирования показали, что алгоритм с динамически вычисляемым *BP* имеет лучшие характеристики. Это и понятно, так как этот алгоритм лучше “чувствует” дисбаланс загрузки.

В процессе моделирования возможна ситуация, когда большинство частиц собирается в одном слое, или в одном столбце ячеек, или даже в одной ячейке. Для организации вычислений в таком случае используется понятие *виртуального слоя*. Слой ПМ (вычисления и сеточные значения, заданные в вершинах ячеек слоя) представляется несколькими копиями (виртуальными слоями), а множество частиц исходного слоя распределяется между этими *виртуальными* слоями и, следовательно, между ПЭ, обеспечивая тем самым их равную нагрузку. Основной недостаток такого решения заключается в том, что появляется потребность в “длинных” взаимодействиях, так как исходно смежные слои могут теперь располагаться не в соседних ПЭ.

Благодаря введению понятия виртуального слоя в начальный момент времени достигается сбалансированная загрузка. Граничные слои дублируются в соседних ПЭ, частицы из этих слоев распределяются между ПЭ таким образом, чтобы в каждом ПЭ находилось равное количество частиц. То

есть алгоритм начальной балансировки загрузки видоизменен следующим образом. Слой 0, как и ранее, назначается в $ПЭ_0$. В цикле по элементам массива A слой j назначается не только в $ПЭ_i$, но и в $ПЭ_{i+1}$, если при добавлении частиц слоя j в $ПЭ_i$ количество частиц в первых i ПЭ становится большим чем $i \times ANP$. При накоплении дисбаланса в каждом ПЭ по массиву A строится новый массив S. Если новый массив S отличается от предыдущего, ПЭ обмениваются слоями. После чего распределяются частицы из граничных дублируемых слоев.

Централизованный алгоритм балансировки при реализации PIC на решетке ПЭ. Если при реализации PIC на мультикомпьютерах с большим количеством ПЭ использовать только линейные связи между ПЭ, то во время выполнения централизованных алгоритмов динамической балансировки загрузки требуются длинные коммуникации. При использовании топологии связи типа “решетка” радиус системы значительно уменьшается. Другим преимуществом реализации PIC на решетке ПЭ является тот факт, что уменьшается размер минимального неделимого фрагмента вычислений.

Динамическая перебалансировка загрузки происходит в два этапа. Сначала перераспределяются слои между строками ПЭ решетки. Используется описанный выше алгоритм динамической балансировки для линейки ПЭ. Затем новые блоки распределяются между ПЭ строк (рис.4).

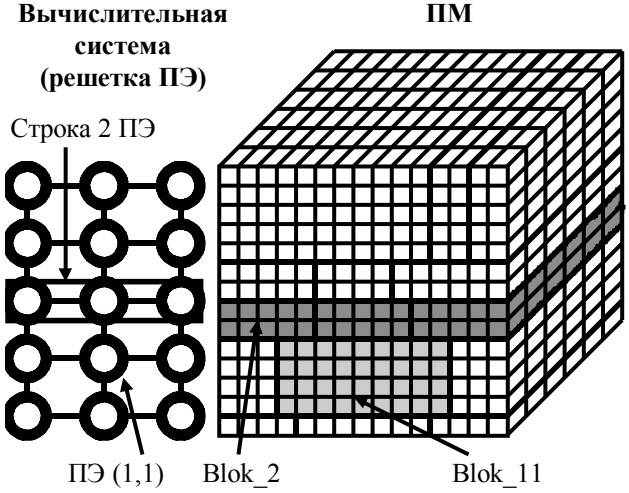

Рис. 4

Основной диффузионный алгоритм. В диффузионных алгоритмах решение о направлении передачи данных из более загруженных ПЭ в менее загруженные принимается локально. Цель – сбалансировать загрузку нескольких ПЭ в небольшой окрестности – локальной области. Все ПЭ мультикомпьютера покрываются такими локальными областями.

В основных диффузионных алгоритмах при выборе локальных областей должно выполняться условие, что нельзя разделить систему на части, не разделив при этом хотя бы одну локальную область (бесшовное покрытие). Балансировка загрузки в области происходит лишь на основе информации о текущей загрузке ПЭ области. Диффузионные алгоритмы отличаются размерами (и формами) локальных областей.

Пусть размер области ограничивается 2 ПЭ. Поведение системы во время принятия решения о направлении передачи данных синхронизировано. Принятие решения производится в несколько шагов. На четном шаге каждая пара ($ПЭ_{2i}$, $ПЭ_{2i+1}$) ($0 \le i < N/2$, где N – количество ПЭ) обменивается информацией о количестве частиц и принимает решение о передаче части частиц из более загруженного ПЭ в менее загруженный с целью сравнять загрузку. После чего каждый ПЭ имеет новую информацию о количестве частиц в нем (рис.5).

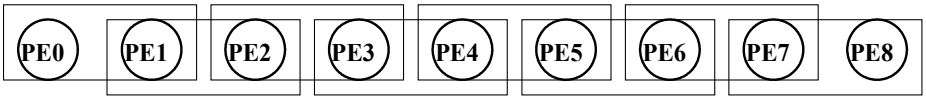

Рис. 5

На нечетном шаге каждая пара ($ПЭ_{2i+1}$, $ПЭ_{2i+2}$) ($0 \le i < (N-1)/2$, где N – количество ПЭ) обменивается информацией о количестве частиц и принимает решение о передаче части частиц из более загруженного ПЭ в менее загруженный с целью сравнять загрузку.

Такая диффузия может продолжаться в течение нескольких шагов, пока, наконец, данные не будут распределены равномерно. А в рассмотренном выше централизованном алгоритме перебалансировка производится за два шага.

Тестирование алгоритма показало, что хотя при увеличении количества шагов качество балансировки улучшается, общее время работы программы изменяется незначительно.

Заключение

В заключение можно сформулировать два рецепта:

- Строить мелко фрагментированные алгоритмы.
- Обеспечивать локальность взаимодействий между фрагментами, не допускать длинных транзитивных зависимостей, как в итеративных процессах.

Такое представление обеспечит хорошую параллельную реализацию алгоритмов. Конечно, совсем не обязательно всегда реализовать динамическую балансировку.

Список литературы

1. *Малышкин В.Э.* Параллельное программирование мультикомпьютеров: Учебное пособие. – Ярославль: Изд-во Ярослав. ун.-та, 1999. – 134 с.
2. *Березин Ю.А., Вшивков В.А.* Метод частиц в динамике разреженной плазмы. – Новосибирск: Наука, 1980. – 94 с.
3. *Вшивков В.А., Краева М.А., Малышкин В.Э.* Параллельная реализация метода частиц //Программирование. – 1997. – № 2. – С. 39-51
4. *Kraeva M.A., Malyshkin V.E.* Implementation of PIC Method on MIMD Multicomputers with Assembly Technology //The International Conference on High Performance Computing and Networking – HPCN Europe. – 1997. LNCS. – Vol.1225. – P. 541-549.
5. *Kraeva M.A., Malyshkin V.E.* Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers //International Journal on Future Generation Computer Systems. – 2001. – Vol.17. – P. 755-765.
6. *Malyshkin V.E.* Functionality in ASSY System and Language of Functional Programming //Proceedings of the First International Symposium on Parallel Algorithms/Architectures Synthesis. 1995, Aizu, Japan. IEEE Comp. Soc. PressLos Alamitos, California. – 1995. – P. 92-97.
7. *Вальковский В.А., Малышкин В.Э.* Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 126 с.

НЕКОТОРЫЕ ПОДХОДЫ К ПОСТРОЕНИЮ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ВЫЧИСЛИТЕЛЬНОЙ ГИДРОДИНАМИКИ*

С.В.Тимченко

НИИ ПММ при Томском госуниверситете

Введение

В последнее время, в связи с широким распространением сравнительно недорогих многопроцессорных систем, резко вырос интерес к параллельным методам вычислительной гидродинамики. Чрезвычайно быстрый рост производительности современных компьютеров дал реальную возможность приступить к решению задач гидро- и газодинамики, которые еще совсем недавно были недоступны для численного моделирования. При этом необходимо отметить, что применение традиционных подходов при решении задач математической физики приводит к тому, что используется лишь малая часть современных вычислительных мощностей.

Следуя [1], отметим четыре требования к вычислительным алгоритмам, обеспечивающие их успешную адаптацию к архитектуре многопроцессорных вычислительных систем с распределенной памятью:

1) *внутренний параллелизм*, позволяющий разбить исходный алгоритм на некоторое количество подзадач (по возможности совпадающее с числом процессоров);
2) обеспечение *равномерности загрузки* процессоров;
3) *минимизация обмена информацией* между процессорами;
4) *простота* алгоритма.

Если первое из этих требований носит алгоритмический характер, то остальные три связаны в значительной степени с экономической стороной вопроса. Действительно, только существенное снижение стоимости расчетов может побудить исследователя перейти на новую вычислительную технику и новые вычислительные алгоритмы. Если в стоимость вычислений закладывать, в частности, оплату труда исследователя, то простота алгоритма (и, как следствие, скорость переноса программ с последовательных компьютеров на параллельные) становится чрезвычайно существенной.

С точки зрения практической методологии при построении параллельных алгоритмов часто выделяют следующие три подхода:

* Работа выполнена при поддержке РФФИ, грант №02-01-01022.

1. *Конвейерное решение.* При решении многих нестационарных задач возможна следующая декомпозиция исходного алгоритма – каждый процессор получает решение некоторой подзадачи в некоторый момент времени, после чего это решение передается другим процессорам для использования при решении других подзадач, а данный процессор продолжает решать свою подзадачу для последующих моментов времени.
2. *Геометрическое решение.* При нахождении решения многих гидродинамических задач приходится сталкиваться с регулярными геометрическими структурами с пространственно-ограниченным взаимодействием. Благодаря этой однородности исходные данные могут быть единообразно распределены между процессорами, каждый из которых будет находить решение поставленной задачи в отведенной ему геометрической подобласти при ограниченном обмене информацией с другими процессорами. Как правило, этот обмен связан с необходимостью аппроксимации производных на границах подобластей.
3. *Коллективное решение.* При решении многих физических задач необходимо неоднократно выполнять одну и ту же часть программы с различными, слабосвязанными друг с другом наборами входных данных. Например, при решении различных задач оптимизации естественным является распределение по процессорам программы вычисления целевой функции.

1. Пример геометрического параллелизма

Рассмотрим задачу расчета течений в пространственном турбулентном пограничном слое (ТПС). На основе принципа геометрического параллелизма построим численный алгоритм для решения этой задачи на параллельной вычислительной системе с распределенной памятью.

1.1. Постановка задачи

Рассмотрим задачу пространственного обтекания затупленных тел сложной формы гиперзвуковым потоком вязкого однородного газа под углами атаки и скольжения. Для численного решения задачи будем использовать криволинейную систему координат $\{x^i\}$, нормальным образом связанную с поверхностью обтекаемого тела: x^3 отсчитывается по нормали к телу, а x^1 и x^2 некоторым образом выбраны на его поверхности. В системе координат $\{x^i\}$ соответствующие уравнения ТПС имеют следующий безразмерный вид:

$$D_i(\rho u^i a^{1/2} a_{(ii)}^{-1/2}) = 0,$$

$$\rho\,(Du^{\alpha} + A^{\alpha}_{\beta\delta}u^{\beta}u^{\delta}) = -\frac{\gamma-1}{2\gamma}a^{\alpha\beta}a^{1/2}_{(\alpha\alpha)}D_{\beta}P + D_3(\mu D_3 u^{\alpha}),$$

$$\rho DT = \frac{\gamma-1}{\gamma}D^{*}P + D_3\left(\frac{\mu}{\mathrm{Pr}}D_3 u^{\alpha}\right) + 2\mu B_{\alpha\beta}D_3 u^{\alpha}D_3 u^{\beta},$$

$$p = \rho T, \mu = \mu_l + \Gamma\mu_t, \left(\frac{\mu}{\mathrm{Pr}}\right) = \left(\frac{\mu}{\mathrm{Pr}}\right)_l + \Gamma\left(\frac{\mu}{\mathrm{Pr}}\right)_t,$$

$$D_i = \partial/\partial x^i, D^{*} = u^{\alpha}a^{-1/2}_{\alpha\alpha}D^{\alpha},$$

$$D = D^{*} + u^3 D_3, B_{\alpha\beta} = a^{\alpha\beta}a^{-1/2}_{(\alpha\alpha)}a^{-1/2}_{(\beta\beta)},$$

$$A^{\alpha}_{\alpha\alpha} = \frac{a_{12}}{a}\left(\frac{\partial\sqrt{a_{(\alpha\alpha)}}}{\partial x^{\beta}} + \frac{a_{12}}{a_{(\alpha\alpha)}}\frac{\partial\sqrt{a_{(\alpha\alpha)}}}{\partial x^{\alpha}} - \frac{1}{\sqrt{a_{(\alpha\alpha)}}}\frac{\partial a_{12}}{\partial x^{\alpha}}\right),$$

$$A^{\alpha}_{12} = A^{\alpha}_{21} = \frac{1}{2a}\left[\sqrt{a_{11}a_{22}}\left(1 + \frac{a^2_{12}}{a_{11}a_{22}}\right)\frac{\partial\sqrt{a_{(\alpha\alpha)}}}{\partial x^{\beta}} - 2a_{12}\frac{\partial\sqrt{a_{(\beta\beta)}}}{\partial x^{\alpha}}\right],$$

$$A^{\beta}_{\alpha\alpha} = \frac{\sqrt{a_{(\beta\beta)}}}{a}\left(\frac{\partial a_{12}}{\partial x^{\alpha}} - \sqrt{a_{(\alpha\alpha)}}\frac{\partial\sqrt{a_{(\alpha\alpha)}}}{\partial x^{\beta}} - \frac{a_{12}}{\sqrt{a_{(\alpha\alpha)}}}\frac{\partial\sqrt{a_{(\alpha\alpha)}}}{\partial x^{\alpha}}\right), \alpha \neq \beta.$$

Данная система решается с граничными условиями на поверхности тела и на внешней границе пограничного слоя:

$$x^3 = \infty : u^{\alpha} = u^{\alpha}_e(x^1, x^2), T = T_e(x^1, x^2);$$

$$x^3 = 0 : u^{\alpha} = u^{\alpha}_w(x^1, x^2), T = T_w(x^1, x^2), (\rho u^3)_w = G(x^1, x^2).$$

Здесь все линейные размеры отнесены к характерному линейному размеру L, а нормальная координата x^3 – к $L/\sqrt{\mathrm{Re}}$; $\sqrt{2c_{p0}T_0}u^{\alpha}$, $\sqrt{2c_{p0}T_0}u^{\alpha}/\sqrt{\mathrm{Re}}$ – физические компоненты вектора скорости; PP_0, $\rho\rho_0, TT_0, \mu\mu_0$ – соответственно давление, плотность, температура и коэффициент вязкости; Г – коэффициент перемежаемости; $a_{\alpha\beta}$ – ковариантные компоненты первой квадратичной формы поверхности тела. Нижние индексы *w, 0, e* здесь и далее соответствуют значениям газодинамических параметров на поверхности тела, в набегающем потоке и на внешней границе пограничного слоя соответственно. Нижние индексы *l* и *t* соответствуют ламинарным и

турбулентным значениям параметров. $\mathrm{Re} = \sqrt{2c_{p0}T_0}\rho_0 L/\mu_0$ – число Рейнольдса, $\mathrm{Pr} = \mu c_p/\lambda$ – число Прандтля.

При определении коэффициента турбулентной вязкости использовалась алгебраическая модель Себеси-Смита, а коэффициент перемежаемости Г в переходной области течения вычислялся на основе формул Чена и Тайсона.

Значения $u^\alpha = u_e^\alpha(x^1,x^2), T_e(x^1,x^2)$ на внешней границе пограничного слоя определяются из решения следующей системы уравнений:

$$\rho_e(Du_e^\alpha + A_{\beta\delta}^\alpha u_e^\beta u_e^\delta) = -\frac{\gamma-1}{2\gamma}a^{\alpha\beta}a_{(\alpha\alpha)}^{1/2}D_\beta P,$$

$$\rho_e DT_e = \frac{\gamma-1}{\gamma}D^* P,$$

где $P(x^1,x^2)$ – известное распределение давления по поверхности тела. В гиперзвуковом случае для определения *P* используется формула Ньютона:

$$P = \cos^2\alpha,$$

где α – угол между вектором скорости набегающего потока и нормалью к поверхности тела.

В рамках данной постановки рассматриваются течения в пространственном пограничном слое около поверхности трехосных эллипсоидов и гиперболоидов, обтекаемых под углами атаки и скольжения.

