Министерство образования и науки Российской Федерации

САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Приоритетный национальный проект «Образование»
Национальный исследовательский университет

В. Л. БАДЕНКО

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Рекомендовано в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки магистров «Системный анализ и управление»

Санкт-Петербург
Издательство Политехнического университета
2010

УДК 681.3.06+519.68
ББК 32.973

Рецензенты:
Доктор технических наук, профессор,
Заведующий кафедрой ГОУ «СПбГПУ» *Ю. Я. Болдырев*
Кандидат технических наук, доцент,
Заведующая кафедрой Балтийского государственного технического университета «ВОЕНМЕХ» им. Д.Ф. Устинова *Н. Н. Смирнова*

Баденко В. Л. **Высокопроизводительные вычисления** : учеб. пособие / В. Л. Баденко. – СПб. : Изд-во Политехн. ун-та, 2010. – 182 с.

Представлено описание современных технологий, используемых в высокопроизводительных вычислениях. Дано описание архитектуры современных суперкомпьютеров, советующих технологий и средств программирования для них. В пособие также включены начальные сведения об исчислении взаимодействующих систем Робина Милнера.

Учебное пособие предназначено для студентов вузов, обучающихся по направлению подготовки магистров «Системный анализ и управление». Оно может быть использовано для обучения по направлению подготовки магистров "Прикладная математика и информатика», а также в системах повышения квалификации и учреждениях дополнительного профессионального образования. Материал, представленный в учебном пособии, будет интересен всем, кто специализируется в области разработки систем для реализации высокопроизводительных вычислений.

Работа выполнена в рамках реализации программы развития национального исследовательского университета «Модернизация и развитие политехнического университета как университета нового типа, интегрирующего мультидисциплинарные научные исследования и надотраслевые технологии мирового уровня с целью повышения конкурентоспособности национальной экономики»

Печатается по решению редакционно-издательского совета Санкт-Петербургского государственного политехнического университета.

ISBN

ОГЛАВЛЕНИЕ

СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ

CC-NUMA – системы, в которых обеспечивается когерентность локальной кэш-памяти разных процессоров

CCS – исчисление взаимодействующих систем Робина Милнера

COMA – системы, где для данных используется только локальная кэш-память процессоров

MPI – интерфейс передачи сообщений (библиотека функций)

MPP – многопроцессорная система с массовым параллелизмом

NCC-NUMA – системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэш-памяти

NUMA – неоднородный доступ к памяти

PVP – векторные параллельные процессоры

SMP – симметричные мультипроцессорные системы

SPMP – модель «одна программа – множество процессов»

UMA – однородный доступ к памяти

ВВЕДЕНИЕ

В настоящее время круг задач, требующих для своего решения применения мощных вычислительных ресурсов, все время расширяется. Это связано с тем, что произошли фундаментальные изменения в самой организации научных исследований. Сюда следует отнести эксперименты на коллайдерах со встречными пучками, расшифровку генома человека. Полностью не решена задача безопасного хранения ядерного оружия, а из-за запрета на ядерные испытания состояние накопленных зарядов можно определить только путем компьютерного моделирования. Очевидно, что решение таких масштабных задач требует значительных вычислительных ресурсов.

Высокопроизводительные вычисления в настоящее время не мыслятся без распараллеливания, ибо наиболее мощные вычислительные системы имеют сотни и тысячи процессоров, работающих одновременно и в тесном взаимодействии, т. е. параллельно. Благодаря распараллеливанию удается достичь производительности в тысячи терафлопс (терафлопс – 10^{12} операций в секунду с плавающей точкой), но для наиболее сложных современных задач и этого недостаточно: требуются вычислительные мощности с быстродействием в сотни пентафлопс (пентафлопс – 10^{15} операций в секунду с плавающей точкой).

Необходимость и естественность широкого использования высокопроизводительных параллельных вычислений связаны с тем, что многое в окружающем нас мире характеризуется параллелизмом – одновременной реализацией процессов и явлений окружающей среды. В частности, в живой природе параллелизм распространен очень широко в дублирующих системах для защиты от случайного воздействия, приводящего к разрушению части системы. Параллельные процессы пронизывают также и общественные отношения, ими характеризуются развитие науки, культуры и экономики в человеческом обществе. Среди этих процессов особую роль играют параллельные информационные потоки. Отдельно следует упомянуть циркуляцию

информации в человеческом обществе и в сети Интернет, потоки информации с искусственных спутников Земли, постоянно ведущих сканирование поверхности нашей планеты в интересах различных пользователей и т.п. При организации проведения высокопроизводительных вычислений стало обычным использование многозадачности и мультипрограммности, мультимедийных средств, компьютерных локальных сетей, а также глобальных сетей, таких, как Интернет. Это показывает, что серьезное изучение вопросов распараллеливания и высокопроизводительных вычислений чрезвычайно важно.

В современном быстро меняющимся мире появляются все новые и новые задачи, которые требуют использования высокопроизводительных вычислений, а имеющихся в распоряжении пользователей мощностей вычислительных систем для их решения не хватает. Кроме того, возрастают требования к точности и скорости решения прежних задач и поэтому вопросы разработки и использования сверхмощных компьютеров, называемых суперкомпьютерами, актуальны сейчас и в будущем. При этом многие проблемы носят не только технологический характер. Имеются чрезвычайно важные и трудно решаемые проблемы организационного плана, проблемы подготовки специалистов и другие.

С появлением высокопроизводительных суперкомпьютеров и компьютерных систем возникли специфические проблемы, которые требуют особого подхода к их решению. Например, как обеспечить эффективное решение задач на той или иной высокопроизводительной параллельной системе, и какими критериями эффективности при этом следует пользоваться. Важной задачей является поиск существующих путей увеличения производительности при модификации данной высокопроизводительной системы и связанная с ней задача обеспечения работоспособности существующих программ при увеличении количества параллельных компонентов системы.

В настоящем учебном пособии представлен подход к поиску ответов на эти и другие вопросы, возникающие при создании высокопроизводительных систем.

Текст учебного пособия во многом опирается на родственные учебные дисциплины, которые реализуются в Московском государственном университете, Санкт-Петербургском государственном университете, Нижегородском государственном университете, а также Южном федеральном университете.

Данное учебное пособие создано на основе курса лекций, который читает автор по учебной дисциплине «Инструментальные средства высокопроизводительных вычислений» для магистров, обучающихся на кафедре «Прикладная математика» Физико-механического факультета ГОУ «СПбГПУ». Данное учебное пособие следует рассматривать как введение в технологии, связанные с высокопроизводительными вычислениями. Оно содержит лишь часть материала программы по учебной дисциплине. При этом автор отдает себе отчет, что материал учебного пособия относится к чрезвычайно быстро меняющейся отрасли компьютерных технологий и поэтому может достаточно быстро устаревать.

В первой главе дается понятие о высокопроизводительных вычислениях. В главе 2 представлена архитектура современных суперкомпьютеров, а глава 3 посвящена проблемам, связанным с оценкой производительности суперкомпьютеров. Описание современных высокопроизводительных компьютерных технологий представлено в главе 4. При этом рассмотрены решения, которые предлагаются как отечественными компаниями, так и мировыми лидерами IBM и Cray. Главы 5-7 посвящены программированию для суперкомпьютеров, в частности, представлены основы интерфейса передачи сообщений (message passing interface – MPI). В главе 8 представлено краткое введение в исчисление взаимодействующих процессов Робина Милнера.

Учебное пособие предназначено для студентов вузов, обучающихся по направлению подготовки магистров «Системный анализ и управление». Оно может быть использовано для обучения по направлению подготовки магистров "Прикладная математика и информатика», а также в системах повышения квалификации и учреждениях дополнительного профессионального образования. Материал, представ-

ленный в учебном пособии, будет интересен всем, кто специализируется в области разработки систем для реализации высокопроизводительных вычислений.

1. ПОНЯТИЕ О ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЯХ

1.1. ПАРАЛЛЕЛЬНЫЕ КОМПЬЮТЕРЫ И СУПЕР-ЭВМ

После появления в середине 20 века первых электронных вычислительных машин – компьютеров, сфера их применения охватила практически все области человеческой деятельности. Однако наиболее важным по-прежнему остается использование их в том направлении, для которого они собственно и создавались, а именно, для решения больших задач численного моделирования, требующих выполнения громадных объемов вычислений. Такие задачи возникли в середине прошлого века в связи с развитием атомной энергетики, авиастроения, ракетно-космических технологий и ряда других областей науки и техники.

Сегодня также имеется большое количество важнейших практических задач, связанных с вычислительными экспериментами, решение которых требует использования огромных компьютерных мощностей. К таким задачам относятся, например, задачи точных долгосрочных прогнозов климатических изменений, геологических катаклизмов – землетрясений, извержений вулканов, столкновений тектонических плит, прогнозов цунами и разрушительных ураганов и др.

При этом следует отметить, что вследствие все более широкого внедрения компьютерной техники и всеобщего повышения требований к уровню и качеству жизни наблюдается значительное усиление внимания к численному моделированию и вычислительным экспериментам.

Численное моделирование, заполняя промежуток между физическими экспериментами и аналитическими подходами, позволило изучать явления, которые являются либо слишком сложными для ис-

следования аналитическими методами, либо слишком дорогостоящими или опасными для экспериментального изучения. При этом вычислительный эксперимент позволил значительно удешевить процесс научного и технологического поиска.

К сожалению, технологические возможности увеличения быстродействия процессоров ограничены по объективным причинам, связанным с физическими основами работы процессоров. Базовая проблема состоит в том, что увеличение быстродействия требует уменьшения размеров процессоров, а при малых размерах полупроводниковых элементов процессоров появляются трудности из-за квантово-механических эффектов, вносящих элементы недетерминированности. Следует констатировать, что эти трудности носят принципиальный характер, и их пока что не удается преодолеть и вряд ли удастся преодолеть в будущем.

Вследствие указанных выше причин для повышения производительности приходится идти по пути создания параллельных вычислительных систем, т.е. систем, в которых предусмотрена одновременная реализация ряда вычислительных процессов, связанных с решением одной задачи, на разных процессорных элементах. На современном этапе развития вычислительной техники такой способ, по-видимому, является основным, если не единственным, способом ускорения вычислений и достижения требуемой производительности.

Высокопроизводительные вычисления – всегда параллельные – реализуются на устройствах, которые принято называть суперкомпьютерами. Подобные компьютеры всегда правильно ассоциируются с чем-то большим: огромные размеры, большие задачи, крупные фирмы и компании, невероятные скорости работы и стоимость установки и обслуживания. К суперкомпьютерам относят лишь те компьютерные системы, которые имеют максимальную производительность в настоящее время.

Первоначально идея распараллеливания вычислительного процесса родилась в связи с необходимостью ускорить вычисления для решения сложных задач при использовании уже имеющейся элемент-

ной базы. Предполагалось, что вычислительные модули – процессоры или компьютеры – можно соединить между собой так, чтобы решение задач на полученной новой вычислительной системе ускорялось во столько раз, сколько использовано в ней вычислительных модулей. Однако, достаточно быстро стало ясно, что для интересующих сложных задач такое ускорение, как правило, достичь невозможно по двум причинам:

1) любая задача распараллеливается лишь частично – всегда имеются части, которые невозможно распараллелить;

2) коммуникационная среда, связывающая отдельные части параллельной системы, работает значительно медленнее процессоров, так что передача информации существенно задерживает вычисления.

Последние годы характеризуются скачкообразным прогрессом в развитии микроэлектроники, что ведет к постоянному совершенствованию вычислительной техники. Появилось большое количество вычислительных систем с разнообразной архитектурой, исследованы многие варианты их использования при решении возникающих задач. При этом в настоящее время все суперкомпьютеры являются параллельными системами.

При изучении высокопроизводительных вычислений следует помнить, что в этой сфере деятельности все начинается с задач, задачами определяется и по результатам решения задач оценивается. Об этом нередко забывают, но это именно та идея, которая должна составлять фундамент любого суперкомпьютерного проекта, иметь максимальный приоритет при принятии решений на каждом этапе эго реализации. Вместе с тем, задачи, для решения которых создаются суперкомпьютер, полностью определяют особенности их компоновки, проектирования, технологического цикла производства и эксплуатации.

В любом компьютере, в том числе и суперкомпьютере, все основные параметры тесно связаны. Трудно себе представить универсальный компьютер, имеющий высокое быстродействие и мизерную оперативную память, либо огромную оперативную память и неболь-

шой объем дисков. Суперкомпьютеры – это компьютерные системы, имеющие в настоящее время не только максимальную производительность, но и максимальный объем оперативной и дисковой памяти. При этом не стоит забывать о специализированном программном обеспечении, с помощью которого можно эффективно всем этим воспользоваться.

В качестве основной характеристики компьютеров для того, чтобы присвоить им префикс "супер-", используется такой показатель, как производительность – величина, показывающая, какое количество арифметических операций он может выполнить за единицу времени. При этом понятие суперкомпьютера не связано с какой-либо конкретной производительностью компьютера и носит исторический характер. Из истории высокопроизводительных вычислений видно как производительность, которая реализовывалась на суперкомпьютере, через 10 лет могла быть доступна на общедоступном персональном компьютере. В общем случае, можно говорить, что суперкомпьютер – это компьютер значительно более мощный, чем доступные для большинства пользователей компьютеры, а скорость технического прогресса сегодня такова, что нынешний лидер по производительности легко может стать через несколько лет обычной компьютерной системой, доступной простому пользователю.

Определений суперкомпьютерам пытались давать много, иногда серьезных, иногда ироничных. Из-за большой гибкости самого термина до сих пор распространены довольно нечёткие представления о понятии «суперкомпьютер». Шутливая классификация Гордона Белла и Дона Нельсона, разработанная приблизительно в 1989 году, предлагала считать суперкомпьютером любой компьютер, весящий более тонны. Современные суперкомпьютеры действительно весят несколько тонн, однако далеко не каждый тяжёлый компьютер достоин чести считаться суперкомпьютером. Кен Батчер в 1998 предложил такой вариант: суперкомпьютер – это устройство, сводящее проблему вычислений к проблеме ввода/вывода. Иными словами, что раньше долго вычислялось, временами сбрасывая нечто на диск, на суперкомпь-

ютере может выполниться мгновенно, переводя стрелки неэффективности на относительно медленные устройства ввода/вывода. Таким образом, подводя небольшой итог, можно утверждать, что определение понятия суперкомпьютер не раз было предметом многочисленных споров и дискуссий.

Авторство термина «суперкомпьютер» приписывается Джорджу Мишелю и Сиднею Фернбачу, в конце 60-х годов XX века работавшим в Ливерморской национальной лаборатории и компании CDC. Тем не менее, известен тот факт, что ещё в 1920 году газета New York World рассказывала о «супервычислениях», выполняемых при помощи табулятора IBM, собранного по заказу Колумбийского университета. В общеупотребительный лексикон термин «суперкомпьютер» вошёл благодаря распространённости компьютерных систем Сеймура Крея, таких как, CDC 6600, CDC 7600, Cray-1, -2, -3, -4. Сеймур Крей разрабатывал вычислительные машины, которые, по сути, становились основными вычислительными средствами правительственных, промышленных и академических научно-технических проектов США с середины 60-х годов до 1996 года. Не случайно в то время одним из популярных определений суперкомпьютера было следующее: «любой компьютер, который создал Сеймур Крей». Отметим, что сам Крей никогда не называл свои детища суперкомпьютерами, предпочитая использовать вместо этого обычное название «компьютер».

Суперкомпьютеры каждого типа создаются в небольшом количестве экземпляров, и обычно каждый тип суперкомпьютеров имеет определенные неповторимые архитектурные, технологические и вычислительные характеристики. В этой связи сравнение суперкомпьютеров весьма сложная задача, не имеющая однозначного решения. Тем не менее, разработаны определенные принципы условного сравнения компьютеров, что важно для их дальнейшего совершенствования и для продвижения на рынке. В соответствии с этими принципами суперкомпьютеры классифицируются в регулярно обновляемом списке Top500 (www.top500.org), о котором более подробно речь пойдет ниже. Этот список обновляется два раза в год и носит общепри-

знанный, достаточно объективный характер. В этой связи можно предложить свое собственное определение суперкомпьютера, которое позволит избежать различных двусмысленностей, и будет вполне объективно: «суперкомпьютер – это высокопроизводительная компьютерная система, входящая в список Top500».

1.2. ВАЖНОСТЬ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Простые расчеты показывают, что даже умеренные конфигурации суперкомпьютеров могут стоить несколько миллионов рублей. Поэтому следует четко прояснить, какие задачи настолько важны, что требуются компьютерные системы стоимостью несколько миллионов рублей, и какие задачи настолько сложны, что хорошего компьютера, который можно купить в магазине не достаточно.

Характерным и типичным примером сложной вычислительной задачи, ради решения которой следует затевать суперкомпьютерный проект и которая требует использования высокопроизводительных вычислений, является задача о компьютерном моделировании климата, в частности, задача о метеорологическом прогнозе. Климатическая задача включает в себя несколько блоков и подсистем, описывающих процессы и явления происходящие в географической оболочке Земли. Климатом называется ансамбль состояний, который система проходит за большой промежуток времени. Под климатической моделью подразумевается математическая модель, описывающая климатическую систему с той или иной степенью точности.

В основе климатической модели лежат уравнения сплошной среды и уравнения равновесной термодинамики. В модели описываются физические процессы, связанные с переносом энергии: перенос излучения в атмосфере, фазовые переходы воды, мелкомасштабная турбулентная диффузия тепла, диссипация кинетической энергии, образование облаков, конвекция и др. Рассматриваемая модель представляет собой систему нелинейных уравнений в частных производных в трехмерном пространстве. Решение этой системы уравнений

позволяет получить все главные характеристики ансамбля состояний климатической системы в искомые моменты времени.

Работая с климатической моделью, приходится принимать во внимание, что в отличие от многих других наук при исследовании климата нельзя поставить глобальный натурный эксперимент, а проведение численных экспериментов над моделями и сравнение результатов экспериментов с результатами наблюдений – единственная возможность изучения климата. Сложность моделирования заключается в том, что климатическая модель включает в себя ряд моделей, которые разработаны неодинаково глубоко. При этом лучше всего разработана модель атмосферы, поскольку наблюдения за ее состоянием ведутся давно и, следовательно, имеется много эмпирических данных. Следует отметить, что общая модель климата далека от завершения; поэтому в исследования включают обычно лишь моделирование состояния атмосферы и моделирование состояния океана.

В общую климатическую модель входит простейшая система уравнений, моделирующая погоду, которая решается в приземном сферическом слое. Система уравнений для погоды состоит из шести нелинейных скалярных уравнений в частных производных относительно шести неизвестных функций, зависящих от трех координат и времени, а именно, относительно компонент вектора скорости и давления, плотности и температуры. К этим уравнениям присоединяются начальные и граничные условия. Полученная система уравнений представляет собой математическую модель погоды и будет намного проще полной климатической модели.

Рассмотрим вычислительную сложность решения системы уравнений, моделирующих погоду. Предположим, что нас интересует развитие атмосферных процессов на протяжении 100 лет. При построении вычислительных алгоритмов используем принцип дискретизации: вся атмосфера разбивается на отдельные элементы (параллелепипеды) с помощью сетки с шагом 1° по широте и по долготе, а по высоте берут 40 слоев. Таким образом, получается $2{,}6 \cdot 10^6$ элементов. Каждый элемент описывается десятью компонентами. В фиксирован-

ный момент времени состояние атмосферы характеризуется ансамблем из $2{,}6 \cdot 10^7$ чисел. Условия развития процессов в атмосфере требуют каждые 10 минут находить новый ансамбль, так что за 100 лет будем иметь $5{,}3 \cdot 10^6$ ансамблей. Таким образом, в течение одного численного эксперимента получим около $1{,}4 \cdot 10^{14}$ числовых результатов. Если учесть, что для получения одного числового результата требуются около тысячи арифметических операций, то приходим к выводу, что для одного варианта вычисления модели состояния атмосферы на интервале 100 лет требуется затратить 10^{17} арифметических действий с плавающей точкой. Следовательно, вычислительная система с производительностью 10^{12} операций в секунду (1 Tflops – терафлопс) при полной загрузке и эффективной программе будет работать около 10^5 секунд, иначе говоря, потребуется от 3 до 30 часов вычислений. Ввиду отсутствия точной информации о начальных и краевых условиях, требуется просчитать сотни подобных вариантов. Еще раз подчеркнем, что расчет полной климатической модели займет на порядок больше времени.

Представленный пример показывает, что высокопроизводительные вычислительные системы необходимы для прогноза погоды, а также для прогнозирования климатических изменений. Нетрудно понять, что задачи прогноза землетрясений, цунами, извержений и других природных катаклизмов требуют решения не менее сложных математических задач. Еще сложнее задачи высоконадежных вычислений, связанных с исследованиями космоса, с экспериментами на субатомном уровне – постройка ускорителей элементарных частиц, ядерных реакторов, с испытанием и хранением ядерного оружия и др. Примеры использования суперкомпьютеров можно найти во многих отраслях. Вот лишь небольшой список областей человеческой деятельности, где использование суперкомпьютеров действительно необходимо:

- автомобилестроение;
- нефте- и газодобыча;
- фармакология;

- прогноз погоды и моделирование изменения климата;
- сейсморазведка;
- проектирование электронных устройств;
- синтез новых материалов;
- и многие, многие другие.

По данным Марка Миллера (Mark Miller, Ford Motor Company), для выполнения crash-тестов, при которых реальные автомобили разбиваются о бетонную стену с одновременным замером необходимых параметров, съемкой и последующей обработкой результатов, компании Форд понадобилось бы от 10 до 150 прототипов новых моделей при общих затратах от 4 до 60 миллионов долларов. Использование суперкомпьютеров позволило сократить число прототипов на одну треть.

Подобные примеры можно найти повсюду. В свое время исследователи фирмы DuPont искали замену хлорофлюорокарбону. Нужно было найти материал, имеющий те же положительные качества: невоспламеняемость, стойкость к коррозии и низкую токсичность, но без вредного воздействия на озоновый слой Земли. За одну неделю были проведены необходимые расчеты на суперкомпьютере с общими затратами около 5 тысяч долларов. По оценкам специалистов DuPont, использование традиционных экспериментальных методов исследований потребовало бы около трех месяцев и 50 тысяч долларов и это без учета времени, необходимого на синтез и очистку необходимого количества вещества.

1.3. МЕТОДЫ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

Вариантов ответа на вопрос о причинах высокой производительности суперкомпьютеров может быть несколько. Среди них два имеют явное преимущество: развитие элементной базы и использование новых решений в архитектуре компьютеров.

При выборе того, какой же из этих факторов оказывается решающим для достижения рекордной производительности можно обратиться к известным историческим фактам. На одном из первых компьютеров мира – EDSAC, появившемся в 1949 году в Кембридже

и имевшем время такта 2 микросекунды ($2 \cdot 10^{-6}$ секунды), можно было выполнить $2 \cdot n$ арифметических операций за $18 \cdot n$ миллисекунд, то есть в среднем 100 арифметических операций в секунду. При сравнении с одним вычислительным узлом компьютера Hewlett-Packard V2600, который в начале 21 века носил приставку супер-, видно, что время такта приблизительно 1,8 наносекунды, а пиковая производительность около 77 миллиардов арифметических операций в секунду (77 Gflops – гигафлопс). За полвека производительность компьютеров выросла более, чем в семьсот миллионов раз. При этом выигрыш в быстродействии, связанный с уменьшением времени такта с 2 микросекунд до 1,8 наносекунд, составляет лишь около 1000 раз. Весь остальной рост производительности достигнут за счет использования новых решений в архитектуре компьютеров.

Основное место среди новых решений в архитектуре компьютеров занимает принцип параллельной обработки данных, воплощающий идею одновременного (параллельного) выполнения нескольких действий. Достижение параллелизма возможно только при выполнимости определенных требований к архитектурным принципам построения вычислительной среды. Среди них отметим независимость функционирования отдельных устройств, а также избыточность элементов вычислительной системы. Избыточность включает использование специализированных устройств, таких как векторные процессоры, позволяющие производить арифметические операции за один такт не над отдельными числами, а над целыми векторами. При этом параллельная обработка данных, воплощая идею одновременного выполнения нескольких действий, имеет две разновидности: конвейерность и собственно параллельность.

Рассмотрим сначала понятие собственно параллельности. Если некое устройство выполняет одну операцию за единицу времени, то тысячу операций оно выполнит за тысячу единиц. Если предположить, что есть пять таких же независимых устройств, способных работать одновременно, то ту же тысячу операций система из пяти устройств может выполнить уже не за тысячу, а за двести единиц време-

ни. В идеальном случае можно предположить, что система из N устройств ту же работу выполнит за $1000/N$ единиц времени. К сожалению это только идеальный вариант. В реальный жизни очень много накладных расходов по времени приходится на коммуникации, и указанное ускорение недостижимо.

Рассмотрение конвейерной обработки начнем с анализа того, что необходимо сделать на процессорном элементе для сложения двух вещественных чисел, представленных в форме с плавающей запятой. Для этого требуется провести целое множество мелких операций таких, как сравнение порядков, выравнивание порядков, сложение мантисс, нормализация и т.п. Процессоры первых компьютеров выполняли все эти "микрооперации" для каждой пары аргументов последовательно одну за другой до тех пор, пока не доходили до окончательного результата, и лишь после этого переходили к обработке следующей пары слагаемых.

Идея конвейерной обработки заключается в выделении отдельных этапов выполнения общей операции. При этом каждый этап, выполнив свою работу, передает результат следующему, одновременно принимая новую порцию входных данных. Получаем очевидный выигрыш в скорости обработки за счет совмещения прежде разнесенных во времени операций. Предположим, что в операции можно выделить пять микроопераций, каждая из которых выполняется за одну единицу времени. Если есть одно неделимое последовательное устройство, то 100 пар аргументов оно обработает за 500 единиц. Если же каждую микрооперацию выделить в отдельный этап (или иначе говорят – ступень) конвейерного устройства, то на пятой единице времени на разной стадии обработки такого устройства будут находиться первые пять пар аргументов. Первый результат будет получен через 5 единиц времени, каждый следующий – через одну единицу после предыдущего, а весь набор из ста пар будет обработан за 5+99=104 единицы времени. Таким образом, будет получено ускорение по сравнению с последовательным устройством почти в пять раз (по числу ступеней конвейера).

Приблизительно также будет и в общем случае. Если конвейерное устройство содержит m ступеней, а каждая ступень срабатывает за одну единицу времени, то время обработки n независимых операций этим устройством составит $m+n-1$ единиц. Если это же устройство использовать в монопольном режиме (как последовательное), то время обработки будет равно $m \cdot n$. В результате получим ускорение почти в m раз за счет использования конвейерной обработки данных.

На первый взгляд, конвейерную обработку можно с успехом заменить обычным параллелизмом, для чего продублировать основное устройство столько раз, сколько ступеней конвейера предполагается выделить. Однако стоимость и сложность получившейся системы будут несопоставимы со стоимостью и сложностью конвейерного варианта, а производительность будет почти такой же.

1.4. ИСТОРИЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ СУПЕРКОМПЬЮТЕРОВ

Сегодня параллелизмом в архитектуре компьютеров уже мало кого удивишь. Все современные микропроцессоры используют тот или иной вид параллельной обработки. Например, в ядре современных процессоров Intel на разных стадиях выполнения может одновременно находиться до 126 микроопераций. На презентациях новых чипов это преподносится как последнее слово техники и передовой край науки, и это действительно так, если рассматривать реализацию этих принципов в миниатюрных рамках одного кристалла. Вместе с тем, сами эти идеи появились давно и развивались постепенно. Изначально новые идеи внедрялись в самых передовых компьютерах своего времени. Затем после должной отработки технологии и удешевления производства они спускались в компьютеры среднего класса, и, наконец, сегодня все это в полном объеме воплощается и в персональных компьютерах. Для того чтобы убедиться, что все основные нововведения в архитектуре современных суперкомпьютеров появлялись постепенно, рассмотрим ряд исторических примеров.

В проектах IBM 701 (1953 год) и IBM 704 (1955 год) появились разрядно-параллельная память, разрядно-параллельная арифметика.

Все самые первые компьютеры (EDSAC, EDVAC, UNIVAC) имели разрядно-последовательную память, из которой слова считывались последовательно бит за битом. Первым коммерчески доступным компьютером, использующим разрядно-параллельную память и разрядно-параллельную арифметику, стал IBM 701, а наибольшую популярность получила модель IBM 704 (продано 150 экз.), в которой, помимо сказанного, была впервые применена память на ферритовых сердечниках и аппаратное арифметическое устройство с плавающей точкой.

В проекте IBM 709 (1958 год) появились независимые процессоры ввода/вывода. Процессоры первых компьютеров сами управляли вводом/выводом. Однако скорость работы самого быстрого внешнего устройства, а по тем временам это магнитная лента, была в 1000 раз меньше скорости процессора, поэтому во время операций ввода/вывода процессор фактически простаивал. В 1958 г. к компьютеру IBM 704 присоединили 6 независимых процессоров ввода/вывода, которые после получения команд могли работать параллельно с основным процессором, а сам компьютер переименовали в IBM 709. Данная модель получилась удивительно удачной. Так, вместе с модификациями, было продано около 400 экземпляров, причем последний был выключен в 1975 году.

В проекте IBM STRETCH (1961 год) появился опережающий просмотр вперед и расслоение памяти. В 1956 году IBM подписывает контракт с Лос-Аламосской научной лабораторией на разработку компьютера STRETCH, имеющего две принципиально важные особенности: опережающий просмотр вперед для выборки команд и расслоение памяти на два банка для согласования низкой скорости выборки из памяти и скорости выполнения операций.

Впервые конвейерный принцип выполнения команд был использован в 1963 году на машине ATLAS, разработанной в Манчестерском университете. Выполнение команд разбито на 4 стадии: выборка команды, вычисление адреса операнда, выборка операнда и выполнение операции. Конвейеризация позволила уменьшить время вы-

полнения команд с 6 мкс до 1,6 мкс. Данный компьютер оказал огромное влияние, как на архитектуру ЭВМ, так и на программное обеспечение: в нем впервые использована мультипрограммная операционная система, основанная на использовании виртуальной памяти и системы прерываний.

В 1964 году в компьютере CDC 6600 появились независимые функциональные устройства. Фирма Control Data Corporation (CDC) при непосредственном участии одного из ее основателей, Сеймура Р. Крэя (Seymour R.Cray) выпускает компьютер CDC-6600 – первый компьютер, в котором использовалось несколько независимых функциональных устройств. Машина имела громадный успех на научном рынке, активно вытесняя машины фирмы IBM. Для сравнения с сегодняшним днем приведем некоторые параметры это исторически значимого компьютера:

- время такта 100 нс,
- производительность 2-3 млн. операций в секунду,
- оперативная память разбита на 32 банка по 4096 60-ти разрядных слов,
- цикл памяти 1 мкс,
- 10 независимых функциональных устройств.

В 1969 году проект CDC-6600 получил свое дальнейшее развитие в компьютере CDC 7600, где были применены конвейерные независимые функциональные устройства. CDC выпускает компьютер CDC-7600 с восемью независимыми конвейерными функциональными устройствами – сочетание параллельной и конвейерной обработки. Основные параметры CDC-7600 следующие:

- такт 27,5 нс,
- 10-15 млн. опер/сек.,
- 8 конвейерных функциональных устройств,
- 2-х уровневая память.

В знаменитом проекте ILLIAC IV (1974 год) впервые были использованы матричные процессоры. В проекте ILLIAC IV было предусмотрено 256 процессорных элементов – 4 квадранта по

64 процессорных элементов с возможностью реконфигурации в 2 квадранта по 128 процессорных элементов или 1 квадрант из 256 процессорных элементов. Среди характеристик отметим тактовую частоту 40 нс и производительность 1 Gflops. Работы были начаты в 1967 году, к концу 1971 изготовлена система из одного квадранта, в 1974 году она введена в эксплуатацию, доводка велась до 1975 года. Центральная часть состояла из устройства управления + матрица из 64 процессорных элементов. Устройство управления представляло собой простую ЭВМ с небольшой производительностью, управляющую матрицей процессорных элементов. Все процессорные элементы матрицы работали в синхронном режиме, выполняя в каждый момент времени одну и ту же команду, поступившую от устройства управления, но над своими данными. Процессорный элемент имел собственное арифметико-логическое устройство с полным набором команд, оперативная память – 2 Кслова по 64 разряда, цикл памяти 350 нс, каждый процессорный элемент имел непосредственный доступ только к своей оперативной памяти. Сеть пересылки данных представляла собой двумерный тор со сдвигом на единицу по границе по горизонтали. В результате по сравнению с проектом стоимость оказалась в 4 раза выше, сделан был лишь один квадрант, такт составил 80 нс, а реальная производительность достигла уровня 50 Mflops. Несмотря на это данный проект оказал огромное влияние на архитектуру последующих машин, построенных по схожему принципу.

И, наконец, упомянем знаменитый проект CRAY-1, в котором впервые были применены векторно-конвейерные процессоры. В 1972 году С. Крэй покидает CDC и основывает свою компанию Cray Research, которая в 1976 году выпускает первый векторно-конвейерный компьютер CRAY-1 со временем такта 12,5 нс, с 12 конвейерными функциональными устройствами, с оперативной памятью до 1 Мслова (слово – 64 разряда) – цикл памяти 50 нс. Компьютер имел пиковую производительность 160 миллионов операций в секунду (160 Mflops). Главным новшеством в CRAY-1 является введение векторных команд, работающих с целыми массивами независимых

данных и позволяющих эффективно использовать конвейерные функциональные устройства.

Контрольные вопросы для самопроверки

Какова тактовая частота Вашего компьютера? Сколько арифметических операций в среднем он делает за одну секунду?

По каким направлениям идет развитие высокопроизводительной вычислительной техники сегодня?

Какого рода ускорение происходит в конвейере? А при параллельной обработке?

Что такое мегагерц, наносекунда, микросекунда, миллисекунда, такт?

Поясните понятие суперкомпьютера.

Чем обусловлено важность внедрения суперкомпьютеров?

Приведите пример задачи, обязательно требующей применения высокопроизводительных вычислений.

Возможно ли увеличение производительности суперкомпьютера прямо пропорционально увеличению количества процессорных элементов? Поясните свой ответ.

2. АРХИТЕКТУРА СУПЕРКОМПЬЮТЕРОВ

2.1. КЛАССИФИКАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ФЛИННА

Одним из наиболее распространенных способов классификации ЭВМ является систематика Флинна (Flynn), в рамках которой основное внимание при анализе архитектуры вычислительных систем уделяется способам взаимодействия последовательностей (потоков) выполняемых команд и обрабатываемых данных. Схематично классификация Флинна показана на рис. 2.1. При этом количество потоков команд и количество данных для каждого потока является классификационным признаком.

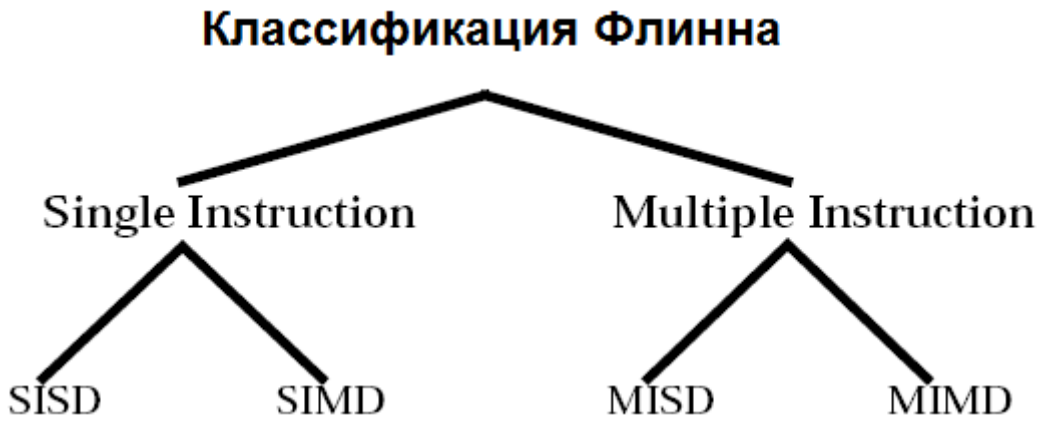

Рис. 2.1. Классификация Флинна

Рассмотрим каждый из видов вычислительных систем, получающихся по классификации Флинна, более подробно.

SISD (Single Instruction, Single Data) – системы, в которых существует одиночный поток команд и одиночный поток данных. К данному типу систем можно отнести обычные последовательные компьютеры.

SIMD (Single Instruction, Multiple Data) – системы с одиночным потоком команд и множественным потоком данных. SIMD компьютеры состоят из одного командного процессора (управляющего модуля), называемого контроллером, и нескольких модулей обработки данных, называемых процессорными элементами. В каждый момент времени может выполняться одна и та же команда для обработки нескольких информационных элементов. Управляющий модуль принимает, анализирует и выполняет команды. Если в команде встречаются данные, контроллер рассылает на все процессорные элементы команду, и эта команда выполняется на нескольких или на всех процессорных элементах. Каждый процессорный элемент имеет свою собственную память для хранения данных. Одним из преимуществ данной архитектуры считается то, что в этом случае более эффективно реализована логика вычислений. До половины логических инструкций обычного процессора связано с управлением выполнением машинных команд, а

остальная их часть относится к работе с внутренней памятью процессора и выполнению арифметических операций.

MISD (Multiple Instruction, Single Data) – системы, в которых существует множественный поток команд и одиночный поток данных. Вычислительных машин такого класса практически нет и трудно привести пример их успешной реализации. Один из немногих – систолический массив процессоров, в котором процессоры находятся в узлах регулярной решетки, роль ребер которой играют межпроцессорные соединения. Все процессорные элементы управляются общим тактовым генератором. В каждом цикле работы каждый процессорный элемент получает данные от своих соседей, выполняет одну команду и передает результат соседям.

MIMD (Multiple Instruction, Multiple Data) – системы с множественным потоком команд и множественным потоком данных. К подобному классу систем относится большинство параллельных многопроцессорных высокопроизводительных вычислительных систем, являющихся предметом рассмотрения настоящего учебного пособия.

Можно также классифицировать суперкомпьютеры в соответствии с тенденциями развития, выделяя при этом следующие четыре направления.

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, т.е. команда вида A = B + C может означать сложение двух массивов, а не двух чисел.

2. Массивно-параллельные компьютеры с распределенной памятью. Идея построения таких компьютеров состоит в соединении серийных микропроцессоров с локальной памятью посредством некоторой коммуникационной среды. Достоинств у такой архитектуры масса: если нужна высокая производительность, то можно добавить еще процессоров, если ограничены финансы или заранее известна требуе-

мая 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 этому же классу можно отнести и сети компьютеров, которые все чаще рассматривают как дешевую альтернативу крайне дорогим суперкомпьютерам.

3. Параллельные компьютеры с общей памятью. 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.

4. Последнее направление 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 не достаточно, то объединим несколько узлов высокоскоростными каналами. Подобную архитектуру называют кластерной. Именно это направление является в настоящее время наиболее перспективным для конструирования компьютеров с рекордными показателями производительности.

2.2. КЛАССИФИКАЦИЯ МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

В категории MIMD, согласно классификации Флинна, проводят еще одну классификацию, представленную на рис. 2.2 – на мультипроцессоры (один компьютер со многими процессорами) и мультикомпьютеры (вычисления проводятся на относительно независимых компьютерах).