1.2. Общий алгоритм решения задачи

Для численного решения исходной задачи использовалась специально выбранная система координат на поверхности тела (подробно описанная в [2]), представляющая собой систему координат полярного типа x^1 – радиальная и x^2 – угловая координата с началом в критической точке $x^1 = 0$. Для удобства численного интегрирования с учетом особенности выбранной системы координат осуществлялся переход к новым переменным типа переменных Дородницына, позволяющим разрешить особенности в исходных уравнениях на критической линии. Использовалась неявная конечно-разностная схема, имеющая четвертый порядок аппроксимации по поперечной координате, второй порядок по окружной координате и первый порядок точности по маршевой координате, представляющая собой обобщение на трехмерный случай схемы Петухова и использующая переменный шаг по нормальной координате. Методические расчеты, проведенные на последовательности сгущающихся сеток, показали, что для

достижения точности в 1% по распределениям теплового потока вдоль поверхности тела достаточно использовать сетку, содержащую 15 точек по нормальной координате, 64 точки по окружной координате и 30 точек по маршевой координате. Поэтому в дальнейшем все серийные расчеты были проведены на этой сетке.

В конвективном операторе производные по маршевой координате заменяются разностями против потока, а производные по окружной координате заменяются центральными разностями на основе решения, полученного на предыдущей итерации на текущей окружности. Так как данная система координат вырождается в критической точке, то для решения исходных уравнений в критической точке использовалась невырожденная криволинейная система координат. После определения решения в невырожденной системе координат производился пересчет в старую систему координат.

Используемый метод решения фактически является вариантом метода релаксации по строкам, построенного на основе метода прогонки из работы Петухова. Несмотря на то, что этот метод является неявным, он очень хорошо может быть адаптирован для параллельных компьютерных систем с распределенной памятью, так как декомпозиция расчетной области в окружном направлении x^2 позволяет получать решение в каждой подобласти на текущей итерации независимо от других подобластей, а из-за использования высокого порядка аппроксимации по поперечной координате количество обменов между отдельными процессорами сравнительно невелико.

1.3. Декомпозиция расчетной области

Вышеописанный численный метод решения уравнений ТПС реализуется в виде программного кода 3DTBL, предназначенного для использования на сети из N компьютеров, обменивающихся пакетами данных при помощи пакета MPI. Вычислительная область Ω разбита на N непересекающихся подобластей Ω_j (j =1,..., N). Каждая подобласть дискретизируется на K сеточных точек в направлении x^2:

$$\Omega = \bigcup_{J=1}^{N} \Omega_j, \Omega_j = \left\{ \left(x^1, x^2, x^3 \right) \mid 0 \le x^1 \le x^1_*; x^2_j \le x_2 \le x^2_{j+1}; 0 \le x^3 \le x^3_e \right\}.$$

1.4. Общая стратегия

Для удобства дальнейшего изложения введем следующие обозначения: *SEND(T_i, T_j, A)* и *RECV(T_i, T_j, A)*, которые соответственно означают посылку (получение) структуры данных на узле кластера T_i на (из) узел(ла) кластера T_j

(нижние индексы i и j отличаются на единицу). Этим операторам соответствуют простейшие функции MPI: *MPI_SEND* и *MPI_RECV*. Будем также считать, что индексная арифметика выполняется по модулю N. Выражение A^m_j означает, что структура данных A относится к точке сетки с номером m в окружном направлении на узле T_j . В сеточных точках с номерами от 1 до K решение вычисляется с использованием процессора с тем же самым номером, а необходимые для замыкания задачи в размещенной на этом кластере вычислительной подобласти структуры данных A^0_j и A^{K+1}_j передаются с "соседних" процессоров. Тогда схема вычислений может быть представлена следующим способом:

1**.** Решение в критической точке независимо получается на каждом процессоре. На этой стадии алгоритм не параллелен.

2. При получении решения системы уравнений ТПС на боковой поверхности тела для конечно-разностного представления производных в окружном направлении необходимо знать значения неизвестных функций на предыдущей итерации в соседних сеточных точках в направлении x^2. Для этого перед каждой итерацией каждый нечетный процессор сначала посылает необходимые значения функций левым и правым "соседним" процессорам, а потом получает соответствующую информацию от них. Каждый четный процессор, наоборот, первоначально получает информацию от правого и левого "соседних" процессоров и после этого посылает соответствующие данные обратно. Посылка и получение информации на процессоре T_j могут быть представлены следующим способом:

IF(odd(j))
    SEND(T_j, T_{j+1}, A^K_j)
    SEND(T_j, T_{j-1}, A^1_j)
    RECV(T_j, T_{j+1}, A^{K+1}_j)
    RECV(T_j, T_{j-1}, A^0_j)
ENDIF
IF(even(j))
    RECV(T_j, T_{j-1}, A^0_j)
    RECV(T_j, T_{j+1}, A^{K+1}_j)
    SEND(T_j, T_{j-1}, A^1_j)
    SEND(T_j, T_{j+1}, A^K_j)
ENDIF

Здесь предикаты *odd* и *even* дают значение "истина" в случаях нечетности и четности своих аргументов соответственно.

3. При этом шаге каждый процессор получает решение уравнений ТПС в вышеописанной подобласти на текущей итерации полностью, независимо от других процессоров.

4. После окончания текущей итерации необходимо проверить определенный критерий *D* сходимости итерационного процесса. Этот процесс заканчивается при достижении сходимости для всех точек x^1=const, т.е. при реализации условия $\forall j : \left| Y_j^{(i)} - Y_j^{(i-1)} \right| < \delta$, где δ – некоторое число; *i* – номер текущей итерации; *j* – номер процессора; Y_j – вектор неизвестных функций. Для синхронизации вычислительных процессов на различных процессорах необходимо одновременное завершение итераций на всех процессорах, для чего при помощи функции *MPI_REDUCE* вычисляется величина $D = \bigwedge_j D_j$, где $D_j = \left(\left| Y_j^{(i)} - Y_j^{(i-1)} \right| < \delta \right)$. Если значение *D*="ложь", то выполняется шаг 2. В противоположном случае переменная x^1 увеличивается на Δx^1, и после этого выполняется шаг 2.

Более подробно данный алгоритм (в том числе для гетерогенных кластерных систем) рассматривается в работе [3].

1.5. Эффективность параллельного алгоритма

Важнейшей характеристикой любого параллельного алгоритма является его эффективность *E,* т.е. величина

$$E = \frac{t_1}{N t_N},$$

где t_i – время решения задачи на *i* процессорах.

В данной задаче проблема уменьшения объема обмена данных между процессорами решается за счет использования схемы высокого порядка аппроксимации в направлении x^3, по которому производится декомпозиция расчетной области. В результате число обменов между процессорами невелико по сравнению с объемом вычислений. Поэтому можно дать приближенную оценку эффективности:

$$E = \frac{1 + K \cdot L}{N + K \cdot L},$$

где *K* – число точек вдоль тела; *L* – число точек в окружном направлении; *N* – число процессоров. Снижение эффективности параллельного алгоритма с увеличением числа процессоров объясняется исключительно наличием нераспараллеленного участка в программе (начальный участок – получение решения в критической точке – дублируется на всех процессорах). Поэтому

увеличение числа точек в продольном либо окружном направлении приводит к увеличению эффективности распараллеливания. Некоторые результаты расчетов на кластере Томского госуниверситета приведены в таблице.

Эффективность параллельного алгоритма, %

Число процессоров	Сетка K=30,L=64	Сетка K=20,L=64	Сетка K=30,L=32
2	99	99	99
4	98	98	97
8	97	96	94
16	94	93	88

2. Пример коллективного решения

Как показывает анализ литературы [4-6], большинство параллельных вычислительных алгоритмов в той или иной мере основано на декомпозиции вычислительной области на ряд подобластей, в каждой из которых решение ищется на отдельном процессоре. Особенно широко распространены параллельные алгоритмы для явных разностных схем. Однако применение

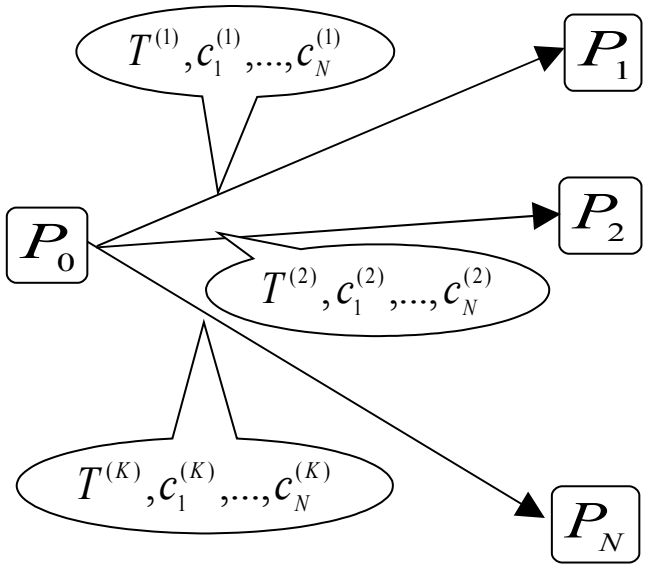

Рис.1. Рассылка нулевым процессом значений температуры и концентраций при помощи функции MPI_SCATTER

различных быстрых методов решения задач линейной алгебры, таких как прогонка, препятствует в этих случаях применению декомпозиции при построении параллельных алгоритмов.

При решении задач гидродинамики многокомпонентных химически

неравновесных реагирующих смесей основное время вычислений тратится на расчет коэффициентов переноса – коэффициентов вязкости, теплопроводности и диффузии. Этот факт позволяет предложить другой подход к построению параллельных вычислительных алгоритмов:

1. Основной вычислительный блок размещается на корневом (нулевом) процессоре, на остальных процессорах размещается программа расчета коэффициентов переноса.
2. После инициализации параллельного приложения (инициализации системы передачи сообщений) на корневом (нулевом) процессоре запускается основной вычислительный блок.
3. На каждой итерации корневой процесс рассылает подчиненным процессам необходимую для расчета коэффициентов переноса информацию (значения температуры и концентраций). В используемой параллельной библиотеке передачи сообщений MPI это осуществляется функцией *MPI_SCATTER* (рис. 1).
4. После расчета коэффициентов переноса их значения собираются на корневом процессе при помощи функции *MPI_GATHER* (рис. 2).

Другим характерным примером построения параллельных алгоритмов на основе коллективного решения являются многие методы глобальной оптимизации (например эволюционные и генетические алгоритмы и методы управляемого случайного поиска [5-6]).

Список литературы

1. *Четверушкин Б.Н.* Кинетически-согласованные схемы в газовой динамике. Новая модель вязкого газа, алгоритмы, параллельная реализация, приложения. – М.: Изд-во Моск. ун-та, 1999. – 232 с.
2. *Бородин А.И., Пейгин С.В.* Пространственный тонкий вязкий ударный слой при отсутствии в течении плоскостей симметрии //Изв. АН СССР. МЖГ. – 1989. – № 2. – С.150-158.
3. *Пейгин С.В., Тимченко С.В.* Расчет пространственного турбулентного пограничного слоя на сети компьютеров различной производительности //ЖВМиМФ. – 1998. – Т.38, №3. – С. 510 - 519.
4. Parallel Computation Fluid Dynamics. New Trends and Advances /J.Periaux et al. (ed). Elsevier Science B.V. Amsterdam, 1995. – 560 p.

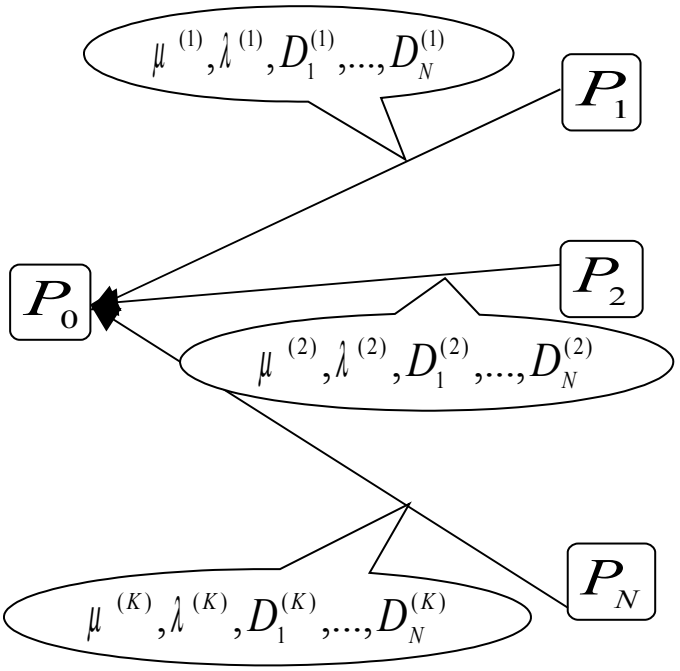

Рис.2. Получение нулевым процессом значений коэффициентов переноса при помощи функции MPI_GATHER

5. Parallel Computation Fluid Dynamics. Resent Developments and Advances Using Parallel Computers /J.Periaux et al. (ed). Elsevier Science B.V. Amsterdam, 1998. – 682 p.
6. Parallel Computation Fluid Dynamics. Developments and Applications of Parallel Technology /J.Periaux et al. (ed). Elsevier Science B.V. Amsterdam, 1999. – 534 p.

РЕАЛИЗАЦИЯ МОДЕЛЕЙ АТМОСФЕРЫ НА ПАРАЛЛЕЛЬНЫХ КОМПЬЮТЕРАХ

М.А. Толстых, В.Н. Глухов

Институт вычислительной математики РАН

Введение

Численный прогноз погоды и моделирование климата всегда были областями, где требуются высокопроизводительные компьютеры. Для их использования программная реализация модели должна учитывать особенности параллельной архитектуры этих компьютеров.

В этой статье представлены параллельные реализации двух глобальных трехмерных атмосферных моделей, развиваемых в Институте вычислительной математики Российской академии наук. Первая модель – это эйлерова климатическая модель, а вторая – полулагранжева, предназначена для использования в численном прогнозе погоды. Мы будем называть их модель климата и модель прогноза погоды соответственно. Обе модели используют одну и ту же систему уравнений – уравнения типа Навье-Стокса на вращающейся сфере с некоторыми дополнительными допущениями (например, гидростатическое приближение). Модели отличаются специфической формой уравнений и способом их дискретизации.

Описываемые подходы к распараллеливанию могут быть также применены для задач несжимаемой вычислительной газовой динамики.

Система уравнений для влажной атмосферы может быть записана как

$$\frac{\partial \varphi}{\partial t} + A\varphi + B\varphi = F, \qquad (1)$$

где $\varphi = (u, v, T, q, \ln \pi)^T$ – вектор состояния системы; u, v – компоненты горизонтального вектора скорости; T – температура; q – удельное влагосодержание воздуха; π – приземное давление; A – трехмерный оператор адвекции (переноса). Для эйлеровой модели в декартовой системе координат он имеет вид

$$A\varphi = u\frac{\partial \varphi}{\partial x} + v\frac{\partial \varphi}{\partial y} + w\frac{\partial \varphi}{\partial z}.$$

Оператор B представляет собой все остальные слагаемые системы уравнений без правых частей (градиент давления, сила Кориолиса). Такие операторы встречаются во многих задачах математической физики. Таким

образом, описываемые подходы к распараллеливанию моделей атмосферы могут быть применены и к другим проблемам.

В векторе правых частей $F = (F_u, F_v, F_T, F_q, 0)^T$ каждый компонент описывает источники и стоки соответствующей переменной вследствие процессов подсеточного масштаба (горизонтальная и вертикальная диффузия, коротковолновая и длинноволновая радиация, глубокая и мелкая конвекция, планетарный пограничный слой, торможение гравитационных волн). В обеих моделях слагаемое, описывающее внешнее воздействие на систему F, может быть вычислено независимо для каждой точки горизонтальной сетки, так как вычисление этого слагаемого в данной точке сетки зависит только от данной вертикальной колонны сетки. Здесь можно легко применить двумерное разбиение расчетной области в горизонтальной плоскости и достичь высокой степени параллелизма без необходимости обменов данными между процессорами. Вычисления правых частей ответственны за 53,3% времени CPU в эйлеровой модели и за 61% в полулагранжевой модели (при расчетах на одном процессоре). Главное затруднение в распараллеливании атмосферных моделей лежит в алгоритме интегрирования по времени дискретизированной системы уравнений (1). Это вызвано тем, что обе модели используют полунеявный алгоритм интегрирования по времени для быстрых гравитационных волн, чтобы увеличить шаг по времени. Это приводит к необходимости решения на каждом шаге по времени двумерных уравнений типа Гельмгольца на сфере. Параллельная реализация прямого алгоритма для решения этой проблемы рассматривается в разд. 2. Можно заметить, что решение уравнений эллиптического типа также часто встречается в задачах математической физики.

Для организации обменов данными между процессорами используются стандарт MPI (Massage Passing Interface – интерфейс передачи сообщений) и подход «одна программа – много данных» (SPMD), где головной процесс также организует ввод/вывод и некоторые дополнительные вычисления. В практических вычислениях применялись две параллельные вычислительные системы. Первая – это Linux-кластер с 18 двухпроцессорными узлами Intel Pentium III/500, соединенных через сеть SCI с пиковой пропускной способностью 88 Мбайт/с и временем задержки обменов 6 мкс поверх MPI. Этот кластер установлен в Научно-исследовательском вычислительном центре Московского государственного университета (в дальнейшем ПЭВМ-кластер). Вторая система – высокопроизводительная система МВС 1000М, основанная на процессорах Alpha21264A, 667 МГц, установленная в Межведомственном суперкомпьютерном центре в Москве. Она состоит из 384 двухпроцессорных узлов, соединенных через сеть Myrinet с пиковой

пропускной способностью 140 Мбайт/с и временем задержки 14 мкс поверх MPI. Пиковая производительность этой системы достигает 1 Тфлопс.

1. Модель климата

Эйлерова модель климата ИВМ РАН предназначена для изучения изменчивости и изменений климатической системы под влиянием различных искусственных факторов и природных явлений [1]. Так как модель необходимо интегрировать на длительные сроки (десятилетия и даже столетия) с различными начальными состояниями, численные эксперименты с моделью предъявляют высокие требования как к вычислительной системе, так и к параллельным алгоритмам модели.

В модели для дискретизации оператора A используются стандартные центральные разности второго порядке на смещенной сетке С. Эта дискретизация обозначена как A_h. Уравнение (1) без правых частей дискретизируется по времени следующим образом:

$$\frac{\varphi^{n+1} - \varphi^{n}}{2\Delta t} + A_h\varphi^{n} + \frac{1}{2}\left(L\varphi^{n+1} + L\varphi^{n}\right) + \left(N\varphi\right)^{n} = 0, \qquad (2)$$

где L – линейная часть оператора B, который ответственен за быстрые гравитационные волны, $L+N=B;$ верхний индекс означает номер шага по времени, Δt – шаг по времени.

Размер вычислительной области модели – 145x73 (2,5 и 2 градуса по долготе и широте соответственно) и 21 уровень по вертикали. Область разбивается по широте и долготе так, чтобы эти измерения были распределены между P_{lon} и P_{lat} процессорами соответственно. Решение системы уравнений (2) занимает 46,7 % времени счета однопроцессорного варианта, при этом примерно половина затрат связана с решением набора уравнений типа Гельмгольца, возникающих при решении неявной по времени части системы (2) .

Расчеты оператора адвекции A_h, нелинейного оператора N и последующие вычисления правой части R для уравнения Гельмгольца не имеют глобальных зависимостей в горизонтальной плоскости, за исключением нескольких особенностей возле полюсов. Вычисления в данной точке сетки (i, j, k) используют известные значения переменных на уровне времени n из области [i –2:i+2]x[j-2:j+2]x[1:K]. Здесь индексы i, j и k означают долготу, широту и вертикальный уровень соответственно; K – количество вертикальных уровней. Рис. 1 представляет шаблоны зависимостей поверхностного давления, температуры, компонент вектора

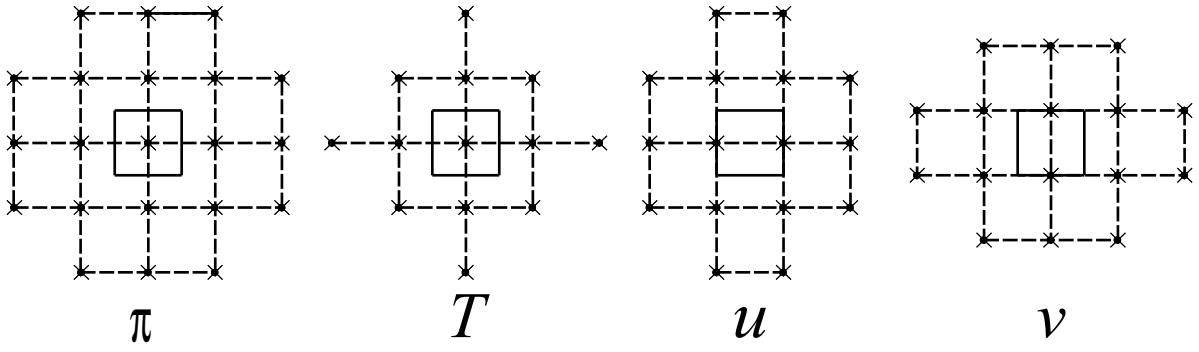

Рис. 1. Шаблоны зависимостей для вычисления правой части уравнения Гельмгольца

горизонтальной скорости *u* и *v*, от которых зависит значение правой части *R* в данной точке сетки.

Перед тем как эти вычисления начинаются, процессоры обменивают необходимые значения переменных на границах подобластей с соседними процессорами. Так же значения в первом и последнем узлах сетки по долготе пересылаются между первыми и последними процессорами разбиения по долготе, чтобы поддерживать периодические граничные условия по этой координате.

Как только явная часть динамики вычислена, решается неявная часть системы (2), сводящаяся к решению одного дискретного трехмерного уравнения типа Гельмгольца относительно одной переменной *D:*

$$D^{n+1} - (\Delta t)^2 G \nabla^2 D^{n+1} = R, \qquad (3)$$

где *D* – горизонтальная дивергенция на модельной σ-поверхности; *G* – матрица $K \times K$; ∇^2 – дискретный аналог горизонтального оператора Лапласа в сферических координатах; *R* объединяет все известные слагаемые системы (2) на уровнях времени *n* и *n-1*. Периодические граничные условия в (3) поддерживаются так же, как в системе (2). Диагонализирующее преобразование, примененное к матрице *G*, позволяет расщепить уравнение (3) в вертикальном направлении и решать его независимо для каждого вертикального уровня прямым методом, включающим быстрое преобразование Фурье по долготе и метод исключения Гаусса (прогонку) в меридиональном направлении. Алгоритм решения уравнения (3) имеет следующую структуру:

1. Преобразование правой части уравнения (3) в базис собственных векторов матрицы G.
2. Прямое быстрое преобразование Фурье.
3. Решение линейных трехдиагональных уравнений по широте.
4. Обратное быстрое преобразование Фурье.
5. Преобразование решения из базиса собственных векторов матрицы *G*.

Шаги 1 и 5 вычисляют с помощью матрично-векторного произведения по вертикальной координате, и они не требуют дополнительных обменов данными между процессорами. Шаги 2 и 4 требуют данных от всех точек по долготе для данной широты и вертикального уровня. Шаг 3, напротив, нуждается во всех значениях в точках сетки по широте для данных коэффициента Фурье по долготе и вертикального уровня.

Чтобы выполнить шаг 2, мы переставляем данные в плоскости «долгота-вертикальная координата», собирая данные по всем узлам по долготе и распределяя данные со всех вертикальных уровней между процессорами (T_φ , рис. 2). Эта операция называется транспонированием данных. По завершении быстрых преобразований Фурье каждый процессор содержит данные по всем долготным коэффициентам Фурье для части уровней и широт. Чтобы приступить к шагу 3, мы снова используем транспонирование, но в плоскости «широта-долгота». Теперь собираются

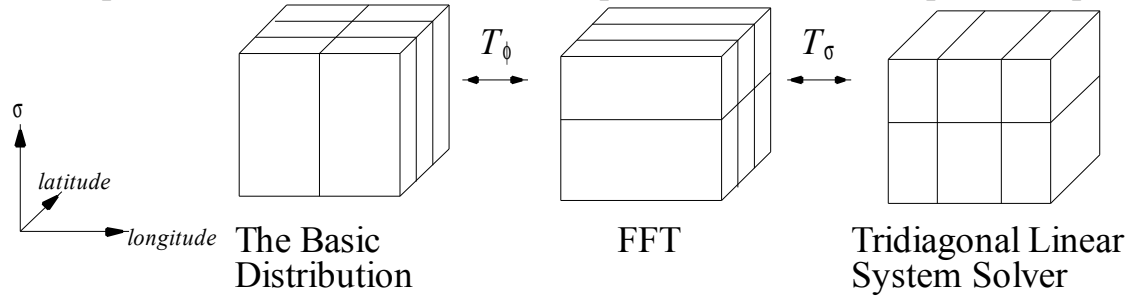

Рис. 2. Распределения данных, используемые при решении уравнения Гельмгольца (3) (пример для $P_{lon}=2$, $P_{lat}=3$)

данные по всем широтам, но распределяются они по долготным коэффициентам Фурье (T_σ, рис. 2). Получив решения трехдиагональных систем уравнений, мы повторно переупорядочиваем данные в исходное распределение, выполняя транспонирования в обратном порядке и вычисляя между ними обратное преобразование Фурье.

Тестирование параллельной реализации модели было выполнено как на вычислительной системе МВС 1000М, так и на ПЭВМ-кластере НИВЦ МГУ. Для данного числа процессоров P мы измерили время интегрирования модели на один шаг на всех сетках из (P_{lon}, P_{lat}) процессоров, таких, что $P_{lon}P_{lat}=P$. Результаты приведены в таблице. Параллельное ускорение определяется как отношение продолжительности вычисления на одном процессоре к минимальному времени, получаемому на P процессорах. Например, на 8 процессорах МВС 1000М мы тестировали сетки процессоров (1,8), (2,4), (4,2) и (8,1) и выбрали конфигурацию (2,4) как наилучшую. Время расчета одного шага по времени на одном процессоре – 11,4 с на МВС и 36,0 с на ПЭВМ-кластере. Чтобы изучить масштабируемость модели, мы также провели вычисления с горизонтальным разрешением 5 градусов по долготе и 4 градуса по широте. Полученное параллельное ускорение изображено на рис. 3 слева.

Время CPU динамического блока модели климата, на вычислительной системе МВС 1000М, с

P_{lon} / INM AGCM dynamics speedup	P_{lat} 1	2	4	8	16
1	11,39	5,97	3,56	2,62	2,08
2	6,62	4,56	2,60	1,74	1,78
4	4,90	3,18	1,70	1,49	0,93
8	4,01	2,64	1,30	0,97	0,96
16	4,24	2,56	1,24	1,58	1,05

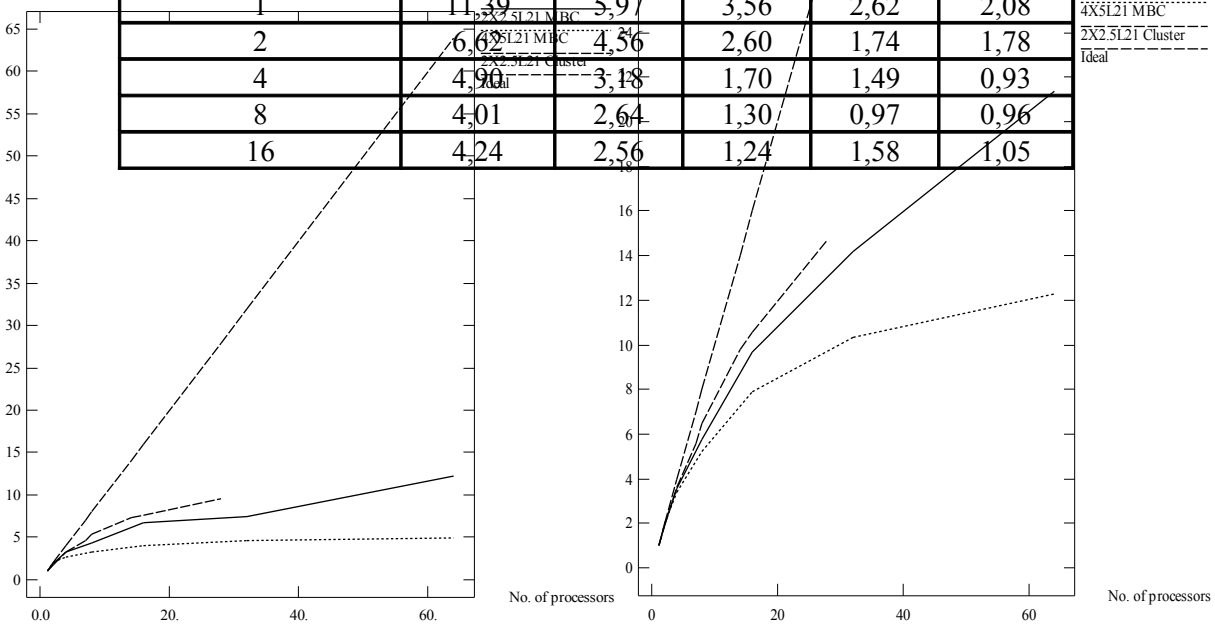

Рис. 3. Параллельное ускорение динамического блока модели климата (слева), экстраполированное ускорение для всей модели (справа)

Справа на рис. 3 показано экстраполированное параллельное ускорение для всей модели, включая учет параметризации процессов подсеточного масштаба, т.е. с расчетом правых частей системы (1). Параметризации не требуют дополнительных обменов данными между процессорами.

2. Модель прогноза погоды

Численный гидродинамический прогноз погоды с пространственным разрешением, необходимым для соответствующего описания конвективных и мезомасштабных процессов, является проблемой, которая требует огромных компьютерных ресурсов. Полулагранжев метод (обратный метод характеристик) для описания адвекции (переноса) позволяет использовать шаг по времени в несколько раз больше, чем определяемый условием Куранта. В этом случае оператор адвекции исчезает из дискретизированного уравнения (1), поскольку полная производная функции дискретизируется как производная вдоль траектории (см. [3]). Рассматриваемая модель подробно изложена в [4]. Отличительной особенностью этой модели является применение компактных конечных разностей четвертого порядка на несмещенной сетке для аппроксимации неадвективных слагаемых в сочетании с использованием вертикального компонента абсолютного вихря и горизонтальной дивергенции в качестве прогностических переменных модели.

Система (1) без правых частей дискретизируется с использованием двуслойной схемы по времени с экстраполяцией ветра и нелинейных слагаемых уравнений на уровень по времени n+1/2 как

$$\frac{\varphi^{n+1} - \varphi^{n^*}}{\Delta t} + \frac{1}{2}\left(L\varphi^{n+1} + L\varphi^{n^*}\right) + \frac{1}{2}\left(\left(N\varphi\right)^{n+1/2} + \left(N\varphi\right)^{n+1/2^*}\right) = 0, \qquad (4)$$

где L – линейная часть оператора B из уравнения (1), ответственная за быстрые гравитационные волны, $L+N=B$; $(\,)^{n+1}$ – значение элементов соответствующего уравнения в конечной точке траектории на новом шаге времени; $(\,)^{n^*}$ – значение в соответствующей исходной точке (которое определяется трехмерной траекторией). Конечные точки траекторий всегда являются точками сетки, в то время как исходные точки чаще всего не совпадают с узлами сетки. Значения в исходных точках получают кубической интерполяцией, используя близлежащие точки сетки.

Неявная часть системы (4) решается аналогично модели климата, описанной в предыдущем разделе. Для этого соответствующая часть системы (4) сводится к дискретному трехмерному уравнению типа

Гельмгольца относительно одной переменной *D*. Это уравнение решается прямым методом с использованием быстрого преобразования Фурье по долготе и решения блочно-трехдиагональных линейных систем с блоками 2x2 по широте (обратите внимание, что в модели климата используется обычная прогонка). Необходимость использования обращения блочно-трехдиагональных матриц вызвана применением компактных разностей высокого порядка. Также правая часть для уравнения Гельмгольца

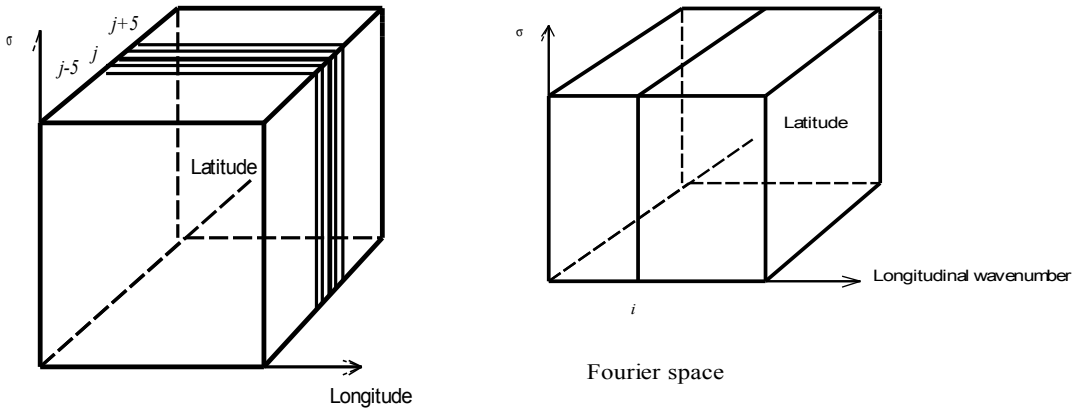

Рис. 4. Разбиение вычислительной области при вычислениях в сеточном пространстве (слева) и пространстве коэффициентов Фурье по долготе (справа)

вычисляется в пространстве коэффициентов Фурье по долготе.