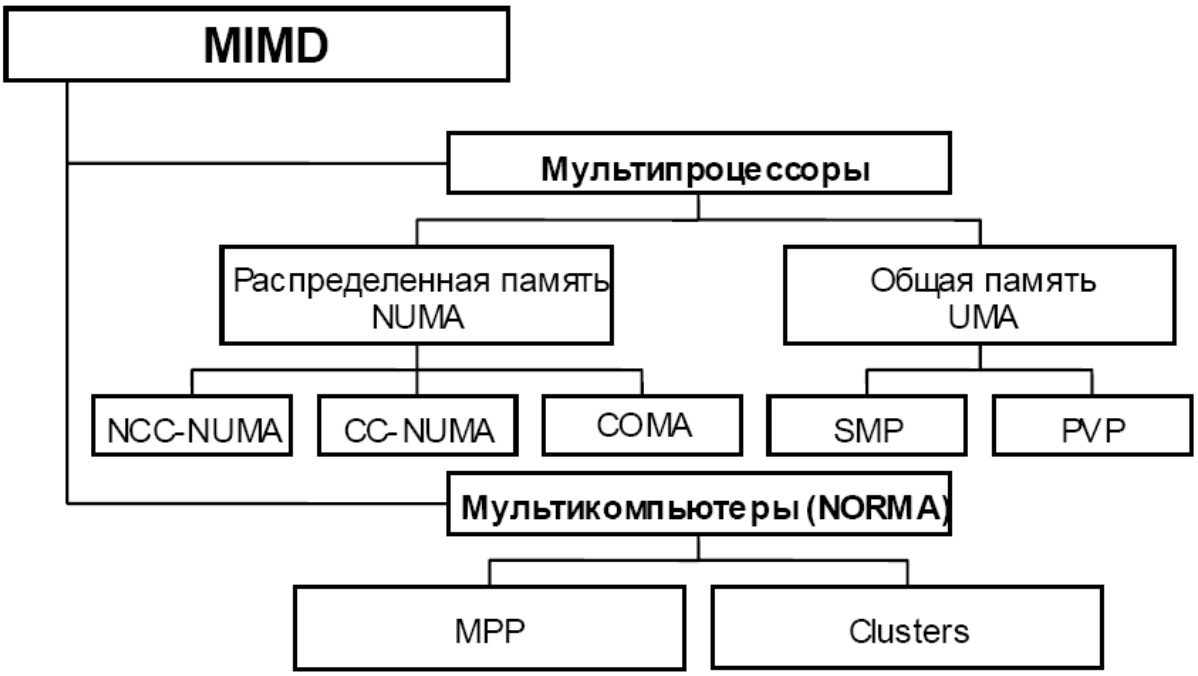

Рис. 2.2. Классификация MIMD систем

При классификации мультипроцессоров учитывается способ построения общей памяти. Возможный подход – использование единой (централизованной) общей памяти. Такой подход обеспечивает однородный доступ к памяти (uniform memory access – UMA) и служит основой для построения векторных параллельных процессоров (parallel vector processor – PVP) и симметричных мультипроцессоров (symmetric multiprocessor – SMP).

Одной из основных проблем, которые возникают при организации параллельных вычислений системах общей памятью, является доступ с разных процессоров к общим данным и обеспечение, в этой связи, однозначности (когерентности) содержимого разных кэшей (cache coherence problem). Дело в том, что при наличии общих данных копии значений одних и тех же переменных могут оказаться в кэш-памяти разных процессоров. Если в такой ситуации (при наличии копий общих данных) один из процессоров выполнит изменение значения разделяемой переменной, то значения копий в кэш-памяти других процессоров окажутся не соответствующими действительности и их использование приведет к некорректности вычислений. Обеспечение

однозначности кэшей обычно реализуется на аппаратном уровне – для этого после изменения значения общей переменной все копии этой переменной в кэш-памяти всех процессоров системы отмечаются как недействительные и последующий доступ к переменной потребует обязательного обращения к основной памяти. Следует отметить, что необходимость обеспечения когерентности приводит к некоторому снижению скорости вычислений и затрудняет создание систем с достаточно большим количеством процессоров.

Наличие общих данных при выполнении параллельных вычислений приводит к необходимости синхронизации взаимодействия одновременно выполняемых потоков команд. Так, например, если изменение общих данных требует для своего выполнения некоторой последовательности действий, то необходимо обеспечить взаимоисключение (mutual exclusion) с тем, чтобы эти изменения в любой момент времени мог выполнять только один командный поток. Задачи взаимоисключения и синхронизации относятся к числу классических проблем, и их рассмотрение при разработке параллельных программ является одним из основных вопросов параллельного программирования.

Общий доступ к данным может быть обеспечен и при физически распределенной памяти. При этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти. Такой подход именуется как неоднородный доступ к памяти (non-uniform memory access – NUMA). Среди систем с таким типом памяти выделяют три типа систем. Системы, в которых для представления данных используется только локальная кэш-память имеющихся процессоров (cache-only memory architecture – COMA). Системы, в которых обеспечивается когерентность локальной кэш-памяти разных процессоров (cache-coherent NUMA – CC-NUMA). И, наконец, системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэш-памяти (non-cache coherent NUMA – NCC-NUMA).

Использование распределенной общей памяти (distributed shared memory – DSM) упрощает проблемы создания мультипроцессоров. Известны примеры систем с несколькими тысячами процессоров. Однако, возникающие при этом проблемы эффективного использования распределенной памяти, например, время доступа к локальной и удаленной памяти может различаться на несколько порядков, приводят к существенному повышению сложности параллельного программирования.

Мультикомпьютеры – многопроцессорные системы с распределенной памятью уже не обеспечивают общий доступ ко всей имеющейся в системах памяти (no-remote memory access – NORMA). При всей схожести подобной архитектуры с системами с распределенной общей памятью, мультикомпьютеры имеют принципиальное отличие – каждый процессор системы может использовать только свою локальную память. При этом для доступа к данным, располагаемых на других процессорах, необходимо явно выполнить операции передачи сообщений (message passing operations). Данный подход используется при построении двух важных типов многопроцессорных вычислительных систем – массивно-параллельных систем (massively parallel processor – MPP) и кластеров (clusters). Это системы с распределенной памятью и с произвольной коммуникационной системой. При этом, как правило, каждый из процессорных элементов MPP системы является универсальным процессором, действующим по своей собственной программе.

Следует отметить чрезвычайно быстрое развитие многопроцессорных вычислительных систем кластерного типа. Под кластером обычно понимается множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (single system image), надежного функционирования (availability) и эффективного использования (performance). Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров, либо же сконструированы из типовых ком-

пьютерных элементов, что обычно не требует значительных финансовых затрат.

Применение кластеров может также в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (lowly parallel processing). Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (coarse granularity), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

2.3. СИММЕТРИЧНЫЕ МУЛЬТИПРОЦЕССОРНЫЕ СИСТЕМЫ

Характерной чертой симметричных многопроцессорных систем (SMP) является то, что все процессоры имеют прямой и равноправный доступ к любой точке общей памяти. Первые промышленные образцы мультипроцессорных систем появились на базе векторно-конвейерных компьютеров в середине 80-х годов 20 века. Наиболее распространенными многопроцессорными вычислительными системами такого типа были суперкомпьютеры фирмы Cray Research. Однако такие системы были чрезвычайно дорогими и производились небольшими сериями. Как правило, в подобных компьютерах объединялось от 2 до 16 процессоров, которые имели равноправный (симметричный) доступ к общей оперативной памяти. Именно в связи с этим они получили название симметричные мультипроцессорные системы (Symmetric Multi-Processing – SMP). Однако очень скоро обнаружи-

лось, что SMP-архитектура обладает весьма ограниченными возможностями по наращиванию числа процессоров в системе из-за резкого увеличения числа конфликтов при обращении к общей шине памяти.

Современные системы SMP-архитектуры состоят, как правило, из нескольких микропроцессоров и массива общей памяти, подключение к которой производится с помощью общей шины или коммутатора. Архитектура SMP-системы представлена на рис. 2.3.

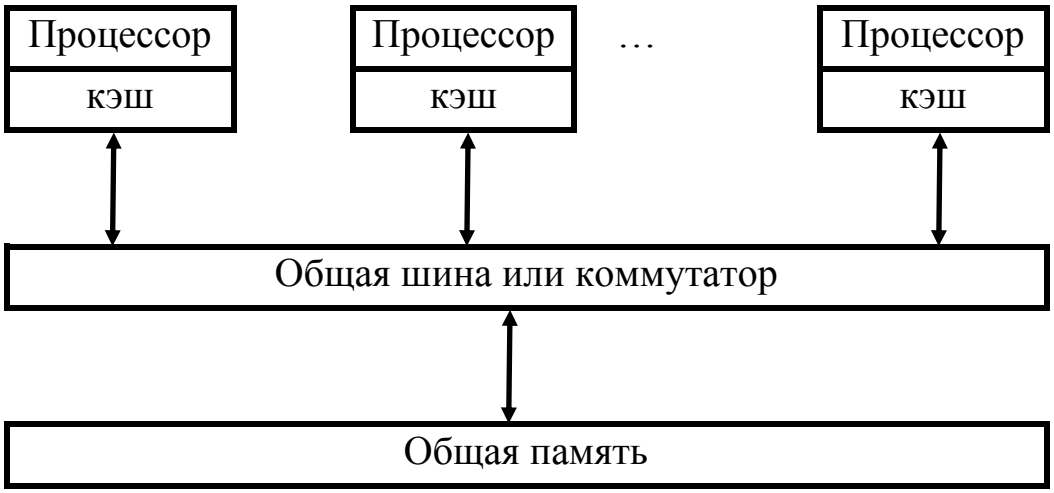

Рис. 2.3. Архитектура SMP-систем с общей памятью

Наличие общей памяти значительно упрощает организацию взаимодействия процессоров между собой и упрощает программирование, поскольку параллельная программа работает в едином адресном пространстве. Однако за этой кажущейся простотой скрываются большие проблемы, присущие системам этого типа. Все они, так или иначе, связаны с оперативной памятью. Дело в том, что в настоящее время даже в однопроцессорных системах самым узким местом является оперативная память, скорость работы которой значительно отстала от скорости работы процессора. Для того чтобы сгладить этот разрыв, современные процессоры снабжаются скоростной буферной памятью (кэш-памятью), скорость работы которой значительно выше, чем скорость работы основной памяти. При этом одной из основных проблем, которые возникают при организации параллельных вычислений на системах такого типа, является доступ с разных процессоров

к общим данным и обеспечение, в этой связи, однозначности (когерентности) содержимого разных кэшей (cache coherence problem). Подробно эта проблема была рассмотрена в предыдущем параграфе.

Помимо хорошо известной проблемы конфликтов при обращении к общей шине памяти возникла и новая проблема, связанная с иерархической структурой организации памяти современных компьютеров. В многопроцессорных системах, построенных на базе микропроцессоров со встроенной кэш-памятью, нарушается принцип равноправного доступа к любой точке памяти. Данные, находящиеся в кэш-памяти некоторого процессора, недоступны для других процессоров. Это означает, что после каждой модификации копии некоторой переменной, находящейся в кэш-памяти какого-либо процессора, необходимо производить синхронную модификацию самой этой переменной, расположенной в основной памяти.

С большим или меньшим успехом эти проблемы решаются в рамках общепринятой в настоящее время архитектуры ccNUMA (cache coherent Non Uniform Memory Access). В этой архитектуре память физически распределена, но логически общедоступна. Это, с одной стороны, позволяет работать с единым адресным пространством, а, с другой, увеличивает масштабируемость систем. Когерентность кэш-памяти поддерживается на аппаратном уровне, что не избавляет, однако, от накладных расходов на ее поддержание. В отличие от классических SMP-систем память становится трехуровневой:

- кэш-память процессора;
- локальная оперативная память;
- удаленная оперативная память.

Время обращения к различным уровням может отличаться на порядок, что сильно усложняет написание эффективных параллельных программ для таких систем.

Перечисленные обстоятельства значительно ограничивают возможности по наращиванию производительности ccNUMA систем путем простого увеличения числа процессоров. Тем не менее, эта технология позволяет в настоящее время создавать системы, содержащие

до 256 процессоров с общей производительностью порядка 200 миллиардов операций в секунду (200 Gflops). Системы этого типа серийно производятся многими компьютерными фирмами как многопроцессорные серверы с числом процессоров от 2 до 128 и прочно удерживают лидерство в классе малых суперкомпьютеров. Неприятным свойством SMP систем является то, что их стоимость растет быстрее, чем производительность при увеличении числа процессоров в системе. Кроме того, из-за задержек при обращении к общей памяти неизбежно взаимное торможение при параллельном выполнении даже независимых программ.

Преимущество такого типа архитектуры состоит в легкости программирования под нее, так как нет необходимости явно определять коммуникации между процессорами, поскольку все коммуникации определяются глобальной памятью. И, кроме того, как описано выше, в этом случае программист избавлен от забот об организации правильного доступа к общей памяти.

2.3. ВЕКТОРНО-КОНВЕЙЕРНЫЕ СУПЕРКОМПЬЮТЕРЫ

Исторически это были первые компьютеры, к которым в полной мере было применимо понятие суперкомпьютер. Первый векторно-конвейерный компьютер Cray-1 появился в 1976 году. Архитектура его оказалась настолько удачной, что он положил начало целому семейству компьютеров. Название этому семейству компьютеров дали два принципа, заложенные в архитектуре процессоров:

1. конвейерная организация обработки потока команд;
2. введение в систему команд набора векторных операций, которые позволяют оперировать с целыми массивами данных.

Длина одновременно обрабатываемых векторов в современных векторных компьютерах составляет, как правило, 128 или 256 элементов. Очевидно, что векторные процессоры должны иметь гораздо более сложную структуру и по сути дела содержать множество арифметических устройств. Основное назначение векторных операций состоит в распараллеливании выполнения операторов цикла, в которых

в основном и сосредоточена большая часть вычислительной работы. Для этого циклы подвергаются процедуре векторизации с тем, чтобы они могли реализовываться с использованием векторных команд. Как правило, это выполняется автоматически компиляторами при генерации исполнимого кода программы. Поэтому векторно-конвейерные компьютеры не требовали какой-то специальной технологии программирования, что и явилось решающим фактором в их успехе на компьютерном рынке. Тем не менее, требовалось соблюдение некоторых правил при написании циклов с тем, чтобы компилятор мог их эффективно векторизовать.

Как правило, несколько векторно-конвейерных процессоров (2-16) работают в режиме с общей памятью (SMP), образуя вычислительный узел, а несколько таких узлов объединяются с помощью коммутаторов, образуя либо NUMA, либо MPP систему. Уровень развития микроэлектронных технологий не позволяет в настоящее время производить однокристальные векторные процессоры, поэтому эти системы довольно громоздки и чрезвычайно дороги. В связи с этим, начиная с середины 90-х годов 20 века, когда появились достаточно мощные суперскалярные микропроцессоры, интерес к этому направлению был в значительной степени ослаблен. Суперкомпьютеры с векторно-конвейерной архитектурой стали проигрывать системам с массовым параллелизмом. Однако в марте 2002 г. корпорация NEC представила систему Earth Simulator из 5120 векторно-конвейерных процессоров, которая в 5 раз превысила производительность предыдущего обладателя рекорда – MPP системы ASCI White из 8192 суперскалярных микропроцессоров. При этом Earth Simulator занимал первое место в Top500 достаточно долго – с июня 2002 по ноябрь 2004. Это, конечно же, заставило многих по-новому взглянуть на перспективы векторно-конвейерных систем. Рассмотрим проект Earth Simulator более подробно.

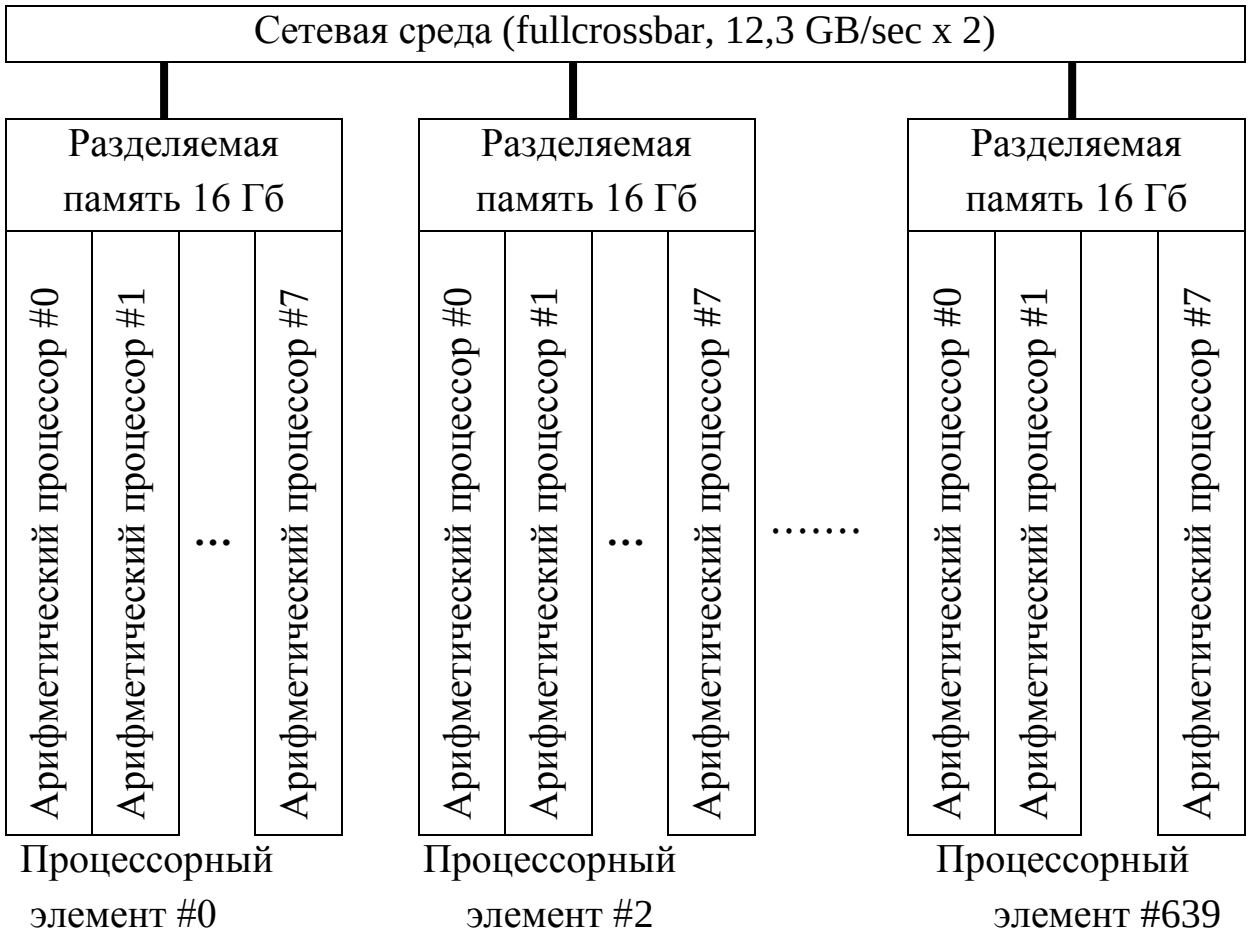

Рис. 2.4. Архитектура Earth Simulator

Разработка суперкомпьютера Earth Simulator началась в 1997 и в марте 2002 он был установлен на территории Института наук о Земле в Иокогаме (Yokohama Institute for Earth Sciences) в японском центре JAMSTEC (Japan Agency for Marine-Earth Science and Technology Center). Earth Simulator – это продолжение серии суперкомпьютеров от NEC на базе операционной системы UNIX. Earth Simulator, по сути, представляет собой вычислительный центр (Earth Simulator Research and Development Center – ESRDC). Архитектура Earth Simulator представлена на рис. 2.4 (640 процессорных узла содержат по 8 векторных арифметических процессоров (всего 640*8=5120 процессоров), работающих на частоте 500МГц).

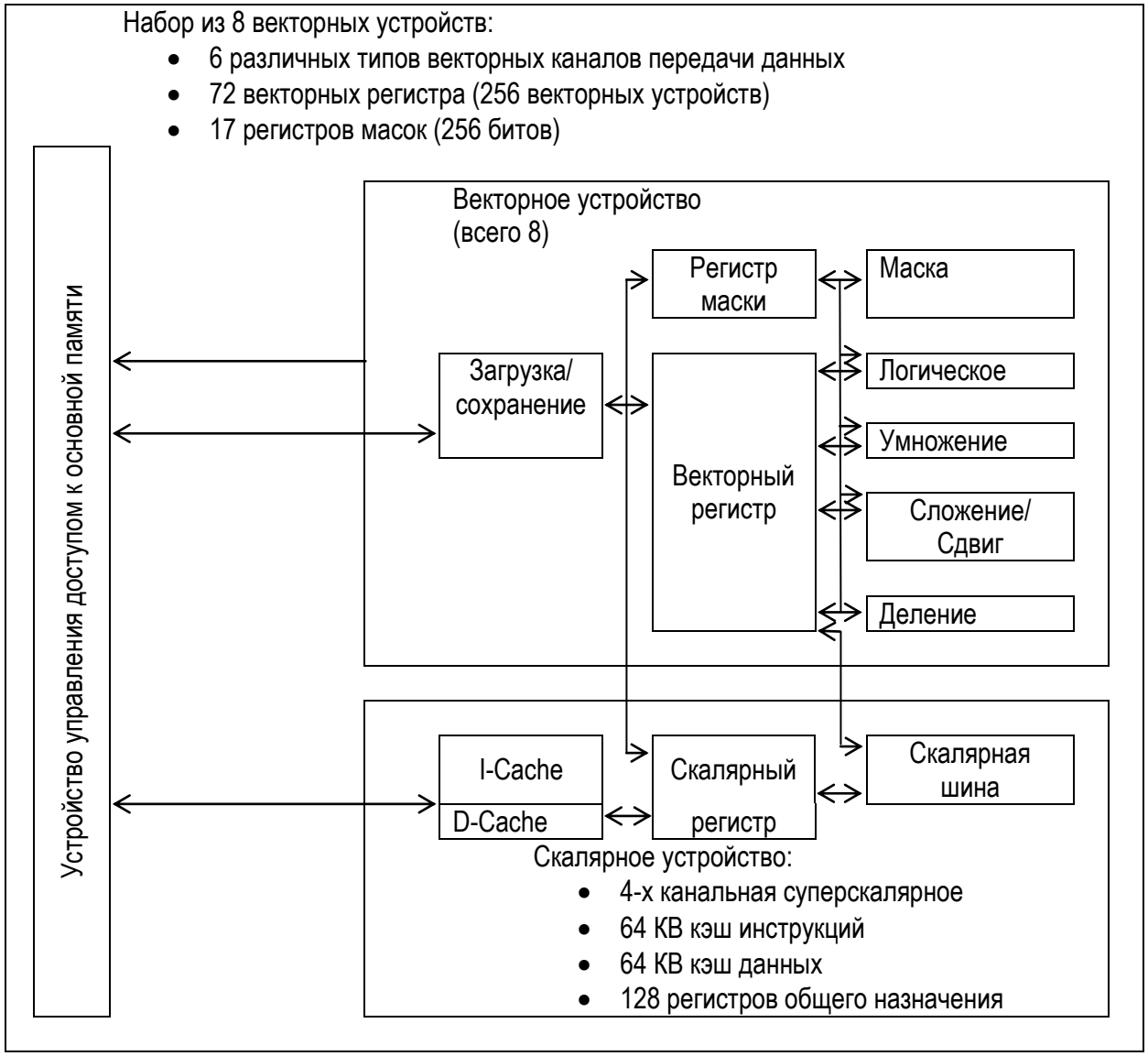

Рис. 2.5. Арифметический векторно-скалярный процессор Earth Simulator

Система архивирования Earth Simulator включает дисковые массивы на 250 Тбайт и ленточную библиотеку StorageTek 9310 на 1,5 Пбайт (1 Петабайт = 1015 байт). Архитектура компьютера объединяет многие известные принципы построения высокопроизводительных систем. В целом, Earth Simulator является массивно-параллельным компьютером с распределенной памятью. Вместе с тем, каждый процессорный узел построен на принципах SMP-

архитектуры, причем основа каждого процессора – векторно-конвейерная обработка, как показано на рис. 2.5.

В первую очередь, Earth Simulator был создан для построения виртуальной модели нашей планеты. Он решает различные задачи глобального (крупномасштабного) моделирования океана и атмосферы с достаточно большим разрешением (шаг сетки 10 км). Используется трехмерная гидростатическая модель, причем указанный шаг сетки – наибольший возможный из используемых сейчас. Ни в какой другой модели не используется столь мелкий шаг сетки и столь крупномасштабное моделирование. Суперкомпьютер способен обрабатывать всю информацию, поступающую со всевозможных "наблюдательных пунктов" – на земле, воде, в воздухе, космосе и т.д. На Earth Simulator можно строить и анализировать модель нашей планеты со всеми процессами: климатическими изменениями, тем же глобальным потеплением, землетрясениями, тектоническими сдвигами, атмосферными явлениями, загрязнением окружающей среды и тому подобным. При помощи Earth Simulator можно прогнозировать все эти глобальные явления и процессы.

Для визуализации модели внутреннего устройства земли была разработана специальная система визуализации BRAVE. Она представляет собой кабину, три стены и пол которой – это экраны размером 3 на 3 метра. На эти экраны проецируется изображение под различными углами, в результате чего для наблюдателя, находящегося внутри кабины, может быть сформирована трехмерная картинка. Пользователь может взаимодействовать с системой, например, увеличивать отдельные области и анализировать результаты в трехмерном виде.

2.4. КЛАСТЕРЫ

Кластер – это группа компьютеров, объединенных в локальную вычислительную сеть и способных работать в качестве единого вычислительного ресурса. Преимущества кластерной системы перед набором независимых компьютеров очевидны. Во-первых, разработано множество диспетчерских систем пакетной обработки заданий, по-

зволяющих послать задание на обработку кластеру в целом, а не какому-то отдельному компьютеру. Эти диспетчерские системы автоматически распределяют задания по свободным вычислительным узлам или буферизуют их при отсутствии таковых, что позволяет обеспечить более равномерную и эффективную загрузку компьютеров. Во-вторых, появляется возможность совместного использования вычислительных ресурсов нескольких компьютеров для решения одной задачи.

Кластерные технологии стали логическим продолжением развития идей, заложенных в архитектуре MPP-систем. Если процессорный модуль в MPP-системе представляет собой законченную вычислительную систему, то следующий шаг напрашивается сам собой: почему бы в качестве таких вычислительных узлов не использовать обычные серийно выпускаемые компьютеры. Развитие коммуникационных технологий, а именно, появление высокоскоростного сетевого оборудования и специального программного обеспечения, реализующего механизм передачи сообщений над стандартными сетевыми протоколами, сделали кластерные технологии общедоступными. Сегодня не составляет большого труда создать небольшую кластерную систему, объединив вычислительные мощности компьютеров отдельной лаборатории или учебного класса.

Привлекательной чертой кластерных технологий является то, что они позволяют для достижения необходимой производительности объединять в единые вычислительные системы компьютеры самого разного типа, начиная от персональных компьютеров и заканчивая мощными суперкомпьютерами. Широкое распространение кластерные технологии получили как средство создания систем суперкомпьютерного класса из составных частей (компьютеров, процессоров и проч.) массового производства, что значительно удешевляет стоимость вычислительной системы.

Как уже указывалось, производительность систем с распределенной памятью (мультикомпьютеры на рис. 2.2) очень сильно зависит от производительности коммуникационной среды. Коммуникаци-

онную среду можно достаточно полно охарактеризовать двумя параметрами: латентностью – временем задержки при посылке сообщения и пропускной способностью – скоростью передачи информации. Например, для компьютера Cray T3D эти параметры составляют соответственно 1 мкс и 480 Мб/сек, а для кластера, в котором в качестве коммуникационной среды использована сеть Fast Ethernet, 100 мкс и 10 Мб/сек. Это отчасти объясняет очень высокую стоимость суперкомпьютеров. При таких параметрах, как у указанного кластера, найдется не так много задач, которые могут эффективно решаться на достаточно большом числе процессоров.

Для создания кластеров часто используются либо простые однопроцессорные персональные компьютеры, либо двух- или четырехпроцессорные SMP-серверы. При этом не накладывается никаких ограничений на состав и архитектуру узлов. Каждый из узлов может функционировать под управлением своей собственной операционной системы. Чаще всего используются стандартные операционные системы из семейства Linux. В тех случаях, когда узлы кластера неоднородны, то говорят о гетерогенных кластерах. В качестве примера в табл. 2.1 сведены характеристики одного российского кластера, установленного еще в 2007 году.

При создании кластеров можно выделить два подхода. Первый подход применяется при создании небольших кластерных систем. В кластер объединяются полнофункциональные компьютеры, которые продолжают работать и как самостоятельные единицы, например, компьютеры учебного класса или рабочие станции лаборатории. Второй подход применяется в тех случаях, когда целенаправленно создается мощный вычислительный ресурс. Тогда системные блоки компьютеров компактно размещаются в специальных стойках, а для управления системой и для запуска задач выделяется один или несколько полнофункциональных компьютеров, называемых хост-компьютерами. В этом случае нет необходимости снабжать компьютеры вычислительных узлов графическими картами, мониторами,

дисковыми накопителями и другим периферийным оборудованием, что значительно удешевляет стоимость системы.

Таблица 2.1

Пример характеристик кластерной системы

Место установки	г. Томск, Томский ГУ
Условное название кластера	СКИФ Cyberia
Год установки кластера	2007
Количество узлов в кластере	283
Количество процессоров на узел	2
Тип процессоров	Intel Xeon 5150, 2,66 ГГц
Объём оперативной памяти на узел	4 ГБ
Тип коммуникационной сети	InfiniBand
Марка сетевых карт и коммутатора	QLogic InfiniPath, Silverstorm InfinI09240
Тип транспортной сети	Gigabit Ethernet
Тип управляющей сети	ServNet
Количество жёстких дисков на узел	1
Форм-фактор узлов	1U
Есть ли отдельный управляющий узел? Форм-фактор?	Да, 2U
Объём общего файлового хранилища	10 ТБ
Тип общей файловой системы	PanFS (Panasas)
Является ли файловое хранилище одновременно и головной машиной и вычислительным узлом?	Нет
Размеры и число стоек	14 стоек 42U, 19": 8 под вычислитель, 6 под систему бесперебойного электропитания
Используется ли UPS?	Да
Суммарная потребляемая мощность кластера	115 кВт
Тип системы кондиционирования	модульная внутрирядная
Общая мощность системы кондиционирования	96 кВт

Площадь помещения	72 кв.м.
Высота потолков в помещении	4 м
Используется ли помещение в иных целях?	Нет
Средства мониторинга состояния помещения	APC InfraStruXure
Средства мониторинга параметров кластера	Ganglia
Версия ОС	SuSe Linux Enterprise Server 10, Microsoft Windows Compute Cluster Server 2003
Версия параллельной среды	QLogic MPI
Версии компиляторов	gcc 4.1.0, Intel FORTRAN compiler 9.1, Intel C/C++ compiler 9.1
Дополнительное прикладное программное обеспечение	Intel MKL, Fluent 6.3, Gambit 2.3
Система управления очередями	Torque
Тип доступа и способ авторизации пользователей	ssh2
Производительность пиковая / HPL	12 / 9 Tflops

Кластерные вычисления представляют собой особую технологию высокопроизводительных вычислений, зародившуюся вместе с развитием коммуникационных средств и ставшую прекрасной альтернативной использованию суперкомпьютеров. Кластер предполагает более высокую надежность и эффективность, нежели локальная вычислительная сеть, и существенно более низкую стоимость в сравнении с другими типами параллельных вычислительных систем за счет использования типовых аппаратных и программных решений. В разряд кластерных вычислений, фактически, входят любые парал-

лельные вычисления, где все компьютеры системы используются как один унифицированный ресурс.

Существует несколько типов кластеров – кластеры высокой доступности, MPP-кластеры, высокопроизводительные кластеры, кластеры распределенной нагрузки. Появлению кластеров способствовал гигантский скачок в развитии аппаратной базы, появление и воцарение на рынке микропроцессоров и персональных компьютеров, накопление критической массы идей и методов параллельного программирования. Все это привело, в конечном счете, к решению извечной проблемы уникальности каждой параллельной вычислительной установки – разработке стандартов на создание параллельных программ для систем с общей и распределенной памятью. Добавим к этому непрерывное улучшение соотношения "цена/производительность" персональных компьютеров. В свете всех этих обстоятельств появление кластеров было неизбежным. Преимущества такого подхода к созданию вычислительных систем большой мощности, получившие признание практически сразу после первого представления первой кластерной высокопроизводительной системы, со временем только возрастали, поддерживаемые непрерывным ростом производительности типовых компонент.

Следует отметить, что кластеры относятся к мультикомпьютерам, согласно классификации на рис. 2.2, и имеют принципиальное отличительное свойство – каждый процессор системы может использовать только свою локальную память, в то время как для доступа к данным, располагаемых на других процессорах, необходимо явно выполнить операции передачи сообщений (message passing operations). Отметим также, что данный подход используется при построении двух важных типов многопроцессорных вычислительных систем – массивно-параллельных систем (massively parallel processor – MPP) и кластеров (clusters).

Кластеры могут быть образованы на базе уже существующих у пользователей отдельных компьютеров, либо же сконструированы из типовых компьютерных элементов, что обычно не требует значитель-

ных финансовых затрат. Применение кластеров может также в некоторой степени снизить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров. Это приводит к тому, что для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов, что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, что накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

2.5. МЕТАКОМПЮТИНГ И GRID-ТЕХНОЛОГИИ

Когда стали доступны каналы связи с высокой пропускной способностью, у компьютеров появилась возможность выполнять задачу совместно. На этой основе появилась концепция виртуального суперкомпьютера, где масштабная задача выполняется совместно в единой сети обычных компьютеров. Для внешнего наблюдателя (пользователя такой системы) создается полная иллюзия, что он работает с одним устройством.

Вычислительные узлы этой сети ведут скоординированную работу, используют ресурсы друг друга и потенциально доступны из любой точки системы. Компьютеры могут быть удалены друг от друга и могут использовать разные типы коммуникаций, однако для конечного программного продукта и пользователя они играют роль единой вычислительной машины. Все это привело к концепции метакомпьютинга в 90-е годы 20 века. Они были ответом на следующие вызовы суперкомпьютерных технологий:

- высокая цена;
- ограниченная масштабируемость;

- развитие элементной базы;
- нехватка мощности для некоторых задач.

Метакомпьютинг – способ соединения суперкомпьютеров, которые используются как единый вычислительный ресурс (коммуникации скрыты) и пользователь получает практически неограниченные вычислительные мощности. Центральное понятие метакомпьютера можно определить как метафору виртуального компьютера, динамически организующегося из географически распределенных ресурсов, соединенных высокоскоростными сетями передачи данных. Отдельные установки являются составными частями метакомпьютера и в то же время служат точками подключения пользователей. Необходимо подчеркнуть принципиальную разницу метакомпьютерного подхода и сегодняшних программных средств удаленного доступа. В метакомпьютере этот доступ прозрачен, то есть пользователь имеет полную иллюзию использования одного, но гораздо более мощного компьютерного устройства, чем то, которое стоит на его столе, и может с ним работать в рамках той же модели, которая принята на его персональном вычислителе.

В качестве примеров можно представить проекты начала 1990-х годов. Это пионерный проект CASA – проект гигабитной связи в США, который создавался с целью обеспечения вычислительными ресурсами набора приложений, требующих высокой производительности, таких как сбор, анализ, обработка данных из удаленных баз данных, а также прогноз погоды, создание лекарств, имитация систем.

Американский проект FAFNER (Factoring via Network-Enabled Recoursion) создавался для решения задачи разложения на множители для алгоритма шифрования RSA, для чего использовался алгоритм NFS (Number Field Sieve). Применялся сценарий CGI (интерфейс для обмена данными) и особый Web-клиент – просеивающий программный демон. Авторы проекта выделяли следующие особенности (преимущества) проекта:

- использование любой рабочей станции с объемом памяти не менее 4 Мб;

- поддержка анонимной регистрации;
- формирование иерархической сети web-серверов;
- минимизация потребности администрирования сети.

Для объединения ресурсов больших суперкомпьютерных центров США был внедрен метакомпьютерный проект I-WAY. Его особенностью было то, что объединялись высокопроизводительные компьютеры и использовались достаточно развитые средства визуализации. В этом проекте были объединены вычислительные ресурсы, расположенные в 17 городах, которые решали задачи связи, управления природными ресурсами, а также удаленного манипулирования данными.

Grid-технологии являются развитием и обобщением идей метакомпьютинга. В качестве процессорных мощностей рассматриваются не только суперкомпьютеры, а вообще любые компьютеры. Разделяемые ресурсы: коммуникации, данные, программное обеспечение, процессорное время. Grid – это согласованная, открытая и стандартизованная среда, которая обеспечивает гибкое, безопасное, скоординированное разделение (общий доступ) ресурсов в рамках виртуальной организации. Для Grid характерно отсутствие центра управления вычислительными ресурсами, использование открытых стандартов и нетривиальный уровень обслуживания.

Вычислительные узлы Grid обычно расположены далеко друг от друга, слабо связаны между собой через интернет-каналы, и доступность того или иного из них в произвольный момент времени не гарантирована. Это накладывает дополнительные требования на управление ресурсами. Структура Grid – это виртуальная организация, образованная над пространством реальных компьютеров, сетей, административных зон.

Задача виртуальной организации – создать среду, где части одного приложения, выполняемые на разных компьютерах, будут взаимодействовать между собой. Более того, подразумевается, что к системе можно динамически подключить произвольный новый ресурс – а значит, модель коммуникаций должна быть строго стандартизиро-

вана. При этом их коммуникации не должны зависеть от среды выполнения. Происходит ли вычисление в рамках одного компьютера, в локальной сети или в глобальной, объединяющей множество организаций, – это должно оставаться прозрачным для приложения.

Grid-системы могут быть классифицированы с точки зрения выделения вычислительных ресурсов следующим образом:

- добровольные,
- научные,
- коммерческие.

С точки зрения решаемых задач Grid-системы могут быть классифицированы следующим образом:

- вычислительные,
- для интенсивной обработки данных,
- семантические – для оперирования данными из различных баз данных.

Крупнейшим разработчиком Grid-технологий является Globus Alliance – сообщество разработчиков, которое распространяет и поддерживает Globus Toolkit – открытое программное обеспечение для построения Grid систем и приложений. Следует отметить, что стандарты Globus признают такие лидеры компьютерной индустрии, как IBM, Sun, Microsoft, Intel. В качестве применения Grid технологий отметим проект SETI (Search for Extraterrestrial Intelligence) – некоммерческий проект, использующий свободные ресурсы на компьютерах добровольцев, для поиска внеземного разума. Другой известный проект – расшифровка геномов, в том числе и человека.

Globus Toolkit представляет собой набор модулей для построения виртуальной организации распределенных вычислений. Каждый модуль определяет интерфейс, используемый высокоуровневыми компонентами, и имеет реализацию для различных сред выполнения. Вместе они образуют виртуальную машину Globus, в которой существуют следующие группы модулей:

- поиска и выделения ресурсов;
- коммуникаций;

- аутентификации;
- информационные;
- доступа к данным;
- создания процессов.

Важнейшая проблема, которую решают Grid технологии – разделение ресурсов. Разделение должно быть жестко контролируемо провайдерами ресурсов и потребителями, определяющими, что разделяется, кому и на каких условиях разрешено разделение. Виртуальная организация (virtual organization) – объединение отдельных специалистов, определенное вышеописанными правилами разделения.

Фундаментальным свойством GRID-систем является интероперабельность (interoperability) – способность к взаимодействию различных программных и аппаратных средств. В контексте сетевых технологий интероперабельность – общность протоколов, поэтому можно сказать, что GRID-архитектура – архитектура протоколов.

Один из идеологов Grid-систем Ян Фостер сформулировал следующие три критерия. Grid-система – это система, которая:

1. Координирует управление ресурсами при отсутствии централизованного управления этими ресурсами.
2. Использует стандартные, открытые, универсальные протоколы и интерфейсы.
3. Нетривиальным образом обеспечивает качество обслуживания.

Так, например Всемирная паутина (Web) – не является Grid системой (критерий 3), а файлообменные системы в локальной вычислительной сети с распределёнными одноранговыми объектами без централизованного управления (peer-2-peer) можно рассматривать как Grid-системы.

Кроме проекта Globus Toolkit, существуют и другие – UNICORE и gLite. Они во многом отличаются по реализации, все они успешно выполняют одну и ту же задачу – предоставляют пользователям полноценное управление распределёнными ресурсами. Систему Globus можно охарактеризовать как набор инструментов для разработки Grid

приложений на основе сервисов. Unicore ориентирован на однородный доступ к компьютерам. gLite предоставляет основу для создания распределенных приложений. Поэтому значимым этапом на пути развития Grid-приложений стала возможность объединить все три проекта. Стартовавший в 2006 году проект OMII-Europe в 2008 году анонсировал создание программных компонентов, поддерживающих совместное функционирование Globus, Unicore и gLite. Таким образом, у пользователей каждой из рассмотренных нами виртуальных платформ Grid появляется потенциальный доступ к сетям остальных платформ. Grid-вычисления становятся всё более интегрированными.

2.6 ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

С тематикой высокопроизводительных вычислений связаны технологии облачных вычислений. Облачные вычисления (cloud computing) – технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис. Характерной особенностью этой технологии является то, что вычисления производятся в готовой инфраструктуре с удаленным доступом. Используется технология распределённой обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис. В 1997 году один из идеологов этой технологии Рамнах Челлаппа (Ramnath K. Chellappa) выдвинул следующий тезис: «Облачные вычисления – это парадигма, при которой границы вычислений будут определяться не техническими ограничениями, а экономическими».

Облачная обработка данных – это парадигма, в рамках которой информация постоянно хранится на серверах в интернет и временно кэшируется на клиентской стороне, например, на персональных компьютерах, игровых приставках, ноутбуках, смартфонах и т. д. Характерными особенностями этой технологии являются:

- транзитивное получение ресурсов,
- не федеративное предоставление ресурсов,
- измеримость ресурсов,
- использование ресурсов в максимальном объеме,

- использование систем виртуализации,
- размытость, нечеткая граница,
- распределенность, удаленность, масштабируемость,
- отказоустойчивость и надежность хранения данных,
- наличие стандартизированного интерфейса,
- оплата только потребления ресурсов.

В этой технологии программное обеспечение предоставляется пользователю как интернет-сервис. Пользователь имеет доступ к собственным данным, но не может управлять операционной системой и собственно программным обеспечением, с которым работает. Заботиться об инфраструктуре ему также не нужно. Непосредственно "облаком" называют глобальную сеть, которая как раз и скрывает многие технические детали.

Существует несколько моделей облаков:

- SaaS (Software as a Service)
- PaaS (Platform as a Service)
- IaaS (Infrastructure as a Service)
- DaaS (Desktop as a Service)
- CaaS (Communications as a Service)

SaaS-приложения в виде сервисов – вариант, при котором пользователю предлагают использовать какое-то конкретное программное обеспечение, например, корпоративные системы, в виде сервиса по подписке. Скажем, у предприятия нет возможности или желания поддерживать внутренний сервер для работы почты, календарей и т.п. В этом случае предприятие может купить такую услугу удаленно, с учетом всей необходимой специфики.

PaaS – платформа как сервис – в отличие от SaaS, предназначенного больше для конечного пользователя, это вариант для разработчиков. В облаке функционирует некоторый набор программ, основных сервисов и библиотек, на основе которых предлагается разрабатывать свои приложения. Самый яркий пример – платформа для создания приложений Google AppEngine. Помимо этого, под PaaS по-

нимают также и отдельные части сложных систем, вроде системы базы данных или коммуникаций.

IaaS – инфраструктура как сервис – этот термин пришел на смену Hardware as a Service (HaaS), подняв его на новый уровень. HaaS – компьютерное оборудование как сервис – один из первых терминов, означающих предоставление некоторых базовых hardware-функций и ресурсов в виде сервисов. Но вместо прямой аренды хостинга используется виртуализация. Поэтому, когда речь идет о конкретном компьютерном оборудовании, понимаются некоторые абстрактные сущности, аналогичные реальным – место под хранение, процессорное время в эквиваленте какого-либо реального процессора, пропускная способность и т.д. В качестве примера нового уровня IaaS по сравнению с HaaS – это системы виртуализации, балансировщики нагрузки и тому подобные системы, лежащие в основе построения других систем.

DaaS – рабочее место как сервис. При предоставлении услуги DaaS клиенты получают полностью готовое к работе («под ключ») стандартизированное виртуальное рабочее место, которое каждый пользователь имеет возможность дополнительно настраивать под свои задачи. Таким образом, пользователь получает доступ не к отдельной программе, а к необходимому для полноценной работы программному комплексу.

CaaS – коммуникации как сервис – подразумевается, что в качестве сервисов предоставляются услуги связи; обычно это IP-телефония, почта и сервисы мгновенных сообщений.

Можно выделить следующие способы развертывания облаков:

- частные,
- публичные,
- партнерское.

Публичное облако предоставляется непосредственно специализированными публичными компьютерными компаниями. Например, в публичных облаках сервис SaaS используется для настройки и адаптация приложения, а также в тех случаях, когда пользователю требу-

ется пакет для разработки программного обеспечения. Партнерское или облако хостера – когда сервис-провайдеры строят облачную платформу на основе крупных провайдеров публичных облаков, а затем продают сервисы этой платформы другим клиентам. Наконец, бывают частные облака – построенные в рамках одного заказчика. Частные облака чаще всего связаны с аппаратной и программной инфраструктурой – по предоставлению сервисов IaaS, PaaS, SaaS. Они принадлежат какому-либо владельцу и используются для целей конкретной корпорации.

На рис. 2.6 представлены проекты, реализованные на основе концепции облачных вычислений. Чем выше уровень абстракции сервиса, тем дальше он находится от инфраструктуры провайдера. В табл. 2.2 представлено сравнение этих технологий по некоторым характеристикам.

В период с апреля по июль 2010 года аналитическое агентство Gartner опросило в 40 странах 1487 специалистов в области компьютерных технологий. Оказалось, что 39% из них уже выделили бюджеты на использование технологий Cloud Computing. Причем компании из Европы, Азии, Среднего Востока и Северной Америки тратят 40-50% «облачных» денег на сервисы внешних провайдеров. Директор по исследованиям Gartner Боб Игоу отметил и другой тренд – переход от затрат на традиционные IT-активы такие, как дата-центры, к активам, доступ к которым осуществляется через «облака». «Это плохие новости для технологических провайдеров и фирм, предоставляющих IT-услуги, которые не инвестируют и не собираются предоставлять эти новые сервисы, глядя на растущий спрос со стороны заказчиков»,– заключает господин Игоу.

В отчете компании Gartner указывается, что в 2010 году мировой доход рынка сервисов облачных вычислений вырастет на 16,6% по сравнению с предыдущим годом – до $68,3 млрд. Российские провайдеры «облачных» услуг также рапортуют о притоке клиентов, хотя отечественный бизнес оценил лишь услуги виртуального хостинга и аренды почтовых приложений. Другие облачные предложения оста-

ются невостребованными. Главным потребителем "облачных" вычислений в России, по мнению экспертов, должны стать небольшие компании. Аренда инфраструктуры и программных продуктов – способ при небольших затратах иметь те же технологии, что доступны лишь крупным корпорациям. Но предложение на рынке пока не достигло достаточной плотности.

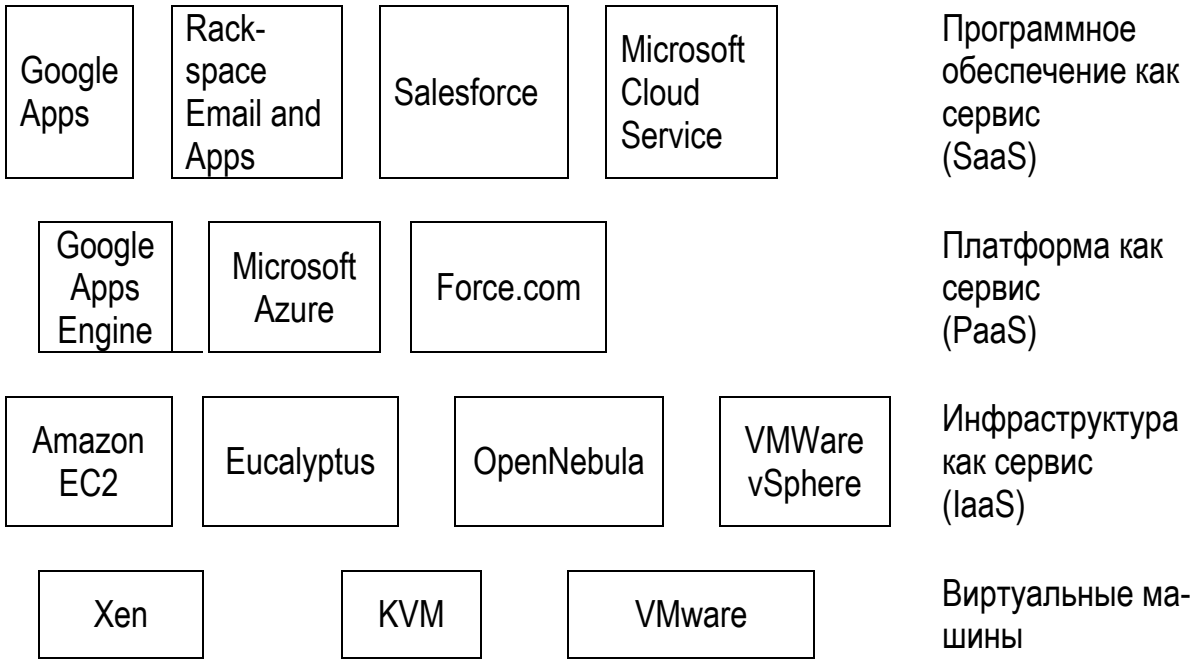

Рис. 2.6. Возможные варианты реализации вычислений в облаке

Следует отметить, что когда "софт как услугу" и "инфраструктуру как услугу" будут предлагать на каждом углу, малый бизнес обратит внимание на новый способ потребления IT-услуг. Компания Parallels (http://www.parallels.com) для России, СНГ и стран Балтии, приводит такую статистику. Емкость глобального рынка облачных услуг в целом составляет $25-30 млрд., из которых $10 млрд. приходится на услуги для малого бизнеса. Именно малый и средний бизнес является локомотивом "облачного" рынка. Это касается и нашей страны.

Таблица 2.2

Сравнение характеристик возможных вариантов реализации вычислений в облаке

Характеристика	Amazon Web Services	Google App Engine	Microsoft Azure	Force.com
1	2	3	4	5
Вычислительная архитектура	Elastic Compute Cloud (EC2) поддерживает загрузку образов виртуальных машин и предоставляет клиентские API для создания экземпляров и управления ими	Распределенная архитектура Google	Размещается в центрах обработки данных Microsoft, предоставляя операционную систему и сервисы разработки (могут быть использованы вместе или по отдельности)	"Коммунальная" (Multitenant) архитектура
Управление виртуализацией	Работа на уровне операционной системы в гипервизоре Xen	Приложения работают на экземплярах виртуальной машины Java или среды времени выполнения Python	Гипервизор (на основе Hyper-V)	Поддерживается инфраструктурой Force.com. встроенные регуляторы резервируют и распределяют ресурсы между пользователями
Сервис	IaaS	PaaS	PaaS	SaaS через среду разработки Force.com (можно рассматривать как PaaS)
Балансировка нагрузки	Гибкая балансировка нагрузки	Автоматическое масштабирование и балансировка нагрузки	Встроенная балансировка нагрузки	Встроенная балансировка нагрузки между участниками

Отказоустойчивость	Размещение экземпляров EC2 во множестве различных «зон доступности»	Автоматически перемещаются на несколько отказоустойчивых серверов, сервис Cron в App Engine	Экземпляры приложений размещаются в независимых отказоустойчивых доменах; все данные копируются трижды	Хранение данных оптимизировано для «коммунальной» архитектуры, кэши и распределение данных поддерживают отказоустойчивость и случаи резкого падения производительности
Интероперабельность	Опубликованные API для взаимодействия с экземплярами; размещаемые приложения могут быть написаны на стандартных языках программирования, работающих в виртуальных машинах	Размещаемые приложения, написанные на Java и Python, доступны через предоставляемые ими интерфейсы; доступ к интернет-ресурсам из кода и сервиса Fetch в App Engine	Размещаемые приложения работают в Windows и доступны через предоставляемые ими интерфейсы, приложения могут также использовать простой API для доступа к структуре Azure	Web Services API в Force.com предоставляет извне доступ к данным, хранимым в Force.com; приложения Web-логики могут быть представлены как стандартный Web-сервис; для сервисов REST предлагается HTTP API
Хранение	Simple Storage Service (S3), Amazon Elastic Block Storage (EBS), Amazon SimpleDB, Amazon Relation Database Service (RDS)	Хранилище данных App Engine (нереляционная, встроенная Bigtable); объекты со с-ми хранятся без схемы, поддер. транзакции	Постоянные данные, хранящ. в нереляционных блобах, таблицах и очередях; ресурсы хранения SQL предостав. SQL Azure	Постоянные данные хранятся в объектах, экземпляры объектов аналогичны таблице реляционной базы данных

Безопасность	SAS 70 Type II Certification, межсетевой экран, сертификат X.509. API для поддержки защиты SSL, список контроля доступа	SAS 70 Type II Certification, защищенный доступ к интранет через Secure Data Connector компании Google	SAS 70 Type II Certification, приложения работают в 64-разрядной MS-Windows Server 2008	SAS 70 Type II Certification, контроль доступа к данным на основе идентификационной информации пользователей и их ролей в организации
Платформа программирования	Amazon Machine Image (AMI), MapReduce	Java и Python, планируемые задачи и очереди, доступ к таким сервисам, как получение URL, почта, Memcache, работа с изображениями	.net и неуправляемый код, если он работает в Windows	Модель разработки на базе метаданных, платформа Visualforce для создания пользовательского интерфейса, язык программирования Apex

Сегодня, по подсчетам компании Parallels, примерно 85% облачного предложения для малого бизнеса формируется в России классическими хостерами, еще 10% – операторами связи и дистрибуторами программного обеспечения (например, Softcloud.ru) и порядка 5% приходится на независимых разработчиков, самостоятельно предоставляющих облачные сервисы. Анализ спроса среднего и малого бизнеса на «облачные» сервисы по типам, показывает, что наибольшей популярностью сейчас пользуется все, что относится к Web-позиционированию бизнеса и хостингу инфраструктуры (аренда выделенных и физических серверов). Далее следуют приложения для совместной работы (корпоративная почта, сервисы мгновенного обмена сообщениями и др.), и только в последнюю очередь – специализированные и индустриальные приложения, например, для поддержки процессов продаж, взаимодействия с клиентами и др.

Контрольные вопросы для самопроверки

В чем заключаются основные способы достижения параллелизма?

В чем могут состоять различия параллельных вычислительных систем?

Что положено в основу классификация Флинна?

В чем состоит принцип разделения многопроцессорных систем на мультипроцессоры и мультикомпьютеры?

Какие классы систем известны для мультипроцессоров?

В чем состоят положительные и отрицательные стороны симметричных мультипроцессоров?

Какие классы систем известны для мультикомпьютеров?

Что такое массивно-параллельный компьютер?

Что такое векторно-конвейерный компьютер?

Какие особенности архитектуры суперкомпьютера Earth Simulator?

В чем состоят положительные и отрицательные стороны кластерных систем?

В чем состоят особенности сетей передачи данных для кластеров?

Каковы причины появления концепции метакомпютинга?

Приведите примеры метакомпьютерных проектов. Каковы их особенности?

Каковы причины появления Grid проектов?

Сравните метакомпютинг и Grid технологии.

Представьте перспективы реализации высокопроизводительных вычислений на основе использования облачных вычислений.

3. ОЦЕНКА ВЫСОКОПРИЗВОДИТЕЛЬНЫХ СИСТЕМ

3.1. ЧИСЛЕННЫЙ ЭКСПЕРИМЕНТ И ПАРАЛЛЕЛЬНАЯ ФОРМА АЛГОРИТМА

Естественным способом решения проблем параллельного программирования для высокопроизводительных вычислительных систем стало создание стандартов как для вычислительной техники, так и для программного обеспечения. При этом требуется выделить класс задач, которые необходимо решать на высокопроизводительной системе. Оптимально, следующим шагом должен быть выбор или конструирование системы для выделенного класса задач. При этом следует помнить, что необходимо создать подходящее математическое обеспечение для конкретного класса задач на выбранной системе. И, наконец, написать программу для данной конкретной задачи с учетом перечисленных факторов.

Высокопроизводительная вычислительная система имеет большую стоимость, а ее создание и эксплуатация требуют обучения большого числа специалистов. Создание математического обеспечения и программ для такой системы – весьма трудоемкая задача. С другой стороны, ряд задач допускает постановку натурного эксперимента, что в ряде случаев быстрее приводит к цели, чем проведение численного эксперимента, хотя стоимость натурного эксперимента может оказаться очень большой.

Например, до запрещения испытаний ядерного оружия был возможен натурный эксперимент. После запрещения испытаний он стал невозможен, так что способы надежного хранения и совершенствования ядерного оружия определяются исключительно численным экспериментом, для проведения которого нужны мощнейшие компьютеры, соответствующие программные разработки, штат специалистов и т.д.

Существует множество областей, в которых невозможно или трудно проводить натурный эксперимент: экономика, экология, астрофизика, медицина; однако во всех этих областях часто возникают большие вычислительные задачи. В некоторых областях, таких как

аэродинамика, часто проводится дорогостоящий натурный эксперимент: "продувка" объектов (самолетов, ракет и т. п.) в аэродинамической трубе. В начале прошлого века «продувка» самолета братьев Райт стоила более 10 тысяч долларов, а «продувка» многоразового корабля «Шаттл» стоит 100 миллионов долларов. Однако, как оказалось, «продувка» не дает полной картины обтекания объекта, так как нельзя установить датчики во всех интересующих точках.

Для преодоления упомянутых выше трудностей приходится создавать математическую модель и проводить численный эксперимент, который обходится недешево, но все же значительно дешевле, чем натурный эксперимент. Типичная ситуация для задач, которые требуют высокопроизводительных вычислений состоит в том, что исследуемые объекты являются трехмерными, а для приемлемой точности приходится использовать сетку с одним миллионом узлов. При этом в каждом узле необходимо найти числовые значения от 5 до 20 функций, а при изучении нестационарного поведения объекта нужно определить его состояние в 10^2-10^4 моментах времени. На вычисление каждого значимого результата в среднем приходится 10^2-10^3 арифметических действий, и эти вычисления могут циклически повторяться для уточнения результата. Этапы численного эксперимента изображены на рис. 3.1. Следует также сделать замечание, что если хотя бы один из этапов выполняется неэффективно, то неэффективным будет и весь численный эксперимент и проводить его, по-видимому, нецелесообразно.

Для реализации алгоритма на параллельной системе его следует представить в виде последовательности групп операций. Для того, чтобы имелась возможность к распараллеливанию, отдельные операции в каждой группе должны обладать ключевым свойством, что их можно выполнять одновременно на имеющихся в системе функциональных устройствах. Пусть операции алгоритма разбиты на группы, а множество групп полностью упорядоченно так, что каждая операция любой группы зависит либо от начальных данных, либо от ре-

зультатов выполнения операций, находящихся в предыдущих группах.

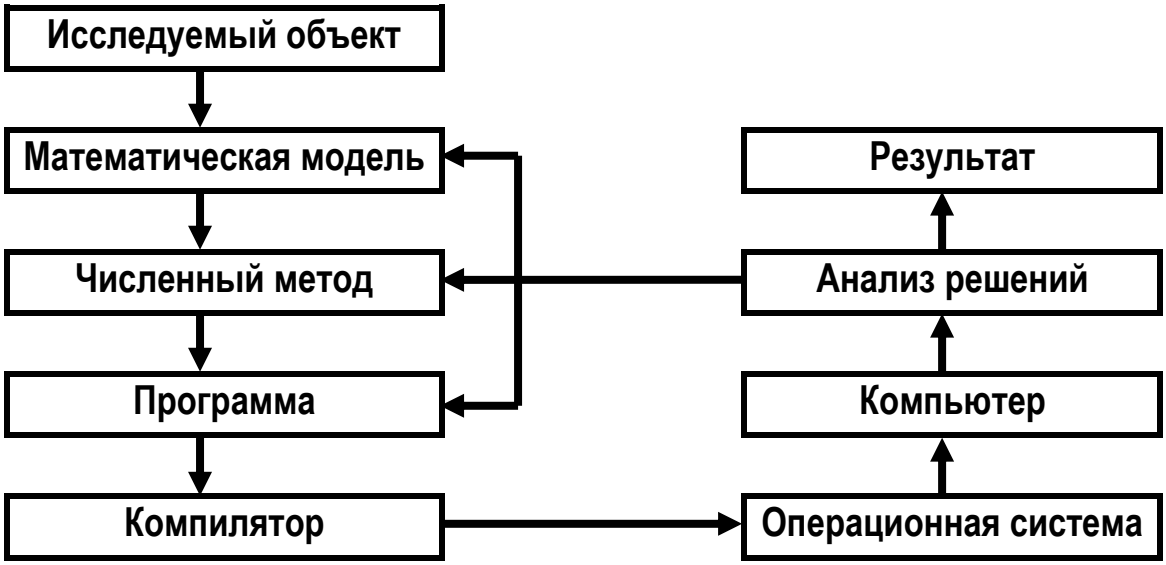

Рис. 3.1. Этапы численного эксперимента

Представление алгоритма в таком виде называется параллельной формой алгоритма. Каждая группа операций называется ярусом, а число таких ярусов – высотой параллельной формы. Максимальное число операций в ярусах (число привлекаемых процессов в ярусах) – шириной параллельной формы. Один и тот же алгоритм может иметь много параллельных форм. Формы минимальной высоты называются максимальными. Очевидно, что удобной формой представления таких алгоритмов является ориентированный граф. Именно такой подход позволяет строить для высокопроизводительных систем оценки, которые рассмотрены ниже.

3.2. СХЕМЫ ПАРАЛЛЕЛЬНОГО ВЫПОЛНЕНИЯ АЛГОРИТМА

При разработке параллельных алгоритмов решения сложных научно-технических задач принципиальным моментом является анализ эффективности использования параллелизма, состоящий обычно в оценке получаемого ускорения процесса вычислений – сокращения времени решения задачи. Формирование подобных оценок ускорения может осуществляться применительно к выбранному вычислитель-

ному алгоритму, т.е. происходит оценка эффективности распараллеливания конкретного алгоритма. Другой важный подход может состоять в построении оценок максимально возможного ускорения процесса решения задачи конкретного типа, т.е. оценка эффективности параллельного способа решения задачи. Модель вычислений в виде графа «операции-операнды» может использоваться для описания существующих информационных зависимостей в алгоритмах решения задач, что позволяет построить оценки эффективности максимально возможного параллелизма, которые могут быть получены в результате анализа имеющихся моделей вычислений.

Для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач традиционно используется модель в виде графа «операции-операнды». Далее будет предполагаться, что время выполнения любых вычислительных операций является одинаковым и равняется 1 (в тех или иных единицах измерения). Кроме того, принимается, что передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени.

Представим множество операций, выполняемых в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости в виде ациклического ориентированного графа $G=(V, R)$. Здесь $V=\{1,...,|V|\}$ – множество вершин графа, представляющих выполняемые операции алгоритма, а R – множество дуг графа, при этом дуга $r=(i, j)$ принадлежит графу только, если операция j использует результат выполнения операции i. Можно утверждать, что для выполнения выбранного алгоритма решения задачи могут быть использованы разные схемы вычислений и построены соответственно разные вычислительные модели. Как будет показано далее, разные схемы вычислений обладают различными возможностями для распараллеливания, и тем самым при построении модели вычислений может быть поставлена задача выбора наиболее подходящей для параллельного исполнения вычислительной схемы алгоритма. В рассматриваемой вычислительной модели алгоритма

вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг – для операций вывода. Обозначим через $\overline{V}$ множество вершин графа без вершин ввода, а через *d(G)* – диаметр (длину максимального пути) графа.

Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены параллельно. Возможный способ описания параллельного выполнения алгоритма может состоять в следующем. Пусть *p* есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество – расписание*:*

$$H_p = \{(i, P_i, t_i) : i \in V\},$$

в котором для каждой операции $i \in V$ указывается номер используемого для выполнения операции процессора P_i и время начала выполнения операции t_i.

Вычислительная схема алгоритма *G* совместно с расписанием H_p может рассматриваться как модель параллельного алгоритма $A_p(G, H_p)$, исполняемого с использованием процессоров. Время выполнения параллельного алгоритма определяется максимальным значением времени, используемым в расписании:

$$T_p(G, H_p) = \max_{i \in V}(t_i + 1).$$

Для выбранной схемы вычислений желательно использование расписания, обеспечивающего минимальное время исполнения алгоритма:

$$T_p(G) = \min_{H_p} T_p(G, H_p).$$

Уменьшение времени выполнения может быть обеспечено и путем подбора наилучшей вычислительной схемы:

$$T_p = \min_{G} T_p(G).$$

Оценки $T_p(G, H_p)$, $T_p(G)$ и T_p могут быть использованы в качестве показателей времени выполнения параллельного алгоритма. Кроме того, для анализа максимально возможного параллелизма можно определить оценку наиболее быстрого исполнения алгоритма:

$$T_\infty = \min_{p \geq 1} T_p .$$

Оценку T_∞ можно рассматривать как минимально возможное время выполнения параллельного алгоритма при использовании неограниченного количества процессоров (концепция вычислительной системы с бесконечным количеством процессоров, обычно называемой паракомпьютером, широко используется при теоретическом анализе параллельных вычислений).

Оценка T_1 определяет время выполнения алгоритма при использовании одного процессора и представляет, тем самым, время выполнения последовательного варианта алгоритма решения задачи. Построение подобной оценки является важной задачей при анализе параллельных алгоритмов, поскольку она необходима для определения эффекта использования параллелизма, т.е. ускорения времени решения задачи. Очевидно, что

$$T_1(G) = \left|\overline{V}\right|,$$

где $\left|\overline{V}\right|$, напомним, есть количество вершин вычислительной схемы G без вершин ввода. Важно отметить, что если при определении оценки T_1 ограничиться рассмотрением только одного выбранного алгоритма решения задачи и использовать величину

$$T_1 = \min_G T_1(G),$$

то получаемые при использовании такой оценки показатели ускорения будут характеризовать эффективность распараллеливания выбранного алгоритма. Для оценки эффективности параллельного решения исследуемой задачи вычислительной математики время последовательного решения следует определять с учетом различных последовательных алгоритмов, т.е. использовать величину

$$T_1^* = \min T_1,$$

где операция минимума берется по множеству всех возможных последовательных алгоритмов решения данной задачи.

3.3. ПОКАЗАТЕЛИ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА

Ускорение (speedup), получаемое при использовании параллельного алгоритма для p процессоров, по сравнению с последовательным вариантом выполнения вычислений определяется величиной:

$$S_p(n) = T_1(n) / T_p(n),$$

т.е. как отношение времени решения задач на скалярной ЭВМ к времени выполнения параллельного алгоритма (величина *n* используется для параметризации вычислительной сложности решаемой задачи и может пониматься, например, как количество входных данных задачи).

Эффективность (efficiency) использования параллельным алгоритмом процессоров при решении задачи определяется соотношением

$$E_p(n) = T_1(n) / \left(pT_p(n) \right) = S_p(n) / p.$$

Величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи.

Из приведенных соотношений можно показать, что в наилучшем случае $S_p(n) = p$ и $E_p(n) = 1$. При практическом применении данных показателей для оценки эффективности параллельных вычислений следует учитывать следующих два важных момента.

При определенных обстоятельствах ускорение может оказаться больше числа используемых процессоров $S_p(n) > p$ – в этом случае говорят о существовании сверхлинейного (superlinear) ускорения. Несмотря на парадоксальность таких ситуаций – ускорение превышает число процессоров – на практике сверхлинейное ускорение может иметь место.

Одной из причин такого явления как сверхлинейное ускорение может быть неравноправность выполнения последовательной и параллельной программ. Например, при решении задачи на одном процессоре оказывается недостаточно оперативной памяти для хранения

всех обрабатываемых данных и, как результат, необходимым становится использование более медленной внешней памяти. В случае же использования нескольких процессоров оперативной памяти может оказаться достаточно за счет разделения данных между процессорами. Еще одной причиной сверхлинейного ускорения может быть нелинейный характер зависимости сложности решения задачи в зависимости от объема обрабатываемых данных. Так, например, известный алгоритм пузырьковой сортировки характеризуется квадратичной зависимостью количества необходимых операций от числа упорядочиваемых данных. Как результат, при распределении сортируемого массива между процессорами может быть получено ускорение, превышающее число процессоров. Источником сверхлинейного ускорения может быть и различие вычислительных схем последовательного и параллельного методов;

При внимательном рассмотрении можно обратить внимание, что попытки повышения качества параллельных вычислений по одному из показателей (ускорению или эффективности) могут привести к ухудшению ситуации по другому показателю, ибо показатели качества параллельных вычислений являются противоречивыми. Так, например, повышение ускорения обычно может быть обеспечено за счет увеличения числа процессоров, что приводит, как правило, к падению эффективности. И, обратно, повышение эффективности достигается во многих случаях при уменьшении числа процессоров. В предельном случае идеальная эффективность $E_p(n) = 1$ легко обеспечивается при использовании одного процессора. Как результат, разработка методов параллельных вычислений часто предполагает выбор некоторого компромиссного варианта с учетом желаемых показателей ускорения и эффективности.

При выборе надлежащего параллельного способа решения задачи может оказаться полезной оценка стоимости (cost) вычислений, определяемая как произведение времени параллельного решения задачи и числа используемых процессоров: $C_p = pT_p$. В этой связи можно определить понятие стоимостно-оптимального (cost-optimal) па-

раллельного алгоритма как метода, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

3.4. ОЦЕНКА ДОСТИЖИМОГО ПАРАЛЛЕЛИЗМА. ЗАКОН АМДАЛА

Оценка качества параллельных вычислений предполагает знание наилучших (максимально достижимых) значений показателей ускорения и эффективности, однако, получение идеальных величин $S_p = p$ для ускорения и $E_p = 1$ для эффективности может быть обеспечено не для всех вычислительно трудоемких задач. Так, для рассматриваемого учебного примера в предыдущем пункте минимально достижимое время параллельного вычисления суммы числовых значений составляет $log_2 n$. Определенное содействие в решении данной проблемы могут оказать теоретические утверждения, приведенные в начале данного раздела. В дополнение к ним рассмотрим еще ряд закономерностей, которые могут быть чрезвычайно полезны при построении оценок максимально достижимого параллелизма.

Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены. Пусть f есть доля последовательных вычислений в применяемом алгоритме обработки данных, тогда в соответствии с законом Амдала (Amdahl) ускорение процесса вычислений при использовании p процессоров ограничивается величиной:

$$S_p \le \frac{1}{f + (1-f)/p} \le S^* = \frac{1}{f}.$$

Так, например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10-кратного ускорения обработки данных. Отсюда можно сделать вывод, что не любая программа может быть эффективно распараллелена. Для того чтобы это было возможно, необходимо, чтобы доля информационно независимых операций была очень большой. В принципе, это не должно отпугивать от параллель-

ного программирования, потому что, как показывает практика, большинство вычислительных алгоритмов устроено в этом смысле достаточно хорошим образом.

Предположим теперь, что в программе относительно немного последовательных операций. Казалось бы, в данном случае все проблемы удалось разрешить. Но представьте, что доступные вам процессоры разнородны по своей производительности. Значит, будет такой момент, когда кто-то из них еще трудится, а кто-то уже все сделал и бесполезно простаивает в ожидании. Если разброс в производительности процессоров большой, то и эффективность всей системы при равномерной загрузке будет крайне низкой.

Но предположим, что все процессоры одинаковы. Процессоры выполнили свою работу, но результатами чаще всего надо обмениваться для продолжения вычислений, а на передачу данных уходит время, и в это время процессоры опять простаивают. Кроме указанных, есть и еще большое количество факторов, влияющих на эффективность выполнения параллельных программ, причем все они действуют одновременно, а значит, все в той или иной степени должны учитываться при распараллеливании.

Таким образом, заставить параллельную вычислительную систему или суперкомпьютер работать с максимальной эффективностью на конкретной программе – это задача не из простых, поскольку необходимо тщательное согласование структуры программ и алгоритмов с особенностями архитектуры параллельных вычислительных систем.

Закон Амдала характеризует одну из самых серьезных проблем в области параллельного программирования (алгоритмов без определенной доли последовательных команд практически не существует). Следует отметить также, что рассмотрение закона Амдала происходит при предположении, что доля последовательных расчетов f является постоянной величиной и не зависит от параметра n, определяющего вычислительную сложность решаемой задачи. Однако для большого ряда задач доля $f = f(n)$ является убывающей функцией от n, и в этом

случае ускорение для фиксированного числа процессоров может быть увеличено за счет увеличения вычислительной сложности решаемой задачи. Данное замечание может быть сформулировано как утверждение, что ускорение $S_p = S_p(n)$ является возрастающей функцией от параметра n. Это утверждение часто именуется как эффект Амдала.

Оценим максимально достижимое ускорение, исходя из имеющейся доли последовательных расчетов в выполняемых параллельных вычислениях:

$$g = \frac{\tau(n)}{\tau(n) + \pi(n)/p},$$

где $\tau(n)$ и $\pi(n)$ есть времена последовательной и параллельной частей выполняемых вычислений соответственно, т.е.

$$T_1 = \tau(n) + \pi(n), T_p = \tau(n) + \pi(n)/p.$$

С учетом введенной величины g можно получить

$$\tau(n) = g \cdot (\tau(n) + \pi(n)/p),\ \pi(n) = (1-g)p \cdot (\tau(n) + \pi(n)/p),$$

что позволяет построить оценку для ускорения:

$$S_p = \frac{T_1}{T_p} = \frac{\tau(n) + \pi(n)}{\tau(n) + \pi(n)/p} = \frac{(\tau(n) + \pi(n)/p)(g + (1-g)p)}{\tau(n) + \pi(n)/p},$$

которая после упрощения приводится к виду закона Густавсона–Барсиса (Gustafson–Barsis's law):

$$S_p = g + (1-g)p = p + (1-p)g.$$

При рассмотрении закона Густавсона–Барсиса следует учитывать еще один важный момент. При увеличении числа используемых процессоров темп уменьшения времени параллельного решения задач может падать после превышения определенного порога. Однако при этом за счет уменьшения времени вычислений сложность решаемых задач может быть увеличена. Оценку получаемого при этом ускорения можно определить при помощи сформулированных закономерностей. Такая аналитическая оценка тем более полезна, поскольку решение таких более сложных вариантов задач на одном процессоре может оказаться достаточно трудоемким и даже невозможным, на-

пример, в силу нехватки оперативной памяти. С учетом указанных обстоятельств оценку ускорения, получаемую в соответствии с законом Густавсона–Барсиса, еще называют ускорением масштабирования (scaled speedup), поскольку данная характеристика может показать, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.

3.5. ТЕСТ LINPACK

Для сравнения высокопроизводительных систем используется тест Linpack. Он рассматривается как некоторое достаточно универсальное средство для измерения производительности компьютера в операциях с плавающей точкой. Его авторство принадлежит Джеку Донгарра, Пётру Лужчеку и Антуану Перитэ. Тест представляет из себя компьютерную программу, которая решает плотную систему линейных алгебраических уравнений. Решение основано на LU-разложении, матрица генерируется с помощью псевдо-случайного генератора.

С годами, естественно, какие-то внутренние аспекты теста претерпевают изменения, поэтому на данный момент тест в действительности включает в себя результаты трёх независимых тестов. Исторически Linpack представлял из себя библиотеку для решения задач линейной алгебры, и естественным образом не являлся самым эффективным способом решения системы линейных алгебраических уравнений, так как для каждого типа систем линейных алгебраических уравнений какой-то метод может оказаться выигрышнее, если не в плане производительности, так в плане точности. Он входит в состав более крупной библиотеки Linpack, которая, помимо решения системы линейных алгебраических уравнений, предоставляла возможность поиска собственных чисел и векторов, метод наименьших квадратов и выделение особенных векторов для вырожденных систем линейных алгебраических уравнений. Подробное описание теста с комментариями и примерами его использования можно найти по ссылке: http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpl.pdf.

Для того чтобы измерить производительность кластерной системы на тесте High Performance Linpack (HPL), потребуется сам тест и библиотека BLAS. Собственно тест HPL доступен по адресу http://www.netlib.org/benchmark/hpl/hpl.tgz. При этом для высоких значений производительности важно подобрать размер задачи так, чтобы использовалась вся оперативная память узлов.

В тесте используется понятие flops (флопс). Оно обозначает количество операций с плавающей точкой в секунду (сокращение от FLOating Point operations per Second). Каждый раз, когда этот термин будет использоваться, будет подразумеваться операция с 64-битным числом с плавающей точкой, а операция – либо сложение, либо умножение. Также используются понятия:

1 Mflops (мегафлопс) = 10^6 (миллион) оп/сек

1 Gflops (гигафлопс) = 10^9 (миллиард) оп/сек

1 Tflops (терафлопс) = 10^{12} (триллион) оп/сек

1 Pflops (пентафлопс) = 10^{15} (квадриллион) оп/сек

Далее, существует понятие «теоретической пиковой» производительности. Она никак не связана с реальной производительностью в приложениях или тестах, а лишь показывает максимально-достижимую (в реальности, практически никогда недостижимую) производительность – как скорость света в физике. Оно означает лишь то, что если бы каждый цикл процессора производил операцию с плавающей точкой, то какая была бы производительность.

Первый тест в Linpack измеряет производительность двух процедур: DGEFA(SGEFA) и DGESL(SGESL) для 64- и 32-битных версий, соответственно. DGEFA выполняет LU-разложение с выбором ведущего элемента по столбцу, DGESL использует данное разложение для решения системы линейных алгебраических уравнений. Большая часть времени требуется на разложение матрицы – DGEFA – *O(N3).* Как только оно завершено, находится решение – DGESL – *O(N2).*

Второй тест работает с матрицей размерности 1000. Третий тест создан для проверки производительности «хорошо» распараллелен-

ных вычислений, и именно по его результату строится рейтинг Top500. Этот тест пытается вычислить максимально-достижимую производительность системы при решении системы линейных алгебраических уравнений и представляет из себя целый набор модулей, для которого пользователем предоставляется реализация недостающих интерфейсов. Размерность задачи и метод решения можно варьировать, чтобы добиться максимальной производительности на данном тесте. Ограничения на решение те же, что и в предыдущем тесте. Так же запрещено пользоваться методом Штрассена умножения матриц, который хоть и даёт лучшую временную асимптотическую оценку, однако обладает в десятки раз большей численной погрешностью, которая ещё и растёт с ростом размерности примерно в 1,5-2 раза быстрее роста аналогичной погрешности привычного алгоритма умножения матриц. Ниже представлен в виде распечатки результат запуска теста Linpack для суперкомпьютера СКИФ, установленного в Московском государственном университете:

```
================================================
HPLinpack 1.0a  --  High-Performance Linpack benchmark  --  January 20, 2004
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Labs.,  UTK
================================================
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.
The following parameter values will be used:
N      :  740000
NB     :     168
PMAP   : Row-major process mapping
P      :      25
Q      :     200
PFACT  :   Right
NBMIN  :       4
NDIV   :       3
RFACT  :   Right
```

```
BCAST  :  1ringM
DEPTH  :       0
SWAP   : Mix (threshold = 256)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
--------------------------------------------------------------------------------
- The matrix A is randomly generated for each test.
- The following scaled residual checks will be computed:
   1) ||Ax-b||_oo / ( eps * ||A||_1  * N        )
   2) ||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  )
   3) ||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo )
- The relative machine precision (eps) is taken to be           2.220446e-16
- Computational tests pass if scaled residuals are less than           16.0
================================================================================
T/V              N    NB     P     Q               Time             Gflops
--------------------------------------------------------------------------------
WR01R3R4      740000   168    25   200            5709.59          4.732e+04
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        0.0003600 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0005678 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0000950 ..... PASSED
================================================================================
Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------
End of Tests.
```

Итак, тест Linpack используется для формирования списка Top500 – пятисот самых мощных вычислительных систем мира. При этом указывается Rpeak – теоретически достижимая (пиковая) производительность и Rmax – равная производительности системы на тесте Linpack. Поэтому к этим показателям и нужно относиться соответственно – они говорят о том, насколько хорошо система может решать системы линейных алгебраических уравнений с плотной матрицей указанным методом. Для других же задач могут быть совсем другие результаты!

Контрольные вопросы для самопроверки

Каковы этапы численного эксперимента?

Как можно определить требуемую производительность для решения конкретной задачи?

Как определяется модель "операция – операнды"?

Как определяется расписание для распределения вычислений между процессорами?

Как определяется время выполнения параллельного алгоритма?

Какое расписание является оптимальным?

Как определить минимально возможное время решения задачи?

Что понимается под паракомпьютером и для чего может оказаться полезным данное понятие?

Какие оценки следует использовать в качестве характеристики времени последовательного решения задачи?

Как определить минимально возможное время параллельного решения задачи по графу "операнды – операции"?

Какие зависимости могут быть получены для времени параллельного решения задачи при увеличении или уменьшения числа используемых процессоров?

Как определяются понятия ускорения и эффективности?

Возможно ли достижение сверхлинейного ускорения?

В чем состоит противоречивость показателей ускорения и эффективности?

Как определяется понятие стоимости вычислений?

Как формулируется закон Амдала? Какой аспект параллельных вычислений позволяет учесть данный закон?

Как оценивается производительность суперкомпьютеров на тесте Linpack?

4. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

4.1. РЕЙТИНГ СУПЕРКОМПЬЮТЕРОВ TOP500

Как уже отмечалось выше, суперкомпьютеры каждого типа создаются в небольшом количестве экземпляров. При этом каждый тип суперкомпьютеров имеет определенные неповторимые архитектурные, технологические и вычислительные характеристики. Поэтому сравнение суперкомпьютеров весьма сложная задача, не имеющая однозначного решения. Тем не менее, разработаны определенные принципы условного сравнения компьютеров, что важно для их дальнейшего совершенствования и для продвижения на рынке. В соответствии с этими принципами суперкомпьютеры классифицируются в списке Top500, который размещен в Интернете по адресу www.top500.org. Этот список содержит описание 500 реализаций суперкомпьютеров, расположенных в порядке убывания мощности. В списке указывается порядковый номер суперкомпьютера, организация, где он установлен, его название и производитель, количество процессоров, максимальная реальная производительность (на пакете Linpack) и теоретическая пиковая производительность.