Рассмотрим вычислительную структуру модели. Сначала вычисляют правые части уравнения (1), используя значения переменных состояния на уровне времени *n*. Эти правые части содержат тенденции прогностических переменных вследствие процессов подсеточного масштаба (солнечное излучение и т.д.). Здесь мы имеем высокую степень параллелизма, так как эти вычисления для данной точки сетки не требуют значений переменных в смежных точках сетки в долготном и широтном направлениях, требуются только значения в вертикальных столбцах. Следует заметить, что эти вычисления занимают 61 % всего времени расчета. Затем вычисляются операторы *L* на шаге по времени *n* и *N* на уровне *n+1/2* из (4). Эти вычисления для данной точки сетки (*i, j, k*) требуют значения переменных из области [*i-4:i+4*]×[*j-4:j+4*]×[*1:K*]. Увеличение горизонтального шаблона по отношению к модели климата, описываемой в предыдущем разделе, вызвано использованием компактных конечных разностей четвертого порядка для вычисления операторов вихря и дивергенции. Они реализованы с помощью приближенного обращения соответствующих

трехдиагональных операторов. Как и в модели климата, процессоры обмениваются необходимыми значениями переменных на границах подобластей с соседями перед тем, как эти вычисления начинаются. Там, где возможно, используются асинхронные обмены, что позволяет одновременно выполнять вычисления и обмены данными. Следующий этап вычислений – поиск исходных точек траекторий и интерполяция вычисленных слагаемых уравнений в эти точки. Эта часть ответственна за 30 % машинного времени. Шаблон интерполяции составляет 4 точки по каждой координате. Исходная точка траектории может лежать за несколько интервалов сетки от конечной точки (которая всегда является точкой сетки в полулагранжевом методе). Максимальная ширина области зависимости вычислений для данной точки сетки зависит от размера шаблона, значения шага по времени и максимальной скорости ветра в атмосфере (около 110 м/с). Типичная половинная ширина области зависимости в нашей модели для горизонтального разрешения 1,5 градуса составляет 5 точек в каждом горизонтальном направлении. Таким образом, количество данных, которое передается между процессорами, существенно выше, чем для модели климата.

Для вычислительной системы со значительным временем задержки представляется выгодным ограничиться одномерным разбиением вычислительной области так, чтобы каждый процессор выполнял вычисления в некоторой полосе широт (рис. 4). Очевидно, что двумерное разбиение является более общим и позволяет использовать большее количество процессоров. Однако это также приводит к большему количеству обменов данными между процессорами, когда количество процессоров относительно мало. Кроме того, двумерное разбиение требует двух прямых и обратных транспонирований данных для решения уравнения Гельмгольца, как было описано в разд. 1, в то время как одномерное разбиение требует только одной пары транспонирований.

Нам необходимо переместить данные из долготных рядов, которые размещаются в других процессорах. Необходимые ряды переносятся после того, как завершаются все вычисления величин на уровнях времени n и n+1/2.

После того как интерполяции и прямые быстрые преобразования Фурье по долготе заканчиваются, нам необходимо перераспределить данные для вычислений в пространстве коэффициентов Фурье по долготе. Вычисления для каждого коэффициента требуют значения всех широт (см. рис. 4), так что нам нужно обеспечить каждый процессор соответствующими данными, используя их транспонирование.

Вычисления в пространстве Фурье включают вычисление правой части и решение уравнения типа Гельмгольца (3), неявной горизонтальной диффузии четвертого порядка и вычисления горизонтальных компонент вектора скорости из дивергенции и завихренности. Эти вычисления выполняются параллельно для каждого коэффициента Фурье. После вычислений в пространстве Фурье нам нужно снова перераспределить данные между процессорами, так как для выполнения обратного преобразования Фурье по долготе нужны все коэффициенты.

С горизонтальным разрешением 1,5 градусов по широте модель теоретически может использовать до 120 процессоров. Однако это будет требовать большого количества обменов данными между процессорами, поскольку необходимые данные будут размещены не только в соседних процессорах, но и в следующих за ними. Практически для упомянутого разрешения можно использовать до 24 процессора.

Тестовые расчеты с количеством процессоров от 1 до 16 проводились на ПЭВМ-кластере и системе МВС 1000М с тремя разрешениями модели: 2 градуса по долготе и широте с 20 вертикальными уровнями, 1,5 градуса с 20

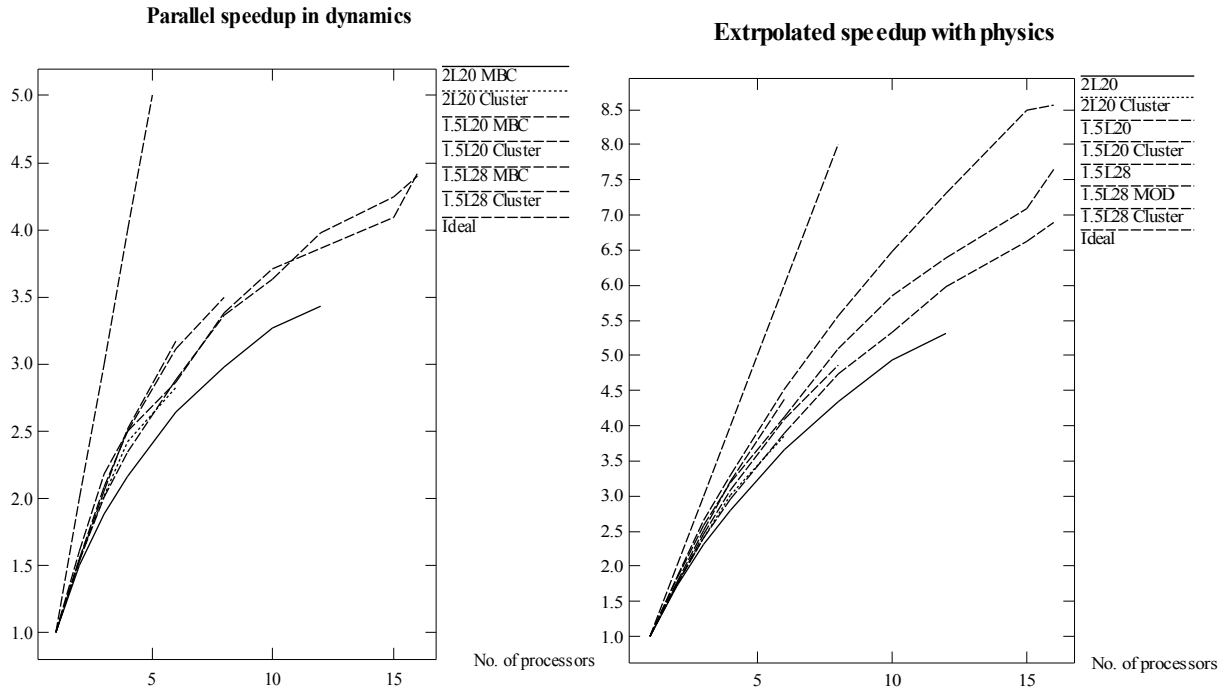

Рис. 5. Параллельное ускорение модели прогноза погоды: ускорение динамики (слева), экстраполированное ускорение всей модели с расчетом правых частей уравнений для различных разрешений модели (справа)

уровнями и 1,5 градуса с 28 уровнями. Параллельное ускорение динамики модели прогноза погоды на ПЭВМ-кластере и системе МВС 1000М представлено на рис. 5 слева. Правая панель рис. 5 показывает теоретическое параллельное ускорение, экстраполированное для всей модели с расчетом правых частей системы (1), т.е. с параметризациями процессов подсеточного масштаба. Кривая 1.5L28 MOD на правой части рис.5 показывает наилучшие результаты (параллельное ускорение 8,5 на 15 процессорах) и получена при оптимизации обменов на МВС 1000М.

Из этого рисунка можно сделать два вывода. Во-первых, эффективность распараллеливания возрастает с увеличением разрешения модели (т.е. размера вычислительной области) на обеих вычислительных системах. Это объясняется уменьшением влияния времени задержки обмена. Во-вторых, параллельная эффективность ПЭВМ-кластера несколько выше, чем системы МВС 1000М. Это вызвано двумя причинами. Первой причиной является большее значение времени задержки обмена на МВС 1000М (14 мкс по сравнению с 6 на ПЭВМ-кластере). Вторая причина состоит в том, что отношение времени получения одного результата процессором к пропускной способности сети на МВС 1000М в 1,6 раза больше, чем на ПЭВМ-кластере. Это означает, что процессор может произвести больше результатов на протяжении времени, требуемого для того, чтобы перенести один результат вычислений между процессорами и ждать завершения обменов. Частично эта проблема решена благодаря использованию асинхронных обменов между процессорами, но на этапе транспонирования данных возможность применения таких обменов ограничена.

Выводы

Модели климата и прогноза погоды ИВМ РАН были реализованы на параллельных компьютерах с распределенной памятью. Результаты указывают на преимущество использования параллельных вычислений в этих задачах. Примененные подходы к распараллеливанию моделей можно использовать и для других задач математической физики.

Полученные параллельные ускорения моделей, будучи еще далеки от идеала, являются разумными и допускают усовершенствование. Улучшение разрешения моделей повышает эффективность распараллеливания. Ведутся работы по оптимизации обменов и вычислительных алгоритмов, включающие адаптацию алгоритма транспонирования данных к конкретной вычислительной системе с использованием асинхронных обменов. Также может быть полезным сравнение полунеявных и явных схем интегрирования по времени.

Для модели прогноза погоды настоящая параллельная реализация не позволяет использовать большое количество процессоров. Ведутся работы по изучению различных возможностей реализации двумерного разбиения расчетной области без ухудшения эффективности распараллеливания при небольшом числе процессоров.

Наконец, можно обратить внимание, что современные атмосферные модели применяют все более совершенные и вычислительно-емкие алгоритмы для расчета компонентов вектора правых частей в уравнении (1). Это означает, что эффективность распараллеливания моделей в дальнейшем будет расти без дополнительных усилий, так как вычисления правых частей не требуют дополнительных обменов между процессорами.

Список литературы

1. *Алексеев В.В., Володин Е.М., Галин В.Я., Дымников В.П., Лыкосов В.Н.* Моделирование современного климата с использованием атмосферной модели ИВМ РАН. Описание модели А5421 (версия 1997 г.) и результаты экспериментов по программе AMIP II. М.: Институт вычислительной математики РАН, 1998. – 200 с. Деп. в ВИНИТИ 03.07.98 N 2086–В98.
2. *Kumar V., Grama A., Gupta A., Karypis G.* Introduction to Parallel Computing: Design and Analysis of Algorithms. Benjamin/Cummings Publ., California (USA), 1994.
3. *Staniforth A., Cote J.* Semi-Lagrangian integration schemes for atmospheric models. A review //Mon. Weather Rev. – 1991. – N119. – P. 2206-2223.
4. *Толстых М.А.* Полулагранжева модель атмосферы с высоким разрешением для численного прогноза погоды //Метеорология и гидрология. – 2001. – № 4. – С. 5-16.

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ MPI ДЛЯ РАС-ПАРАЛЛЕЛИВАНИЯ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ НА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С НЕВЫСОКОЙ СКОРОСТЬЮ ОБМЕНОВ МЕЖДУ ПРОЦЕССОРАМИ

М.А. Толстых

Институт вычислительной математики РАН

Создание программных комплексов задач математической физики на параллельных вычислительных системах с распределенной памятью и ограниченной пропускной способностью связующей сети имеет ряд особенностей, которые рассматриваются в данной статье. Предполагается, что читатель знаком в общих чертах со стандартом передачи данных MPI [1, 2, 3]. Предполагается также, что алгоритм задачи допускает распараллеливание.

Рассмотрим аппаратные особенности типичных вычислительных систем данного класса. Как правило, число процессоров не превышает 40, время задержки обмена (т.е. время между поступлением команды пользователя на пересылку данных и реальным началом этой пересылки) 6-15 мкс, реальная (ощущаемая пользователем) пропускная способность сети, связывающей процессоры, 10-200 Мбайт/с. Данные о некоторых существующих сетях приведены в табл. 1. Более подробную и регулярно обновляемую информацию можно получить на сайте http://parallel.ru. Следует учитывать, что приведенные данные показывают максимальные значения.

Современные процессоры успевают выполнить порядка 10 операций с плавающей точкой за время, необходимое для пересылки одного результата вычислений (на параллельных системах, где узлы соединены сетью Fast Ethernet, – порядка 100 операций). Это правило верно и для очень дорогих вычислительных систем – например, на системе Fujitsu VPP5000 пиковая производительность процессора составляет 9,6 Гфлопс (на реальных задачах, как правило, в два раза меньше), а пиковая пропускная способность сети – 2 по 1,6 Гбайт/с (при одновременном двустороннем обмене). Этот фактор нужно обязательно учитывать при создании параллельных программных комплексов.

Таблица 1

Характеристики типичных сетей, используемых в параллельных вычислительных системах

Тип сети	Время задержки, мкс	Пиковая пропускная способность поверх MPI, Мбайт/с	Где установлена
Fast Ethernet	85 (LAM MPI)	10,5	Кластер ТГУ
Gigabit Ethernet		60	
SCI	7	88	Кластер НИВЦ МГУ
Myrinet	18	140	МВС 1000М, МСЦ
Myrinet 2000	9	210	МВС 1000М, МСЦ с 2002 г.

В классическом учебнике по использованию интерфейса передачи сообщений [1] для распараллеливания многомерных задач математической физики рекомендуется применять двумерное разбиение расчетной области. Однако преимущество двумерного разбиения сильно зависит от задачи, ее размерности и типа используемых обменов. Проиллюстрируем сказанное на примере. Рассмотрим модифицированный тест из упомянутого учебника. Его исходные тексты находятся в файле

ftp://ftp.mcs.anl.gov/pub/mpi/using/UsingMPI.tar.gz.

В этом тесте решается уравнение Пуассона в квадрате итерационным методом Якоби. Аналогичный подход к распараллеливанию применяется и для многих других задач математической физики (например, решения нестационарного уравнения или системы уравнений в частных производных с использованием явной схемы по времени).

Число арифметических операций, приходящееся на один узел расчетной области в исходном тесте, составляет всего 6, поэтому для приближения к реальным задачам тест был усложнен путем введения дополнительных операций. Математически модифицированный тест эквивалентен исходной задаче, так как правая часть уравнения задается равной нулю, но число операций на один узел сетки составляет теперь 23, кроме того, вычисляются 6 раз значения тригонометрических функций. Тексты подпрограмм этого теста при одномерном и двумерном разбиениях

расчетной области приведены на рис. 1 и 2 соответственно. Для обменов используются блокирующие процедуры двустороннего обмена MPI (*MPI_SENDRECV*). Это означает, что выход из подпрограммы обмена осуществляется только после ее завершения.

```
c**************************************************************
c   oned.f – a solution to the Poisson problem using Jacobi
c   iteration on a 1-d decomposition
c
c  The size of the domain is set by processor 0 and broadcast
c  to all other processors.  The Jacobi iteration is run until
c  the change in successive elements is small or a maximum
c  number of iterations is reached.
c**************************************************************
      program main
!
      include "mpif.h"
      integer maxn
      parameter (maxn = 202)
      double precision a(maxn,maxn),b(maxn,maxn),
     $                 f(maxn,maxn)
      integer nx, ny
      integer myid, numprocs, ierr
      integer comm1d, nbrbottom, nbrtop, s, e, it
      double precision diff, diffnorm, dwork
      double precision t1, t2
      external diff

      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
c
      if (myid .eq. 0) then
c
c         The size of the problem
c
          nx = 200
      endif
   call MPI_BCAST(nx,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
```

```
      ny   = nx
c
c Get a new communicator for a decomposition of the domain
c
      call MPI_CART_CREATE( MPI_COMM_WORLD, 1, numprocs,
     $           .false.,.true., comm1d, ierr )
c
c Get my position in this communicator, and my neighbors
c
      call MPI_COMM_RANK( comm1d, myid, ierr )
      call MPI_Cart_shift(comm1d,0,1, nbrbottom, nbrtop, ierr)
c
c Compute the actual decomposition
c
      call MPE_DECOMP1D( ny, numprocs, myid, s, e )
!
! Initialize the right-hand-side (f) and the initial solution
! guess (a)
      call onedinit( a, b, f, nx, s, e )
!
! Actually do the computation.  Note the use of a collective
! operation to check for convergence, and a do-loop to bound
! the number of iterations.
!
      call MPI_BARRIER( MPI_COMM_WORLD, ierr )
      t1 = MPI_WTIME()
      do 10 it=1, 100
        call exchng1(a, nx, s, e, comm1d, nbrbottom, nbrtop )
          call sweep1d( a, f, nx, s, e, b )
          call exchng1( b, nx, s, e, comm1d, nbrbottom, nbrtop )
          call sweep1d( b, f, nx, s, e, a )
          dwork = diff( a, b, nx, s, e )
          call MPI_Allreduce( dwork, diffnorm, 1,
     $           MPI_DOUBLE_PRECISION,MPI_SUM, comm1d, ierr )
          if (diffnorm .lt. 1.0e-5) goto 20
!           if (myid .eq. 0) print *, 2*it, ' Difference is ',
!     &     diffnorm
10     continue
      if (myid .eq. 0) print *, 'Failed to converge'
```

```
20    continue
      t2 = MPI_WTIME()
      if (myid .eq. 0) then
         print *,'Converged after ',2*it,' Iterations in '
     $      , t2 - t1,   ' secs '
      endif
!
      call MPI_FINALIZE(ierr)
      end
!
! Perform a Jacobi sweep for a 1-d decomposition.
! Sweep from a into b
! Modified by M.Tolstykh – increase operation count
!
      subroutine sweep1d( a, f, nx, s, e, b )
      integer nx, s, e
      double precision a(0:nx+1,s-1:e+1), f(0:nx+1,s-1:e+1),
     +                 b(0:nx+1,s-1:e+1)
!
      integer i, j
      double precision h,c,x,y,pi2
!
      h = 1.0d0 / dble(nx+1)
      pi2 = asin(1.d0)
      do j = s, e
        do i = 1, nx
         x = dble(i-1)*h*pi2
         y = dble(j-1)*h*pi2
         c = cos(x)**3*cos(y)**2
         b(i,j) = 0.25*c*(a(i-1,j)+a(i,j+1)+a(i,j-1)+a(i+1,j))
     +  - h *h* f(i,j)*cos(x**3)*sin(x**2)*cos(y**3)*sin(y**2)
        enddo
      enddo
      return
      end
!
       subroutine exchng1(a, nx, s,e,comm1d, nbrbottom, nbrtop)
!
!   exchanges for 1d decomposition
```

```
!
      include 'mpif.h'
      integer nx, s, e
      double precision a(0:nx+1,s-1:e+1)
      integer comm1d, nbrbottom, nbrtop
      integer status(MPI_STATUS_SIZE), ierr
!
      call MPI_SENDRECV(
     &         a(1,e), nx, MPI_DOUBLE_PRECISION, nbrtop, 0,
     &     a(1,s-1), nx, MPI_DOUBLE_PRECISION, nbrbottom, 0,
     &           comm1d, status, ierr )
      call MPI_SENDRECV(
     &        a(1,s), nx, MPI_DOUBLE_PRECISION, nbrbottom, 1,
     &        a(1,e+1), nx, MPI_DOUBLE_PRECISION, nbrtop, 1,
     &           comm1d, status, ierr )
      return
      end
c
c  The rest of the 1-d program
c
      double precision function diff( a, b, nx, s, e )
      integer nx, s, e
      double precision a(0:nx+1, s-1:e+1), b(0:nx+1, s-1:e+1)
c
      double precision sum
      integer i, j
c
      sum = 0.0d0
      do 10 j=s,e
        do 10 i=1,nx
          sum = sum + (a(i,j) - b(i,j)) ** 2
 10     continue
c
      diff = sum
      return
      end
      subroutine onedinit( a, b, f, nx, s, e )
      integer nx, s, e
      double precision a(0:nx+1, s-1:e+1), b(0:nx+1, s-1:e+1),
```

```
     &                f(0:nx+1, s-1:e+1)
!
      integer i, j
!
      do 10 j=s-1,e+1
         do 10 i=0,nx+1
            a(i,j) = 0.0d0
            b(i,j) = 0.0d0
            f(i,j) = 0.0d0
 10       continue
c
c     Handle boundary conditions
c
      do 20 j=s,e
         a(0,j) = 1.0d0
         b(0,j) = 1.0d0
         a(nx+1,j) = 0.0d0
         b(nx+1,j) = 0.0d0
 20    continue
      if (s .eq. 1) then
         do 30 i=1,nx
            a(i,0) = 1.0d0
            b(i,0) = 1.0d0
 30       continue
      endif
      return
      end
```

Рис. 1. Распечатка программы решения уравнения Пуассона с одномерным разбиением области

```
c**********************************************************
c   twod.f – a solution to the Poisson problem by using Jacobi
c   iteration on a 2-d decomposition
c
c   Node 0 prints the result.
c
c**********************************************************
```

```
      program main
      include "mpif.h"
      integer maxn
      parameter (maxn = 202)
      double precision a(maxn,maxn),b(maxn,maxn),f(maxn,maxn)
      integer nx, ny
      integer myid, numprocs, it, rc, comm2d, ierr, stride
      integer nbrleft, nbrright, nbrtop, nbrbottom
      integer sx, ex, sy, ey
      integer dims(2)
      logical periods(2)
      double precision diff2d, diffnorm, dwork
      double precision t1, t2
      external diff2d
      data periods/2*.false./

      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
      if (myid .eq. 0) then
c
c        The size of the problem
c
        nx = 200
      endif
      call MPI_BCAST(nx,1,MPI_INTEGER,0,MPI_COMM_WORLD,
     $                    ierr)
      ny   = nx
c
c Get a new communicator for a decomposition of the domain.
C Let MPI find a "good" decomposition
c
      dims(1) = 0
      dims(2) = 0
      call MPI_DIMS_CREATE( numprocs, 2, dims, ierr )
      call MPI_CART_CREATE( MPI_COMM_WORLD, 2, dims,
     $                              periods, .true.,  comm2d, ierr )
c
c Get my position in this communicator
```

```
c
      call MPI_COMM_RANK( comm2d, myid, ierr )
c
c My neighbors are now +/- 1 with my rank. Handle the case
c of the boundaries by using MPI_PROCNULL.
      call fnd2dnbrs(comm2d,nbrleft,nbrright,nbrtop,nbrbottom)
c
c Compute the decomposition
c
      call fnd2ddecomp( comm2d, nx, sx, ex, sy, ey )
c
c Create a new, "strided" datatype for the exchange
c  in the "non-contiguous" direction
c
      call mpi_Type_vector( ey-sy+1, 1, ex-sx+3,
     $                 MPI_DOUBLE_PRECISION, stride, ierr)
      call mpi_Type_commit( stride, ierr )
c
c Initialize the right-hand-side (f) and the initial
c solutionguess (a)
c
      call twodinit( a, b, f, nx, sx, ex, sy, ey )
c
c Actually do the computation. Note the use of a collective
c operation to check for convergence, and a do-loop to bound
c   the number of iterations.
c
      call MPI_BARRIER( MPI_COMM_WORLD, ierr )
      t1 = MPI_WTIME()
      do 10 it=1, 100
       call exchng2( b, sx, ex, sy, ey, comm2d, stride,
     $               nbrleft, nbrright, nbrtop, nbrbottom)
       call sweep2d( b, f, nx, sx, ex, sy, ey, a )
       call exchng2( a, sx, ex, sy, ey, comm2d, stride,
     $               nbrleft, nbrright, nbrtop, nbrbottom)
       call sweep2d( a, f, nx, sx, ex, sy, ey, b )
       dwork = diff2d( a, b, nx, sx, ex, sy, ey )
       call MPI_Allreduce( dwork, diffnorm, 1,
     $    MPI_DOUBLE_PRECISION, MPI_SUM, comm2d, ierr)
```

```
      if (diffnorm .lt. 1.0e-5) goto 20
!     if (myid.eq.0) print *,2*it,' Difference is ',diffnorm
10    continue
      if (myid .eq. 0) print *, 'Failed to converge'
20    continue
      t2 = MPI_WTIME()
      if (myid .eq. 0) then
        print *, 'Converged after ', 2*it, ' Iterations in ',
     $         t2 - t1,    ' secs '
      endif
c
      call MPI_Type_free( stride, ierr )
      call MPI_Comm_free( comm2d, ierr )
      call MPI_FINALIZE(rc)
      end
!  Perform a Jacobi sweep for a 2-d decomposition
!  Modified by M.Tolstykh – increase operation count
!
      subroutine sweep2d( a, f, n, sx, ex, sy, ey, b )
      integer n, sx, ex, sy, ey
      double precision a(sx-1:ex+1, sy-1:ey+1),
     +     f(sx-1:ex+1, sy-1:ey+1),b(sx-1:ex+1, sy-1:ey+1)
!
      integer i, j
      double precision h, c, x, y, pi2
!
      h = 1.0d0 / dble(n+1)
      pi2 = asin(1.d0)
      do j= sy, ey
        do i= sx, ex
          x = dble(i-1)*h*pi2
          y = dble(j-1)*h*pi2
          c = cos(x)**3*cos(y)**2
          b(i,j) =0.25*c*(a(i-1,j)+a(i,j+1)+a(i,j-1)+a(i+1,j))
     +   - h*h*f(i,j)*cos(x**3)*sin(x**2)*cos(y**3)*sin(y**2)
        enddo
      enddo
      return
      end
```

```
      subroutine exchng2( a, sx, ex, sy, ey,
     &                    comm2d, stridetype,
     &                    nbrleft, nbrright, nbrtop, nbrbottom )
       include 'mpif.h'
       integer sx, ex, sy, ey, stridetype
       double precision a(sx-1:ex+1, sy-1:ey+1)
       integer nbrleft, nbrright, nbrtop, nbrbottom, comm2d
       integer status(MPI_STATUS_SIZE), ierr, nx
c
       nx = ex - sx + 1
c  These are just like the 1-d versions, except for less data
       call MPI_SENDRECV( a(sx,ey), nx, MPI_DOUBLE_PRECISION,
     &                    nbrtop, 0,
     &                    a(sx,sy-1), nx, MPI_DOUBLE_PRECISION,
     &                    nbrbottom, 0, comm2d, status, ierr )
       call MPI_SENDRECV( a(sx,sy), nx, MPI_DOUBLE_PRECISION,
     &                    nbrbottom, 1,
     &                    a(sx,ey+1), nx, MPI_DOUBLE_PRECISION,
     &                    nbrtop, 1, comm2d, status, ierr )
c
c This uses the vector datatype stridetype
         call MPI_SENDRECV(a(ex,sy), 1, stridetype, nbrright, 0,
     &                    a(sx-1,sy), 1, stridetype, nbrleft, 0,
     &                    comm2d, status, ierr )
         call MPI_SENDRECV( a(sx,sy),  1, stridetype, nbrleft,
     &                    1, a(ex+1,sy), 1, stridetype, nbrright, 1,
     &                    comm2d, status, ierr )
         return
       end
c
c  The rest of the 2-d program
c
       double precision function diff2d(a,b, nx,sx, ex, sy, ey)
       integer nx, sx, ex, sy, ey
       double precision a(sx-1:ex+1, sy-1:ey+1),
     &                  b(sx-1:ex+1, sy-1:ey+1)
c
       double precision sum
       integer i, j
```

```
c
      sum = 0.0d0
      do 10 j=sy,ey
        do 10 i=sx,ex
          sum = sum + (a(i,j) - b(i,j)) ** 2
 10     continue
c
      diff2d = sum
      return
      end
      subroutine twodinit( a, b, f, nx, sx, ex, sy, ey )
      integer nx, sx, ex, sy, ey, i, j
      double precision a(sx-1:ex+1, sy-1:ey+1),
     &       b(sx-1:ex+1, sy-1:ey+1),f(sx-1:ex+1, sy-1:ey+1)
c
      do 10 j=sy-1,ey+1
        do 10 i=sx-1,ex+1
          a(i,j) = 0.0d0
          b(i,j) = 0.0d0
          f(i,j) = 0.0d0
 10     continue
c
c     Handle boundary conditions
c
      if (sx .eq. 1) then
        do 20 j=sy,ey
          a(0,j) = 1.0d0
          b(0,j) = 1.0d0
 20     continue
      endif
      if (ex .eq. nx) then
        do 21 j=sy,ey
          a(nx+1,j) = 0.0d0
          b(nx+1,j) = 0.0d0
 21     continue
      endif
      if (sy .eq. 1) then
        do 30 i=sx,ex
          a(i,0) = 1.0d0
```

```
          b(i,0) = 1.0d0
 30       continue
      endif
      return
      end
c
c This routine show how to determine the neighbors in a 2-d
c  decomposition of the domain. This assumes that
c   MPI_Cart_create has already been called
c
      subroutine fnd2dnbrs( comm2d,
     $                 nbrleft, nbrright, nbrtop, nbrbottom )
      integer comm2d, nbrleft, nbrright, nbrtop, nbrbottom
c
      integer ierr
c
      call MPI_Cart_shift(comm2d, 0,1,nbrleft,nbrright,ierr)
      call MPI_Cart_shift(comm2d, 1,1,nbrbottom,nbrtop,ierr)
c
      return
      end
c
      subroutine fnd2ddecomp( comm2d, n, sx, ex, sy, ey )
      integer comm2d
      integer n, sx, ex, sy, ey
      integer dims(2), coords(2), ierr
      logical periods(2)
c
      call MPI_Cart_get(comm2d,2, dims, periods, coords,ierr)
      call MPE_DECOMP1D( n, dims(1), coords(1), sx, ex )
      call MPE_DECOMP1D( n, dims(2), coords(2), sy, ey )
c
      return
      end
```

Рис. 2. Распечатка программы решения уравнения Пуассона с двумерным разбиением области

Оба теста используют подпрограмму разбиения расчетной области по данной координате из библиотеки расширений MPI *MPE_DECOMP1D* (рис. 3).

```
C ******************************************************************
c  This routine is for producing a decomposition of
c a 1-d array when given a number of processors. It
c may be used in "direct" product  decomposition.
c The values returned assume a "global" domain in [1:n]
c
******************************************************************
      subroutine MPE_DECOMP1D(n,numprocs, myid, s, e)
      integer n, numprocs, myid, s, e
      integer nlocal
      integer deficit
c
      nlocal  = n / numprocs
      s       = myid * nlocal + 1
      deficit = mod(n,numprocs)
      s       = s + min(myid,deficit)
      if (myid .lt. deficit) then
         nlocal = nlocal + 1
      endif
      e = s + nlocal - 1
      if (e .gt. n .or. myid .eq. numprocs-1) e = n
      return
      end
```

Рис. 3. Распечатка программы разбиения расчетной области

Результаты запуска этого теста на кластере НИВЦ МГУ при количестве процессоров от 1 до 8 приведены в табл. 2 (вторая и третья прографка). Параллельное ускорение определяется как отношение продолжительности вычислений на одном процессоре ко времени вычислений на данном числе процессоров.

Таблица 2

Параллельное ускорение для различных версий программы

Кол-во процессоров	Одномерное разбиение	Двумерное разбиение	Одномерное разбиение, асинхронные обмены
2	1,99	1,84	1,92
3	2,67	2,55	2,74
4	3,32	3,2	3,42
6	4,48	4,15	4,84
8	5,18	4,79	5,62

Мы видим, что при данной размерности задачи (200) и небольшом количестве процессоров параллельное ускорение больше для одномерного разбиения расчетной области даже при числе процессоров 2 и 3, когда эти разбиения совпадают. Это вызвано большими накладными расходами на организацию обменов. Накладные расходы включают в себя как аппаратную часть (время задержки начала обмена), так и программную (время работы подпрограмм MPI). Сравним одномерное и двумерное разбиения при числе процессоров, равном 8. В случае одномерного разбиения на каждой итерации рассматриваемого алгоритма всем процессорам необходимо переслать $2\times7\times nx$ чисел, при этом 2×7 раз вызывается подпрограмма обменов. Для двумерного разбиения, когда число процессоров по координате x составляет 4, а по y – 2, общий объем пересылаемой информации $3\times2\times ny+2\times nx$ чисел ($=4\times2\times nx$, когда $nx=ny$), т.е. в 1,75 раз меньше, чем для одномерного случая. Зато при двумерном разбиении необходимо 2×10 обращений к подпрограмме обменов.