Рейтинг составляется два раза в год и включает только общественно известные компьютерные системы в мире – участники рейтинга самостоятельно присылают составителям информацию о своих суперкомпьютерах. Проект был запущен в 1993 году, и с июня 1993 Top500 составляется два раза в год – в июне и ноябре публикуется обновлённый список суперкомпьютеров, который основывается только на информации от узлов сети и производителей. Этот проект направлен на обеспечение надёжной основы для выявления и отслеживания тенденций в области высокопроизводительных вычислений. Россия по данным на июнь 2010 года занимает 7 место по числу установленных систем (11 суперкомпьютеров в списке). Лидирует по этому показателю США – 282 системы. В табл. 4.1 представлен рейтинг суперкомпьютеров на лето 2010 года.

Таблица 4.1

Рейтинг суперкомпьютеров из Top500 на лето 2010

	Rmax Rpeak (Tflops)	Название	Компьютер Число ядер и процессор	Производитель	Страна, год
1	1759.00 2331.00	*Jaguar*	Cray XT5 224162 (Opteron)	Cray	США, 2009
2	1271.00 2984.30	*Nebulae*	Intel X5650, NVidia Tesla C2050 GPU 120640	Dawning	Китай, 2010
3	1042.00 1375.78	*Roadrunner*	BladeCenter QS22/LS21 122400 (Cell/Opteron)	IBM	США, 2009
4	831.70 1028.85	*Kraken*	Cray XT5 98928 (Opteron)	Cray	США, 2009
5	825.50 1002.70	*JUGENE*	Blue Gene/P Solution 294912 (POWER)	IBM	Германия, 2009

В качестве комментария к таблице отметим, что все проекты представляют собой кластерные решения. Во второй колонке показана производительность – максимально достигнутая Rmax и теоретически достижимая (пиковая) – Rpeak на тесте Linpack. Приведем также список систем № 1 в Top500, начиная с 1993 года:

Cray Jaguar (с 2009.11)

IBM Roadrunner (с 2008.06—2009.11)

IBM Blue Gene/L (2004.11-2008.06)

NEC Earth Simulator (2002.06 — 2004.11)

IBM ASCI White (2000.11 — 2002.06)

Intel ASCI Red (1997.06 — 2000.11)

Hitachi CP-PACS (1996.11 — 1997.06)

Hitachi SR2201 (1996.06 — 1996.11)

Fujitsu Numerical Wind Tunnel (1994.11 — 1996.06)

Intel Paragon XP/S140 (1994.06 — 1994.11)

Fujitsu Numerical Wind Tunnel (1993.11 — 1994.06)

ТМС СМ-5 (1993.06 — 1993.11)

На рис. 4.1 представлено распределение по производительности систем, попавших в рейтинг, видно, что лидерство США очевидно. В табл. 4.2 показано распределение показателей систем из Top500 (лето 2010) по производителям.

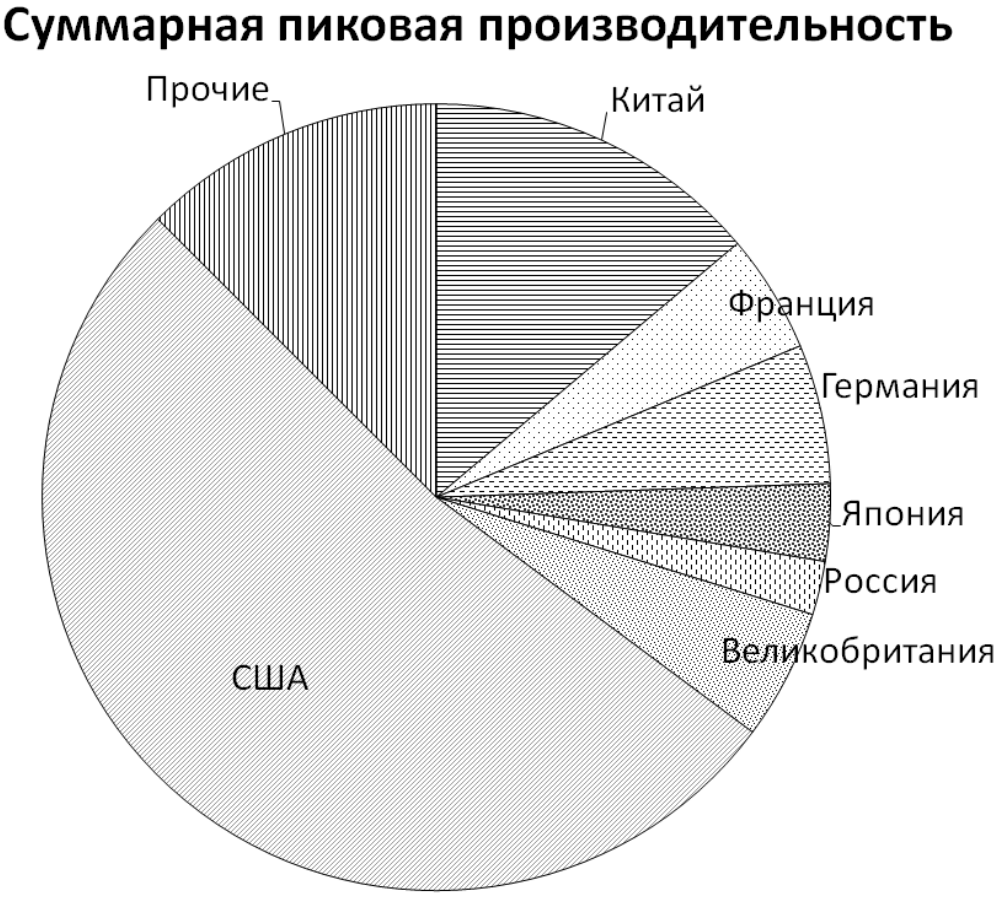

Рис. 4.1. Распределение суммарной пиковой производительности по странам для суперкомпьютеров из Top500 (лето 2010)

Находящийся на 1 месте (Top500 - лето 2010) суперкомпьютер Jaguar размещен в Национальном центре компьютерных исследований в Окридже (США) и используется Министерством энергетики США для моделирования физических систем. Jaguar содержит 18688 вычислительных ячеек по 2 четырехъядерных процессора и еще 7832 ячеек с одним четырехъядерным процессором. Конструктивно 4 вычислительных процессорных элемента объединяются в Cray XT4 в один вычислительный blade-сервер.

Таблица 4.2

Производители систем из рейтинга Top500

Производитель	Количество	Суммарная мощность (Tflops)	Суммарное количество ядер
IBM	198	10891	2241840
Hewlett-Packard	185	6618	1060400
Cray Inc	21	4786	649805
SGI	17	2150	232192
Dell	17	821	106514

Китайская система Nebulae расположена в Национальном суперкомпьютерном центре в Шензене (National Supercomputing Centre in Shenzhen – NSCS). Собран этот кластер китайской фирмой Dawning и носит название Dawning TC3600 Blade System, а область применения не конкретизируется – просто «исследования». В качестве операционной системы используется Linux, для коммуникаций используется оборудование Infiniband QDR, а процессоры – Intel EM64T Xeon X56xx (Westmere-EP) 2660 MHz (10.64 GFlops).

Находящийся на 3 месте суперкомпьютер Roadrunner построила IBM компьютер для Министерства энергетики США. Roadrunner используется для расчёта старения ядерных материалов, а также для анализа безопасности и надёжности ядерного арсенала США. Кроме того, планируется использование Roadrunner для научных, финансовых, транспортных и аэрокосмических расчётов.

4.2. РЕЙТИНГ ДЛЯ СТРАН СНГ TOP50

Цель рейтинга Top50 – получить список 50 наиболее мощных компьютеров России и СНГ для оценки развития отрасли высокопроизводительных вычислений. Формирование списка 50 наиболее мощных компьютеров СНГ призвано акцентировать внимание пользователей, разработчиков, поставщиков компьютерной техники и широкой общественности на колоссальных возможностях современных суперкомпьютеров и параллельных вычислительных технологий, на

перспективности данного направления в разработке наукоемких технологий и развитии инновационной деятельности, на новых областях реальной интеграции промышленности, науки и образования.

В список могут включаются компьютеры, установленные на территории СНГ. Список составляется и поддерживается Межведомственным суперкомпьютерным центром РАН и Научно-исследовательским вычислительным центром МГУ имени М. В. Ломоносова. В табл. 4.3 представлена первая тройка систем из рейтинга Top50 на март 2010.

Первый рейтинг Top50 был составлен в мае 2004 года. Рейтинг обновляется два раза в год в конце марта и в конце сентября.

На рис. 4.2 показаны 4 графика – показывающие рост производительности относительно или всех 50, или первых 5 участников Top50. Показан рост пиковой производительности (Peak) и на тесте Linpack. Легко наблюдается значительное ускорение роста производительности.

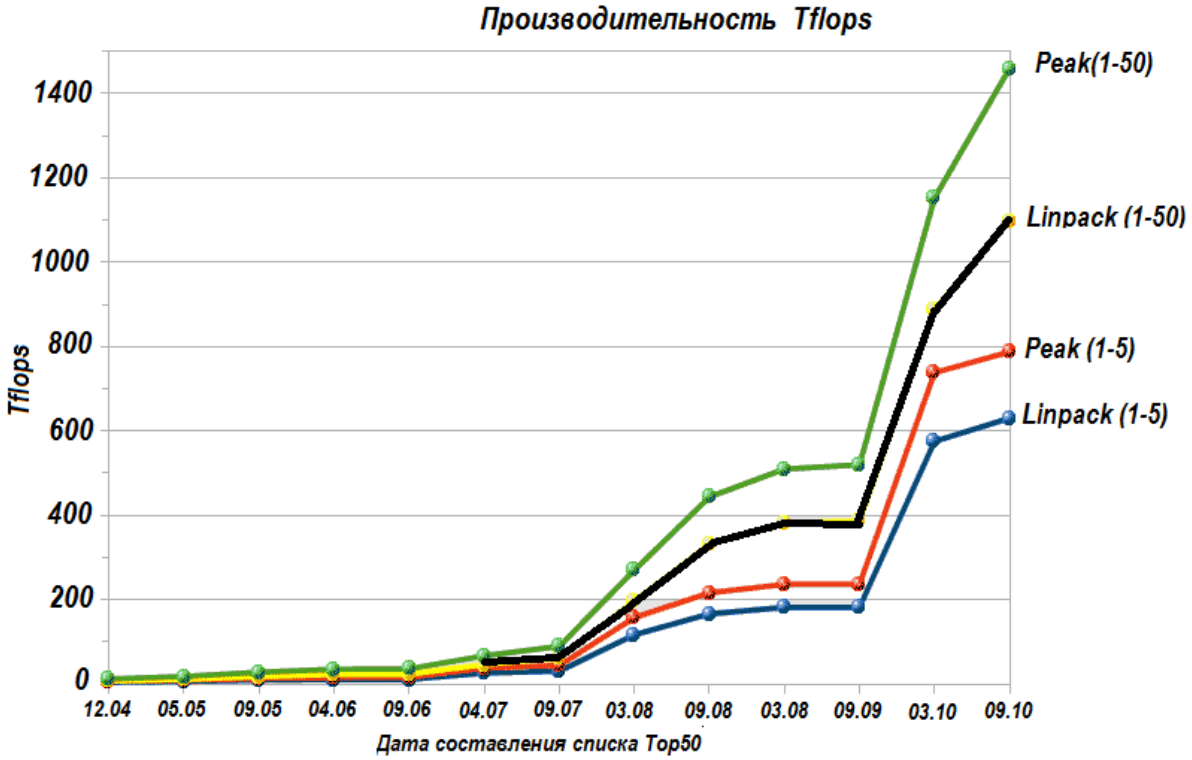

Рис. 4.2. Рост производительности в списке Top50

Таблица 4.3

Лидеры рейтинга Top50 на март 2010

№	Место	Кол-во CPU/ ядер	Архитектура	Rmax Rpeak (Tflops)	Разработ-чик
1	Москва Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова 2009 г.	8840 / 35360	узлов: 4160 (2xXeon X5570 2.933 GHz 12.28 GB RAM) узлов: 260 (2xXeon X5570 2.933 GHz 24.576 GB RAM) сеть: QDR Infini-band/QDR Infini-band/Fast Ethernet	350,1 414,2	Т-Платфор-мы
2	Москва Межведомственный суперкомьютерный центр Российская академия наук 2009 г	2920 / 11680	узлов: 990 (2xXeon E5450 3 GHz 8.192 GB RAM) узлов: 470 (2xXeon 5365 3 GHz 8.192 GB RAM) сеть: Infiniband 4x DDR/2xGigabit Ether-net/Gigabit Ethernet	107,5 140,2	Hewlett-Packard
3	Москва Научно-исследовательский вычислительный центр МГУ имени М.В.Ломоносова 2008 г	1250 / 5000	узлов: 529 (2xXeon E5472 3 GHz 8.192 GB RAM) узлов: 64 (2xXeon E5472 3 GHz 8.192 GB RAM) узлов: 32 (2xXeon E5472 3 GHz 16.384 GB RAM) узлов: 8 (2xXeon E5472 3 GHz 32.768 GB RAM) сеть: InfiniBand/Gigabit Ethernet/СКИФ-ServNet + IPMI	47.3 60	СКИФ

Следует отметить, что все лидеры рейтинга расположены в Москве. Занимающий первое место суперкомпьютер «Ломоносов» на момент установки стал самым мощным в странах Восточной Европы: пиковая производительность системы – 420Tflops, реальная – 350Tflops. В ноябре 2009 суперкомпьютер занял 12-е место в рейтинге Top500. Суперкомпьютер построен на базе инновационной blade-системы T-Blade2, которая была сконструирована инженерами компании «Т-Платформы» «с нуля». Все платы и механические компоненты системы, за исключением микросхем, вентиляторов и блоков питания, являются запатентованными разработками «Т-Платформы». При этом лидер рейтинга Top50 уступает по производительности лидеру Top500 примерно в 5 раз.

В России 25 ноября 2003 года создан Научный Совет РАН «Высокопроизводительные вычислительные системы, научные телекоммуникации и информационная инфраструктура» Постановлением Президиума РАН №325 в целях координации работ по созданию, использованию и развитию высокопроизводительных вычислительных, телекоммуникационных и информационных ресурсов.

4.3. ПРИМЕРЫ ПРИМЕНЕНИЯ СУПЕРКОМПЬЮТЕРОВ В РОССИИ

Современный уровень развития вычислительной техники и методов математического моделирования дает уникальную возможность для перевода как промышленного производства, так и научных исследований на качественно новый уровень. Цифровые модели сложных конструкций, точное описание и воспроизведение природных явлений и процессов, тонкая многопараметрическая оптимизация – все это становится реальным сегодня. Для индустрии – это повышение конкурентоспособности на мировом рынке, для науки – это завоевание лидирующих позиций нашими учеными, и все это вместе составляет один из важных элементов перехода к инновационной экономике. Именно поэтому суперкомпьютерные технологии отнесены руководством страны к приоритетным направлениям модернизации экономики и технологического развития. Для координации усилий в облас-

ти суперкомпьютерных технологий в России был создан соответствующий консорциум. Ниже представлены некоторые примеры достижений его участников.

Центр коллективного пользования Санкт-Петербургского государственного политехнического университета развивает работы во многих направлениях. В частности, занимается решением ряда вычислительно ресурсоемких задач для ОАО «Силовые машины» – «ЛМЗ».

Проблема защиты Санкт-Петербурга от наводнений привела к насущной необходимости построить в Финском заливе защитную дамбу. Ее составной частью являются судопропускные сооружения С1 и С2. Важнейшим элементом большого судопропускного сооружения С1 является плавучий затвор (батопорт), состоящий из двух подвижных створок. В процессе затопления створок могут возникать интенсивные колебания, грозящие разрушением всей конструкции. Причиной этих колебаний является то, что в процессе погружения створок между дном канала и нижней границей створок образуется сильное течение. Предметом проводимых исследований является изучение процесса обтекания затвора и нахождение действующих на него гидродинамических сил. Работы в данном направлении ведутся коллективом межкафедеральной лаборатории прикладной математики и механики физико-механического факультета СПбГПУ, начиная с 2004 года. В частности были рассмотрены два случая – заглубления 2 и 10 м при перепаде уровней воды на затворе в 1 м. Как выяснилось в ходе расчетов, общая картина течения для этих случаев существенно различается.

Результаты вычислений хорошо показывают, как распределяются вихревые структуры в пространстве. Если в случае заглубления 10 м наблюдается единый «вихревой жгут», сходящий с нижней кромки створки, то в случае заглубления 2 м этот жгут разбивается на множество мелких вихрей.

В расчетах использовались многопроцессорные кластеры Лаборатории прикладной математики и механики. Всего в расчете был за-

действован практически весь ресурс двухсот пятидесяти шести процессорных ядер. Общее процессорное время, затраченное на проведение расчетов, составляет около ста пятидесяти тысяч часов. При этом среднее время численного решения каждого отдельного варианта задачи составляло около полутора месяцев.

4.4. ПРОИЗВОДСТВО СУПЕРКОМПЬЮТЕРОВ В РОССИИ

Компания «Т-Платформы» является крупнейшим российским холдингом на рынке высокопроизводительных вычислений. Холдинг предоставляет полный комплекс решений и услуг в области суперкомпьютерных технологий и ресурсоемких расчетов. На российском рынке он занимает практически монопольное положение среди отечественных компаний в области проектирования высокопроизводительных программно-аппаратных решений, обеспечивающих максимальную реальную производительность приложений, а также разработки и производства суперкомпьютерных платформ. Компания также занимается разработкой системного программного обеспечения для высокопроизводительных систем, а также непосредственно проектированием и созданием суперкомпьютерных центров, включая управление ими для оптимизации затрат и получения прибыли.

«Т-Платформы» – единственный российский разработчик, высокопроизводительные системы которого вошли в рейтинг самых мощных суперкомпьютеров мира Top500. Наличие собственных уникальных разработок и высокий профессиональный уровень позволяют холдингу реализовывать собственные технологические решения любой сложности и обеспечивать оптимальное соотношение «цена/производительность». На счету компании более 150 успешных комплексных проектов, а также ряд собственных патентов на суперкомпьютерные технологии и электронные компоненты.

Последним достижением компании является инновационная блейд-система T-Blade2 (TB2), которая была сконструирована инженерами «Т-Платформы» «с нуля». Все платы и механические компоненты системы, за исключением микросхем, вентиляторов и блоков питания, являются запатентованными разработками «Т-Платформы».

T-Blade2 – универсальная платформа для строительства суперкомпьютеров высшего диапазона производительности. Шасси TB2 включает 16 вычислительных модулей в форм-факторе 7U с 32 или 64 процессорами, два 36-портовых коммутатора QDR InfiniBand с избыточным количеством внешних портов суммарной пропускной способностью 1.6 Тб/сек, модуль управления и вспомогательные сети стандарта Ethernet. Высокая вычислительная плотность достигается за счет использования запатентованной системной платы L-образной формы, со специализированными компактными модулями памяти DDR3 и инновационной конструкции радиатора, обеспечивающей отвод тепла от всего вычислительного модуля.

В отличие от большинства вычислительных платформ, предназначенных для суперкомпьютеров высшего диапазона производительности, шасси T-Blade2 является независимым вычислительным блоком с воздушным охлаждением, устанавливаемым в стандартную стойку шириной 19”, что обеспечивает дополнительную гибкость развертывания. Новое системное программное обеспечение Clustrx обеспечивает мониторинг всех подсистем суперкомпьютера в режиме, близком к реальному времени, и позволяет контролировать до 12000 вычислительных узлов с помощью одного управляющего сервера и увеличивать скорость работы приложений.

Среди реализованных проектов отметим суперкомпьютер «Ломоносов», который на момент установки стал самым мощным в странах Восточной Европы: пиковая производительность системы – 420Tflops, реальная – 350Tflops. В ноябре 2009 суперкомпьютер занял 12-е место в рейтинге ТОП500 мощнейших компьютеров мира.

Суперкомпьютерный комплекс, поставленный компанией «Т-Платформы» для МГУ им. М.В. Ломоносова, обладает пиковой производительностью 420Tflops. Реальная производительность системы на тесте Linpack – 350Tflops. Таким образом, эффективность суперкомпьютера, то есть соотношение реальной и пиковой производительности составляет 83%. Этот показатель на сегодня является одним из самых высоких в мире: аналогичный показатель суперкомпь-

ютера Jaguar, лидера списка Top500 на июнь 2010 года, составляет лишь 75.46%.

Говоря о вычислительном ядре суперкомпьютера «Ломоносов», следует отметить, что это первый гибридный суперкомпьютер такого масштаба в России и Восточной Европе. В нем используется 3 вида вычислительных узлов и процессоры с различной архитектурой. Перед установкой в МГУ им. М.В. Ломоносова система прошла тщательное тестирование на производстве компании «Т-Платформы». В качестве основных узлов, обеспечивающих свыше 90% производительности системы, используется инновационная blade-платформа T-Blade2.

По вычислительной плотности на квадратный метр занимаемой площади – 30Tflops/м2 – эта система превосходит все мировые аналоги. T-Blade2 на базе процессоров Intel Xeon X5570 обеспечивает производительность 18Tflops в стандартной стойке высотой 42U. В суперкомпьютере также задействованы blade-системы T-Blade 1.1 с увеличенным объемом оперативной памяти и локальной дисковой памятью для выполнения специфических задач, особенно требовательных к этим параметрам системы. Третий тип узлов – платформы на базе многоядерного процессора PowerXCell 8i, использующиеся в качестве мощных ускорителей для ряда задач. Все три типа вычислительных узлов были разработаны компанией «Т-Платформы».

В качестве системной сети, связывающей узлы суперкомпьютера «Ломоносов», используется интерконнект QDR Infiniband с пропускной способностью до 40Гб/сек. Для максимально бесконфликтной передачи данных в интегрированных коммутаторах InflniBand предусмотрено избыточное количество внешних портов: их суммарная пропускная способность составляет 1,6Тбит/сек.

Суперкомпьютер использует трехуровневую систему хранения данных суммарным объемом до 1 350ТБ с параллельной файловой системой Lustre. Система хранения данных обеспечивает одновременный доступ к данным для всех вычислительных узлов суперком-

пьютера с агрегированной скоростью чтения данных – 20Гб/сек и агрегированной скоростью записи – 16Гб/сек.

Суперкомпьютер работает под управлением пакета Clustrx – разработки компании T-Massive Computing, входящей в состав холдинга «Т-Платформы». Clustrx ОС устраняет критические ограничения масштабируемости, присущие современным операционным системам, обеспечивая более эффективное использование ресурсов крупных инсталляций размером до 25000 узлов. Пакет Clustrx содержит все необходимые компоненты для управления суперкомпьютером и организации удобного доступа пользователей к системе. Clustrx обеспечивает ежесекундный мониторинг до 300 метрик на каждом вычислительном узле, использует технологии агрессивного энергосбережения и автоматического реагирования на критические ситуации.

Добиться требуемого уровня отказоустойчивости позволило резервирование всех критических подсистем и компонентов суперкомпьютерного комплекса – от вентиляторов и блоков питания в вычислительных узлах до систем электропитания и охлаждения. Высокую надежность blade-систем обеспечивает отсутствие кабельных соединений и жестких дисков внутри шасси, а также целый ряд конструктивных решений, таких как специально разработанные разъемы для модулей памяти.

Система будет использоваться для решения ресурсоемких вычислительных задач в рамках фундаментальных научных исследований, а также для проведения научной работы в области разработки алгоритмов и программного обеспечения для мощных вычислительных систем.

Другой проект компании «Т-Платформы», работающий в настоящее время – это работающий в Межведомственном суперкомпьютерном центре суперкомпьютер «МВС-100К», который предназначен для решения сложных научно-технических задач. Его пиковая производительность составляет 140,16 TFlops. Программные и аппаратные средства «МВС-100К» позволяют решать одну задачу с ис-

пользованием всего вычислительного ресурса, а также разделять решающее поле на части требуемого размера и предоставлять их нескольким пользователям.

В состав технических средств СК "МВС-100К" входят:

- решающее поле из 1460 вычислительных модулей (11680 процессорных ядер);
- управляющая станция и узел доступа на базе двух процессоров Intel Xeon;
- коммуникационная сеть Infiniband DDR, построенная с использованием коммутаторов Voltaire и Cisco;
- транспортная сеть Gigabit Ethernet;
- управляющая сеть Gigabit Ethernet;
- системная консоль.

Базовый вычислительный модуль СК "МВС-100К" представляет собой сервер HP Proliant, содержащий:

- два четырёхядерных микропроцессора Intel Xeon, работающих на частоте 3 ГГц;
- оперативную память DDR2 объёмом не менее 4 ГБайт;
- жёсткий диск объёмом не менее 36 ГБайт;
- интерфейсную плату HP Mezzanine Infiniband DDR;
- два интегрированных контроллера Gigabit Ethernet.

Вычислительные модули связаны между собой высокоскоростной коммуникационной сетью Infiniband DDR, транспортной и управляющей сетями Gigabit Ethernet. Коммуникационная сеть Infiniband DDR предназначена для высокоскоростного обмена между вычислительными модулями в ходе вычислений. Сеть реализована двумя уровнями коммутаторов. Скорость двунаправленных обменов данными между двумя вычислительными модулями с использованием библиотек интерфейса передачи сообщений – MPI – находится на уровне 1400 Мбайт/сек. Латентность между двумя соседними узлами составляет 3.2 мкс, самыми дальними 4.5 мкс.

Транспортная сеть Gigabit Ethernet предназначена для соединения решающего поля с управляющей станцией, параллельной файло-

вой системой и файл-сервером. Управляющая сеть, построенная с использованием технологии Gigabit Ethernet, предназначена для запуска программ на счёт вычислительными модулями, а также для передачи служебной информации о ходе вычислительного процесса и состоянии подсистем.

Программное обеспечение "МВС-100К" поддерживает все этапы разработки параллельных программ, а также обеспечивает выполнение процессов обработки данных на решающем поле. При выборе программного обеспечения использовался принцип преемственности с системой "МВС-6000IM" для облегчения переноса программ на новый кластер.

На "МВС-100К" установлено следующее программное обеспечение:

- операционная система вычислительных модулей – ОС CentOS 5.3;
- программные средства коммуникационных сетей Infiniband, Ethernet;
- среда параллельного программирования – пакет MVAPICH;
- инструментальные программные средства разработки системного и прикладного программного обеспечения, включающие оптимизирующие компиляторы с языков Си, C++ (icc) и Фортран-77, 90 (ifc) фирмы Intel;
- математические библиотеки MKL фирмы Intel;
- система коллективного использования ресурсов СК – система управления прохождением пользовательских задач (СУППЗ), разработанная ИПМ РАН;
- программные средства удаленного доступа (ssh).

На кластере также установлены средства профилирования параллельных программ, инструменты параллельного администрирования, управления и тестирования кластера, позволяющие осуществлять проверку состояния и диагностику узлов кластера, создание и модификацию пользовательских бюджетов на узлах кластера, параллель-

ные операции над файлами и выполнение операций на всех узлах кластера. В заключении отметим, что взаимодействие удаленных пользователей с суперкомпьютером осуществляется по протоколу ssh по адресу mvs100k.jscc.ru.

Компиляция заданий осуществляется на узле доступа (mvs100k.jscc.ru). Выполнение заданий производится на остальных узлах кластера. Для планирования выполнения заданий используется подсистема коллективного доступа.

4.5. СУПЕРКОМПЬЮТЕРЫ ФИРМЫ IBM

Долгое время среди суперкомпьютеров в рейтинге Top500 лидировали кластерные решения фирмы IBM семейства Blue Gene. Blue Gene – проект компьютерной архитектуры, разработанный для создания нескольких суперкомпьютеров и направленный на достижение скорости обработки данных, превышающей 1 петафлопс. На данный момент успешно достигнута скорость почти в 500 терафлопс. Blue Gene является совместным проектом фирмы IBM, Ливерморской национальной лаборатории, Министерства энергетики США (которое частично финансирует проект) и академических кругов. Предусмотрено четыре этапа проекта: Blue Gene/L, Blue Gene/C, Blue Gene/P и Blue Gene/Q. Проект был награждён Национальной Медалью США в области технологий и инноваций 18 сентября 2009 года. Президент Барак Обама вручил награду 7 октября 2009.

Blue Gene/L – это первый компьютер серии IBM Blue Gene, разработанный совместно с Ливерморской национальной лабораторией. Его теоретическая пиковая производительность составляет 360 терафлопс, а реальная производительность, полученная на тесте Linpack, около 280 терафлопс. После обновления в 2007 году реальная производительность увеличилась до 478 терафлопс при пиковой производительности в 596 терафлопс. В ноябре 2006 года 27 компьютеров из списка TOP500 имели архитектуру Blue Gene/L. Блок-схема чипа Blue Gene/L показана на рис. 4.3.

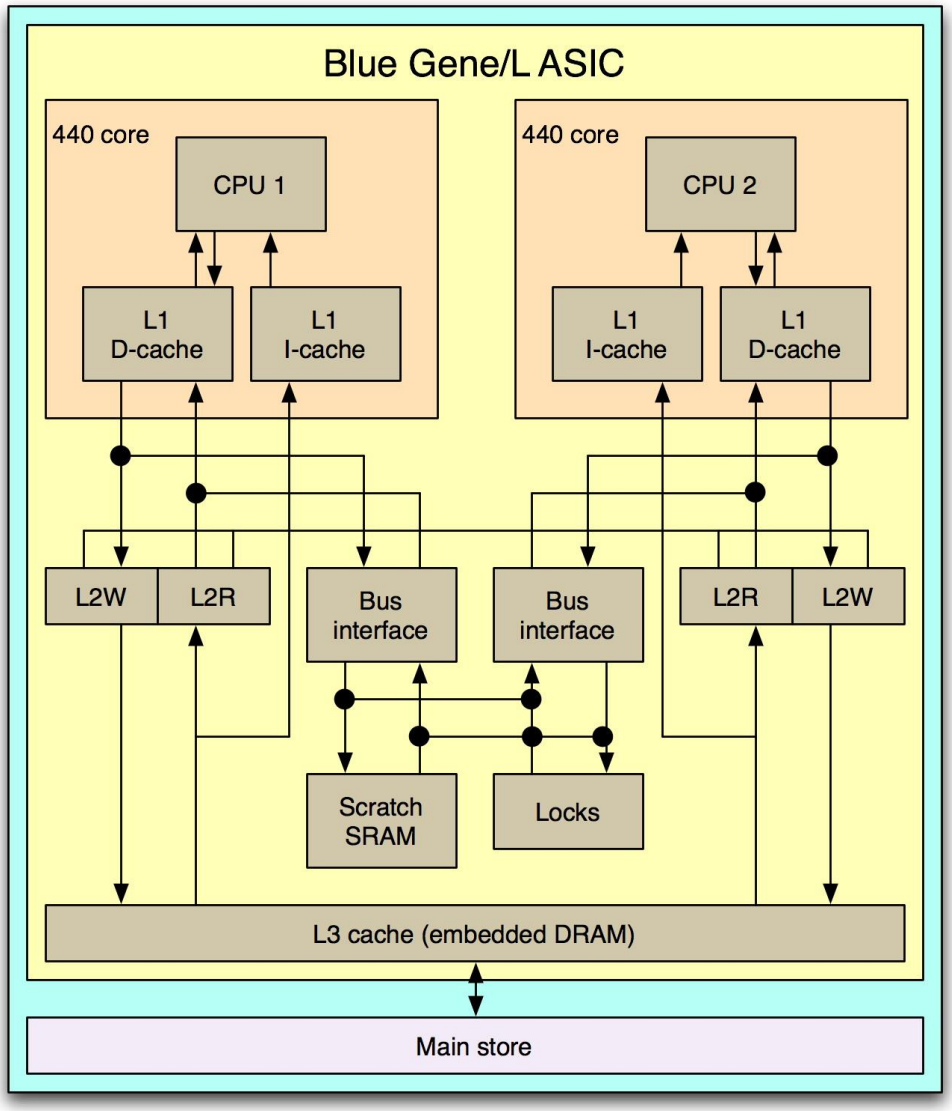

Рис. 4.3. Вычислительный узел Blue Gene/L

Новое поколение MPP-систем IBM представила 26 июня 2007 года. Оно получило название Blue Gene/P – второе поколение суперкомпьютеров Blue Gene. Разработан для работы с пиковой производительностью в 1 петафлопс. Blue Gene/P может быть сконфигурирован для достижения пиковой производительности более, чем 3 петафлопса. Кроме того, он в семь раз более энергетически эффективен, чем любые другие суперкомпьютеры. Blue Gene/P выполнен с использованием большого числа небольших, маломощных чипов, связывающихся через пять специализированных сетей.

Система включает следующие ключевые особенности. Увеличенное число ядер на стойку составляет 4096 штук. Используется микропроцессор IBM PowerPC, Book E compliant, 32-bit, 850 MHz. Двойная точность, двойной канал ускорения с плавающей запятой на каждом ядре. Для системы характерна низкая мощность рассеивания на каждую микросхему – показатель эффективности 1.8 Ватт на Gflops.

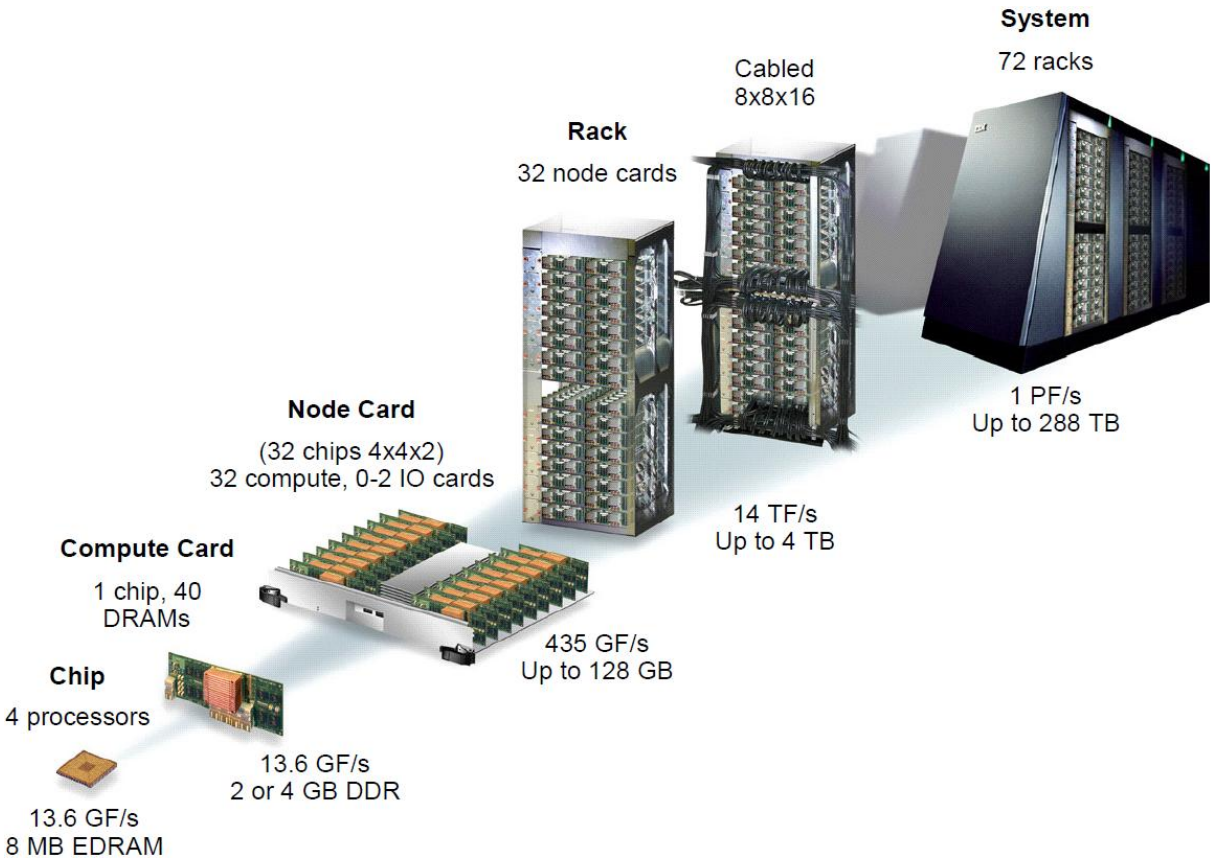

Рис. 4.4. Архитектура суперкомпьютера Blue Gene/P

На рис. 4.4 показана архитектура Blue Gene/P. Каждый чип (Chip) Blue Gene/P состоит уже из четырёх процессорных ядер PowerPC 450 с тактовой частотой 850 МГц. Чип с 2 или 4 ГБ оперативной памяти и сетевые интерфейсы образуют вычислительный узел суперкомпьютера (Compute card). 32 вычислительных узла объединяются в карту (Node card), к которой можно подсоединить от 0 до 2 узлов ввода-вывода.

Системная стойка (Rack) вмещает в себя 32 таких карты (Node card). Конфигурация Blue Gene/P с пиковой производительностью 1 петафлопс представляет собой 72 системные стойки (System), содержащие 294,912 процессорных ядер, объединённых в высокоскоростную оптическую сеть.

Конфигурация Blue Gene/P может быть расширена до 216 стоек с общим числом процессорных ядер 884,736, чтобы достигнуть пиковой производительности в 3 петафлопса. В стандартной конфигурации системная стойка Blue Gene/P содержит 4,096 процессорных ядер.

Окружение системы Blue Gene/P представлено на рис. 4.5 и состоит из множества компонент, без которых системы не будет – сетевая топология тор (torus).

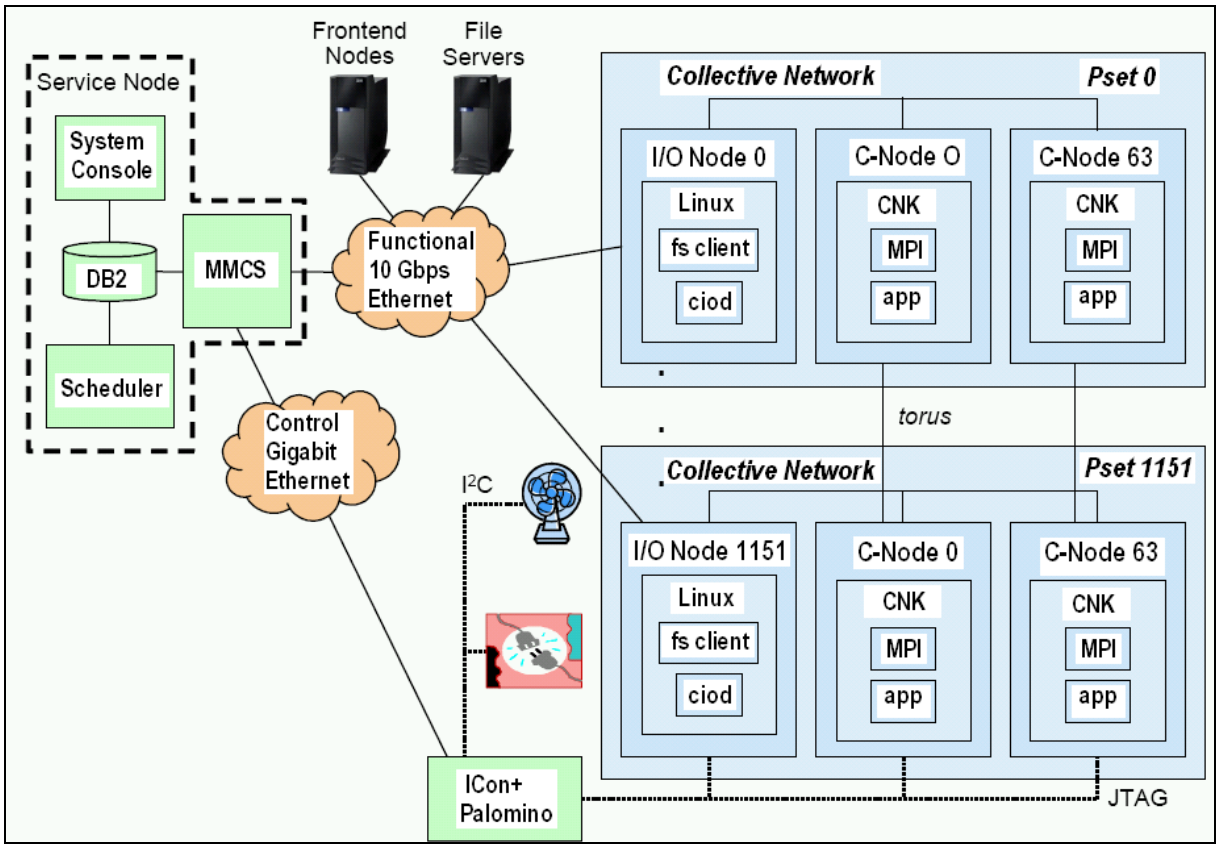

Рис. 4.5. Окружение суперкомпьютера Blue Gene/P

На рис. 4.5 показаны следующие составляющие окружения. Service node – этот узел обеспечивает управление системой. Front end

node – этот узел обеспечивает для пользователей доступ к выполнению приложений. Compute node – этот узел запускает приложения, и у пользователей нет прямого доступа к нему. I/O Node – этот узел обеспечивает доступ к внешним устройствам, и все запросы на ввод/вывод проходят через этот узел. Functional network – эта сеть используется всеми компонентами системы Blue Gene/P, кроме Compute node. Control network – эта сеть является обслуживающей для особенных управляющих функций системы между Service node и I/O Node.