Теперь рассмотрим обмены, совершаемые каждым процессором. В одномерном случае на каждой итерации каждый процессор пересылает и получает по $2\times nx$ чисел (объем пересылок $4\times nx$) независимо от числа процессоров, подпрограммы двунаправленных обменов (*MPI_SENDRECV*) вызываются 2 раза. Для двумерного случая объем пересылок для каждого процессора уменьшается при росте числа процессоров и составляет $2\times2\times(nx+ny)/P$, где P – число процессоров ($=8nx/P$, если $nx=ny$), а подпрограммы обменов вызываются 4 раза. Казалось бы, что двумерное разбиение должно быть выгоднее уже при небольшом числе процессоров. Однако надо помнить, что на вычислительных системах рассматриваемого класса одно- или двухпроцессорные вычислительные узлы имеют всего одно соединение с остальными процессорами (через коммутатор) и скорость обмена по нему ограничена. Таким образом, на однопроцессорном вычислительном узле все 4

двухсторонних обмена при двумерном разбиении будут выполняться через одну сетевую карту. Ситуация еще хуже для двухпроцессорного вычислительного узла, где через одну сетевую карту будет выполняться сразу 6 обменов. Очевидно, что реальная скорость заданного обмена при одномерном разбиении будет выше в 2 раза для однопроцессорного узла и в 3 раза для двухпроцессорного. Также влияние накладных расходов на обмены будет всегда больше для двумерного разбиения.

Отметим, что в дорогих вычислительных системах вычислительные узлы имеют несколько соединений. Например, на Cray T3E каждый двухпроцессорный узел имеет 4 связи, топология соединительной сети представляет собой двумерный тор.

Теоретически при заданном числе процессоров по мере увеличения размерности задачи *nx* накладные расходы на каждый обмен останутся неизменными, а время, затрачиваемое на пересылку данных, будет расти. При увеличении числа процессоров и заданной размерности время на пересылку будет сокращаться при двумерном разбиении. Тогда, начиная с некоторой размерности и некоторого числа процессоров, для данной задачи более эффективным должно стать двумерное разбиение области. Однако может оказаться, что эта размерность и число процессоров превосходят разумные для данной задачи и вычислительной системы значения.

Для повышения эффективности распараллеливания необходимо использовать асинхронные обмены, т.е. совершать пересылки данных в приграничных частях на фоне расчетов во внутренних частях подобластей. Для этого используются подпрограммы *MPI_ISEND* и *MPI_IRECV*, также изменяется порядок вычислений в расчетной подобласти: сначала рассчитываются приграничные узлы, затем запускаются обмены вычисленных значений с соседними процессорами. Во время выполнения этих обменов совершаются расчеты в остальной (внутренней) части расчетной области. Проиллюстрируем это на нашем примере. Подпрограмма, объединяющая вычисления и асинхронные обмены для одномерного разбиения, приведена на рис. 4.

```
! Perform a Jacobi sweep for a 1-d decomposition,
! exchange boundary values.
! Sweep from a into b
! by M.Tolstykh from s/r sweep and exch1d
!
      subroutine swexch1(a,f,nx,s,e,b,comm1d,nbrbottom,nbrtop)
      integer nx, s, e
      double precision a(0:nx+1,s-1:e+1), f(0:nx+1,s-1:e+1),
```

```
     +                    b(0:nx+1,s-1:e+1)
!
      include ”mpif.h”
      integer i, j,comm1d,nbrbottom,nbrtop
      integer status_arr(MPI_STATUS_SIZE,4),ier,req(4)
      double precision h,c,x,y,pi2
! initialize receive
      call MPI_IRECV(b(1,s-1),nx,MPI_DOUBLE_PRECISION,
     &                    nbrbottom, 0, comm1d,req(1),ier)
      call MPI_IRECV(b(1,e+1),nx,MPI_DOUBLE_PRECISION,
     &                    nbrtop, 1, comm1d,req(2),ier)

      h = 1.0d0 / dble(nx+1)
      pi2 = asin(1.d0)
! calculate boundary values
      do i = 1, nx
        x = dble(i-1)*h*pi2
        y = dble(s-1)*h*pi2
        c = cos(x)**3*cos(y)**2
        b(i,s) =0.25*c*(a(i-1,s)+a(i,s+1)+a(i,s-1)+a(i+1,s))-
     +    h*h* f(i,s)*cos(x**3)*sin(x**2)*cos(y**3)*sin(y**2)
        y = dble(e-1)*h*pi2
        c = cos(x)**3*cos(y)**2
        b(i,e) =0.25*c*(a(i-1,e)+a(i,e+1)+a(i,e-1)+a(i+1,e))-
     +     h*h* f(i,e)*cos(x**3)*sin(x**2)*cos(y**3)*sin(y**2)
      enddo
! initialize send of boundary values
        call MPI_ISEND(b(1,e),nx,MPI_DOUBLE_PRECISION,
     &                     nbrtop,0,comm1d,req(3),ier)
        call MPI_ISEND(b(1,s),nx,MPI_DOUBLE_PRECISION,
     &                    nbrbottom, 1, comm1d,req(4),ier)
! then calculate the rest of the domain
      do j = s+1, e-1
        do i = 1, nx
          x = dble(i-1)*h*pi2
          y = dble(j-1)*h*pi2
          c = cos(x)**3*cos(y)**2
          b(i,j)=0.25*c*(a(i-1,j)+a(i,j+1)+a(i,j-1)+a(i+1,j))
     +    -h*h* f(i,j)*cos(x**3)*sin(x**2)*cos(y**3)*sin(y**2)
```

```
      enddo
    enddo
! waiting for completion of exchanges
    call MPI_WAITALL( 4, req, status_arr, ier)
    return
    end
```

Рис. 4. Текст подпрограммы вычислений и асинхронных обменов swexch1

В головной программе первые 4 строки цикла с меткой 10 заменяются двумя вызовами этой подпрограммы:

```
call swexch1(a,f,nx,s,e,b,comm1d,nbrbottom,nbrtop)
call swexch1(b,f,nx,s,e,a,comm1d,nbrbottom,nbrtop)
```

Результаты расчетов по этой программе приведены в табл. 2, четвертая прографка. Мы видим, что эффективность распараллеливания по сравнению с исходной версией программы возросла. Можно использовать асинхронные обмены и для двумерного разбиения расчетной области, но в нашем примере для данной размерности задачи это вряд ли оправдано по причине небольшого числа арифметических операций (во время выполнения которых должна осуществляться пересылка данных).

В случае же решения трехмерных задач или двумерных задач с большим объемом вычислений в промежутках времени между обменами и/или с большой размерностью массивов использование двумерного разбиения области в сочетании с асинхронными обменами, как правило, приводит к большей эффективности распараллеливания. Это утверждение справедливо лишь для данной структуры обменов, когда обмены совершаются только с процессорами, ведущими вычисления в подобластях, которые находятся слева, справа, сверху и снизу от подобласти данного процессора. Если же, например, шаблон разностной схемы требует пересылки данных между процессорами, которые ведут расчеты в подобластях, находящихся по диагонали от данной, то число обменов, совершаемых каждым процессором при двумерном разбиении, вырастет с 4 до 8. Это скорее всего сделает реализацию двумерного разбиения на вычислительных системах рассматриваемого класса невыгодным из-за больших накладных расходов на обмены. Для одномерного разбиения число обменов в этом случае останется неизменным.

Из результатов этого теста можно также сделать вывод, что в случае решения системы уравнений на параллельном компьютере с невысокими

характеристиками связующей сети для уменьшения числа обменов (и, следовательно, затрат на их организацию) полезно все расчетные переменные помещать в один массив. При этом индексы данного массива, соответствующие координатам, по которым производится разбиение области, следует делать последними.

Выводы

Прежде чем приступить к программированию параллельного алгоритма, нужно выбрать оптимальную стратегию распараллеливания, основываясь на технических характеристиках процессоров и связующей сети используемой вычислительной системы, а также на сведениях о свойствах алгоритма, размерности и вычислительной сложности решаемой задачи. При этом можно заранее оценить эффективность параллельной реализации модели. Для достижения высокой эффективности распараллеливания следует использовать асинхронные обмены данными между процессорами. Это может потребовать изменения расчетных подпрограмм исходной последовательной версии программы.

Список литературы

1. *Gropp W., Lusk E., Skjellum A*. Using MPI-2: Portable Parallel Programming with the Message-Passing Interface. – 1999. – MIT Press. – 382 p.
2. http://parallel.ru/vvv/mpi.html
3. http://www-unix.mcs.anl.gov/mpi/index.html

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ В ЗАДАЧАХ ОПЕРАТИВНОГО ПРОГНОЗА ВРЕМЕНИ СУШКИ СЛОЯ ЛЕСНЫХ ГОРЮЧИХ МАТЕРИАЛОВ

Н.В. Барановский

Томский государственный университет

Введение

Так как воспламеняются только достаточно сухие лесные горючие материалы (ЛГМ) [1], то прогноз лесной пожарной опасности невозможен без определения времени сушки слоя ЛГМ на подстилающей поверхности. В России 1807 лесхозов, 7851 лесничество, а в каждом лесхозе может быть до 100 выделов [2]. Технология параллельных вычислений позволит оперативно прогнозировать время сушки слоя ЛГМ при лесопожарном мониторинге в масштабах крупных регионов.

Наиболее полная и поэтому наиболее точная математическая модель сушки слоя ЛГМ дана в работе [3]. Недостатком этой постановки является обилие эмпирических констант и функций и большое количество нелинейных дифференциальных уравнений в частных производных. В [4] сделаны допущения, что: 1) конвективный теплообмен между слоем ЛГМ и приземным слоем атмосферы удовлетворительно описывается граничными условиями третьего рода; 2) давление P, температура T и плотность ρ газовой фазы в слое ЛГМ совпадают с соответствующими метеорологическими данными (P_e, T_e и ρ_e); 3) излучение в слое ЛГМ подчиняется закону Бугера - Ламберта [4]; 4) испарение связанной воды и капелек воды, прилипших к элементам ЛГМ, описывается одним и тем же законом Герца - Кнудсена.

Однако численное решение задачи в постановках [3, 4] не позволяет говорить о сколько-нибудь оперативном прогнозе лесной пожарной опасности.

1. Аналитическая формула для времени сушки слоя ЛГМ

В работе [5] дается уточнение постановки задачи [4], в рамках которого учитывается тот факт, что одной из движущих сил процесса сушки является разность давления насыщенного пара и парциального давления паров воды.

В результате интегрирования дифференциальных уравнений в частных производных [5] по ζ от 0 до 1 с учетом граничных условий получены

обыкновенные дифференциальные уравнения для средней по слою ЛГМ безразмерной температуры θ_s и безразмерного объемного влагосодержания слоя φ_2 [5].

В результате аналитического решения системы уравнений получена формула для определения безразмерного времени сушки слоя лесных горючих материалов [5]:

$$\tau_* = \frac{\sqrt{(1-\pi_{e0})^2 + 2\{\theta'_{s0} - \pi_{e0}\}\frac{1}{\gamma} ln\frac{\varphi_{2H}}{\varphi_{2*}}} - (1-\pi_{e0})}{\{\theta'_{s0} - \pi'_{e0}\}}, \qquad (1)$$

$$\pi'_{e0} = \left.\frac{d\pi_e(\tau)}{d\tau}\right|_{\tau=0}, \quad \theta'_{s0} = \left.\frac{d\theta_s(\tau)}{d\tau}\right|_{\tau=0}.$$

Размерное время сушки в этом случае имеет вид [8]:

$$t_* = \frac{\tau_{*2}\rho_1\varphi_{1H}C_{p1}RT_{S*}^{5/2}}{q_2 k'_{02}\rho_2 E} exp\left(\frac{E}{RT_{S*}}\right). \qquad (2)$$

2. Параллельная реализация

Использование формулы (1) позволяет получить оперативный прогноз, когда речь идет о небольшой контролируемой территории. Вследствие большой вычислительной нагрузки использование однопроцессорной компьютерной техники затруднительно. Чтобы обеспечить равномерную загрузку процессорных узлов МВС вычислительной нагрузкой, используем SPMD-модель вычислений [6]. Исходные данные разрезаются и рассылаются корневым процессом параллельной программы. По окончании вычислений корневой процесс собирает результаты со всех процессорных узлов и сохраняет их в файле результатов (рис. 1).

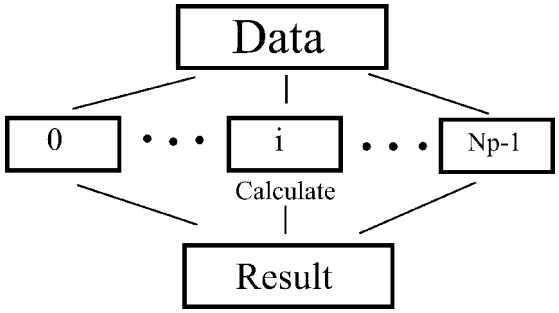

Рис. 1. Алгоритм работы параллельной программы

Параллельная программа реализована на Си с применением функций MPI. Характеристика отладочной машины: CPU Intel PentiumIII-800MHz, RAM 128 Mb, OS Linux, LAM 6.5.1 Характеристики кластера ТГУ: 18 CPU Intel PentiumIII-650MHz (9 Dual MB), RAM 256 Mb, 36 Gb HDD, OS FreeBSD 4.3, LAM-MPI 6.2.

Полученные результаты анализа работы параллельной программы представлены на рис. 2.

а *б*

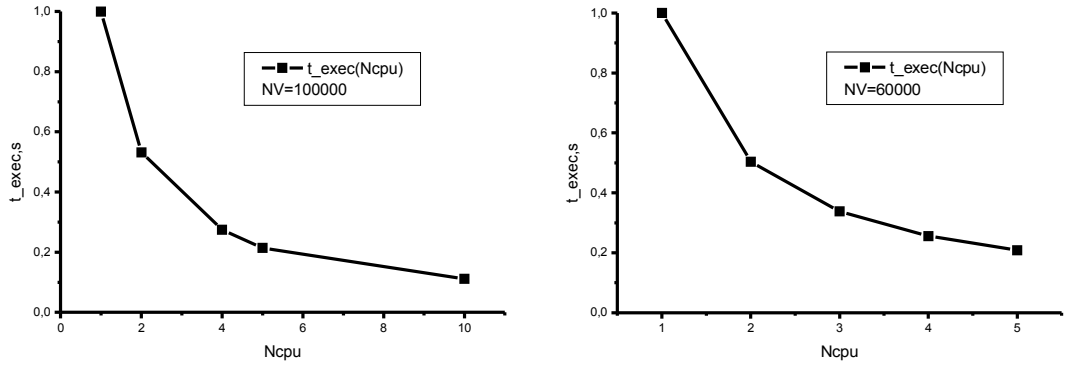

Рис.2. Зависимость времени численного расчета от числа задействованных процессоров: а – расчет по 100000 выделов; б – расчет по 60000 выделов

На рис. 3 приведены зависимости времени работы параллельной программы от величины вычислительной нагрузки.

а *б*

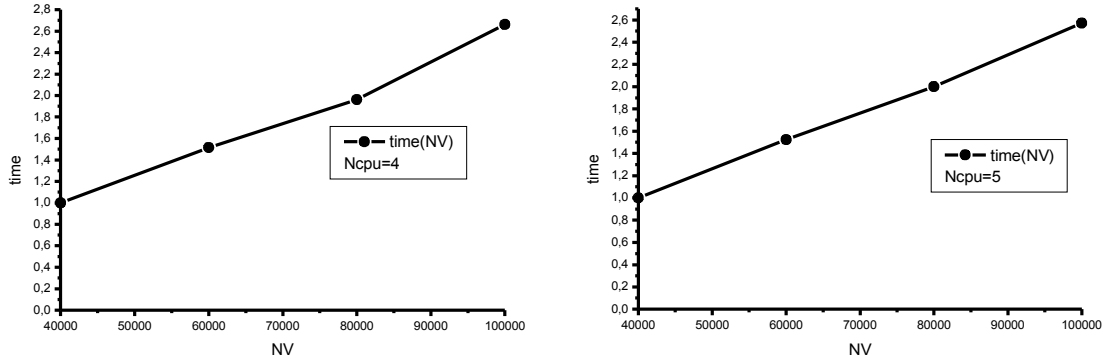

Рис.3. Зависимость времени работы параллельной программы на 4 (а) и 5 (б) процессорах от величины вычислительной нагрузки в пределах 40000-100000 выделов

На рис. 4 представлены зависимости эффективности и ускорения параллельной программы от числа процессоров.

а *б*

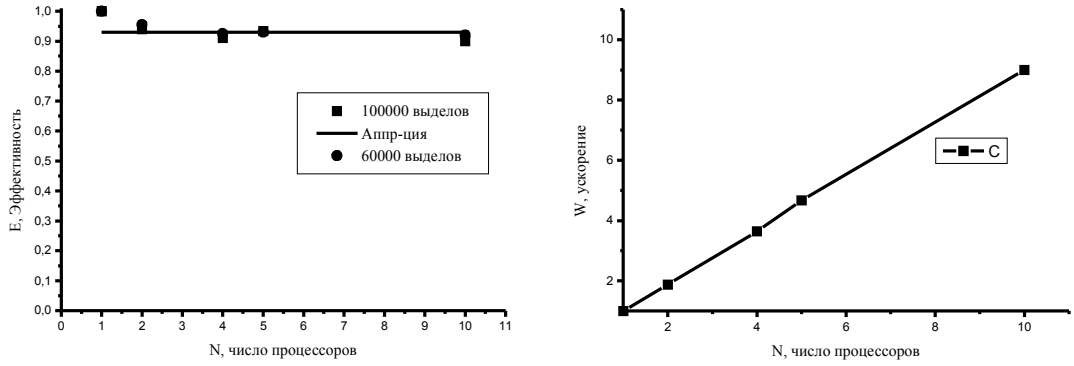

Рис.4. Эффективность (а) и ускорение (б) в зависимости от числа процессоров

Алгоритм задачи очень хорошо подходит для параллельной реализации. В процессе работы параллельной программы происходит минимум передач данных. Передача идет на этапе рассылки исходных данных корневым процессом процессорным узлам и на завершающем этапе при сборе данных корневым процессом с процессорных узлов кластера, а в процессе вычислений нет никаких обменов данными между ветвями параллельной программы. В результате имеем почти пропорциональное числу процессоров уменьшение временных затрат на работу параллельной программы. Иными словами, имеет место ускорение, практически пропорциональное числу задействованных процессоров кластера. Под эффективностью понимается отношение $E = \frac{t_1}{pt_p}$, где p – число задействованных процессоров вычислительной системы; t_1 – время работы параллельной программы при использовании одного процессора; t_p – время работы параллельной программы при использовании p процессоров. Эффективность практически не зависит от числа задействованных процессоров, так как в процессе работы программы нет межпроцессорных передач данных.

Заключение

Таким образом, рассмотрены основные моменты разработки параллельной вычислительной программы для прогноза времени сушки слоя ЛГМ на территориях большого масштаба, программное и аппаратное

обеспечение, использованное для параллельных вычислений. Приведены результаты анализа работы параллельной MPI-программы.

Список литературы

1. *Гришин А.М*. Физика лесных пожаров. – Томск: Изд-во Том. ун-та,1994. – 218 с.
2. *Исаев А.С., Коровин Г.Н., Титов С.П. и др.* Экологические проблемы поглощения углекислого газа посредством лесовосстановления и лесоразведения в России: Аналитический обзор. – М.: Центр экологической политики, 1995. – 156 с.
3. *Гришин А.М., Голованов А.Н., Катаева Л.Ю., Лобода Е.Л.* Постановка и решение задачи о сушке слоя лесных горючих материалов //Физика горения и взрыва. – 2001. – Т.37, №1. – С.65-76.
4. *Гришин А.М., Катаева Л.Ю., Лобода Е.Л.* Математическое моделирование сушки слоя лесных горючих материалов //Proceeding of International Conference RDAMM-2001. – 2001. – Vol.6. pt.2. Special Issue. – P.140-144.
5. *Гришин А.М., Барановский Н.В., Мейнерт И.В., Павшук Ю.Ю.* Аналитическое и численное решение задачи о сушке слоя ЛГМ //Лесные и степные пожары: возникновение, распространение, тушение и экологические последствия //Материалы 4-й Международной конференции. – Томск: Изд-во Том. ун-та, 2001. – С. 47-51.
6. *Малышкин В.Э.* Основы параллельных вычислений: Учеб. пособие. – Новосибирск: Изд-во НГТУ, 1998. – 60 с.

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ВЫЧИСЛИТЕЛЬНОЙ ГИДРОДИНАМИКИ SIMPLE

Н.Н.Богословский, А.О.Есаулов, А.В.Старченко

Томский государственный университет

Решение многих задач, в основу которых положена система уравнений Навье - Стокса, всегда являлось непростой вычислительной проблемой. Решение этой системы без существенных упрощений возможно при условии взаимодополнения эффективных численных методов применением высокопроизводительных компьютеров. Суперкомпьютерная техника с самого своего появления находила широкое применение в этой области математического моделирования. В последние годы в нашей стране резко повысилось число относительно недорогих многопроцессорных вычислительных систем, что, разумеется, сказалось и на общем уровне их доступности для проведения трудоемких расчетов.

Использование разработанных ранее эффективных последовательных методов существенно ограничивается тем, что параллельные реализации алгоритмов, как правило, существенно отличаются от аналогичных последовательных. По сути дела, зачастую проще разработать новую параллельную программу, чем распараллелить старую последовательную.

Самой распространенной задачей, на которой принято тестировать численные схемы решения системы уравнений Навье - Стокса, является задача о вихревом движении в прямоугольной полости (каверне), целиком заполненной вязкой несжимаемой жидкостью [1, 2].

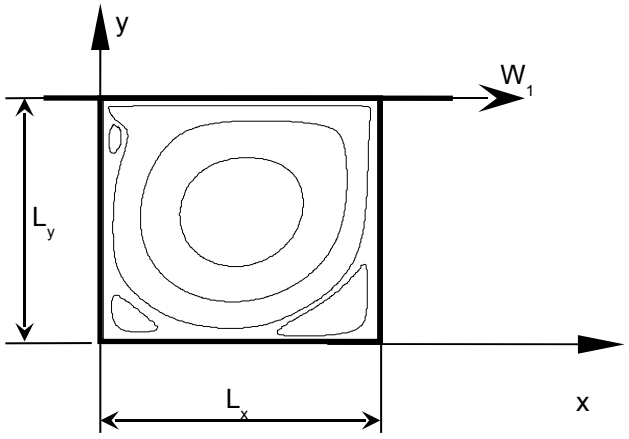

Рис. 1. Каверна

Авторами рассматривалась двумерная задача о стационарном течении несжимаемой вязкой жидкости с постоянными свойствами в прямоугольной полости (ка-

верне) с верхней стенкой, движущейся с постоянной скоростью (рис. 1). Жидкость, целиком заполняющая каверну, вовлекается в движение силами вязкости (действие массовых сил незначительное). Кроме того, требуется выполнение условия прилипания частиц жидкости к твердой стенке.

Рассматриваемое течение (в так называемых примитивных переменных) определяется уравнениями Навье - Стокса:

уравнение неразрывности

$$\frac{\partial(\rho u)}{\partial x}+\frac{\partial(\rho v)}{\partial y}=0;$$

уравнения движения

$$\frac{\partial(\rho u^2)}{\partial x}+\frac{\partial(\rho uv)}{\partial y}=\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)-\frac{\partial p}{\partial x};$$

$$\frac{\partial(\rho uv)}{\partial x}+\frac{\partial(\rho v^2)}{\partial y}=\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)-\frac{\partial p}{\partial y};$$

а также *уравнениями состояния*:

$$\rho = \text{const};\quad \mu = \text{const}$$

и граничными условиями (условиями прилипания):

$$x=0:\begin{cases}u=0,\\ v=0,\end{cases}\qquad x=L_x:\begin{cases}u=0,\\ v=0,\end{cases}$$

$$y=0:\begin{cases}u=0,\\ v=0,\end{cases}\qquad y=L_y:\begin{cases}u=W_1,\\ v=0,\end{cases}$$

где u,v – горизонтальная и вертикальная компоненты скорости; μ – коэффициент динамической вязкости; ρ – плотность; p – давление.

Для дискретизации исходной системы дифференциальных уравнений применялся метод контрольного объема. Конечно-разностный аналог получался на разнесенных сетках (в центре контрольного объема вычисляется давление, а на гранях – компоненты скорости). Сетка строилась с узлами, сгущающимися к границам расчетной области. Для аппроксимации

конвективных потоков применялись обычные противопотоковые разности (выбор схемы первого порядка обусловлен тем, что авторы в первую очередь стремились получить простую параллельную реализацию классического алгоритма, тогда как в будущем предполагается использование схем более высокого порядка [2]).

Решение систем алгебраических уравнений, получающихся в результате дискретизации, производилось методом Гаусса - Зейделя с нижней релаксацией и явным методом Булеева.

Для получения поля давления, согласованного с векторным полем скорости, использовался алгоритм SIMPLE Патанкара [3], основные этапы которого можно представить следующей последовательностью шагов:

1) задаются начальные значения полей скорости и давления;
2) решаются уравнения движения и рассчитываются компоненты вектора скорости;
3) решается уравнение для поправки давления;
4) корректируются компоненты вектора скорости так, чтобы они удовлетворяли уравнению неразрывности;
5) в случае, если максимальное абсолютное значение поправки давления велико, производится возврат ко второму шагу.

Распараллеливание производилось по данным, то есть расчетная область разделялась на подобласти и в каждой подобласти вычисления производились одновременно. Из практики параллельных вычислений хорошо известно, что в целом более эффективная двумерная декомпозиция расчетной области теряет свои преимущества перед одномерной в том случае, если число процессоров параллельного компьютера невелико [4]. А так как изначально ставилась задача проведения расчетов на имеющихся в Томске кластерных системах, то использовалось одномерное разрезание вдоль горизонтальной координаты (рис. 2), так как такому разрезанию свойственна простота программной реализации.

На рис. 2 в правой части схематично изображено распределение данных между процессами. Левая половинка выделенного круга представляет область, рассчитываемую 0-процессом, а правая половинка – рассчитываемую 1- процессом. Кружками обозначены точки, в которых

вычисляется давление, а горизонтальными и вертикальными штрихами – точки, соответствующие горизонтальной и вертикальной компонентам скорости.

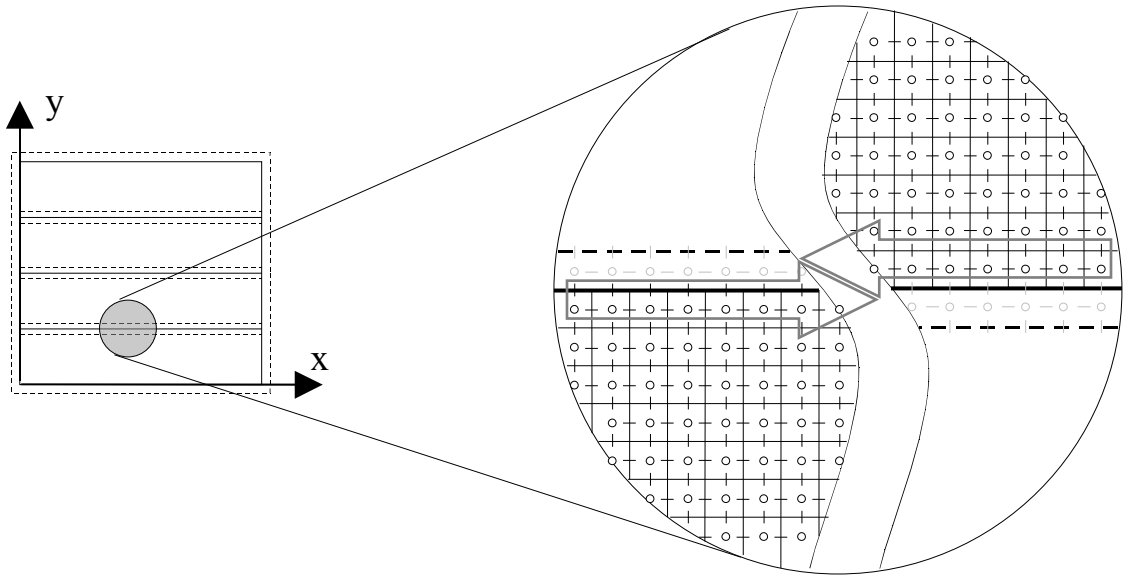

Рис. 2. Одномерная декомпозиция расчетной области

На разных этапах расчетов требовалась синхронизация, то есть обмен данными, расположенными в прилежащих к границам разрезов областях, между соседними процессами. Например, для расчета давления в граничном сеточном слое требуются четыре смежных значения (пятиточечный шаблон), три из которых рассчитываются и расположены в данном процессе, а четвертое надлежит получить от соседнего процесса. По этой причине необходима пересылка последней строки 0-процесса 1-процессу, равно как и получение от него его первой строки. Для этой цели использовались операции блокирующей парной пересылки данных MPI (функция *SendRecv*). Аналогичным образом выглядит схема расчета и обмена значениями для горизонтальной компоненты скорости. Для вертикальной компоненты скорости в принципе схема остается той же, с той лишь разницей, что вычисление скорости на границе возлагается на предыдущий процесс (в данном случае на 0-процесс), тогда как последующий ведет расчеты, начиная с первой сеточной линии, отличной от «линии разреза».

Обобщая, можно сказать, что для каждой характеристики (компонента скорости, давление) каждый процесс должен осуществлять операции посылки

первой и последней строк массива данных, так же как и принимать граничные условия, представляющие собой данные, присланные соседними процессами.

Последовательность шагов алгоритма SIMPLE при его параллельной реализации претерпит определенные изменения.

Во-первых, расчеты компонент скорости и поправки давления производятся каждым процессом в своей подобласти.

Во-вторых, после расчета и коррекции скорости требуется обмен приграничными значениями (после пп. 2 и 4).

В-третьих, после расчета поля давления (п. 3) необходима пересылка граничных значений давления; кроме того, в ходе решения уравнения для поправки давления нужно производить синхронизацию операции поправки давления.

Так как компоненты скорости не обязательно вычислять точно на п.2, было решено применить простой метод Гаусса - Зейделя для вычисления компонент скорости. Точность численного решения уравнения Пуассона для поправки давления определяет степень удовлетворения уравнения неразрывности, поэтому его итерационное решение нужно проводить достаточно точно. По этой причине использовался явный метод Н.И.Булеева. На «разрезах» ставились условия первого рода для поправки давления, то есть в качестве граничных условий использовались результаты расчетов соседних процессов. Так как в ходе решения уравнения Пуассона эти «граничные» условия менялись, потребовалась синхронизация поправки давления на каждой итерации метода Булеева. Как было практически установлено для вычислительных систем, на которых производились расчеты, время выполнения парной операции пересылки одного элемента массива примерно в 10-15 раз больше времени выполнения одной арифметической операции над этим элементом. Была произведена попытка уменьшить число пересылок путем проведения синхронизации не на каждой итерации, а скажем, через 5. Но, к сожалению, она не привела к уменьшению времени расчетов, так как ухудшились свойства сходимости алгоритма и в целом выигрыш оказался сомнительным.

Предложенная методика, без сомнения, может быть применена не только к изучению течения жидкости в каверне, но и к более сложным задачам.

На исходной задаче (течение вязкой жидкости в каверне) исследовалось ускорение параллельного алгоритма по сравнению с последовательным алгоритмом.

Расчеты велись на нерегулярных сетках (от 40x40 до 200x200 узлов) при Re=1000 с использованием удвоенной точности. Итерационный процесс продолжался до тех пор, пока максимальное значение поправки давления не становилось меньше 10^{-4}. Релаксационные коэффициенты в методах Гаусса - Зейделя и Булеева составляли соответственно 0,5 и 0,8.

Как известно, анализ параллельного алгоритма позволяет получить аналитическую формулу ускорения в виде

$$S_p = \frac{p}{1 + \gamma \frac{p(p-1)}{N_x} \cdot \frac{t_{\text{п}}}{t_{\text{а}}}},$$

где N_x – размерность расчетной сетки; p – число процессоров; $t_{\text{п}}$ – время выполнения операции пересылки; $t_{\text{а}}$ – время выполнения арифметической операции, а полуэмпирический параметр γ характеризует соотношение объема пересылаемых к общему объему данных, то есть свойства самого алгоритма. Данному алгоритму соответствует $\gamma \approx 0{,}21$.