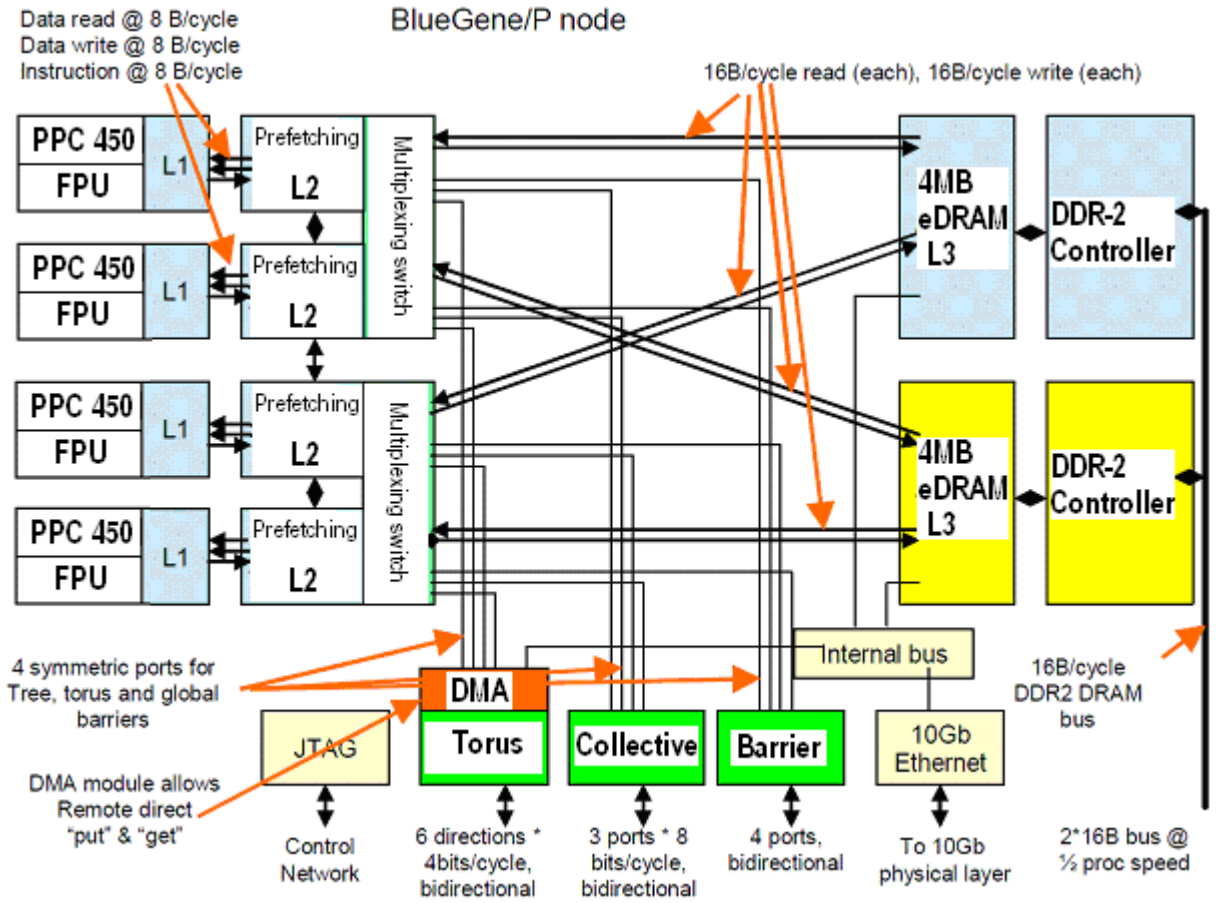

Рис. 4.6. Вычислительный узел Blue Gene/P

Вычислительный узел суперкомпьютера Blue Gene/P показан на рис. 4.6. Можно сравнить его с узлом Blue Gene/L, представленным на рис. 4.3, и убедиться насколько была усложнена конструкция. В узле Blue Gene/P имеется шесть коннекторов для сети в виде тора – по

3.6 Гбит/с на каждый. Имеется также три коннектора к глобальной сети – по 6.8 Гбит/с.

В системе Blue Gene/P используется пять видов сетевой коммуникации для различных задач. Среди них отметим трехмерный тор (соединение точка-точка). Эта сеть для передачи сообщений точка-точка, а также передачи сообщений выбранному «классу» узлов. Топология в виде трехмерного тора жестко связана со структурой узла Blue Gene/P. Поэтому каждый узел имеет шесть соединений с ближайшими соседями, некоторые из них могут соединяться с помощью относительно длинного кабеля. Проектная скорость передачи внутри тора составляет 425 Мбит/с в каждом направлении – суммарно получается 5.1 Гбит/с двунапраправленного пропускания на узел.

Третье место в рейтинге Top500 за июнь 2010 занимает суперкомпьютер фирмы IBM Roadrunner. Он стал первым компьютером, преодолевшим на тесте Linpack рубеж производительности в 1 Pflops. Суперкомпьютер создан компанией IBM для Министерства Энергетики США и установлен в Лос-Аламосской национальной лаборатории в Нью-Мексико, США. Суперкомпьютер Roadrunner построен по гибридной схеме из 6120 двухъядерных процессоров AMD Opteron и почти 12240 процессоров IBM Cell 8i в специальных блэйд-модулях TriBlades, соединенных с помощью коммуникационной сети Infiniband. Установка занимает площадь приблизительно 560 квадратных метров и весит 226 тонн. Общее знергопотребление установки – 2.35 МВт, при этом энергоэффективность составляет 437 Mflops/Вт. Стоимость IBM Roadrunner составила 133 миллиона долларов. Пиковая производительность суперкомпьютера составила 1.376 Pflops, производительность на тесте Linpack – 1.026 Pflops.

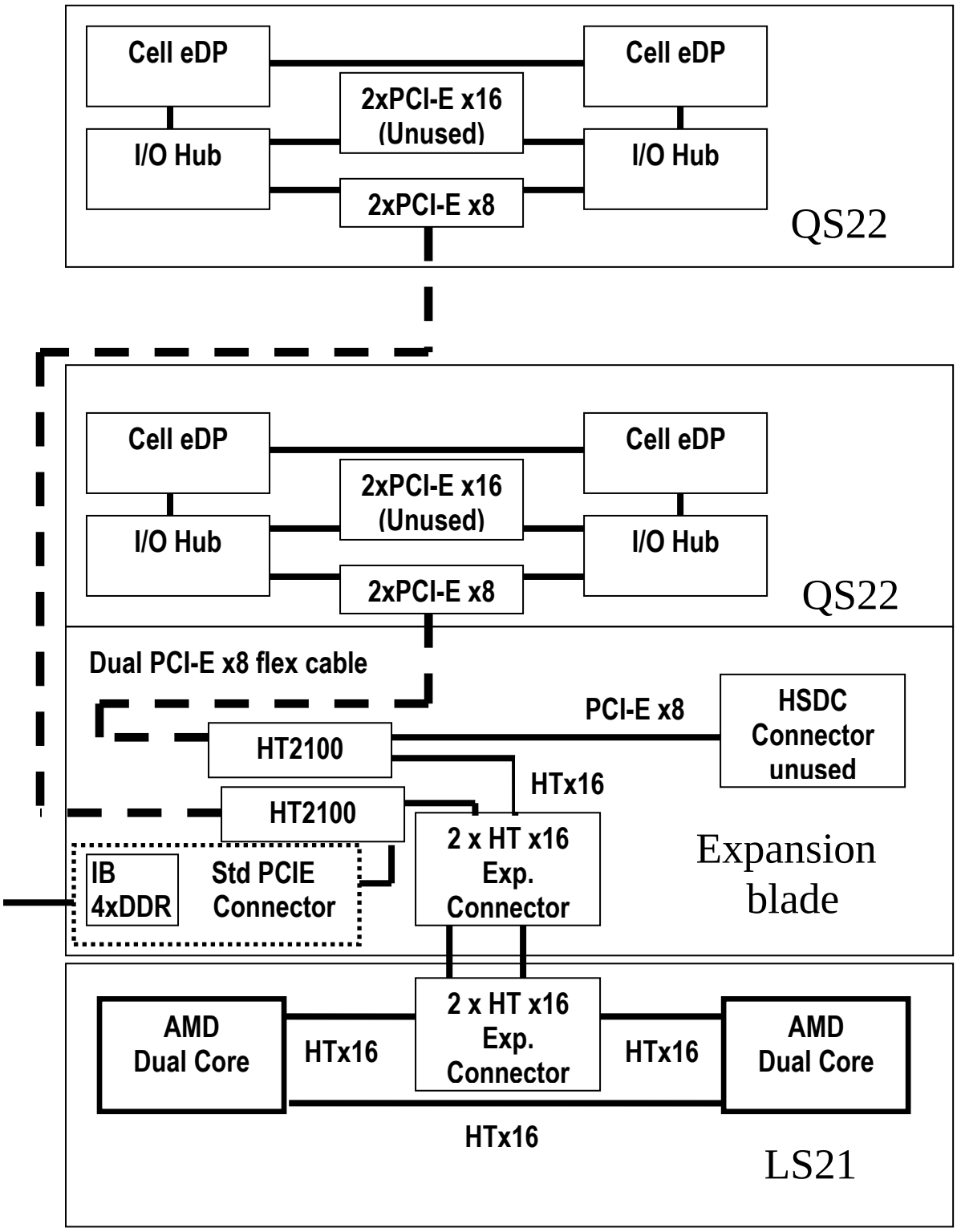

Рис. 4.7. Блэйд-модуль TriBlade суперкомпьютера Roadrunner

Министерство Энергетики планирует использовать RoadRunner для расчёта старения ядерных материалов и анализа безопасности и

надёжности ядерного арсенала США. Также планируется использование для научных, финансовых, транспортных и аэрокосмических расчетов.

В суперкомпьютере Roadrunner использованы двухъядерные процессоры AMD Opteron 2210, работающие на частоте 1.8 ГГц. Всего задействовано 6912 таких процессоров: 6120 вычислительных и 442 процессора для системных функций (13124 процессорных ядра: 12240 и 884, соответственно).

Также в суперкомпьютере Roadrunner задействованы процессоры IBM PowerXCell 8i, работающие на частоте 3.2 ГГц. Каждый такой процессор включает в себя одно универсальное ядро Power (PPE) и 8 специальных ядер для операций с плавающей точкой (SPE). Всего задействовано 12240 таких процессоров, что составляет 12240 ядер PPE и 97920 ядер SPE (всего 110160 ядер).

Логически блэйд-модуль TriBlade, представленный на рис.4.7, состоит из четырёх ядер Opteron, четырёх PowerXCell 8i процессоров, 16 Гбайт памяти для Opteron и 16 Гбайт памяти для Cell. Физически TriBlade состоит из одной платы LS21, платы расширения и двух плат QS22.

LS21 содержит два двухъядерных процессора Opteron с 16 Гбайт памяти, по 4 Гбайт на ядро. Каждая плата QS22 содержит два процессора PowerXCell 8i и 8 Гбайт памяти, по 4 Гбайт на каждый процессор. Плата расширения соединяет QS22 через четыре разъёма PCI x8 с LS21, по два разъёма на QS22. Также плата расширения (Expansion Blade) обеспечивает подключение Infiniband 4x DDR. В результате один блэйд-модуль TriBlade занимает четыре слота, и три TriBlades помещаются в шасси BladeCenter H.

Объединённый модуль – это 60 шасси BladeCenter H с установленными блэйд-модулями TriBlades (всего 180 TriBlades). Все TriBlades подсоединены к 288-портовому маршрутизатору Voltaire ISR2012 Infiniband. Каждый объединённый модуль также подсоединён к файловой системе Panasas через 12 серверов System x3755.

Приведем системную информацию по объединённому модулю:

- 360 двухъядерных процессоров Opteron с 2.88 Тбайт памяти;
- 720 процессоров PowerXCell с 2.88 Тбайт памяти;
- 12 серверов System x3755 с двумя 10 Гбит Ethernet каждый;
- 288-портовый маршрутизатор Voltaire ISR2012 с 192 Infiniband 4x DDR (180 TriBlades и 12 узлов ввода/вывода).

Подводя итоги, можно сделать вывод, что суперкомпьютер Roadrunner представляет собой кластер кластеров.

4.6. СУПЕРКОМПЬЮТЕРЫ ФИРМЫ CRAY

Cray Research, Inc. была основана в 1972 году проектировщиком компьютеров Сеймуром Креем (Seymour Cray). В 1989 году Cray Research выделила разработку и производство суперкомпьютера Cray-3 в отдельную компанию, Cray Computer Corporation, которую возглавил Сеймур Крей и базирующуюся в Колорадо-Спрингс (штат Колорадо). В феврале 1996 года Cray Research слилась с компанией Silicon Graphics (SGI). Сеймур Крей работал над разработками компании вплоть до своей смерти, при автомобильной аварии в сентябре 1996 года в возрасте 71 года. Компания Cray Inc. была зарегистрирована в 2000 году, когда компания Tera Computer Company перекупила Cray Research Inc. у SGI и объединилась с купленной компанией под общим названием Cray Inc. Первый компьютер Cray-1 – 1976 год, стоимость $8.8 млн., 160 Mflops, 8 Mb – память.

Представляют интерес основные характеристики выпущенных Cray систем за последние десятилетия, т.к. эта компания всегда была некоторым «законодателем мод» на рынке суперкомпьютеров. В табл. 4.4 представлены характеристики Cray T3E Jaromir (1995 год), в табл. 4.5 – характеристики Cray SV1 (1998 год), а в табл. 4.6– характеристики Cray X1 (2003 год).

Суперкомпьютер Cray XT3 является дальнейшим развитием линии массивно-параллельных компьютеров Cray T3D и Cray T3E. Вычислительный узел Cray XT3 включает в себя процессор AMD Opteron, локальную память (от 1 до 8 Гбайт) и канал HyperTransport к коммуникационному блоку Cray SeaStar, представленный на рис. 4.8.

Таблица 4.4

Характеристики Cray T3E Jaromir

Класс архитектуры	Масштабируемая массивно-параллельная система, состоит из процессорных элементов.
Процессорный элемент	Состоит из процессора, блока памяти и устройства сопряжения с сетью. Используются процессоры Alpha 21164 (EV5) с тактовой частотой 450 MHz (T3E-900), 600 MHz (T3E-1200), 675 MHz (T3E-1350) пиковая производительность которых составляет 900, 1200, 1350 MFlops соответственно. Локальная память (DRAM) от 256MB до 2GB.
Число процессоров	Системы T3E масштабируются до 2048 процессорных элементов.
Коммутатор	Процессорные элементы связаны высокопроизводительной сетью GigaRing с топологией трехмерного тора и двунаправленными каналами. Скорость обменов по сети достигает 500MB/c в каждом направлении.
Системное программное обеспечение	Используется операционная система UNICOS/mk.
Средства программирования	Явное параллельное программирование с помощью Message Passing Toolkit (MPT) - реализации интерфейсов передачи сообщений MPI, MPI-2 и PVM, библиотека Shmem. Для Фортрана возможно также неявное распараллеливание в моделях CRAFT и HPF. Имеется также набор визуальных средств для анализа и отладки параллельных программ.

Таблица 4.5

Характеристики Cray SV1

Класс архитектуры	Масштабируемый векторный суперкомпьютер.
Процессор	Используются 8-конвейерные векторные процессоры MSP (Multi-Streaming Processor) с пиковой производительностью 4.8 Gflops. Каждый MSP может быть подразделен на 4 стандартных 2-конвейерных процессора с пиковой производительностью 1.2 Gflops. Тактовая частота процессоров - 250MHz.
Число процессоров	Процессоры объединяются в SMP-узлы, каждый из которых может содержать 6 MSP и 8 стандартных процессоров. Система (кластер) может содержать до 32 таких узлов.
Память	SMP-узел может содержать от 2 до 16GB памяти. Система может содержать до 1TB памяти. Вся память глобально адресуема - архитектура DSM (Distributed Shared Memory — распределенная совместно используемая память).
Системное программное обеспечение	Используется операционная система UNICOS.
Средства программирования	Поставляется векторизующий и распараллеливающий компилятор CF90. Поддерживается также явное параллельное программирование с использованием интерфейсов MPI, OpenMP или Shmem. Имеется также набор визуальных средств для анализа и отладки параллельных программ.

Таблица 4.6

Характеристики Cray SX1Cray

Класс архитектуры	Масштабируемый векторный суперкомпьютер.
Процессор	Используются 16-конвейерные векторные процессоры с пиковой производительностью 12.8 Gflops. Тактовая частота процессоров - 800MHz.
Число процессоров	В максимальной конфигурации - до 4096.
Память	Каждый процессор может содержать до 16GB памяти. В максимальной конфигурации система может содержать до 64TB памяти. Вся память глобально адресуема (архитектура DSM). Максимальная скорость обмена с оперативной памятью составляет 34.1 Гбайт/сек. на процессор, скорость обмена с кэш-памятью 76.8 Гбайт/сек. на процессор.
Системное программное обеспечение	Используется операционная система UNICOS/mp.
Средства программирования	Реализованы компиляторы с языков Фортран и Си++, включающие возможности автоматической векторизации и распараллеливания, специальные оптимизированные библиотеки, интерактивный отладчик и средства для анализа производительности. Приложения могут писаться с использованием MPI, OpenMP, Co-array Fortran и Unified Parallel C (UPC).

Коммуникационная технология Cray SeaStar позволяет объединить все вычислительные узлы Cray XT3 по топологии трехмерного тора. Коммуникационная плата Cray SeaStar включает в себя канал HyperTransport, Direct Memory Access (DMA), коммуникационный микропроцессор, interconnect router и управляющий порт. Interconnect router обеспечивает 6 высокоскоростных каналов связи с пиковой пропускной способностью каждого в двунаправленном режиме 7,6

Гбайт/сек. При этом из приложений на MPI достигается латентность около 3 мкс.

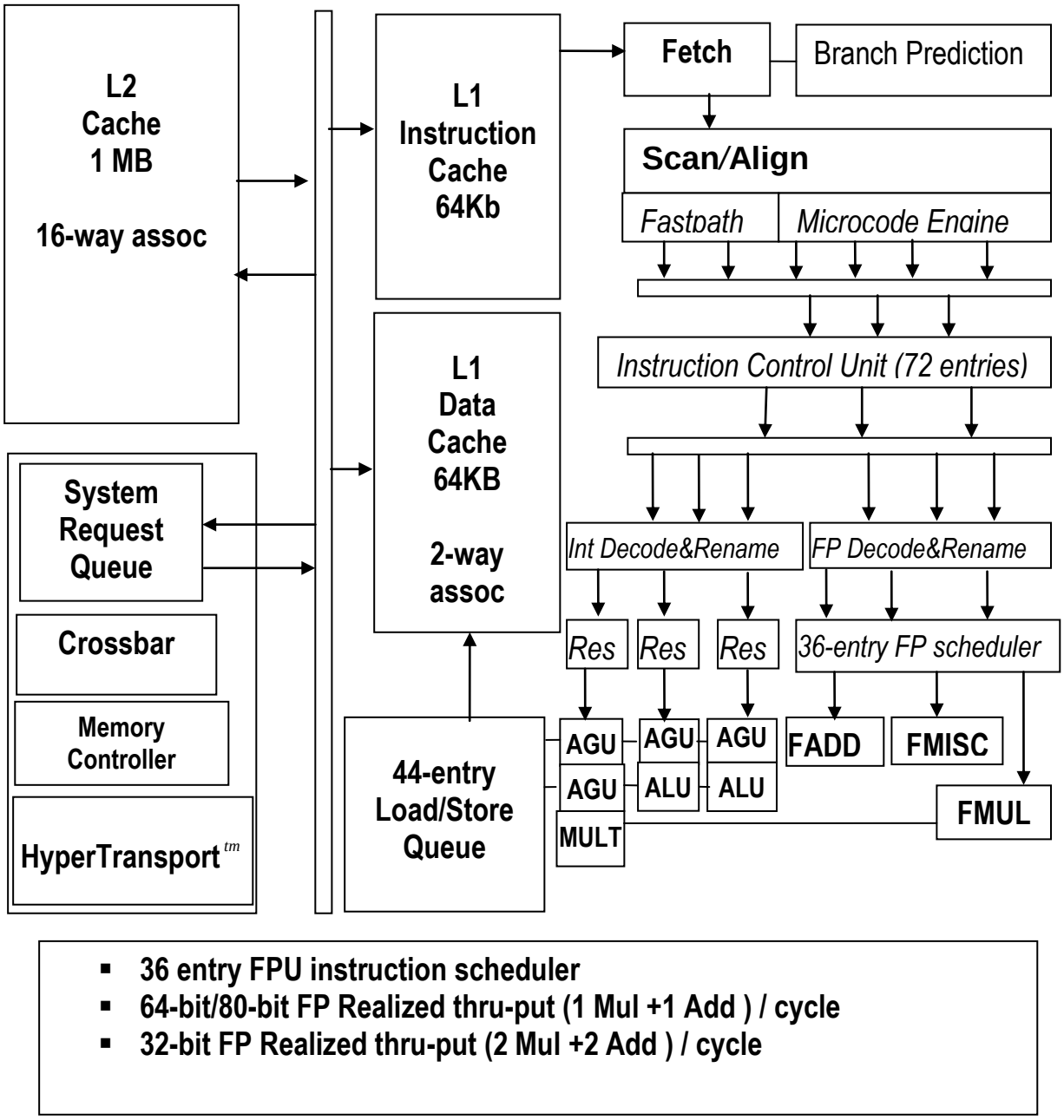

Рис. 4.8. Схема процессор AMD Opteron, используемого в Cray XT3

Вычислительные узлы Cray XT3 компонуются в стойки (до 96 вычислительных узлов на стойку) (рис. 4.8). Возможные конфигурации приводятся в табл. 4.7. Пиковая производительность рассчитывалась для конфигураций, использующих процессор AMD Opteron 2,4 ГГц.

Таблица 4.7

Возможные конфигурации вычислительных узлов в стойках Cray XT3

Число стоек	Число процессоров	Объем памяти	Пиковая производительность	Топология
6	548	4,3 Тбайт	2,6 Tflops	6*12*8
24	2260	17,7 Тбайт	10,8 Tflops	12*12*16
96	9108	71,2 Тбайт	43,7 Tflops	24*16*24
320	30508	239 Тбайт	147 Tflops	40*32*24

Cray XT3 работает под управлением ОС UNICOS/lc, позволяющей эффективно объединять до 30000 вычислительных узлов. На компьютере устанавливаются компиляторы Fortran 77, 90, 95, C/C++, коммуникационные библиотеки MPI (с поддержкой стандарта MPI 2.0) и SHMEM, а также оптимизированные версии библиотек BLAS, FFTs, LAPACK, ScaLAPACK и SuperLU. Для анализа производительности системы устанавливается система Cray Apprentice performance analysis tools.

Суперкомпьютер XT4 является дальнейшим развитием системы Cray XT3, отличаясь от неё в основном использованием двухъядерных процессоров AMD Opteron. Cray XT4 состоит из следующих компонентов: процессорные элементы, в каждом из которых есть процессор и память; коммуникационная сеть с малой латентностью и высокой пропускной способностью; оптимизированная версия операционной системы; специальные средства управления и мониторинга системы; высокоскоростные подсистемы ввода/вывода.

Вычислительные узлы Cray XT4 компонуются в стойки (до 96 вычислительных узлов на стойку). Возможные конфигурации приводятся в табл. 4.8. Пиковая производительность рассчитывалась для конфигураций, использующих двухъядерный процессор AMD Opteron 2,6 ГГц.

Таблица 4.8

Возможные конфигурации вычислительных узлов в стойках Cray XT4

Число стоек	Число процессоров	Объем памяти	Пиковая производительность	Топология
6	548	4.3 Тбайт	5.6 Tflop/s	6*12*8
24	2260	17.7 Тбайт	23.4 Tflop/s	12*12*16
96	9108	71.2 Тбайт	94.6 Tflop/s	24*16*24
320	30508	239 Тбайт	318 Tflop/s	40*32*24

Как и в предыдущих MPP-системах компании Cray, базовым элементом систем Cray XT4 является однопроцессорный элемент. Каждый процессорный элемент состоит из одного микропроцессора AMD (одно-, двух- или четырехъядерного), имеющего собственную память и средства связи с другими процессорными элементами. Такая архитектура устраняет проблему асимметричности, которая есть в кластерах SMP-узлов, поскольку производительность приложений будет одинаковой вне зависимости от конкретного распределения процессов по процессорам, что важно для обеспечения масштабируемости.

В системах Cray XT4 есть два типа процессорных элементов: вычислительные и сервисные.

На вычислительных процессорных элементах работает легковесное ядро операционной системы UNICOS, имеющее минимальные накладные расходы, основной задачей которого является эффективная поддержка работы приложений. На сервисных процессорных элементах установлена ОС Linux, а сами сервисные процессорные элементы могут быть настроены для выполнения сетевых, системных функций, для идентификации пользователей, а также для выполнения функций ввода/вывода.

Конструктивно 4 вычислительных процессорных элемента объединяются в Cray XT4 в один вычислительный сервер-лезвие, что обеспечивает хорошую масштабируемость системы при очень небольшом объеме. Сервисные серверы-лезвия содержат по 2 сервис-

ных процессорных элемента и поддерживают прямую связь с устройствами ввода/вывода.

Рассмотрим несколько более подробно процессорный элемент.

В процессорных элементах используются процессоры AMD Opteron. Кэш-память данных с высокой степенью ассоциативности, расположенная на кристалле процессора AMD, поддерживает мощные алгоритмы внеочередного выполнения команд и может выдавать до 9 инструкций одновременно. Встроенный контроллер памяти устраняет необходимость в использовании отдельного чипа контроллера памяти типа Northbridge, и при этом обеспечивает очень низкую латентность при доступе к локальной памяти – менее 60 наносекунд, что является существенным подспорьем для эффективного выполнения программ, особенно для алгоритмов с неоднородным доступом к памяти. Контроллер памяти с шириной тракта 128 бит обеспечивает каждому процессору AMD Opteron доступ к локальной памяти со скоростью от 10,6 до 12,8 ГБ/с, т.е. более одного байта на каждый Флоп. Такое соотношение позволяет получить высокую производительность на алгоритмах, для которых необходим интенсивный доступ к памяти.

Технология HyperTransport обеспечивает прямое соединение со скоростью 6,4 ГБ/с между процессором и коммуникационной сетью Cray XT4, устраняя распространенное узкое место, которым в большинстве сетей является интерфейс PCI.

Каждый процессорный элемент Cray XT4 может использовать от 1 до 8 ГБ оперативной памяти DDR2. Поскольку в вычислительных процессорных элементах память не буферизуется, задержка при доступе к ней минимальна.

В системах Cray XT4 используется коммуникационная сеть, с высокой пропускной способностью и малой латентностью. Она построена на основе микросхемы Cray SeaStar2, представленной на рис. 4.9, и высокоскоростных соединений, использующих технологию HyperTransport.

Коммуникационная сеть соединяет все процессорные элементы согласно топологии трехмерного тора, исключая необходимость использования коммутаторов. Такой подход повышает надежность системы и позволяет без особых затрат увеличивать количество узлов до десятков тысяч, что намного превышает возможности архитектур с переключателями типа fat-tree. Поскольку эта сеть является основным коммуникационным каналом систем Cray XT4, она отвечает и за все проходящие сообщения, и за весь входящий/исходящий трафик, связанный с глобальной файловой системой.

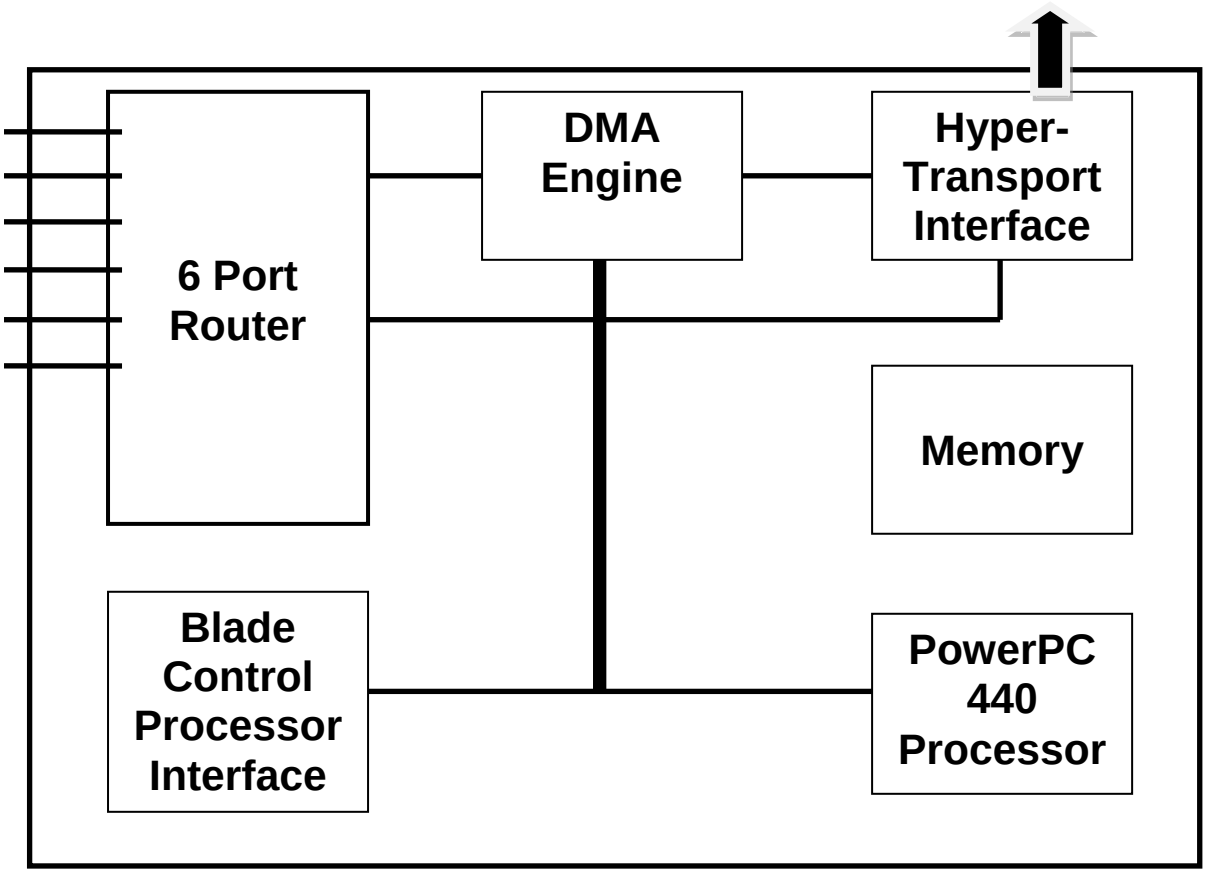

Рис. 4.9. Архитектура микросхемы SeaStar2

Микросхема Cray SeaStar2 выполняет две функции: обработку данных, поступающих по каналам связи, и высокоскоростную маршрутизацию. Каждая микросхема состоит из канала HyperTransport, модуля прямого доступа к памяти (DMA), процессора передачи дан-

ных и управления, высокоскоростного маршрутизатора сети и сервисного порта.

Маршрутизатор микросхемы SeaStar2 создает 6 высокоскоростных соединений с шестью непосредственными соседями по топологии трехмерного тора. Пиковая пропускная способность каждого канала при двусторонней передаче данных – 7,6 ГБ/с. Маршрутизатор использует протокол связи, поддерживающий контроль и коррекцию ошибок, а также повторную передачу.

Микросхема Cray SeaStar2 включает в себя модуль DMA и связанный с ним процессор PowerPC 440. Их задачей является снятие с процессора функций подготовки и демультиплексирования сообщений, что позволяет сосредоточить все его ресурсы исключительно на вычислениях. Логика SeaStar2 соответствует логике отправки и получения MPI-сообщений, поэтому нет необходимости использовать большие буфера памяти, которые необходимы при работе обычных кластерных систем. Модуль DMA и операционная система Cray XT4 позволяют минимизировать латентность путем установления прямой связи между приложением и аппаратурой сети, минуя проход через ядро операционной системы и связанную с этим систему прерываний.

Каждый из шести каналов микросхемы использует протокол, поддерживающий вычисление контрольной суммы и автоматическую повторную передачу. При наличии неустойчивого соединения канал может быть запущен в режиме с сокращенным набором ресурсов, но при этом соединение не будет разорвано.

Микросхема Cray SeaStar2 содержит сервисный порт, который соединяет независимую сеть управления и локальную шину Cray SeaStar2. Этот порт обеспечивает системе управление доступом к памяти и ко всем регистрам компьютера, облегчает его начальную загрузку, поддержку текущей работы и мониторинг.

Операционная система UNICOS/lc систем Cray XT4 разработана для выполнения ресурсоемких приложений при масштабировании до 120000 процессоров. Как и в предыдущих поколениях MPP-систем компании Cray, UNICOS/lc состоит из двух основных компонентов –

микроядра для вычислительных процессорных элементах и полнофункциональной операционной системы для сервисных процессорных элементов.

Микроядро взаимодействует с процессом приложения лишь ограниченным набором функций таким, как управление адресацией, виртуальной памятью, предоставление средств безопасности и обеспечение базовых средств планирования. Микроядро является легковесным, причем никаких дополнительных слоев между приложением и аппаратурой нет.

На сервисных процессорных элементах, показанных на рис. 4.10, как правило, работает полнофункциональная версия операционной системы Linux. Процессорные элементы этого типа могут быть настроены на выполнение сетевых или системных служб, а также функций ввода/вывода и идентификации пользователей в системе.

Процессорные элементы, предназначенные для идентификации, предоставляют программисту систему с полным доступом к среде разработки и ко всем стандартным утилитам, командам и консолям Linux. Это сделано для создания привычного окружения, удобства разработки нового и переносимости уже существующего программного обеспечения. Сетевые процессорные элементы обеспечивают высокоскоростное соединение с другими системами, а процессорный элемент ввода/вывода поддерживают масштабируемую связь с глобальной параллельной файловой системой. Системные процессорные элементы используются для поддержки работы глобальных системных служб таких, как работа с базами данных. Эти службы могут масштабироваться до размеров всей системы или лишь до того уровня, который необходим пользователям.

Приложения пользователей передаются на исполнение интерактивно через процессорный элемент идентификации с помощью явных команд запуска заданий Cray XT4 или с помощью системы пакетной обработки PBS Pro, тесно связанной с планировщиком системных процессорных элементов. Для выполнения каждого приложения на компьютере выделяется отдельный набор вычислительных процессор-

ных элементов, не пересекающийся с наборами, выделенными для других приложений. Конфигурации разделов для пакетной обработки и интерактивного исполнения заданий определяются системным администратором.

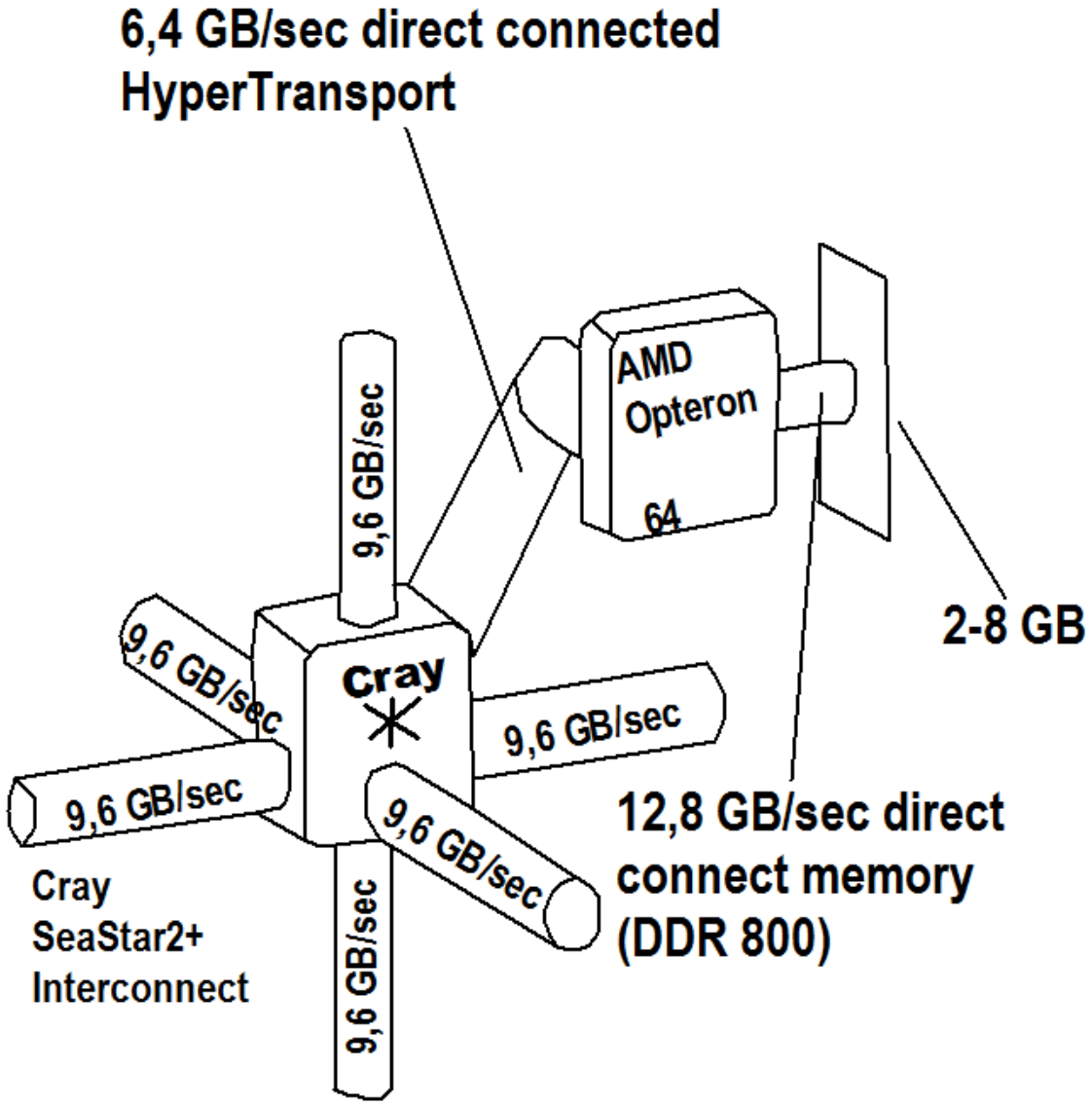

Рис. 4.10. Коммуникационная сеть на основе микросхем Cray SeaStar2+

Системы Cray XT4 поддерживают единую файловую систему с общим по всем узлам корневым разделом, поэтому нет необходимости пересылать изменения в каждый процессорный элемент отдельно, поскольку изменения становятся видны сразу во всей системе.

Среда разработки систем Cray XT4 включает в себя программные инструменты, которые хорошо дополняют друг друга, помогая в разработке масштабируемых приложений. Поскольку процессоры AMD поддерживают 32- и 64-битные приложения и полностью совместимы с линейкой x86-64, на системах Cray XT4 работает большинство известных компиляторов и библиотек, в том числе оптимизирующие компиляторы для C, C++, Fortran90, а также многие математические библиотеки, например, оптимизированные версии BLAS, FFTs, LAPACK, ScaLAPACK и SuperLU.

Библиотеки передач сообщений включают MPI и SHMEM. Реализация MPI соответствует стандарту 2.0 и оптимизирована под особенности коммуникационной сети систем Cray XT4. Пропускная способность по количеству проходящих сообщений может меняться в зависимости от размеров системы, достигающей десятков тысяч процессорных элементов. Библиотека SHMEM совместима с предыдущими версиями систем Cray и использует все возможности микросхемы Cray SeaStar2 для минимизации накладных расходов.