Увеличение числа процессоров уменьшает отношение количества внутренних узлов к количеству граничных, откуда нетрудно сделать вывод, что эффективность параллельной реализации данного алгоритма будет снижаться с увеличением числа процессоров при фиксированном N_x. Наоборот, увеличение размерности расчетной сетки при фиксированном p влечет повышение эффективности.

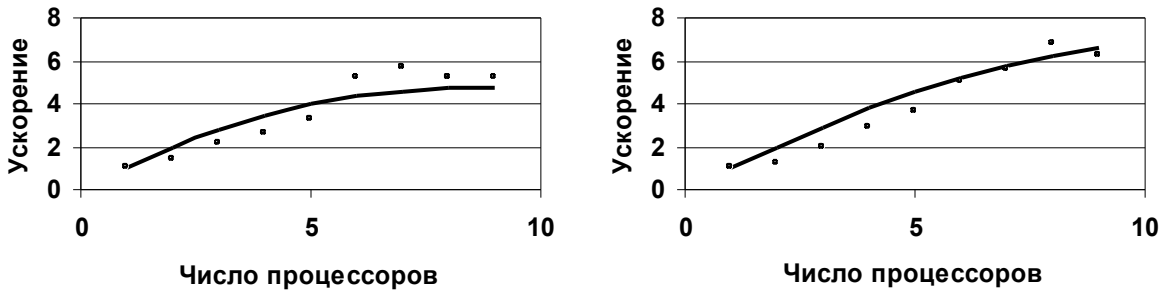

Рис. 3. Ускорение параллельного алгоритма, слева - N_x=40, справа - N_x=100

На рис. 3 изображены зависимости ускорения, показываемого параллельным алгоритмом для расчетных сеток размерностей 40x40 и 100x100 узлов. Сплошная линия соответствует аналитической зависимости, а кружки – экспериментальным данным. Рис. 4 представляет зависимость времени расчетов от размерности сетки.

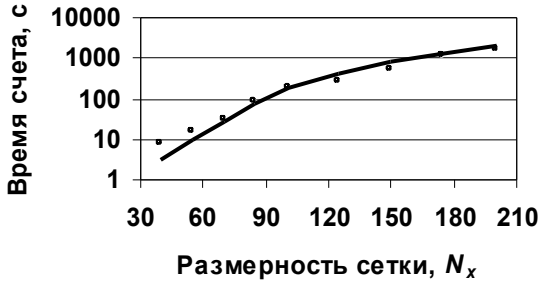

Рис. 4. Ускорение параллельного алгоритма на 9 процессорах при различных размерностях сетки

В итоге можно сделать вывод, что, несмотря на сложность рассматриваемого алгоритма, его параллельная реализация достаточно эффективна: на 9 процессорах ускорение может происходить в 6-7 раз по сравнению с последовательным алгоритмом.

Расчеты производились на кластере Томского государственного университета (9 процессорных пар Pentium-III-650).

Список литературы

1. *Лойцянский Л.Г.* Механика жидкости и газа. – М.: Наука, 1987. – 840 с.

2. *Есаулов А.О., Старченко А.В.* К выбору схемы для численного решения уравнений переноса //Вычислительная гидродинамика. – Томск: Изд-во Том. ун-та, 1999. – С.27-32.
3. *Патанкар С.* Численные методы решения задач теплообмена и динамики жидкости /Пер. с англ. – М.: Энергоатомиздат, 1984. – 152 с.
4. *Корнеев В.Д.* Параллельное программирование в MPI. – Новосибирск: Изд-во СО РАН, 2000. – 213 с.

НЕКОТОРЫЕ АСПЕКТЫ МОДЕЛИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ И КЛАСТЕРНЫХ ВЫЧИСЛЕНИЙ

А.В. Малышев

Институт вычислительного моделирования СО РАН

1. О моделировании параллельных вычислений

С проблемой моделирования параллельных вычислений так или иначе сталкивается всякий, кто ставит и пытается решать вычислительные задачи на кластерной системе. Однако в большинстве случаев математик-вычислитель ограничивается эмпирической оценкой, выполняемой в соответствии с личным опытом и интуицией, полагая, что данный аспект постановки задачи не играет существенной роли и может быть пройден мимоходом или вообще пропущен. При решении пробных или небольших по размеру задач можно действовать наугад, но при планировании масштабного вычислительного эксперимента, время счета которого исчисляется днями и неделями, выбор способа разбиения, размера подзадач или способа организации собственно вычислений может существенно повлиять на время счета и, как следствие, на результаты эксперимента в целом.

Построение модели параллельных вычислений даст возможность:

- проводить качественный и количественный анализ хода процесса вычислений до начала вычислений;
- выполнять априорную и апостериорную оценки эффективности параллельных вычислений;
- делать обоснованный выбор алгоритма решения, способа и параметров разбиения этого алгоритма для достижения оптимальных значений ускорения и/или эффективности.

Требования к модели:

- **полнота**, то есть учет основных технических характеристик кластера: объема памяти, типа, тактовой частоты и размера кэша процессоров и плат вычислительных узлов, латентности и скорости сетевых обменов и дисковых операций;
- **конструктивность**, то есть возможность использовать модель для анализа вычислительного процесса и определения его качественных и

количественных характеристик (временная диаграмма, ускорение, эффективность, оптимальность);

- **адекватность**, то есть соответствие результатов, полученных при моделировании, ходу реального вычислительного процесса;
- **компаративность** – соотносимость результатов, даваемых моделью, с результатами наиболее распространенных тестов производительности, применяемых для кластерных систем.

2. Общие подходы к моделированию параллельных вычислений

В самом общем виде можно выделить два подхода к моделированию: это подходы «от инструмента» и «от задачи».

Подход «от инструмента» включает в себя анализ характеристик кластера как программно-аппаратной системы, выполняющей некоторый набор операций. К числу таких характеристик относятся:

● *быстродействие узла* кластера: количество арифметических операций (умножений и/или сложений) в секунду.

Этот параметр зависит не только от тактовой частоты процессора, но и от множества факторов, включая размеры кэша, оперативной памяти, размеров блоков данных и способов их организации. В качестве иллюстрации рассмотрим рис. 1, на котором показана зависимость быстродействия от размера данных решаемой узлом подзадачи, выявленная в ходе

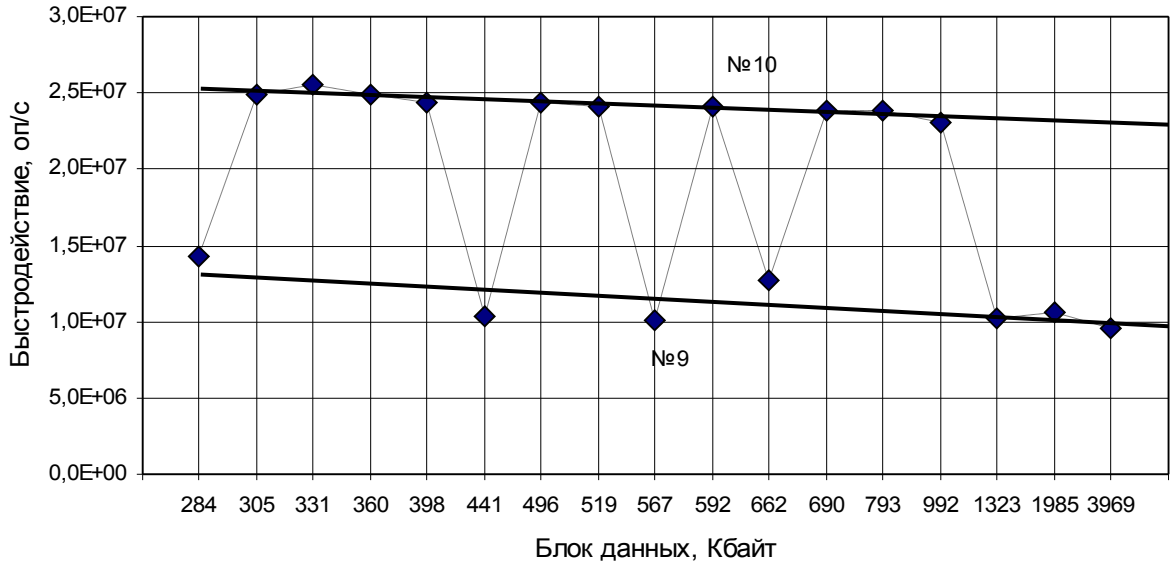

Рис. 1. Зависимость быстродействия узла от размера подзадачи

экспериментов.

Из рис. 1 видно, что, во-первых, существует резкий скачок производительности (она изменяется почти в два раза) при малом изменении размера данных. Так, точки №9 и 10 отделяют друг от друга 25 Кбайт и $1.4 \cdot 10^7$ оп/с. При этом на обоих уровнях точки выстраиваются в прямые, которые на рассмотренном участке практически параллельны. Объяснить этот скачок без дополнительных исследований трудно; можно сделать предположение, что он связан с попаданием либо непопаданием в кэш процессора соответствующих блоков данных.

Во-вторых, видно, что функция быстродействия монотонно (линейно) убывает при увеличении размера блока данных, независимо от уровня. Это лишний раз говорит как о «моделируемости» параметра, так и о пользе распараллеливания: превращение крупной задачи в набор мелких позволяет «бесплатно» получать увеличение быстродействия;

- *латентность* коммуникационной подсистемы и практическая *скорость* обмена (вещественных чисел в секунду).

Эти параметры зависят не только от типа и скорости используемого коммуникационного оборудования, но и от используемой коммуникационной библиотеки. Так, считается, что пакет LAM для протокола TCP/IP дает существенно меньшую латентность, чем пакет библиотек MPICH для того же сетевого протокола.

Подход «от задачи» состоит в рассмотрении вычислительного процесса с точки зрения его основной функции – решения вычислительной задачи, в известной степени абстрагируясь от программно-аппаратного обеспечения.

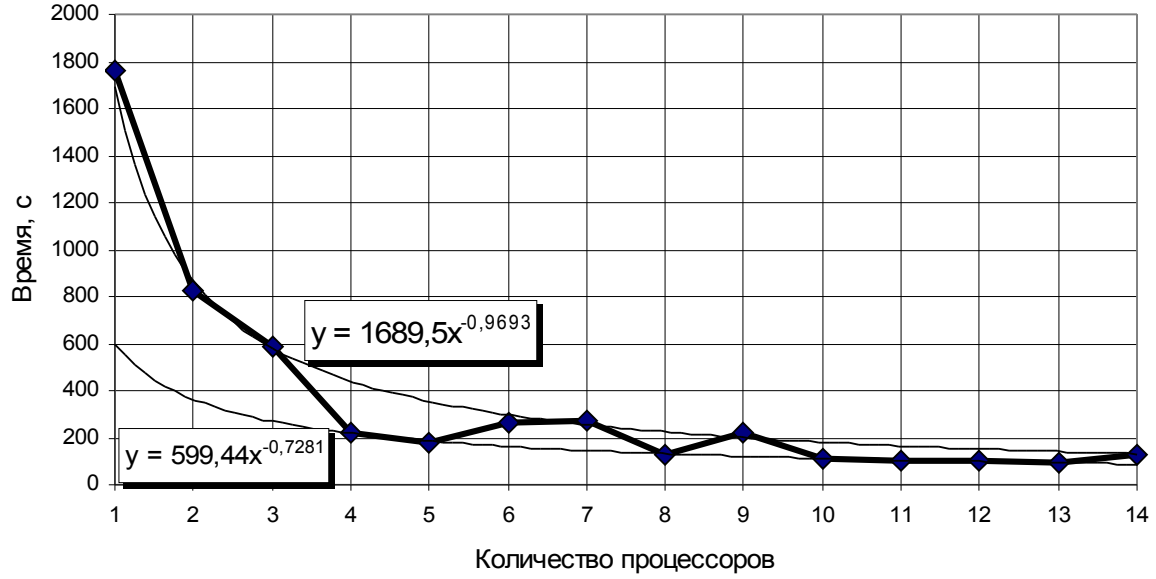

Рис. 2. Зависимость времени счета от количества процессоров

Здесь рассматриваются характеристики задачи или класса задач, связанные с внутренним параллелизмом известных алгоритмов их решения, возможностями конструирования новых параллельных алгоритмов и т.п.

На рис. 2 изображен график зависимости времени перемножения двух матриц размером 2016×2016 от количества задействованных процессоров. Жирной линией показана фактическая кривая, снятая по результатам вычислений. На первый взгляд ее вид не позволяет говорить об эффективном аналитическом моделировании; но, сопоставив с рис.1, нетрудно увидеть две кривые: одна составлена точками, которые соответствуют попаданию быстродействия на верхний уровень, другая – точками, где быстродействие оказалось на нижнем уровне. Эти кривые - по отдельности - очень хорошо приближаются степенными функциями; их уравнения показаны на графике. В данной задаче вклад межпроцессных обменов и связанных с ними нелинейностей оказался достаточно мал, поэтому время счета оказалось зависящим преимущественно от быстродействия. Важно отметить еще и тот факт, что, как показано в [1], дальнейшее увеличение количества задействованных процессоров в данной задаче должно привести к увеличению времени счета из-за роста межпроцессных обменов. В реальных задачах, отличающихся существенным межпроцессным трафиком, этот фактор может иметь очень большое значение.

Заключение

Адекватное конструктивное моделирование параллельных вычислительных систем и параллельных вычислений – актуальная и пока в целом неразрешенная проблема. Используемый в настоящее время эмпирический подход к планированию и постановке параллельного вычислительного эксперимента должен и, скорее всего, будет постепенно вытесняться использованием теоретически обоснованных и экспериментально проверенных моделей, тем более что принципиальных препятствий к построению таких моделей пока не обнаруживается. Основные параметры, используемые в подходе «от инструмента», во многих случаях хорошо приближаются аналитическими кривыми. Когда же это не так, можно пытаться построить зависимости специального вида либо использовать вероятностные методы оценки. Рассмотрение же «от задачи» может опираться на известные методы последовательной вычислительной математики, дополненные или переработанные в соответствии со спецификой параллельного вычисления.

Список литературы

1. *Малышев А.В., Шайдуров В.В.* Параллельные вычисления на кластерах из персональных ЭВМ //Вычислительные технологии. – 2001. – Т.6, ч.2. – Спец. выпуск. – С. 287-293.

СОДЕРЖАНИЕ

1. ПЛЕНАРНЫЕ ДОКЛАДЫ

Тихонов А.Н., Скуратов А.К., Домрачев В.Г., Ретинская И.В. К проблеме подготовки специалистов по высокопроизводительным вычислениям…….……3
Воеводин Вл.В. Учебно-научный центр МГУ по высокопроизводительным вычислениям..……..…….……...24

2. ДОКЛАДЫ ПО МЕТОДАМ И ТЕХНОЛОГИЯМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Андрианов А.Н., Бугеря А.Б., Ефимкин К.Н. НОРМА – специализированная система параллельного программирования ...……...…….……. 33
Вшивков В.А. О распараллеливании вычислительных алгоритмов ……..……...46
Малышкин В.Э. Параллельная реализация алгоритмов численного моделирования……..……60
Тимченко С.В. Некоторые подходы к построению параллельных алгоритмов вычислительной гидродинамики…….72
Толстых М.А., Глухов В.Н. Реализация моделей атмосферы на параллельных компьютерах…….…… …...82
Толстых М.А. Особенности использования MPI для распараллеливания задач математической физики на вычислительных системах с невысокой скоростью обменов между процессорами…….……94

3. ДОКЛАДЫ МОЛОДЫХ УЧЕНЫХ

Барановский Н.В. Параллельные вычислительные технологии в задачах оперативного прогноза времени сушки слоя лесных горючих материалов.……113
Богословский Н.Н., Есаулов А.О., Старченко А.В. Параллельная реализация алгоритма вычислительной гидродинамики SIMPLE…….……...118

Малышев А.В. Некоторые аспекты моделирования параллельных и кластерных вычислений……125

Научное издание

Сибирская школа семинар
по параллельным вычислениям

Томск, 17-20 декабря 2001 года

Редактор **Е.В.Лукина**
Компьютерная верстка: **А.О.Есаулов**

Лицензия ИД 04617 от 24.04.01 г. Подписано в печать 14.06.02 г.
Формат 60x80 1/16. Бумага офсетная №1. Ризография.
П. л. 8,25; уч.-изд. л. 8,5; усл. п. л. 7,67.
Тираж 300 экз. Заказ

Издательство ТГУ, 634029, г. Томск, ул. Никитина, 4
Типография «Иван Федоров», 634003, г. Томск, Октябрьский взвоз, 1