В составе штатного программного обеспечения систем Cray XT4 традиционно присутствуют средства анализа производительности семейства Cray Apprentice2. Они позволяют пользователям оценивать использование ресурсов компьютера их программами и помогают обнаружить многие узкие места, в частности, несбалансированность загрузки процессоров.

Система управления и мониторинга (CRMS) объединяет аппаратные и программные компоненты для мониторинга состояния компьютера, обнаружения неисправностей и восстановления после сбоев. Будучи независимой системой с собственными управляющими процессорами и выделенной сетью, CRMS следит и управляет всеми основными аппаратными и программными компонентами систем Cray XT4.

Помимо функций восстановления после сбоев, CRMS контролирует процессы включения, выключения и начальной загрузки системы, управляет коммуникационной сетью и отображает состояние

компьютера. Все критические компоненты систем Cray XT4 продублированы, чтобы минимизировать количество сбоев. Если из-за неполадок перестанет работать процессорный элемент ввода/вывода, это не скажется на задании, использовавшем этот процессорный элемент. Могут выйти из строя процессор или локальная память некоторого процессорного элемента, но это никак не повлияет на задания, использующие этот узел для транзита сообщений. На системных платах нет ни одной двигающейся компоненты, что также повышает надежность системы в целом.

Процессор и платы ввода/вывода систем Cray XT4 используют компоненты, подключаемые через сокеты, поэтому микросхемы SeaStar2 и DIMM, модули контроля напряжения и процессоры AMD в случае необходимости могут быть легко заменены и модернизированы.

Подсистема ввода/вывода компьютеров Cray XT4 хорошо масштабируется, подстраиваясь под требования самых тяжелых приложений. Архитектура подсистемы включает дисковые массивы, которые напрямую соединены с процессорным элементом ввода/вывода через высокоскоростную сеть. Файловая система Lustre управляет распределением операций с файлами по всему хранилищу. Масштабируемая архитектура ввода/вывода позволяет в каждой конфигурации компьютера получить требуемую пропускную способность путем подбора соответствующего числа массивов и сервисных процессорных элементов.

Для повышения производительности средств ввода/вывода модули Lustre встраиваются прямо в приложения и выполняются на системном микроядре. Данные передаются напрямую из приложений и серверов Lustre на процессорные элементы ввода/вывода, без необходимости создания промежуточной копии данных в самом ядре.

Система Cray XT5 под кодовым названием Jaguar, созданная на базе шестиядерных процессоров AMD Opteron, признана самым мощным в мире суперкомпьютером согласно версии списка TOP500 за июнь 2010. Эта система, работающая в Национальной лаборатории

Oak Ridge (ORNL) в США, была недавно модернизирована с четырехъядерных до шестиядерных процессоров AMD Opteron 2.6GHz. Jaguar содержит 224,256 высокопроизводительных ядер и теоретически обладает пиковым быстродействием 2,3 Pflops, демонстрируя быстродействие 1,75 Pflops в тестах Linpack. Отметим, что 4-е место в рейтинге TOP-500 за июнь 2010 занимает Cray XT5 Craken с 98 928 ядрами, производительностью 0,83 Pflops в тестах Linpack и пиковой производительностью 1,03 Pflops.

Узел XT5, представленный на рис. 4.10, содержит ASIC-микросхему, обеспечивающую работу с коммуникационным соединением SeaStar2+, два процессорных разъема для Opteron и слоты для модулей DIMM. В Cray XT4 узел был устроен аналогичным образом, но имел один процессорный разъем.

Для связи коммуникационного соединения с Opteron естественным образом применяются интегрированные в микропроцессор интерфейсы каналов HyperTransport 2.0 (прямое подсоединение оперативной памяти к процессору – одна из основных особенностей всех моделей Opteron). В результате пропускная способность оперативной памяти масштабируется с числом процессоров, и в расчете на узел пропускная способность памяти составляет 25,6 Гбайт/с (применяется защищенная кодами ECC регистровая память DDR2-800). Емкость оперативной памяти узла составляет от 8 Гбайт до 32 Гбайт.

Процессоры в узле XT5 могут применяться разные – четырехъядерные Barcelona и Shanghai с разными частотами, а также шестиядерные Istanbul. Соответственно суммарное число ядер в узле составляет восемь или двенадцать, а пиковая производительность узла лежит в диапазоне примерно от 70 Gflops до 124 Gflops.

Следующий уровень конструктива над узлом XT5 – лезвие (blade), содержащее четыре узла. При использовании Istanbul производительность лезвия достигает 500 Gflops; ему отвечает решетка узлов 1x2x2.

В стойке можно реализовать решетку узлов 1x4x24 (24 лезвия по четыре узла) с производительностью до 12 Tflops при емкости оперативной памяти 1,54 Тбайт (из расчета 16 Гбайт на узел), а в системе XT5 в целом – решетку 25x32x24. Такой системе отвечает производительность порядка 2 Gflops и емкость памяти 300 Тбайт. В соответствии с указанными на сайте

Cray спецификациями XT5, стойка занимает площадь примерно 0,6м x 1,4 м при высоте около 2 м, весит порядка 700 кг и потребляет не более 43 кВт.

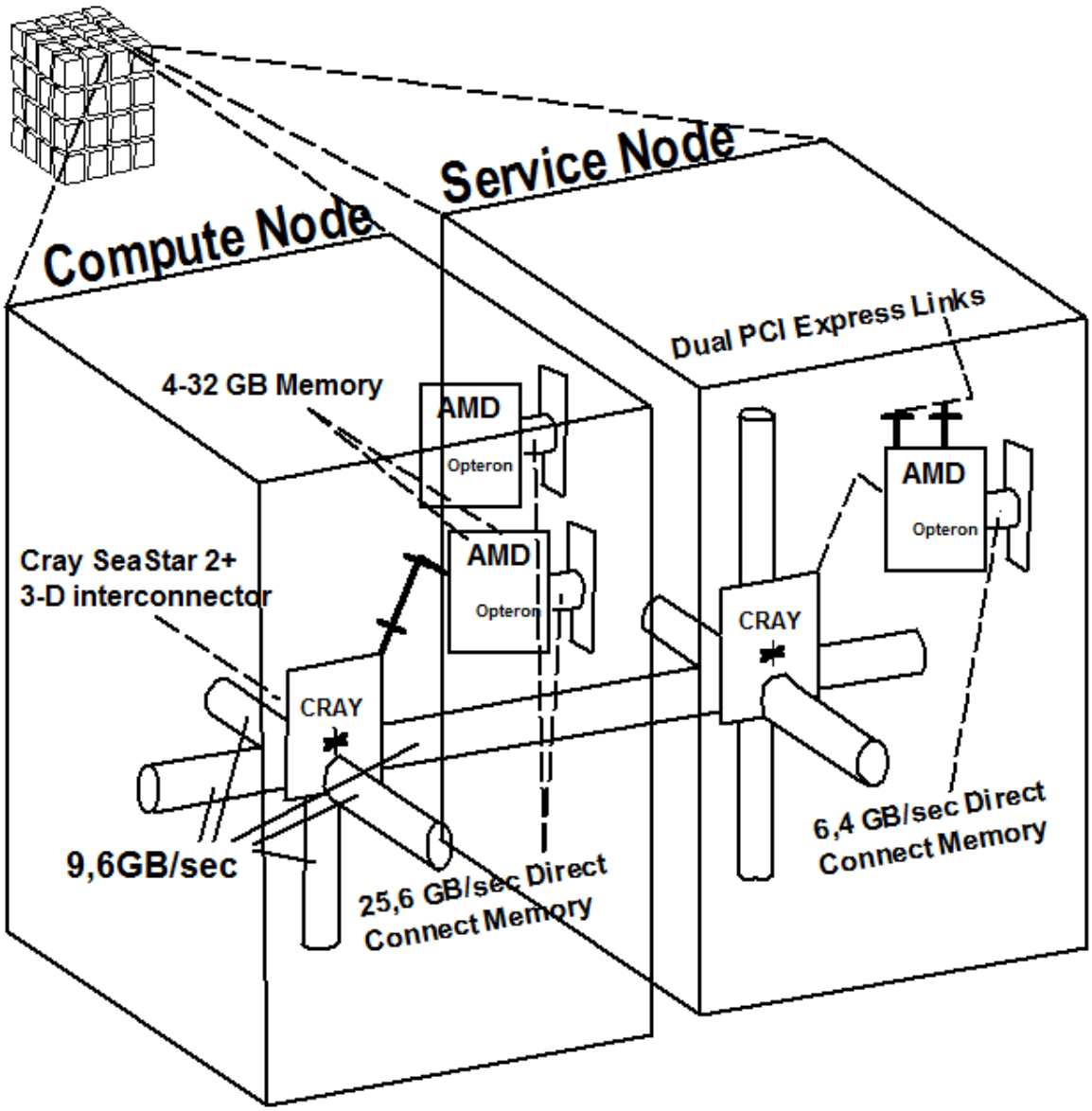

Рис. 4.11. Архитектура и вычислительный узел Cray XT5

Коммуникационное соединение SeaStar2+ поддерживается благодаря соответствующей микросхеме, обеспечивающей маршрутизацию и имеющей шесть портов с пропускной способностью 9,6 Гбайт/с на порт. Это отвечает суммарной пропускной способности микросхемы в 57,6 Гбайт/с, в то время как задержка передачи из узла в узел не превышает 2 мкс. Кроме шестипортового маршрутизатора и интерфейса HyperTransport, микросхе-

ма SeaStar2+ содержит механизм прямого доступа в память (Direct Memory Access, DMA), поддержку интерфейса управления лезвием и другие средства.

Для повышения надежности в SeaStar2+ применяются коды ECC, используется адаптивная маршрутизация, а сбойные каналы можно «обойти» без перезагрузки. В SeaStar для передачи данных не надо устанавливать соединения, поэтому нет кэширующих очередей между парами узлов. Это облегчает работу Cray XT с большим числом узлов. Удвоение числа виртуальных каналов (до четырех) по сравнению с предыдущим поколением SeaStar позволило поднять поддерживаемую пропускную способность на 30%, которая в результате составила 6 Гбайт/с.

Заметим, что более высокая пропускная способность коммуникационного соединения уже могла бы натолкнуться на узкое место в виде канала HyperTransport (6,4 Гбайт/с для HyperTransport 2.0). Для сравнения, пропускная способность современного варианта InfiniBand 4x QDR составляет 4 Гбайт/с для однонаправленной передачи при аппаратных задержках на уровне меньше 2 мкс.

Таким образом, аппаратура и узлов, и коммуникационного соединения Cray XT5 по производительности не превосходит соответствующие параметры InfiniBand-кластеров. Выгоды Cray XT5 связаны в основном с возможностями построения систем со сверхбольшим числом узлов.

В заключение отметим, что в настоящее время фирмой Cray продолжаются разработки и уже конструируются системы Cray XT6.

Контрольные вопросы для самопроверки

Какова производительность и другие характеристики самого мощного суперкомпьютера из списка Top500?

Какова производительность и другие характеристики самого мощного суперкомпьютера из списка Top50?

Сравните производительность суперкомпьютеров из списков Top500 и Top50. Во сколько раз отличается производительность?

Каковы принципы формирования списка Top500?

Как формируется список Top50?

Какова архитектура суперкомпьютеров BlueGene фирмы IBM?

Каковы отличия характеристик суперкомпьютеров BlueGene/L и BlueGene/P?

Почему суперкомпьютер Roadrunner можно называть кластером кластеров?

Какова архитектура суперкомпьютеров серии Cray XT4?

За счет чего увеличена производительность суперкомпьютеров Cray XT5?

Какова архитектура суперкомпьютеров Jaguar фирмы Cray?

5. ПРОГРАММИРОВАНИЕ ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ

5.1. ДВЕ ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

С появлением параллельных систем возникли новые проблемы, связанные с вопросами, как обеспечить эффективное решение задач на той или иной параллельной системе, и какими критериями эффективности следует пользоваться. Появились проблема описания класса тех задач, которые естественно решать на данной параллельной системе, а также класс задач, не поддающихся эффективному распараллеливанию. Следующая проблема связана с собственно методом распараллеливания алгоритма. Затем желательно обеспечить переносимость полученной программы на систему с другой архитектурой. И, наконец, требуется сохранить работоспособность программы и улучшить ее характеристики при модификации данной системы; в частности, как обеспечить работоспособность программы при увеличении количества параллельных модулей.

Распараллеливание алгоритмов и написание параллельных программ – весьма сложное дело. Причины возникающих трудностей различны: с одной стороны, идеология распараллеливания трудно воспринимается после приобретения навыков последовательного программирования, а с другой стороны, методы распараллеливания задач недостаточно разработаны.

В настоящее время существуют два основных подхода к программированию параллельных вычислений, эти подходы в литературе

часто называют парадигмами программирования. Прежде чем обсудить эти подходы, сделаем несколько замечаний общего характера.

Развитие фундаментальных и прикладных наук, технологий требует применения все более мощных и эффективных методов и средств обработки информации. В качестве примера можно привести разработку реалистических математических моделей, которые часто оказываются настолько сложными, что не допускают точного аналитического их исследования. Единственная возможность исследования таких моделей, их верификации (то есть подтверждения правильности) и использования для прогноза – компьютерное моделирование, применение методов численного анализа.

Другая важная проблема – обработка больших объемов информации в режиме реального времени. Все эти проблемы могут быть решены лишь на достаточно мощной аппаратной базе, с применением эффективных методов программирования. Основными особенностями модели параллельного программирования являются высокая эффективность программ, применение специальных приемов программирования и, как следствие, более высокая трудоемкость программирования, проблемы с переносимостью программ.

В настоящее время существуют два основных подхода к распараллеливанию вычислений. Это параллелизм данных и параллелизм задач. В англоязычной литературе соответствующие термины – data parallel и message passing. В основе обоих подходов лежит распределение вычислительной работы по доступным пользователю процессорам параллельного компьютера.

При этом приходится решать разнообразные проблемы. Прежде всего, это достаточно равномерная загрузка процессоров, так как если основная вычислительная работа будет ложиться на один из процессоров, мы приходим к случаю обычных последовательных вычислений, и в этом случае никакого выигрыша за счет распараллеливания задачи не будет. Сбалансированная работа процессоров – это первая проблема, которую следует решить при организации параллельных вычислений.

Другая и не менее важная проблема – скорость обмена информацией между процессорами. Если вычисления выполняются на высокопроизводительных процессорах, загрузка которых достаточно равномерная, но скорость обмена данными низкая, основная часть времени будет тратиться впустую на ожидание информации, необходимой для дальнейшей работы данного процессора. Рассматриваемые парадигмы программирования различаются методами решения этих двух основных проблем. Разберем более подробно параллелизм данных и параллелизм задач.

Модель параллелизма данных основывается на параллелизме, который заключается в регулярном манипулировании элементами больших монолитных структур данных – в применении одной и той же операции к множеству элементов структур данных таких, как массивы. Желательно различные фрагменты такого массива обрабатывать на векторно-конвейерном процессоре или на разных процессорах параллельной машины. Распределением данных между процессорами также занимается программа. Векторизация или распараллеливание в этом случае чаще всего выполняется уже на этапе компиляции – переводе исходного текста программы в машинные команды. Роль программиста в этом случае обычно сводится к заданию опций векторной или параллельной оптимизации компилятору, директив параллельной компиляции, использованию специализированных языков для параллельных вычислений.

Модель параллельных данных не обладает свойствами, привязывающими ее к какой-то конкретной параллельной архитектуре. Основным архитектурным признаком этой модели является то, что компьютер, реализующий эту модель, работает одновременно с набором слов памяти, а не с каждым словом отдельно, как это происходит в последовательном компьютере. Это в свою очередь находит отражение в языках программирования с параллелизмом данных, в том смысле, что в них набор элементов составляет "параллельный" объект, в котором каждый элемент можно интерпретировать как содержимое памяти одного из процессоров машины с параллелизмом дан-

ных. Наиболее распространенными языками для параллельных вычислений являются Высокопроизводительный ФОРТРАН (High Performance FORTRAN – HPF) и параллельные версии языка C (это, например, C*).

Стиль программирования, основанный на параллелизме задач, подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач, и каждый процессор загружается своей собственной подзадачей. Компьютер при этом представляет собой MIMD-машину.

Аббревиатура MIMD обозначает в известной классификации архитектур компьютеров Флинна компьютер, выполняющий одновременно множество различных операций над множеством, вообще говоря, различных и разнотипных данных. Для каждой подзадачи пишется своя собственная программа на обычном языке программирования, например на ФОРТРАНе или C. Чем больше подзадач, тем большее число процессоров можно использовать, тем большей эффективности можно добиться.

Важно то, что все эти программы должны обмениваться результатами своей работы, практически такой обмен осуществляется вызовом процедур специализированной библиотеки. Программист при этом может контролировать распределение данных между процессорами и подзадачами и обмен данными. Очевидно, что в этом случае требуется определенная работа для того, чтобы обеспечить эффективное совместное выполнение различных программ. По сравнению с подходом, основанным на параллелизме данных, данный подход более трудоемкий.

Привлекательными особенностями подхода на основе параллелизма задач являются большая гибкость и большая свобода, предоставляемая программисту в разработке программы, эффективно использующей ресурсы параллельного компьютера и, как следствие, возможность достижения максимального быстродействия.

Примерами специализированных библиотек являются библиотеки MPI (Message Passing Interface) и PVM (Parallel Virtual Machines).

Эти библиотеки являются свободно распространяемыми и существуют в исходных кодах. Библиотека MPI разработана в Арагонской Национальной Лаборатории (США), а PVM – разработка Окриджской Национальной Лаборатории, университетов штата Теннеси и Эмори (Атланта).

5.2. МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Самый простой вариант попробовать ускорить имеющуюся программу – это воспользоваться встроенными в транслятор (обычно с ФОРТРАНа или Си) средствами векторизации или распараллеливания. При этом никаких изменений в программу вносить не придется. Однако вероятность существенного ускорения в разы или десятки раз невелика. Трансляторы с ФОРТРАНа и Си векторизуют и распараллеливают программы очень аккуратно, и при любых сомнениях в независимости обрабатываемых данных оптимизация не проводится. Поэтому, кстати, и не приходится ожидать ошибок от компиляторов, если программист явно не указывает компилятору выполнить векторную или параллельную оптимизацию какой-либо части программы.

Второй этап работы с такой программой – анализ затрачиваемого времени разными частями программы и определение наиболее ресурсопотребляющих частей. Последующие усилия должны быть направлены именно на оптимизацию этих частей. В программах наиболее затратными являются циклы, и усилия компилятора направлены, прежде всего, на векторизацию и распараллеливание циклов. Диагностика компилятора поможет установить причины, мешающие векторизовать и распараллелить циклы. Возможно, что простыми действиями удастся устранить эти причины. Это может быть простое исправление стиля программы, перестановка местами операторов (цикла и условных), разделение одного цикла на несколько, удаление из критических частей программы лишних операторов типа операторов отладочной печати. Небольшие усилия могут дать здесь весьма существенный выигрыш в быстродействии.

Третий этап – замена алгоритма вычислений в наиболее критичных частях программы. Способы написания оптимальных с точки зрения быстродействия программ существенно отличаются в двух парадигмах программирования – в последовательной и в параллельной (векторной). Поэтому программа, оптимальная для скалярного процессора, с большой вероятностью не может быть векторизована или распараллелена. В то же время специальным образом написанная программа для векторных или параллельных компьютеров будет исполняться на скалярных машинах довольно медленно.

Замена алгоритма в наиболее критических частях программы может привести к серьезному ускорению программы при относительно небольших потраченных усилиях. Дополнительные возможности предоставляют специальные векторные и параллельные библиотеки подпрограмм. Используя библиотечные функции, которые оптимизированы для конкретной ЭВМ, можно упростить себе задачу по написанию и отладке программы. Единственный недостаток данного подхода состоит в том, что программа может стать непереносимой на другие машины (даже того же класса), если на них не окажется аналогичной библиотеки.

Написание программы "с нуля" одинаково сложно (или одинаково просто) для машин любых типов. Этот способ является идеальным для разработки эффективных, высокопроизводительных векторных или параллельных программ. Начинать надо с изучения специфики программирования для векторных и параллельных ЭВМ, изучения алгоритмов, которые наиболее эффективно реализуются на ЭВМ данных типов. После этого надо проанализировать поставленную задачу и определить возможность применения векторизуемых и распараллеливаемых алгоритмов для решения конкретной задачи. Возможно, что придется переформулировать какие-то части задачи, чтобы они решались с применением векторных или параллельных алгоритмов. Программа, специально написанная для векторных или параллельных ЭВМ, даст наибольшее ускорение при ее векторизации и распараллеливании.

Описываемая методология проектирования предлагает подход к распараллеливанию, который рассматривает машинно-независимые аспекты реализации алгоритма, такие как параллелизм, на первой стадии, а особенности проектирования, связанные с конкретным параллельным компьютером, – на второй. Данный подход разделяет процесс проектирования на четыре отдельных этапа: декомпозиция (partitioning), коммуникации (communications), кластеризация (agglomeration) и распределение (mapping). Первые два этапа призваны выделить в исходной задаче параллелизм и масштабируемость, остальные этапы связаны с различными аспектами производительности алгоритма.

На этапе декомпозиции, который будет рассмотрен более подробно в следующем параграфе, общая задача вычислений и обработки данных делится на подзадачи меньшего размера. При этом игнорируются проблемы практической реализации, например, число процессоров используемого в будущем компьютера. Напротив, все внимание сосредотачивается на возможном параллелизме исходной задачи. Далее определяется требуемая связь между подзадачами, ее структура и конкретные алгоритмы коммуникаций. При кластеризации структура подзадач и коммуникаций оценивается с учетом требований производительности алгоритма и затрат на реализацию.

Если необходимо, отдельные подзадачи комбинируются в более крупные с целью улучшения производительности и снижения затрат на разработку. Каждая подзадача назначается определенному процессору, при этом стараются максимально равномерно загрузить процессоры и минимизировать коммуникации.

Распределение может быть статическим или формироваться в процессе выполнения программы на основе алгоритмов балансировки загрузки (load balancing algorithms). Следует заметить, что при прохождении последних двух этапов необходимо учитывать архитектуру параллельного компьютера, имеющегося в распоряжении разработчика, другими словами параллельный алгоритм должен быть адекватен используемой параллельной системе.

5.3. ДЕКОМПОЗИЦИЯ ДЛЯ ВЫДЕЛЕНИЯ ПАРАЛЛЕЛИЗМА

Предположим, что мы имеем последовательную программу, которую мы хотим выполнить быстрее, исполняя ее на параллельном оборудовании, в частности, на нескольких компьютерах с распределенной памятью. Мы предполагаем, что все последовательные программы состоят из сугубо последовательных частей и потенциально параллельных частей. Поэтому выполнение последовательной программы – это выполнение ее сугубо последовательных частей и ее потенциально параллельных частей.

Так или иначе, при расщеплении (или декомпозиции) имеются действия, включенные в потенциально параллельную часть, которые могут выполняться на нескольких процессорах одновременно, тогда наша цель состоит в сокращении времени выполнения этой части. Однако декомпозиция часто включает определенные накладные расходы: например, передачу сообщений между процессорами. Мы должны рассматривать эти накладные расходы во времени выполнения параллельной версии подпрограммы, и они не должны превышать выигранного времени исполнения в результате использования нескольких процессоров параллельно. Если это достигается, то мы будем иметь сокращение времени исполнения подпрограммы и соответственно времени исполнения всей программы целиком. На рис. 5.1 показаны важнейшие методы декомпозиции и их связь.

Тривиальная декомпозиция предполагает, что имеется последовательная программа, которая может исполняться независимо от большого числа различных исходных данных. Можно ввести параллелизм путем запуска некоторого числа этих программ параллельно. Так как нет зависимости между разными запусками программы, то число процессоров, которые могут быть использованы, ограничено только числом исполняемых запусков. В этом случае время исполнения набора запусков будет временем исполнения самого длинного по времени запуска в наборе.

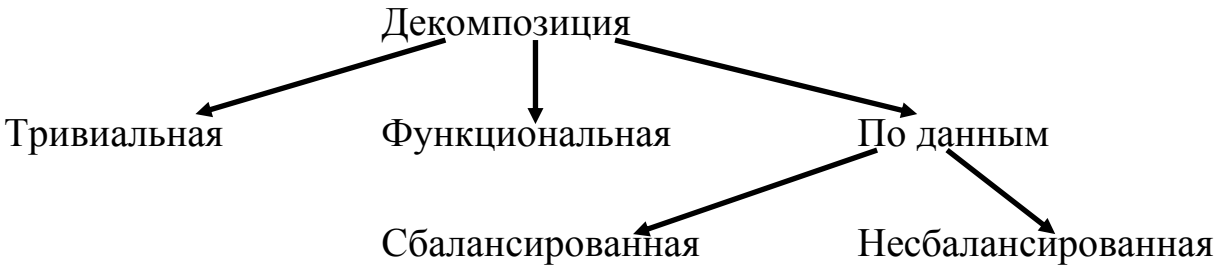

Рис. 5.1. Методы декомпозиции

Ясно, что тривиальный параллелизм может быть использован для почти линейного ускорения, если запуски требуют примерно одинакового времени. Поскольку не требуется никаких связей между процессами, этот метод может быть эффективно использован на кластерах рабочих станций со слабой пропускной способностью сети.

Функциональная декомпозиция – это первый настоящий метод декомпозиции, разбивающий программу на подпрограммы. Простая форма функциональной декомпозиции называется "конвейер", которая очень схожа с конвейерной обработкой команд в процессоре, рассмотренной выше. Пример конвейера для обработки изображения представлен на рис. 5.2. Пример демонстрирует, как входные данные проходит через каждую подпрограмму в заданном порядке.

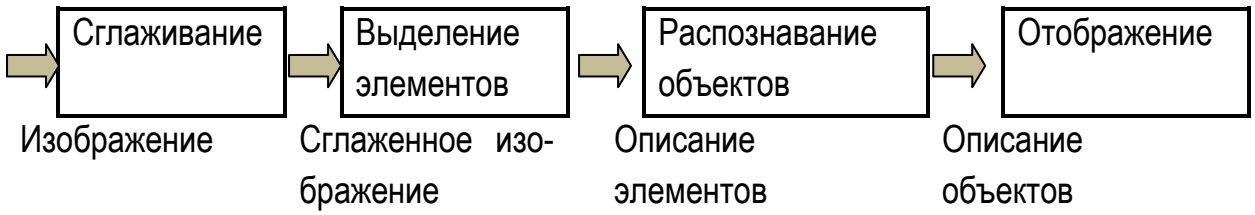

Рис. 5.2. Пример функциональной декомпозиции

В примере на рис. 5.2 параллелизм появляется, когда несколько входных порций данных перемещаются по конвейеру совместно. Конвейер первоначально пуст. Первая порция данных поступает в первую стадию конвейера (сглаживание). Как только эта порция сглажена, она передается на следующую стадию конвейера (выделение элементов). В то время как первая порция проходит стадию выде-

ления элементов, вторая порция может поступать на стадию сглаживания. Параллелизм появляется, когда вторая порция данных, проходя через конвейер, находится там одновременно с первым элементом. Этот процесс заполнения продолжается, пока все части конвейера продвигают вперед данные. Когда набор данных исчерпывается, появляется период просачивания, во время которого число занятых стадий в конвейере падает до нуля.

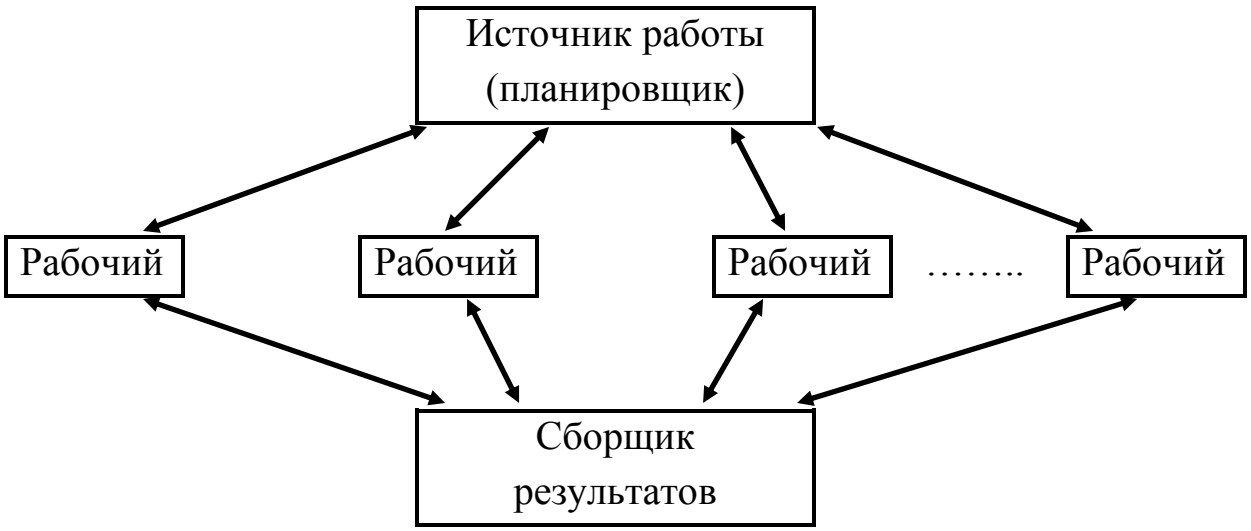

Рис. 5.3. Простейшая ферма задач

Другой вид функциональной декомпозиции – это «фермы задач». В английском языке существует термин "*farm*", дословно переводящийся как "ферма", для обозначения способа организации многомашинной системы, при котором одна из машин действует как планировщик, а другие – как рабочие. Как уже говорилось выше, отдельная задача в функциональной декомпозиции может быть распределена между некоторым числом процессоров. Эта технология обычно так и называется – ферма задач (*task farming)*. Для обработки выбираются небольшие порции данных и один процессор, *источник работы*, или планировщик, который управляет набором необработанных порций данных. Некоторое число *рабочих* процессоров периодически запрашивают порцию данных из источника, обрабатывают ее и затем отправляют результаты к *сборщику* результатов, который может совпа-

дать или нет с источником работы. Изображение основных компонент этого метода показано на рис. 5.3.

Преимущество этой технологии состоит в том, что она не требует предварительных предположений о наборе данных. Если порции данных достаточно малы, равномерная загрузка будет обеспечена за счет того, что рабочие процессоры, которые получают порции данных, не требующих большого объема работы над ними, будут просто запрашивать больше порций данных. Часто возможно построить независимые задания для процессоров по задачам, в которых имеется зависимость данных, путем включения дополнительной информации (например, значений из соседних граничных точек) в обрабатываемые порции данных.

Однако работа фермы задач требует постоянного потока запросов и ответов для частиц между рабочими машинами и управляющей. Стоимости, связанные с поддержанием динамического баланса загрузки, являются существенными в случае, когда выполняется много итераций над набором данных. Вследствие этого необходима повторная передача на каждой итерации.

Рассматривая декомпозицию по данным, следует отметить, что многие задачи связаны с обработкой очень больших наборов данных, распределенных в регулярной сеточной структуре, при этом применяются некоторые операции трансформации над элементами данных. Когда данные могут быть разделены на регулярные подсетки и распределены между несколькими процессами, тогда преобразования могут быть применены параллельно, что позволяет решить задачу в меньшие сроки, чем это было бы возможно в обычных условиях. В методе декомпозиции регулярной области берется большая сетка элементов данных, разбивается на регулярные подсетки (блоки данных), и эти блоки распределяются по отдельным процессам, где они обрабатываются.

5.4. КОНЦЕПЦИЯ ПЕРЕДАЧИ СООБЩЕНИЙ

В модели программирования на основе передачи сообщений, каждый процесс имеет локальную память, и никакие другие процессы

не могут непосредственно иметь доступ на чтение/запись к этой локальной памяти. Более того, модель программирования с распределенной памятью предполагает, что нет глобально адресуемой памяти как в модели с параллельной памятью. Довольно общая ситуация для параллельных компьютеров – поддерживать несколько моделей программирования; и таким образом ответственность за решение, какая из этих моделей подходит лучше для приложения, перекладывается на программиста.

Для прояснения целей, мы будем обращаться к отдельному процессу, координированному в параллельных вычислениях, как образующему параллельную программу или просто программу. Термин *процесс* будет использоваться только для обозначения единичной вычислительной активности на отдельном процессоре. Программы с передачей сообщений пишутся на тех же самых языках, что и обычные последовательные программы, однако в любой точке программист определяет действия процесса, выполняющего часть вычисления, а не описывает действия всей программы.

Параллельное программирование по определению включает взаимодействие между программами для решения общей задачи. Имеется две стороны вопроса программирования процессов, которые взаимодействуют друг с другом. Сначала программист должен определить процессы, которые будут выполняться процессорами, а также определить, как эти процессоры синхронизируются и обмениваются данными между собой. Центральный момент модели обмена сообщениями состоит в том, что процессы связываются и синхронизируют свою работу путем посылки друг другу сообщений. Поскольку процессоры всегда чем-то заняты, операции передачи сообщений только посылают запрос на интерфейс передачи сообщения, который ответственен за физическое соединение по сети реальных процессоров вместе. Архитектура параллельных компьютеров, поддерживающих модель передачи, сообщений представлена на рис. 5.4.

Такая архитектура уже рассматривалась выше и называется архитектурой с распределенной памятью, а соответствующие парал-

лельные компьютеры называются *мультикомпьютерами*, когда топология соединительной сети известна перед началом вычислений. Но иногда возникает ситуация, когда топология сети рабочих станций может меняться во время вычислений и, обычно, не известна программисту.

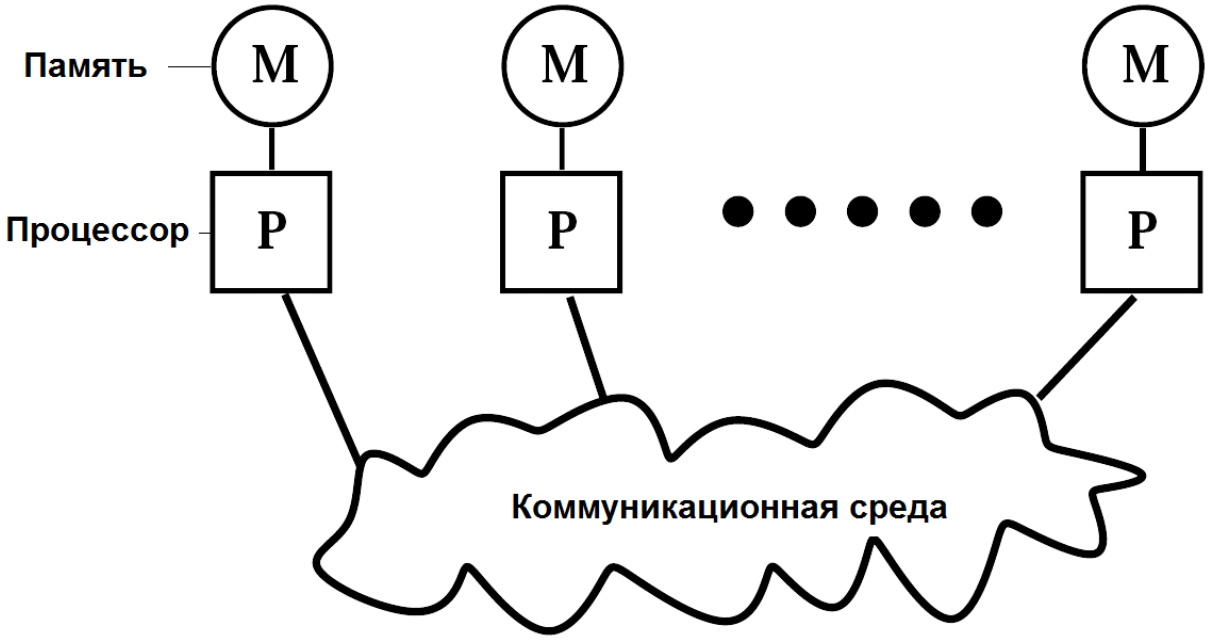

Рис. 5.4. Архитектура параллельных компьютеров, для модели передачи сообщений

Чаще всего встречается ситуация, в которой программист пишет единственный исходный код программы, который компилируется и соединяется на клиентском компьютере. Получающийся в результате объектный код копируется в локальную память каждого процессора, принимая участие в вычислениях. Параллельная программа выполняется путем принуждения всех процессоров исполнять тот же самый объектный код. Эта модель параллельных вычислений называется *Одна-Программа-Много-Данных (Single-Program*-Multiple-Data – *SPMD)*. Если все переменные с одним и тем же именем имели одно и то же значение, и если все эти процессы имели доступ к тем же самым данным, они не должны быть параллельными вовсе, каждый процесс должен действовать точно также как все остальные.

В контексте передачи сообщений коммуникационный интерфейс обычно включает функции или процедуры, которые возвращают значение, идентифицирующее процесс, которое называет эту функцию или процедуру. Более того, то же самое значение будет всегда возвращаться к тому же самому процессу. В этом случае будем называть это значение идентификатором процесса. Обычно, последовательность инструкций, выполняемая процессом, будет определяться входными данными и идентификатором этого процесса.

Отличительной особенностью модели SPMD является то, что все процессы являются процессами того же самого типа. В теории, это ограничение не влияет на выразительность парадигмы передачи сообщений. Все мы должны надлежащим образом запускать процессы различного типа, записывая условные операторы, чьи ветви вызывают соответствующий код для каждого процесса, в зависимости от его соответствующего идентификатора. На практике имеются случаи, когда SPMD модель не подходит.

Например, согласно SPMD модели выполнение всех типов процессов должно сопровождаться их загрузкой на каждый процессор. В зависимости от приложения это может занимать слишком большое количество памяти каждого процессора. Альтернативная модель параллельных вычислений называется *Много-Программ-Много-Данных* (Multiple-Program-Multiple-Data – SPMD), где различные загрузочные модули могут быть исполнены на различных процессорах.

Модель MPMD предполагает возможность загружать объектный код на процессор во время параллельных вычислений и является очень естественной на сети рабочих станций. Многие поставщики мультикомпьютеров упрощают своё системное программное обеспечение путем обеспечения только параллельной среды, которая поддерживает SPMD программы, обеспечение MPMD модели на мультикомпьютерах является активной исследовательской задачей.

Сообщения являются центральным понятием для модели передачи сообщений – они передаются между процессорами. Когда два процесса обмениваются сообщением, данные копируются из локаль-

ной памяти одного процесса в локальную память другого. Данные пересылаются в сообщении, состоящем из двух частей: содержание и оболочка.

Содержание сообщения – это данные пользователя, которые не интерпретируются ни коммуникационным интерфейсом, ни коммуникационной системой, стоящей за этим интерфейсом. Напротив, данные в оболочке используются коммуникационной системой для копирования содержания сообщения между локальными памятями, в частности, должна быть определена следующая информация:

- Какой процессор посылает сообщение?
- Где находятся данные на посылающем процессоре?
- Какой тип данных посылается. Сколько этих данных?
- Какой процессор принимает сообщение?
- Где данные должны быть помещены на принимающем процессоре?
- Сколько данных принимающий процессор готовится принять?

В общем случае процессы посылки и приема будут кооперироваться в обеспечении этой информации. Некоторая из этой информации, которая обеспечивается посылающим процессом, будет присоединяться к сообщению при его путешествии через систему. Другая информация может быть обеспечена принимающим процессом.

5.5. ПЕРЕДАЧА СООБЩЕНИЙ

Простейшая форма сообщения – это связь точка-точка, в котором сообщение посылается от посылающего процесса к принимающему. Только эти два процесса должны знать об этом сообщении. Сообщение само по себе состоит из двух операций: отсылка и получение. *Синхронная* отсылка завершается только тогда, когда о соответствующем сообщении заботится некоторый принимающий процесс, как показано на рис. 5.5.

Асинхронная отсылка завершается, как только соответствующее сообщение доставлено коммуникационной системой, как показано на рис. 5.6.

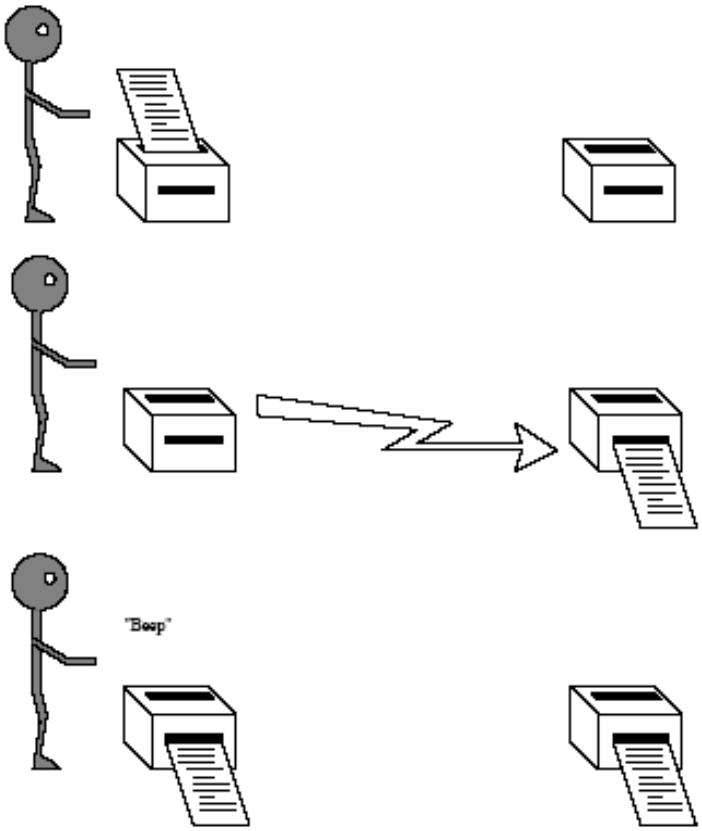

Рис. 5.5. Синхронная связь не завершается до тех пор, пока сообщение не будет принято

Например, факсовое сообщение или заказное письмо являются синхронными операциями. Отправитель всегда знает, было ли его сообщение доставлено. Почтовая карточка является асинхронным сообщением. Отправитель только знает, что положил его в почтовый ящик, но не имеет информации относительно того, было ли оно доставлено, если только получатель не пошлет ответ.

Имеется две основные формы коммуникационных процедур: блокирующие и неблокирующие. Это разделение основано на времени завершении операции коммуникации и процедуре, которая инициирует коммуникационную операцию. Блокирующие процедуры отдают управление только тогда, когда соответствующая коммуникация завершается. Неблокирующие процедуры, как показано на рис. 5.7, отдают управление сразу же и позволяют процессу продолжать дру-

гие действия. Некоторое время спустя процесс может протестировать завершение соответствующей коммуникации.

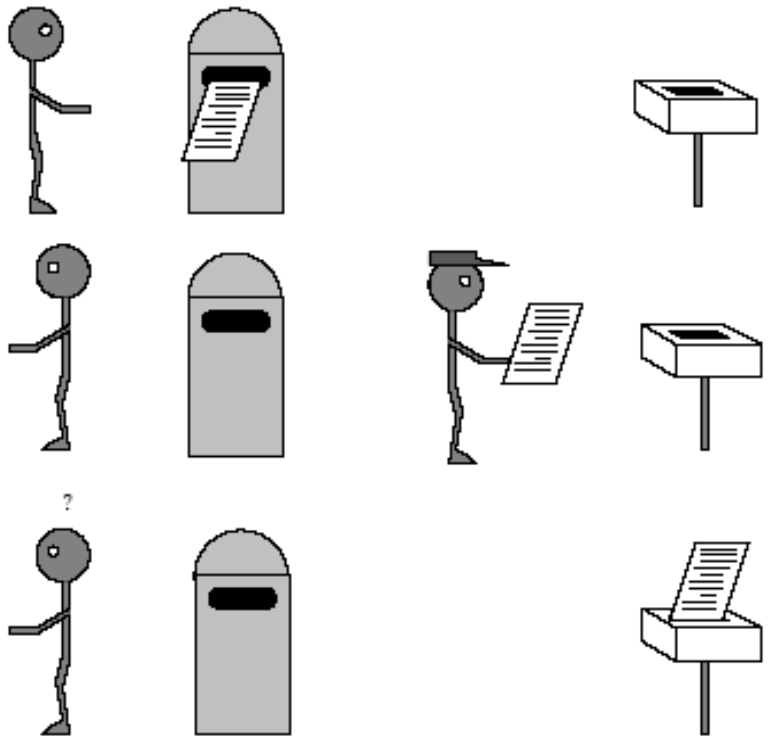

Рис. 5.6. Асинхронная связь завершается, как только сообщение находится в конце своего пути

Обыкновенная факс-машина обеспечивает блокирующую коммуникацию. Факс остается занятым до тех пор, пока сообщение не будет отослано. Это позволяет вам загружать документы в память, и если удаленный номер занят, машина может продолжать пытаться справиться с отсылкой, пока вы ходите и делаете что-либо более важное. Это является неблокирующей операцией. Получение сообщения может быть неблокирующей операцией. Например, если Вы включили факс-машину и оставили её включенной, так что сообщение может прибыть. Вы затем периодически тестируете её путем захода в комнату с факсом и просмотра – пришло ли сообщение.

До этого момента мы рассматривали коммуникации, которые включали только пары процессов. Многие системы передачи сообщений также обеспечивают операции, которые позволяют коммуника-

ции для большого числа процессов. Ниже мы рассмотрим 3 основных типа коллективных коммуникаций.

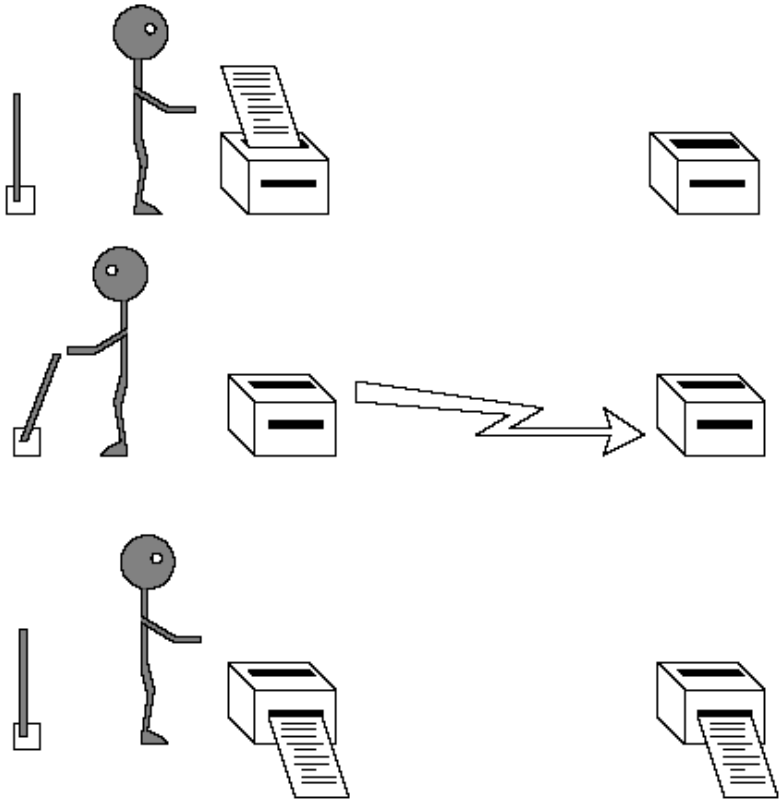

Рис. 5.7. Неблокирующие коммуникации позволяют выполнить полезные действия во время ожидания завершения коммуникации

Барьерная операция синхронизирует процессоры. Не происходит обмена данными, но барьер блокирует до тех пор пока все участвующие процессы не вызовут процедуру барьера, как показано на рис. 5.8.

Вещание – это коммуникация «один-ко-многим». Один процесс посылает сообщение нескольким адресатам за одну операцию, как показано на рис. 5.9.

Операции сокращения берут несколько единиц данных от нескольких процессов и сокращают их до одной единицы данных, которая может стать, а может и не стать доступной для всех участвующих процессов, как показано на рис. 5.10.

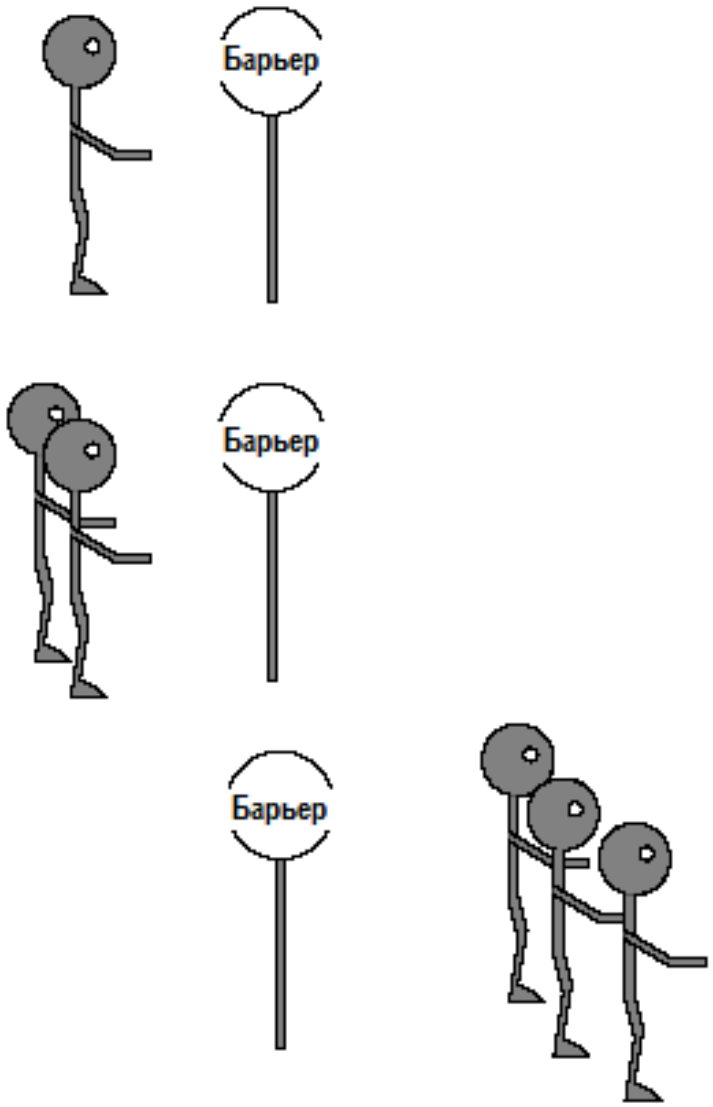

Рис. 5.8. Операция барьера синхронизирует процессы

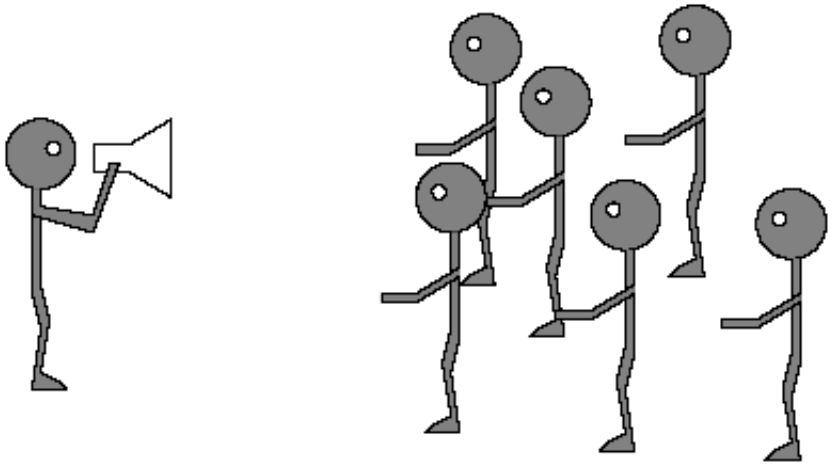

Рис. 5.9. Вещание посылает сообщение нескольким адресатам

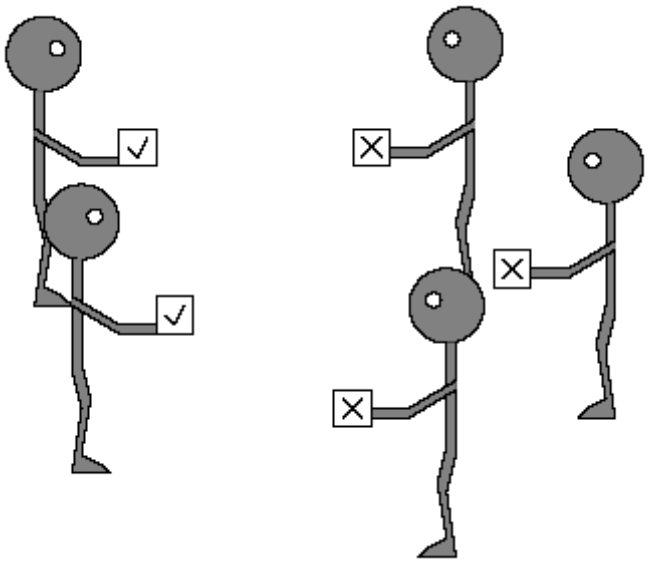

Рис. 5.10. Операция сокращения – на выходе одно значение

Один пример операции сокращения – это голосование нажатием, где тысячи голосов сокращаются до одного решения. Другой пример операции сокращения – это суммирование всех значений, сохраненных во всех переменных, с именем *x*, в каждом процессе, принимающем участие в операции.

Парадигма передачи сообщений становится все более популярной в последнее время. Одна причина этого состоит в том, что широкий круг платформ поддерживает эту модель. Программы, написанные в стиле передачи сообщений, могут выполняться на распределенных процессорах или с разделяемой памятью, на сети рабочих станций, или даже на однопроцессорных системах. Аргумент переносимости в пользу парадигмы передачи сообщений может ослабевать с годами, по мере того как другие модели программирования становятся более широко поддерживаемыми поставщиками. С другой стороны, две характеристики парадигмы передачи сообщений кажутся важными в настоящее время: возможность писать коммуникационные ядра, которые полностью используют топологию коммуникационной сети и возможность менять по время выполнения программы способы распределения данных по процессорам.

Передача сообщений была описана как программирование низкого уровня для параллельных компьютеров. Конечно, возможно использование парадигмы сообщений для написания исходного кода, который использует взаимосвязанную сеть мультикомпьютеров. С другой стороны, интерфейс передачи сообщений, такой, как MPI или PVM, делает возможным написание высококачественного программного обеспечения, которое является переносимым на различные платформы без жертвования производительностью. Передача сообщений популярна не потому, что она чрезвычайно проста, а т.к. она так обща.

Контрольные вопросы для самопроверки

Расскажите о двух парадигмах программирования

Что такое параллелизм данных?

Что такое параллелизм задач?

На каком наборе операций базируется подход, основанный на параллелизме данных?

Расскажите методологию проектирования параллельных алгоритмов.

Что такое коммуникации, кластеризация, распределение?

Что такое балансировка нагрузки процессоров?

Какие парадигмы программирования Вы знаете?

Какие типы декомпозиция вы знаете?

Когда применяется тривиальная декомпозиция?

Каковы особенности функциональной декомпозиции?

Что такое фермы задач?

Какие бывают виды межкомпьютерного общения?

В чём разница синхронной/асинхронной отсылок?

Какие бывают блокирующие и неблокирующие процедуры?

Какие могут быть операции коллективных коммуникаций?

6. ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ОСНОВЕ MPI

6.1. ПРЕИМУЩЕСТВА ПРОГРАММИРОВАНИЯ НА MPI

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга. Для организации параллельных вычислений в таких условиях необходимо иметь возможность распределять вычислительную нагрузку и организовать информационное взаимодействие (передачу данных) между процессорами. Решение этих вопросов и обеспечивает интерфейс передачи сообщений (message passing interface – MPI). При этом для распределения вычислений между процессорами рамках MPI принят простой подход – для решения поставленной задачи разрабатывается одна программа, и эта единственная программа запускается одновременно на выполнение на всех имеющихся процессорах.

В MPI для того, чтобы избежать идентичности вычислений на разных процессорах, можно сделать следующее. Во-первых, подставлять разные данные для программы на разных процессорах. Во-вторых, в MPI имеются средства для идентификации процессора, на котором выполняется программа и тем самым, предоставляется возможность организовать различия в вычислениях в зависимости от используемого программой процессора. Подобный способ организации параллельных вычислений получил наименование, как уже указывалось выше, модели «одна программа – множество процессов» (SPMP).

Для организации информационного взаимодействия между процессорами в самом минимальном варианте достаточно операции приема и передачи данных. В MPI существует целое множество операций передачи данных. Они обеспечивают разные способы пересылки данных, реализуют практически все рассмотренные в предыдущем разделе коммуникационные операции. Именно данные возможности являются наиболее сильной стороной MPI – об этом, в частности, свидетельствует и само название MPI.

Следует отметить, что попытки создания программных средств передачи данных между процессорами начались предприниматься практически сразу с появлением локальных компьютерных сетей. Однако подобные средства часто были неполными и, самое главное, являлись несовместимыми. Таким образом, одна из самых серьезных проблем в программировании – переносимость программ при переводе программного обеспечения на другие компьютерные системы – проявлялась при разработке параллельных программ в самой максимальной степени.

В результате, уже с 90-х годов 20 века стали предприниматься усилия по стандартизации средств организации передачи сообщений в многопроцессорных вычислительных системах. Началом работ, непосредственно приведших к появлению MPI, послужило проведение рабочего совещания по стандартам для передачи сообщений в среде распределенной памяти (the Workshop on Standards for Message Passing in a Distributed Memory Environment, Williamsburg, Virginia, USA, April 1992). По итогам совещания была образована рабочая группа, позднее преобразованная в международное сообщество MPI Forum, результатом деятельности которого явилось создание и принятие в 1994 г. стандарта интерфейса передачи сообщений (message passing interface - MPI) версии 1.0. В последующие годы стандарт MPI последовательно развивался, и в 1997 г. был принят стандарт MPI версии 2.0.

Существуют также некоторые диалекты (реализации библиотек), среди которых отметим MPICH2 – быстродействующая и широко портируемая реализация MPI. Для нее характерна эффективная поддержка различных платформ: общедоступные кластеры, высокоскоростные сети и эксклюзивные высокопроизводительные системы, например, Blue Gene, Cray, SiCortex. Отметим также легко расширяемую модульную структуру. MPICH2 – это развитие предыдущей версии. В MPICH2 по сравнению с MPICH1 добавлен новый внутренний механизм управления процессами (процессы на узлах управляются специальным демоном mpd). Для MPICH2 характерно ускорение

межпроцессных обменов данными, особенно, коллективных, а также поддержка стандарта MPI-2.

В современном виде MPI – это стандарт, которому должны удовлетворять средства организации передачи сообщений. MPI – это программные средства, которые обеспечивают возможность передачи сообщений и при этом соответствуют всем требованиям стандарта MPI. Так, по стандарту, эти программные средства должны быть организованы в виде библиотек программных модулей (библиотеки MPI) и должны быть доступны для наиболее широко используемых алгоритмических языков C и Fortran. Подобную "двойственность" MPI следует учитывать при использовании терминологии. Как правило, аббревиатура MPI используется для упоминания стандарта, а сочетание "библиотека MPI" указывает на ту или иную программную реализацию стандарта. Однако достаточно часто для краткости обозначение MPI используется и для библиотек MPI и, тем самым, для правильной интерпретации термина следует учитывать контекст.

Среди достоинств MPI можно отметить, что он позволяет в значительной степени снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами – параллельная программа, разработанная на алгоритмическом языке C или Fortran с использованием библиотеки MPI, как правило, будет работать на разных вычислительных платформах. Кроме того, MPI содействует повышению эффективности параллельных вычислений, поскольку в настоящее время практически для каждого типа вычислительных систем существуют реализации библиотек MPI, в максимальной степени учитывающие возможности используемого компьютерного оборудования. И, наконец, MPI уменьшает, в определенном плане, сложность разработки параллельных программ, т.к., с одной стороны, большая часть основных операций передачи данных предусматривается стандартом MPI, а с другой стороны, уже имеется большое количество библиотек параллельных методов, созданных с использованием MPI.

6.2. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько процессов – в этом случае их исполнение осуществляется в режиме разделения времени. В предельном случае для выполнения параллельной программы может использоваться один процессор – как правило, такой способ применяется для начальной проверки правильности параллельной программы.

Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (модель SPMP). Данный программный код, представленный в виде исполняемой программы, должен быть доступен в момент запуска параллельной программы на всех используемых процессорах. Исходный программный код для исполняемой программы разрабатывается на алгоритмических языках C или Fortran с использованием той или иной реализации библиотеки MPI.

Количество процессов и число используемых процессоров определяются в момент запуска параллельной программы средствами среды исполнения MPI-программ и в ходе вычислений меняться не могут (в стандарте MPI-2 предусматривается возможность динамического изменения количества процессов). Все процессы программы последовательно перенумерованы от 0 до p-1, где p есть общее количество процессов. Номер процесса именуется рангом процесса.

Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов. Для выполнения парных операций могут использоваться разные режимы передачи, среди которых синхронный, блокирующий и др.

Процессы параллельной программы объединяются в группы. Под коммуникатором в MPI понимается специально создаваемый

служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных. Как правило, парные операции передачи данных выполняются для процессов, принадлежащих одному и тому же коммуникатору.

Коллективные операции применяются одновременно для всех процессов коммуникатора. Как результат, указание используемого коммуникатора является обязательным для операций передачи данных в MPI. В ходе вычислений могут создаваться новые и удаляться существующие группы процессов и коммуникаторы. Один и тот же процесс может принадлежать разным группам и коммуникаторам. Все имеющиеся в параллельной программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI_COMM_WORLD. При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (intercommunicator).

При выполнении операций передачи сообщений, для указания передаваемых или получаемых данных в функциях MPI необходимо указывать тип пересылаемых данных. MPI содержит большой набор базовых типов данных, во многом совпадающих с типами данных в алгоритмических языках C и Fortran. Кроме того, в MPI имеются возможности для создания новых производных типов данных для более точного и краткого описания содержимого пересылаемых сообщений.

Как уже отмечалось ранее, парные операции передачи данных могут быть выполнены между любыми процессами одного и того же коммуникатора, а в коллективной операции принимают участие все процессы коммуникатора. В этом плане, логическая топология линий связи между процессами имеет структуру полного графа (независимо от наличия реальных физических каналов связи между процессорами).

В MPI имеется возможность представления множества процессов в виде решетки произвольной размерности. При этом граничные процессы решеток могут быть объявлены соседними и, тем самым, на

основе решеток могут быть определены структуры типа тор. Кроме того, в MPI имеются средства и для формирования логических (виртуальных) топологий любого требуемого типа.

Описание функций и все приводимые примеры программ далее будут представлены на алгоритмическом языке С. Можно отметить, что, с одной стороны, MPI достаточно сложен – в стандарте MPI предусматривается наличие более 125 функций. С другой стороны, структура MPI является тщательно продуманной – разработка параллельных программ может быть начата уже после рассмотрения всего лишь 6 функций MPI. Все дополнительные возможности MPI могут осваиваться по мере роста сложности разрабатываемых алгоритмов и программ.

6.3. БАЗОВЫЕ ФУНКЦИИ MPI

Для инициализации среды выполнения MPI-программы первой вызываемой функцией MPI должна быть функция (алгоритмический язык С):

```
int MPI_Init (int *agrc, char ***argv);
```

Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

```
int MPI_Finalize (void);
```

Как результат, можно отметить, что структура параллельной программы, разработанная с использованием MPI, должна иметь следующий вид:

```
#include "mpi.h"
int main  ( int argc,  char *argv[]  )  {
<программный код без использования MPI функций>
  MPI_Init( &agrc,  &argv ;
<программный код с использованием MPI функций>
   MPI_Finalize();
<программный код без использования MPI функций>
  return 0;
}
```

Следует отметить, что файл mpi.h содержит определения именованных констант, прототипов функций и типов данных библиотеки MPI. Кроме того, функции MPI_Init и MPI_Finalize являются обязательными и должны быть выполнены (и только один раз) каждым процессом параллельной программы. Перед вызовом MPI_Init может быть использована функция MPI_Initialized для определения того, был ли ранее выполнен вызов MPI_Init.

Рассмотренные примеры функций дают представление синтаксиса именования функций в MPI. Имени функции предшествует префикс MPI, далее следует одно или несколько слов названия, первое слово в имени функции начинается с заглавного символа, слова разделяются знаком подчеркивания. Названия функций MPI, как правило, поясняют назначение выполняемых функцией действий.

Определение количества процессов в выполняемой параллельной программе осуществляется при помощи функции:

```
int MPI_Comm_size(MPI Comm comm, int *size);
```

Для определения ранга процесса используется функция:

```
int MPI_Comm_rank(MPI Comm comm, int *rank);
```

Как правило, вызов функций MPI_Comm_size и MPI_Comm_rank выполняется сразу после MPI_Init:

```
#include "mpi.h"
int main   int argc,   char *argv[]   )   { int ProcNum,   ProcRank;
<программный код без использования MPI функций> MPI_Init(&agrc,   &argv );
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
MPI_Comm_rank( MPI_COMM_WORLD,   &ProcRank);
<программный код с использованием MPI функций>
MPI_Finalize();
<программный код без использования MPI функций> return 0;
}
```

Следует отметить, что коммуникатор MPI_COMM_WORLD, как отмечалось ранее, создается по умолчанию и представляет все процессы выполняемой параллельной программы. Ранг, получаемый при помощи функции MPI_Comm_rank, является рангом процесса, вы-

полнившего вызов этой функции, т.е. переменная ProcRank будет принимать различные значения в разных процессах.

Для передачи сообщения процесс-отправитель должен выполнить функцию:

```
int MPI_Send(void *buf, int count, MPI Datatype type, int dest, int tag,
                                                    MPI Comm comm);
```

Здесь buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения, count – количество элементов данных в сообщении, type – тип элементов данных пересылаемого сообщения, dest – ранг процесса, которому отправляется сообщение, tag – значение-тег, используемое для идентификации сообщений, comm – коммуникатор, в рамках которого выполняется передача данных. При этом следует отметить, что отправляемое сообщение определяется через указание блока памяти (буфера), в котором это сообщение располагается. Используемая для указания буфера триада (buf, count, type) входит в состав параметров практически всех функций передачи данных. Процессы, между которыми выполняется передача данных, в обязательном порядке должны принадлежать коммуникатору, указываемому в функции MPISend. Параметр tag используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное целое число. Сразу же после завершения функции MPISend процесс-отправитель может начать повторно использовать буфер памяти, в котором располагалось отправляемое сообщение. Вместе с этим, следует понимать, что в момент завершения функции MPISend состояние самого пересылаемого сообщения может быть совершенно различным – сообщение может располагаться в процессе-отправителе, может находиться в процессе передачи, может храниться в процессе-получателе или же может быть принято процессом-получателем при помощи функции MPIRecv. Тем самым, завершение функции MPISend означает лишь, что операция передачи начала выполняться и пересылка сообщения будет рано или поздно выполнена.

Для приема сообщения процесс-получатель должен выполнить функцию:

int MPI_Recv(void *buf, int count, MPI Datatype type, int source, int tag,
MPI Comm comm, MPI Status *status);

Здесь buf, count, type – буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в MPI Send, source – ранг процесса, от которого должен быть выполнен прием сообщения, tag – тег сообщения, которое должно быть принято для процесса, comm – коммуникатор, в рамках которого выполняется передача данных, status – указатель на структуру данных с информацией о результате выполнения операции приема данных. При этом отметим, что буфер памяти должен быть достаточным для приема сообщения, а типы элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна, и в коде завершения функции будет зафиксирована ошибка переполнения. При необходимости приема сообщения от любого процесса-отправителя для параметра source может быть указано значение MPI_ANY_SOURCE. При необходимости приема сообщения с любым тегом для параметра tag может быть указано значение MPI_ANY_TAG. Параметр status позволяет определить ряд характеристик принятого сообщения: status.MPI_SOURCE – ранг процесса-отправителя принятого сообщения, status.MPI_TAG – тег принятого сообщения.

Функция MPI_Get_count (MPI_Status *status, MPI_Datatype type, int *count) возвращает в переменной count количество элементов типа type в принятом сообщении. Вызов функции MPI_Recv не должен согласовываться со временем вызова соответствующей функции передачи сообщения MPI_Send – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.

По завершении функции MPI_Recv в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция MPI_Recv является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то при-

чинам ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.

6.4. ПРИМЕР ПРОГРАММЫ НА MPI

Рассмотренный набор функций оказывается достаточным для разработки параллельных программ. Приводимая ниже программа является стандартным начальным примером для алгоритмического языка С.

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[]){
   int ProcNum, ProcRank, RecvRank;
   MPI_Status Status;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
   MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
   if ( ProcRank == 0 ){
      // Действия, выполняемые только процессом с рангом 0
      printf ("\n Hello from process %3d", ProcRank);
     for ( int i=1; i<ProcNum; i++ ) {
         MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
                        MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
        printf("\n Hello from process %3d", RecvRank);
      }
 }
     else // Сообщение, отправляемое всеми процессами,
          // кроме процесса с рангом 0
      MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

Как следует из текста программы, каждый процесс определяет свой ранг, после чего действия в программе разделяются. Все процессы, кроме процесса с рангом 0, передают значение своего ранга нулевому процессу. Процесс с рангом 0 сначала печатает значение своего

ранга, а далее последовательно принимает сообщения с рангами процессов и также печатает их значения. При этом важно отметить, что порядок приема сообщений заранее не определен и зависит от условий выполнения параллельной программы (более того, этот порядок может изменяться от запуска к запуску). Так, возможный вариант результатов печати процесса 0 может состоять в следующем (для параллельной программы из четырех процессов):

```
Hello from process    0
Hello from process    2
Hello from process    l
Hello from process    3
```

Такой "плавающий" вид получаемых результатов существенным образом усложняет разработку, тестирование и отладку параллельных программ, т. к. в этом случае исчезает один из основных принципов программирования – повторяемость выполняемых вычислительных экспериментов. Как правило, если это не приводит к потере эффективности, следует обеспечивать однозначность расчетов и при использовании параллельных вычислений. Так, для рассматриваемого простого примера можно восстановить постоянство получаемых результатов при помощи задания ранга процесса-отправителя в операции приема сообщения:

```
MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
        MPI_COMM_WORLD, &Status);
```

Указание ранга процесса-отправителя регламентирует порядок приема сообщений, и, как результат, строки печати будут появляться строго в порядке возрастания рангов процессов. Повторим, что такая регламентация в отдельных ситуациях может приводить к замедлению выполняемых параллельных вычислений.

Следует отметить еще один важный момент – разрабатываемая с использованием MPI программа как в данном частном варианте, так и в самом общем случае используется для порождения всех процессов параллельной программы и, как результат, должна определять вычисления, выполняемые во всех этих процессах. Можно сказать, что MPI-программа является некоторым "макро-кодом", различные части ко-

торого используются разными процессами. Так, например, в приведенном примере программы существуют участки программного кода, которые не выполняются одновременно ни в одном процессе. Первый участок с функцией приема MPISend исполняется только процессом с рангом 0, второй участок с функцией приема MPIRecv используется всеми процессами, за исключением нулевого процесса.

Для разделения фрагментов кода между процессами обычно используется подход, примененный в только что рассмотренной программе – при помощи функции MPI_Comm_rank определяется ранг процесса, а затем в соответствии с рангом выделяются необходимые для процесса участки программного кода. Наличие в одной и той же программе фрагментов кода разных процессов также значительно усложняет понимание и, в целом, разработку MPI-программы. Как результат, можно рекомендовать при увеличении объема разрабатываемых программ выносить программный код разных процессов в отдельные программные модули (функции). Общая схема MPI программы в этом случае будет иметь вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

Во многих случаях, как и в рассмотренном примере, выполняемые действия являются отличающимися только для процесса с рангом 0. В этом случае общая схема MPI программы принимает более простой вид:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoManagerProcess();
else DoWorkerProcesses();
```

В завершение обсуждения примера поясним использованный в MPI подход для контроля правильности выполнения функций – все функции MPI возвращают в качестве своего значения код завершения. При успешном выполнении функции возвращаемый код равен MPI_SUCCESS. Другие значения кода завершения свидетельствуют об обнаружении тех или иных ошибочных ситуаций в ходе выполне-

ния функций. Для выяснения типа обнаруженной ошибки используются предопределенные именованные константы, полный список которых для проверки кода завершения содержится в файле mpi.h.

6.5. КОЛЛЕКТИВНЫЕ ОПЕРАЦИИ ПЕРЕДАЧИ ДАННЫХ

Функции MPISend и MPIRecv обеспечивают возможность выполнения парных операций передачи данных между двумя процессами параллельной программы. Для выполнения коммуникационных коллективных операций, в которых принимают участие все процессы коммуникатора, в MPI предусмотрен специальный набор функций. Для демонстрации примеров применения рассматриваемых функций MPI будет использоваться учебная задача суммирования элементов вектора x. Разработка параллельного алгоритма для решения данной задачи не вызывает затруднений – необходимо разделить данные на равные блоки, передать эти блоки процессам, выполнить в процессах суммирование полученных данных, собрать значения вычисленных частных сумм на одном из процессов и сложить значения частичных сумм для получения общего результата решаемой задачи. При последующей разработке демонстрационных программ данный рассмотренный алгоритм будет несколько упрощен – процессам программы будут передаваться весь суммируемый вектор, а не отдельные блоки этого вектора.

Первая проблема при выполнении рассмотренного параллельного алгоритма суммирования состоит в необходимости передачи значений вектора x всем процессам параллельной программы. Конечно, для решения этой проблемы можно воспользоваться функциями парных операций передачи данных:

```
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
for (i=1; i<ProcNum; i++)
MPI_Send(&x,n,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
```

Однако такое решение будет крайне неэффективным, поскольку повторение операций передачи приводит к суммированию затрат (латентностей) на подготовку передаваемых сообщений. Кроме того, данная операция может быть выполнена всего за $\log_2 p$ итераций пере-

дачи данных. Достижение эффективного выполнения операции передачи данных от одного процесса всем процессам программы (широковещательная рассылка данных) может быть обеспечено при помощи функции MPI:

int MPI_Bcast(void *buf, int count, MPI_Datatype type, int root,
MPI_Comm comm),

где buf, count, type – буфер памяти с отправляемым сообщением (для процесса с рангом 0) и для приема сообщений для всех остальных процессов, root – ранг процесса, выполняющего рассылку данных, comm – коммуникатор, в рамках которого выполняется передача данных.

Функция MPI_Bcast осуществляет рассылку данных из буфера buf, содержащего count элементов типа type с процесса, имеющего номер root, всем процессам, входящим в коммуникатор comm. При этом следует отметить, что функция MPI_Bcast определяет коллективную операцию и, тем самым, при выполнении необходимых рассылок данных вызов функции MPI_Bcast должен быть осуществлен всеми процессами указываемого коммуникатора. Указываемый в функции MPI_Bcast буфер памяти имеет различное назначение в разных процессах. Для процесса с рангом root, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение. Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

Приведем программу для решения учебной задачи суммирования элементов вектора с использованием рассмотренной функции.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char* argv[]){
  double x[100], TotalSum, ProcSum = 0.0;
  int ProcRank, ProcNum, N=100;
  MPI_Status Status;
      // инициализация
```

```
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
// подготовка данных
  if ( ProcRank == 0 ) DataInitialization(x,N);
// рассылка данных на все процессы
  MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// вычисление частичной суммы на каждом из процессов
// на каждом процессе суммируются элементы вектора x от i1 до i2
  int k = N / ProcNum;
  int i1 = k * ProcRank;
  int i2 = k * ( ProcRank + 1 );
  if ( ProcRank == ProcNum-1 ) i2 = N;
  for ( int i = i1; i < i2; i++ )
      ProcSum = ProcSum + x[i];
// сборка частичных сумм на процессе с рангом 0
  if ( ProcRank == 0 ) {
     TotalSum = ProcSum;
     for ( int i=1; i < ProcNum; i++ ) {
         MPI_Recv(&ProcSum, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0,
                  MPI_COMM_WORLD,  &Status);
         TotalSum = TotalSum + ProcSum;
     }
  }
  else // все процессы отсылают свои частичные суммы
   MPI_Send(&ProcSum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
  // вывод результата
  if ( ProcRank == 0 )
      printf("\nTotal Sum = %10.2f",TotalSum);
    MPI_Finalize();
 }
```

В приведенной программе функция DataInitialization осуществляет подготовку начальных данных. Необходимые данные могут быть введены с клавиатуры, прочитаны из файла или сгенерированы при

помощи датчика случайных чисел – подготовка этой функции предоставляется как задание для самостоятельной разработки.

В рассмотренной программе суммирования числовых значений имеющаяся процедура сбора и последующего суммирования данных является примером часто выполняемой коллективной операции передачи данных от всех процессов одному процессу. В этой операции над собираемыми значениями осуществляется та или иная обработка данных. Для уточнения последнего момента данная операция именуется операцией редукции данных. Как и ранее, реализация операции редукции при помощи обычных парных операций передачи данных является неэффективной и достаточно трудоемкой. Для наилучшего выполнения действий, связанных с редукцией данных, в MPI предусмотрена функция:

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
MPI_Datatype type, MPI_Op op,int root,MPI_Comm comm);
```

Здесь sendbuf – буфер памяти с отправляемым сообщением, recvbuf – буфер памяти для результирующего сообщения (только для процесса с рангом root), count – количество элементов в сообщениях, type – тип элементов сообщений, op – операция, которая должна быть выполнена над данными, root – ранг процесса, на котором должен быть получен результат, comm – коммуникатор, в рамках которого выполняется операция.

В качестве операций редукции данных могут быть использованы предопределенные в MPI операции, представленные в табл. 6.1.

Следует отметить, что функция MPI_Reduce определяет коллективную операцию и, тем самым, вызов функции должен быть выполнен всеми процессами указываемого коммуникатора, все вызовы функции должны содержать одинаковые значения параметров count, type, op, root, comm. Передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом root. Выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений. Так, например, если сообщения содержат по два элемента данных и выполняется

операция суммирования MPI_SUM, то результат также будет состоять из двух значений, первое из которых будет содержать сумму первых элементов всех отправленных сообщений, а второе значение будет равно сумме вторых элементов сообщений, соответственно.

Таблица 6.1

Типы операций MPI для функций редукции данных

Операция	Описание
MPI MAX	Определение максимального значения
MPI MIN	Определение минимального значения
MPI SUM	Определение суммы значений
MPI PROD	Определение произведения значений
MPI LAND	Выполнение логической операции "И" над значениями сообщений
MPI BAND	Выполнение битовой операции " И" над значениями сообщений
MPI LOR	Выполнение логической операции "ИЛИ" над значениями сообщений
MPI BOR	Выполнение битовой операции " ИЛИ" над значениями сообщений
MPI LXOR	Выполнение логической операции исключающего "ИЛИ" над значениями сообщений
MPI BXOR	Выполнение битовой операции исключающего "ИЛИ" над значениями сообщений
MPI MAXLOC	Определение максимальных значений и их индексов
MPI MINLOC	Определение минимальных значений и их индексов

Применим полученные знания для переработки ранее рассмотренной программы суммирования. Как можно увидеть, часть программного кода, который отвечает за вычисления частичных сумм, может быть теперь заменена на вызов одной лишь функции MPI_Reduce:

```
// сборка частичных сумм на процессе с рангом 0
```

```
MPI_Reduce(&ProcSum,&TotalSum, 1, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
```

В ряде ситуаций независимо выполняемые в процессах вычисления необходимо синхронизировать. Так, например, для измерения времени начала работы параллельной программы необходимо, чтобы для всех процессов одновременно были завершены все подготовительные действия, перед окончанием работы программы все процессы должны завершить свои вычисления и т.п. Синхронизация процессов (барьер), т.е. одновременное достижение процессами тех или иных точек процесса вычислений, обеспечивается при помощи функции MPI:

```
int MPI_Barrier(MPI_Comm comm);
```

Функция MPI_Barrier определяет коллективную операцию и, тем самым, при использовании должна вызываться всеми процессами используемого коммуникатора. При вызове функции MPI_Barrier выполнение процесса блокируется, продолжение вычислений процесса произойдет только после вызова функции MPI_Barrier всеми процессами коммуникатора.

Контрольные вопросы для самопроверки

Каковы преимущества программирования на MPI?

Какой минимальный набор средств является достаточным для организации параллельных вычислений в системах с распределенной памятью?

В чем различие понятий процесса и процессора?

Как описываются в MPI передаваемые сообщения?

В чем различие парных и коллективных операций передачи данных?

Какая функция MPI обеспечивает передачу данных от одного процесса всем процессам?

Что понимается под операцией редукции?

В каких ситуациях следует применять барьерную синхронизацию?

Какие режимы передачи данных поддерживаются в MPI?

Как организуется неблокирующий обмен данными в MPI?

7. ПРОГРАММИРОВАНИЯ С ПАРАЛЛЕЛЬНЫМИ ДАННЫМИ

7.1. КОНЦЕПЦИЯ ПАРАЛЛЕЛЬНЫХ ДАННЫХ

Модель программирования с параллельными данными берет свое начало из векторного программирования, где программист писал свои приложения в терминах высоко оптимизированных векторных операций. Программирования с параллельными данными стало широко известным в связи с компьютерами с архитектурой SIMD. Сущность данной модели программирования состоит в обеспечении языка программирования параллельными встроенными средствами и процедурами, которые оперируют с данными, которые являются известными для всех процессов и подпроцессов вычисления.

Координация процессов и обмен данными между процессами определяется посредством операций, выполняемых над данными и аннотаций (комментариев) к программам, написанных программистом.

В своей простейшей форме модель программирования с параллельными данными расширяет язык последовательного программирования с помощью параллельных конструкций для обращения с большими агрегатами данных, такими как массивы. Программа, написанная на основе этого подхода схожа с обыкновенной последовательной программой. Управляющая логика программы подчиняется ограниченному последовательному порядку, за исключением, когда вызываются параллельные встроенные средства и процедуры. В этой точке инструкции запускаются на доступных процессорах для вычисления требуемого результата или трансформации. Программисту нет необходимости быть осведомленным о существующем множестве процессоров, а необходимо знать только о доступных некоторых быстрых встроенных средствах/операциях для некоторых определенных типов данных.

Эта форма программирования с параллельными данными с единственным потоком управления является очень естественным для

программирования SIMD-компьютеров, которые состоят из большого числа простых процессоров, которые плотно синхронизированы контроллером. Хост компьютер интерпретирует последовательную часть программы с параллельными данными до тех пор пока не вызывается параллельная операция, и в этой точке соответствующие инструкции отправляются на доступные процессоры. Первые компиляторы с параллельными данными были написаны для обеспечения использования SIMD-машин, но т.к. затем модель параллельных данных была внедрена на большом числе MIMD-платформ.

MIMD-компьютеры обычно имеют меньшее количество, но более мощные процессоры, чем SIMD-компьютеры. Эти машины часто программируются с использованием пересылки сообщений, но большинство последних проектов также поддерживают модель программирования с параллельными данными. SIMD-реализация такой модели может быть не самой подходящей, т.к. время, необходимое для копирования единичного слова данных по локальным памятям, является относительно большим по сравнению со временем, необходимым для одной арифметической операции.

В MIMD-реализации парадигмы параллельных данных процессы синхронизируются только в начале и в конце параллельных операций в отличие от прохождения одной и той же последовательности всех операций в одно и то же место. Процессы в MIMD-реализации модели программирования с параллельными данными, как говорят, являются слабо синхронизированными, то есть они выполняют те же самые операции над различными данными примерно за то же самое время.

Некоторые скалярные данные, такие как индексы массивов и индексы циклов могут быть более затратными для передачи, чем перевычисление их в каждом процессе, запускаемом на MIMD-компьютере. По этой причине, является общим для применения модели программирования с параллельными данными на таких компьютерах – включать SPMD-модель, и поддерживать частные данные для каждого процесса; SPMD-модель программирования описана ранее.

Параллельные встроенные средства и процедуры обрабатывают данные, которые уже идентифицированы компилятором как параллельные данные. Программист может давать инструкции компилятору посредством директив компилятора, что такие и только такие данные является параллельными или частными.

7.2. ОПЕРАЦИИ С ПАРАЛЛЕЛЬНЫМИ ДАННЫМИ

Основные компоненты параллельных языков следующие: последовательный язык и расширение для этого языка, параллельные данные и директивы компилятору. Эти концепции представлены в этом разделе с определенным уровнем детализации.

Параллельный Фортран пришел в различных диалектах с большой долей общности, но также с некоторой существенной разницей в синтаксисе и в функциональности. Самый важный диалект это High Performance Fortran (HPF). Это по существу F77 с расширением для массивов и встроенными средствами от F90, особенностями для данных и распределением работ и научными библиотеками. Все диалекты поддерживают или будут скоро поддерживать особенности, описанные в предыдущей подсекции, хотя и со слабой разницей в синтаксисе. Однако другие особенности, такие как распределение итерационных циклов по процессорам является значительно различным в подходах – HPF имеет новый оператор (FORALL).

Другая важная проблема совместимости связана с тем, что имеется несколько стандартных библиотечных процедур и много процедур, поддерживаемых под одним диалектом и не имеющих эквивалента в других. Даже там, где переносимость кода не является непосредственным и прямым, переносимость алгоритмов осуществима. По этой причине мы концентрируемся здесь на основных концепциях, которые являются общими для широкого круга языков и диалектов. Примеры приведены в основном на параллельном Фортране.

Одна из наиболее важных особенностей параллельных языков – это наличие средств для обработки массивов как "отдельных примитивов". Массивы могут быть физически распределены по процессорам, хотя все процессы могут иметь доступ к любой части массивов.

Однако в большинстве случаев каждый процесс обрабатывает конкретные элементы массива, которые содержатся в его локальной памяти.

В связи с тем, что массивы являются отдельными примитивами, они могут быть назначены и обработаны тем же самым способом как любые другие типы данных. Например, следующие два куска кода существенно идентичны по функциональности. С левой стороны это F77, а справа – показано как подобные операции выполняются с использованием параллельного Фортрана.

REAL a(64), b(64) DO i = 1, 64 a(i) = 2.0 b(i) = b(i)*a(i) END DO	REAL a(64),b(64) a = 2.0 b = b*a

Выражение в правой части a = 2.0 устанавливает каждый элемент a в 2.0. Подобным же образом, b = b * a выполняет поэлементное умножение двух массивов как в последовательном коде. Компилятор параллельных данных будет пытаться распределить работу так, что ссылки на память будут являться в основном локальными.

Программирование с параллельными данными было бы до некоторой степени ограниченным, если бы целый массив всегда должен был бы обрабатываться одновременно. Имеется некоторое число общих особенностей, которые допускают подходящие операции для выполнения их только с некоторыми элементами массива.

Сечения массива: большинство языков с параллельными данными имеют расширения для упрощения синтаксиса работы с массивами, позволяющие программисту описывать части массива простым способом. Это может быть использовано, например, для ссылки на третью колонку целиком и назначения ее в ноль или всех нечетных колонок. Один из способов сделать это состоит в использовании нотации триплетов для массивов, которые имеют форму min : max : stride(шаг). Например,

a(1:7:2) = 0 would zero a(1), a(3), a(5) and a(7).

Оператор WHERE позволяет элементам массива быть выбранными по значению, вместо позиции, как и в сечениях массива – WHERE является параллельным аналогом IF. Например, когда делятся элементы двух массивов, важно, чтобы делитель был бы не ноль. Следующий фрагмент кода показывает последовательную и параллельную версию этого:

DO i = 1, Max
IF (a(i) .GT. 0) THEN
b(i) = b(i)/a(i)
END IF
END
	WHERE (a .GT. 0)
	b = b/a
	END WHERE

Многие параллельные языки, использующие концепцию параллельных данных, обеспечивают довольно общие способы выбора элементов с помощью логических масок. При этом используется массив того же самого размера как массивы данных, но операции выполняются только над элементами данных, которые соответствуют элементу маски, который принимает значение истина на этих элементах данных.

Директивы компилятору для языков с параллельными данными в основном бывают двух типов – те, которые относятся к схеме параллельных данных в локальной памяти процессора и те, которые относятся к управлению.

Простейшая форма директив компилятора для данных сообщает компилятору, нужно ли или нет для некоторых данных, обычно массивов, иметь доступ ко всем процессам. Более замысловатая форма директив компилятору позволяет программисту определять размер и форму сечения массива, которая будет постоянно находится в локальной памяти какого-либо процессора.

Некоторые языки программирования, такие как CRAFT расширяют модель параллельных данных с помощью директив компилято-

ру для распределения работы итеративных циклов по доступным процессам.

MIMD-реализация модели с параллельными данными может обеспечивать некоторые директивы компилятора для определения главных областей, где поток управления возвращается от SPMD-исполнения к последовательному посредством специализированного мастер-процесса и наоборот. Могут также быть директивы компилятору для определения критических частей кода, которые гарантированно должны исполняться только на одном процессе в любое время.

В параллельных вычислениях важно иметь возможность эффективно перемещать данные. Многие вычисления, например клеточные автоматы, обновляют значения элементов массива на основе значений соседних элементов. Так как элементы массива физически могут располагаться в памяти нескольких процессоров, они должны быть перенесены к обновляемым элементам. В параллельных вычислениях это осуществляется с помощью процедуры, именуемой сдвигом данных (data shifts). Эта процедура перемещает все элементы массива в определенном направлении на некоторое одинаковое для всех элементов число ячеек. Такой подход значительно менее гибок, чем передача сообщений.

Выше были рассмотрены три типа коллективных коммуникаций – барьеры, широковещание и операция сокращения. Они имеют эквиваленты в большинстве языков программирования с параллельными данными.

Барьеры обычно расставляются компилятором до и после каждой параллельной операции с данными. Некоторые компиляторы могут быть созданы таким образом, чтобы не вставлять нефункциональные лишние барьеры; другие позволяют пользователю добавлять директивы, которые определяют, следует или нет вставлять барьеры в определенных местах кода.

Когда бы ни появилась скалярная переменная в качестве аргумента в выражении с массивом, как, например, в выражении 2xA, где A – это некий массив, вполне возможно, что это скалярное значение

будет разослано всем процессам. Некоторые компиляторы языков с параллельными данными поддерживают директивы для явной рассылки значения переменной, хранящегося у главного процесса, методом широковещания.

Многие языки программирования с параллельными данными также включают возможности вычисления суммы элементов в массиве или вычисления наибольшего элемента в массиве. Также существуют операции, называемые сканированием (scans), которые пробегают массив, вычисляя, например, частичные суммы. Глобальные процедуры очень важны в программировании с параллельными данными, и часто реализации их являются эффективными.

Поскольку языки программирования с параллельными данными естественным образом позволяют производить операции над большими массивами данных параллельно, они в первую очередь используются в приложениях, в которых требуется выполнение схожих операций на каждом элементе. Объем работы, требующейся для каждого элемента массива, должен быть приблизительно одинаков для обеспечения равномерной загруженности процессоров. Для того, чтобы позволить множеству процессоров выполнять операции над массивом, большинство операций должно быть *независимыми*, то есть не должны зависеть от результатов предыдущих операций. Эти требования означают, что программирование с параллельными данными более ограничено, чем программирование с передачей сообщений. Тем не менее, программирование с параллельными данными хорошо подходит для многих систем, основанных на сеточных вычислениях таких, как клеточная физика, например, квантовая хронодинамика, атмосферное моделирование, базы данных и клеточные автоматы.

7.3. ТЕХНОЛОГИЯ OPENMP

Одним из наиболее популярных средств программирования для компьютеров с общей памятью, базирующихся на традиционных языках программирования и использовании специальных комментариев, в настоящее время является технология OpenMP. Она базируется на концепции параллельных данных. За основу берётся последователь-

ная программа, а для создания её параллельной версии пользователю предоставляется набор директив, функций и переменных окружения. Предполагается, что создаваемая параллельная программа будет переносимой между различными компьютерами с разделяемой памятью, поддерживающими OpenMP API.

Разработкой стандарта занимается некоммерческая организация OpenMP ARB (Architecture Review Board), в которую вошли представители крупнейших компаний-разработчиков SMP-архитектур и программного обеспечения. OpenMP поддерживает работу с языками Фортран и Си/Си++. Первая спецификация для языка Фортран появилась в октябре 1997 года, а спецификация для языка Си/Си++ – в октябре 1998 года. На данный момент последняя официальная спецификация стандарта OpenMP 3.0 принята в мае 2008 года.

Интерфейс OpenMP задуман как стандарт для программирования на масштабируемых SMP-системах (SSMP, ccNUMA и других) в модели общей памяти (shared memory model). В стандарт OpenMP входят спецификации набора директив компилятора, вспомогательных функций и переменных среды. OpenMP реализует параллельные вычисления с помощью многопоточности, в которой «главный» (master) поток создает набор «подчиненных» (slave) потоков, и задача распределяется между ними. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами, причём количество процессоров не обязательно должно быть больше или равно количеству потоков.

До появления OpenMP не было подходящего стандарта для эффективного программирования на SMP-системах. Как уже указывалось выше, наиболее гибким, переносимым и общепринятым интерфейсом параллельного программирования является интерфейс передачи сообщений. Однако модель передачи сообщений недостаточно эффективна на SMP-системах, а также относительно сложна в освоении, так как требует мышления в "невычислительных" терминах. Проект стандарта ХЗН5 провалился, так как был предложен во время всеобщего интереса к МРР-системам, а также из-за того, что в нем

поддерживается только параллелизм на уровне циклов. Технология OpenMP развивает многие идеи X3H5.

POSIX-интерфейс для организации нитей (Pthreads) поддерживается практически на всех UNIX-системах, однако по многим причинам не подходит для практического параллельного программирования. В нём нет поддержки языка Фортран, слишком низкий уровень программирования, нет поддержки параллелизма по данным, а сам механизм нитей изначально разрабатывался не для целей организации параллелизма. OpenMP можно рассматривать как высокоуровневую надстройку над Pthreads (или аналогичными библиотеками нитей). В технологии OpenMP используется терминология и модель программирования, близкая к Pthreads, например, динамически порождаемые нити, общие и разделяемые данные, механизм «замков» для синхронизации.

Многие поставщики SMP-архитектур (Sun, HP, SGI) в своих компиляторах поддерживают спецдирективы для распараллеливания циклов. Однако эти наборы директив, как правило, весьма ограничены и несовместимы между собой. В результате чего разработчикам приходится распараллеливать приложение отдельно для каждой платформы. Технология OpenMP является во многом обобщением и расширением упомянутых наборов директив и является шагом вперед.

Рост популярности технологии OpenMP обеспечивается также за счет идеи "инкрементального распараллеливания". OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы. При этом, OpenMP – достаточно гибкий механизм, предоставляющий разработчику большие возможности контроля над поведением параллельного приложения.

Предполагается, что OpenMP-программа на однопроцессорной платформе может быть использована в качестве последовательной

программы, т.е. нет необходимости поддерживать последовательную и параллельную версии. Директивы OpenMP просто игнорируются последовательным компилятором, а для вызова процедур OpenMP могут быть подставлены заглушки (stubs), текст которых приведен в спецификациях.

Также одним из достоинств OpenMP его разработчики считают поддержку так называемых "orphan" (оторванных) директив, то есть директивы синхронизации и распределения работы могут не входить непосредственно в лексический контекст параллельной области.

Ниже представлен достаточно простой пример программы для вычисления числа "Пи". В существующую последовательную программу на языке Фортран вставлены две строчки, и она становится распараллеленной:

```
program compute_pi
parameter (n = 1000)
integer i
double precision w,x,sum,pi,f,a
f(a) = 4.d0/(1.d0+a*a)
w = 1.0d0/n
sum = 0.0d0;
!$OMP PARALLEL DO PRIVATE(x), SHARED(w)
!$OMP& REDUCTION(+:sum)
do i=1,n
        x = w*(i-0.5d0)
        sum = sum + f(x)
Enddo
pi = w*sum
print *,'pi = ',pi
Stop
end
```

Директивы OpenMP с точки зрения Фортрана являются комментариями и начинаются с комбинации символов "!$OMP". При этом директивы можно разделить на 3 категории:

- определение параллельной секции;
- разделение работы;

- синхронизация.

Каждая директива может иметь несколько дополнительных атрибутов – клауз. Отдельно специфицируются клаузы для назначения классов переменных, которые могут быть атрибутами различных директив.

Программа начинается с последовательной области – сначала работает один процесс (нить), при входе в параллельную область порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область. В программе может быть любое количество параллельных и последовательных областей.

Кроме того, параллельные области могут быть также вложенными друг в друга. В отличие от полноценных процессов, порождение нитей является относительно быстрой операцией, поэтому частые порождения и завершения нитей не так сильно влияют на время выполнения программы.

Для написания эффективной параллельной программы необходимо, чтобы все нити, участвующие в обработке программы, были равномерно загружены полезной работой. Это достигается тщательной балансировкой загрузки, для чего предназначены различные механизмы OpenMP.

Существенным моментом является также необходимость синхронизации доступа к общим данным. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе. Поэтому значительная часть функциональности OpenMP предназначена для осуществления различного рода синхронизаций работающих нитей.

OpenMP не выполняет синхронизацию доступа различных нитей к одним и тем же файлам. Если это необходимо для корректности программы, пользователь должен явно использовать директивы синхронизации или соответствующие библиотечные функции. При дос-

тупе каждой нити к своему файлу никакая синхронизация не требуется.

Рассмотрим пример на языке Фортран для перемножения матриц:

```
      program matrmult
      include "omp_lib.h"
      integer N
      parameter(N=4096)
      common /arr/ a, b, c
      double precision a(N, N), b(N, N), c(N, N)
      integer i, j, k
      double precision t1, t2
C инициализация матриц
      do i=1, N
      do j=1, N
      a(i, j)=i*j
      b(i, j)=i*j
      end do
      end do
      t1=omp_get_wtime()
C основной вычислительный блок
!$omp parallel do shared(a, b, c) private(i, j, k)
      do j=1, N
        do i=1, N
          c(i, j) = 0.0
          do k=1, N
            c(i, j)=c(i, j)+a(i, k)*b(k, j)
          end do
        end do
      end do
      t2=omp_get_wtime()
      print *, "Time=", t2-t1
      end
```

Нотация PARALLEL ... END PARALLEL определяет параллельную область программы. При входе в эту область порождаются новые

(N-1) нити, образуется "команда" из N нитей, а порождающая нить получает номер 0 и становится основной нитью команды ("master thread"). При выходе из параллельной области основная нить дожидается завершения остальных нитей, и продолжает выполнение в одном экземпляре. Предполагается, что в SMP-системе нити будут распределены по различным процессорам (однако это, как правило, находится в ведении операционной системы). Каким образом между порожденными нитями распределяется работа, которая определяется директивами DO, SECTIONS и SINGLE.

Директива DO ... [ENDDO] определяет параллельный цикл. Клауза SCHEDULE определяет способ распределения итераций по нитям: STATIC, m – статически, блоками по m итераций; DYNAMIC, m – динамически, блоками по m (каждая нить берет на выполнение первый еще невзятый блок итераций); GUIDED, m – размер блока итераций уменьшается экспоненциально до величины m; RUNTIME – выбирается во время выполнения. По умолчанию, в конце цикла происходит неявная синхронизация, эту синхронизацию можно запретить с помощью ENDDO NOWAIT.

Разделение работы осуществляется с помощью определения параллельной секции SECTIONS ... END SECTIONS. Это неитеративная параллельная конструкция, которая определяет набор независимых секций кода ("конечный" параллелизм). Секции отделяются друг от друга директивой SECTION. Если внутри PARALLEL содержится только одна конструкция DO или только одна конструкия SECTIONS, то можно использовать укороченную запись: PARALLEL DOили PARALLEL SECTIONS.

Можно также задать исполнение одной нитью: SINGLE ... END SINGLE, что определяет блок кода, который будет исполнен только одной нитью (первой, которая дойдет до этого блока). Явное управление распределением работы осуществляется с помощью функций OMP_GET_THREAD_NUM() и OMP_GET_NUM_THREADS – нить может узнать свой номер и общее число нитей, а затем выполнять свою часть работы в зависимости от своего номера.

В OpenMP переменные в параллельных областях программы разделяются на два основных класса:

SHARED – общие; под именем A все нити видят одну переменную) и PRIVATE – приватные (локальные); под именем A каждая нить видит свою переменную.

Отдельные правила определяют поведение переменных при входе и выходе из параллельной области или параллельного цикла: REDUCTION, FIRSTPRIVATE, LASTPRIVATE, COPYIN. По умолчанию, все COMMON-блоки, а также переменные, порожденные вне параллельной области, при входе в эту область остаются общими (SHARED). Исключение составляют переменные – счетчики итераций в цикле, по очевидным причинам. Переменные, порожденные внутри параллельной области, являются приватными (PRIVATE). Явно назначить класс переменных по умолчанию можно с помощью клаузы DEFAULT.

Полное описание OpenMP можно найти в соответствующих справочниках. Всего в стандарте можно формально подсчитать 21 директиву, 31 функцию, а также опции и переменные окружения. Полное описание всех подробности выходит за пределы данного издания.

Контрольные вопросы для самопроверки

Каковы преимущества концепции параллельных данных?

В чем особенность реализации MIMD?

Какие языки с параллельными данными вы знаете?

Какие основные операции с параллельными данными?

Каковы особенности команды WHERE? В чем ее привлекательность?

Каковы преимущества программирования с использованием OpenMP?

В чем состоит концепция нитей?

Как достигается балансировка нагрузки в OpenMP?

Какие особенности организации параллельного цикла в OpenMP?

Какие похожие операции есть в MPI и OpenMP?

8. ИСЧИСЛЕНИЕ ВЗАИМОДЕЙСТВУЮЩИХ СИСТЕМ

8.1. ИСЧИСЛЕНИЕ ВЗАИМОДЕЙСТВУЮЩИХ СИСТЕМ И ВЫСОКОПРИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Как видно из предыдущего изложения высокопроизводительные вычисления всегда связаны с использованием многопроцессорных систем. В общем случае можно сказать, что многопроцессорные системы могут использоваться для решения задач, которые условно разделим на «вычислительные» и «невычислительные». Ранее в изложении материала основное внимание уделялось вычислительным задачам, подразумевающим, в том числе, использование численных методов для решения дифференциальных уравнений в частных производных в трехмерной постановке для сложных областей и т.д.

Невычислительные задачи подразумевают проведение несложных вычислений, но для больших объемов данных. В этот же класс попадают задачи, возникающие при управлении базами данных, а также NP-трудные задачи. Среди задач, для решения которых используются многопроцессорные системы можно выделить задачи управления различными производственными процессами. В этом случае процессоры представляют собой специализированные датчики, контролирующие отдельные стадии производственного процесса, они обмениваются между собой сообщениями и решают определённую производственную задачу. Здесь важно, чтобы не возникало известных проблем «тупика» и «бесконечного переспроса». Особенностями «поведения» таких систем является то, что на каждом специализированном процессоре действует своя собственная программа. Отсутствует понятие параллельного программирования – часто процессоры программируются разными производителями, а когда соединяются и образуют систему, о поведении которой хотелось бы иметь достаточно полное представление.

Для решения, в том числе, и указанных выше проблем английским математиком Робином Милнером (Robin Milner) было разработано и опубликовано в 1982 году исчисление взаимодействующих

систем (Calculus for Communicating Systems – CCS) – одно из первых алгебраических исчислений для анализа параллельных процессов.

В своих предшествующих работах Милнер исследовал денотационную семантику параллельных процессов и ввел понятие потокового графа (flow graph) с синхронизированными портами. В своих исследованиях Милнер отметил, что параллельные процессы имеют алгебраическую структуру. Например, если у нас есть два процесса P и Q, то мы можем создать новый процесс, соединив процессы P и Q последовательно или параллельно. Результатом этого соединения будет новый процесс, поведение которого зависит от P и Q и операции, которая была использована при их соединении.

Этот пример наглядно показывает то, что данное исчисление является алгебраическим: исчисление состоит из набора операций для построения описаний новых процессов на основе уже существующих. CCS является очень простым и эффективным способом описания поведения параллельных процессов на абстрактном уровне.

Подчеркнем, что именно работы Робина Милнера послужили толчком для развития теории процессов, являющихся одним из бурно развивающихся разделов математической теории программирования. Эта теория изучает модели поведения динамических систем, называемые процессами. Говоря неформально, процесс представляет собой модель такого поведения, которое заключается в исполнении действий, например, прием и передача каких-либо объектов, или преобразование этих объектов.

Аппарат теории процессов хорошо подходит для формального описания и анализа поведения распределенных динамических систем, состоящих из нескольких взаимодействующих компонентов, причем все эти компоненты работают параллельно, и взаимодействие компонентов происходит путем пересылки сигналов или сообщений от одних компонентов к другим. Таким образом, высокопроизводительные программно-аппаратные мультипроцессорные вычислительные комплексы являются примером таких систем.

8.2. МАТЕМАТИЧЕСКИЕ КОНСТРУКЦИИ

Описание математических конструкций CCS начнем с примера – представления бесконечных часов:

$$C1 \overset{def}{=} tick.C1.$$

Здесь *tick* – это имя действия, *C1* – идентификатор (имя) процесса, *tick.C1* – процесс. Процессы в CCS стоит воспринимать как «чёрные ящики». «Чёрный ящик» может иметь имя, которое определяет его, а также интерфейс процесса. Интерфейс процесса описывает набор коммуникационных портов, также называемых каналами, которые процесс может использовать для взаимодействия с другими процессами, которые находятся в его окружении, вместе с индикацией того, используются ли данные порты для ввода или для вывода информации.

На рис. 8.1 изображен интерфейс для процесса в виде диаграммы, имя которого CS (Computer Scientist). Этот процесс может взаимодействовать с его окружением посредством трех портов или коммуникационных каналов, называемых *coffe*, $\overline{coin}$ и $\overline{pub}$. Порт *coffe* используется на вход, а порты $\overline{coin}$ и $\overline{pub}$ используются процессом CS на выход. В общем случае для порта с именем α, мы используем надчеркивание для обозначения выхода порта α. Обычно мы будем ссылаться на имена, такие как *coffe* или $\overline{coin}$ как на действия.

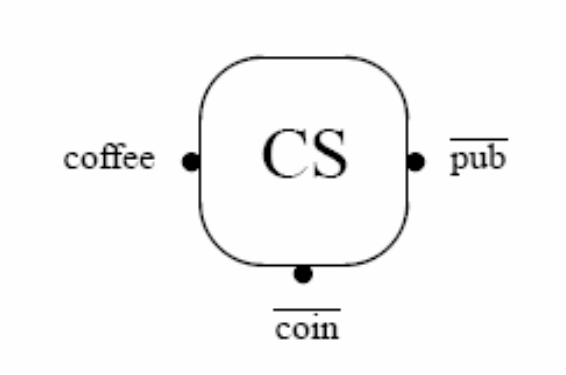

Рис. 8.1. Пример диаграммы CCS

Описание, подобное тому, что приведено на рис. 8.1 дает только статическую информацию о процессе. Больше всего нас интересует

поведение описываемого процесса. Поведение процесса описывается с помощью «CCS программы». Как мы вскоре увидим, её идея в том, чтобы конструкции процессов, которые используются при построении программы, позволяют нам описывать как структуру процесса, так и его поведение.

Теперь будем раскрывать суть конструкций CCS на примерах. Самый простой процесс из всех – это *0* процесс (нулевой процесс). Это самый неинтересный процесс, из всех, что можно себе представить, так как он не представляет никакого действия. Нулевой процесс представляет типичный пример тупикового поведения – то, что не может продолжаться в любых дальнейших вычислениях.

Наиболее типичный способ создания процесса в CCS – это использование префикса действия. Ниже представлены два примера процессов, построенных из нулевого процесса и префикса действия:

1. одиночный префикс: *strike.0*
2. множественный префикс: *take.strike.0*

В примере 1 приведен процесс, который умирает (становится нулевым процессом), когда будет выполнено действие *strike*. В примере 2 приведен процесс, для которого сначала должно быть выполнено действие *take*, после чего мы получим процесс из примера 1. В общем случае правило построения префикса действия таково: если P – процесс и α – имя действия, то $\alpha.P$ – это процесс.

Идея в том, что имя действия вроде *strike* или $\overline{pub}$, будет обозначать входное или выходное действие в коммуникационном порт, и процесс $\alpha.P$ – это такой процесс, начинающийся с выполнения действия α и ведущий себя как процесс P впоследствии.

В качестве примера рекурсивного задания процессов можно привести пример кофейного автомата (Coffe Machine – CM), продающего кофе:

$$CM \overset{def}{=} coin.\overline{coffe}.CM\ .$$

Это пример машины, которая готова в обмен на брошенную монетку наливать кофе покупателю и впоследствии возвращаться в исходное состояние.

Конструкции CCS, которые мы обсуждали до сих пор, не позволяют нам описать поведение кофейного автомата, позволяющего покупателю выбирать между чаем и кофе. Для описания процессов, чье поведение может следовать различным шаблонам взаимодействия с окружением в CCS, имеется оператор выбора, обозначаемый «+». Например, кофейный автомат, предлагающий чай или кофе может быть описан следующим образом:

$$CTM \overset{def}{=} coin.(\overline{coffe}.CTM + \overline{tea}.CTM).$$

Идея такого описания в том, что после того как покупатель бросил монетку, процесс *CTM* (Coffe or Tea Machine) готов налить чай или кофе в зависимости от выбора покупателя. В общем случае правило для выбора состояний таково: если P и Q процессы, то $P+Q$ – процесс. Процесс $P+Q$ будет иметь характеристики обоих процессов P и Q. Однако, выбрав изначальное выполнение действия из P, мы заблокируем дальнейшее выполнение действий из Q и наоборот. Формальное определение оператора + будет выглядеть следующим образом:

$$\mathrm{R}(+) \qquad \frac{P_1 \xrightarrow{\alpha} Q}{P_1 + P_2 \xrightarrow{\alpha} Q} \qquad \frac{P_2 \xrightarrow{\alpha} Q}{P_1 + P_2 \xrightarrow{\alpha} Q}$$

В качестве ещё одного примера использования оператора выбора, можно привести пример автомата по продаже Coca-Cola:

$$\begin{aligned} \mathtt{Ven} &\overset{\text{def}}{=} \mathtt{2p.Ven_b} + \mathtt{1p.Ven_l} \\ \mathtt{Ven_b} &\overset{\text{def}}{=} \mathtt{big.collect_b.Ven} \\ \mathtt{Ven_l} &\overset{\text{def}}{=} \mathtt{little.collect_l.Ven} \end{aligned}$$

Диаграмма для данного примера показана на рис. 8.2.

Оператор выбора может быть обобщен на случай, когда мы хотим одним выражением описать множество однотипных выражений. Для этого вводится индекс:

$$\mathtt{Ct}_0 \stackrel{\text{def}}{=} \mathtt{up}.\mathtt{Ct}_1 + \mathtt{round}.\mathtt{Ct}_0$$
$$\mathtt{Ct}_{i+1} \stackrel{\text{def}}{=} \mathtt{up}.\mathtt{Ct}_{i+2} + \mathtt{down}.\mathtt{Ct}_i$$

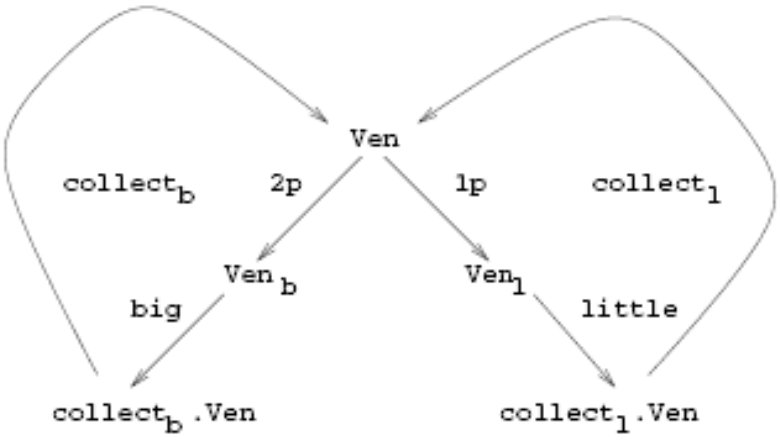

Рис. 8.2. Диаграмма для автомата, торгующего Coca-Cola

Всем известно, что ученый, работающий в университете – это машина для преобразования кофе в публикации. Поведение такого ученого может быть описано процессом CCS (диаграмма – на рис.8.1):

$$CS \stackrel{def}{=} \overline{pub}.\overline{coin}.coffe.CS\ .$$

Как видно из этого описания ученый увлечен написанием публикации или, возможно, своей докторской диссертации, но ему нужно кофе для написания следующей публикации. Для него кофе доступно только через взаимодействие с кофейным автоматом *CM* (см. описание выше), стоящим на кафедре. Для описания двух и более параллельных процессов и их возможных взаимодействий друг с другом, CCS предоставляет оператор параллельной композиции «|».:

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}.$$

Приведем пример. На рис. 8.3 приведен пример системы, состоящей из двух процессов (кофейного автомата *CM* и компьютерного ученого *CS*), которые запущены параллельно друг другу.

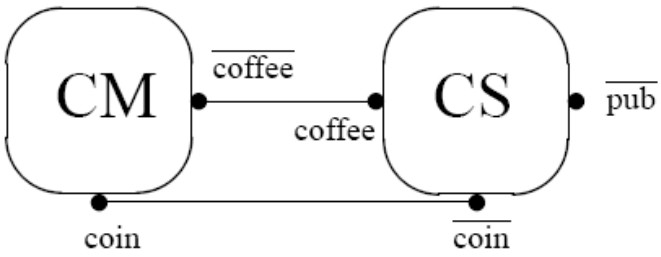

Рис. 8.3. Пример композиции двух процессов

Данную систему можно описать следующим CCS выражением: *CM*|*CS*. Эти два процесса могут связываться друг с другом через коммуникационные порты, которые они делят друг с другом и используют в дополняющем виде. Порты называются *coffe* и *coin*. Под дополнением мы имеем в виду то, что один из процессов использует порт для ввода информации, а другой для вывода. Возможные взаимодействия между процессами изображены на рис. 8.3 линиями, соединяющими дополняющие порты. Порт pub используется университетским ученым для взаимодействия с его исследовательским окружением, или, более прозаично, с другими процессами, которые могут быть представлены в его окружении и будут использоваться для ввода через этот порт. Стоит отметить ещё одну важную вещь – связь между дополняющими портами на рис. 8.3 означает, что существует возможность для компьютерного ученого и кофейного автомата взаимодействовать в параллельной композиции *CM*|*CS*. Однако мы не требуем их обязательного взаимодействия друг с другом. Ученый и кофейный автомат могут использовать их дополняющие порты для взаимодействия с другими системами в их окружении. Например, ещё один ученый *CS'* может использовать кофейный автомат *CM*, а, следовательно, он тоже должен писать публикации и расширять свое резюме, поэтому он будет достойным конкурентом для компьютерного ученого *CS* в конкурсе на место штатного научного сотрудника, как показано на рис. 8.4.

В качестве альтернативы ученый может иметь доступ к другому кофейному автомату в его окружении, как показано на рис. 8.5.

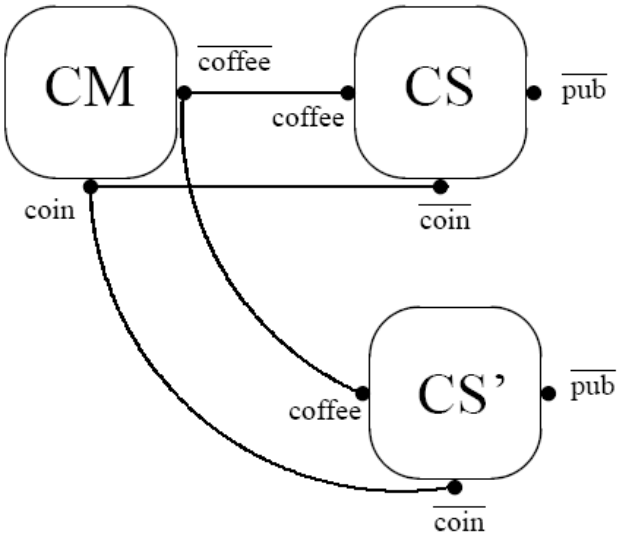

Рис. 8.4. Композиция трех процессов

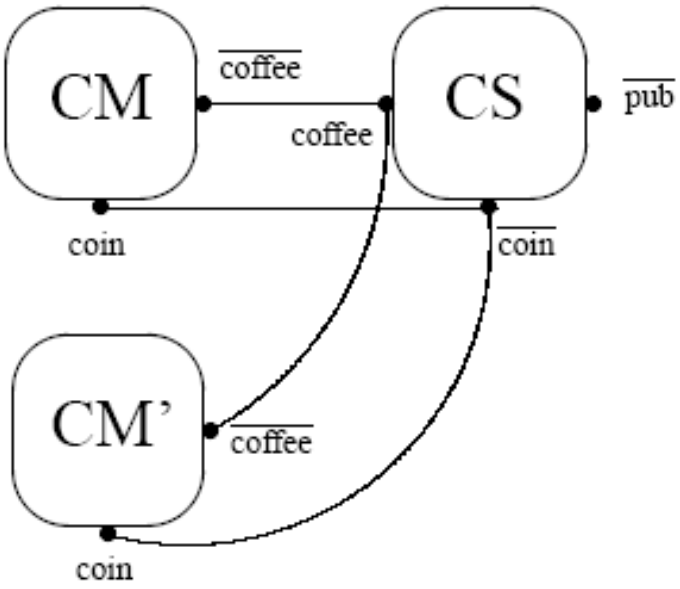

Рис. 8.5. Альтернативная композиция трех процессов

В общем случае, дано два CCS выражения *P* и *Q*, процесс *P*|*Q* описывает систему, в которой:

- *P* и *Q* выполняют независимо;
- могут взаимодействовать через дополняющие порты.

Ученых часто живут в очень конкурентной среде – «публикуй или погибай» – и у них может возникнуть желание иметь выделенный кофейный автомат только для себя и недоступный никому другому. Чтобы сделать это возможным в CCS имеется операция *ограничения*,

которая разделяет область видимости имен каналов (наподобие области видимости переменных в C++).

Например, используя операции *\coin* и *\coffe*, мы можем скрыть порты *coin* и *coffe* в окружении процессов *CM* и *CS*. Определение такого процесса будет следующим:

$$SmUni \stackrel{def}{=} (CM \mid CS) \setminus coin \setminus coffe .$$

Как показано на рис. 8.6, ограниченные порты *coin* и *coffe* могут быть использованы только для коммуникаций между компьютерным ученым и кофейным автоматом, и не доступны для взаимодействия с их окружением. Их область видимости ограничена процессом *SmUni*. Единственный порт процесса *SmUni*, который является видимым для его окружения, например, для конкурирующего университетского ученого CS', это порт через который компьютерный ученый выпускает свои публикации.

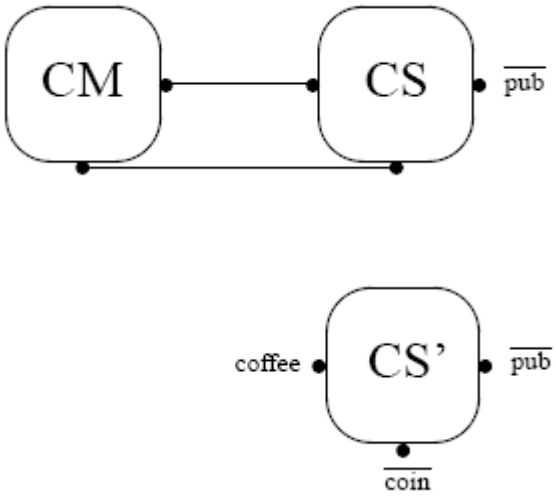

Рис. 8.6. Композиция процессов с ограничением

В общем случае, в формальном правиле для ограничения утверждается, что если *P* – это процесс и *L* – это набор имен портов, то *P\L* – это процесс. А если есть процесс *P\L*, то область видимости имен портов в *L* ограничивается процессом *P* – эти имена портов могут быть использованы для коммуникаций с *P*.

Университетский ученый не может жить только на кофе, поэтому для него было бы неплохо иметь доступ к другим типам торговых автоматов, которые торгуют, например, шоколадом, сухофруктами, чипсами. Поведение этих автоматов может быть легко описано по аналогии с кофейным автоматом. Например, мы можем определить эти процессы так:

$$CHM \overset{def}{=} coin.\overline{choc}.CHM$$
$$DFM \overset{def}{=} coin.\overline{figs}.DFM$$
$$CRM \overset{def}{=} coin.\overline{crisps}.CRM$$

Однако, отметим, что все эти торговые автоматы следуют общему поведенческому шаблону и могут рассматриваться как особые образцы обобщенного торгового автомата, который принимает монеты, выдает покупку и начинает всё с начала. Такой обобщенный торговый автомат может быть описан следующим выражением:

$$VM \overset{def}{=} coin.\overline{item}.VM .$$

Все вышеупомянутые особые торговые автоматы могут быть получены с помощью соответствующего переименования в определении *VM*. Например:

$$CHM \overset{def}{=} VM[choc \setminus item],$$

где *VM [choc\item]* – это процесс, который ведет себя как *VM*, но выдает шоколад, в то время как *VM* выдает некую абстрактную покупку. В общем случае, если *P* – это процесс и *f* – это функция, определенная на множестве имен и действующая в множество новых имен, удовлетворяющая определенным ограничениям, то *P[f]* – это процесс.

На этом обсуждение операций CCS, определяющих поведение процесса, заканчивается. В данном разделе мы рассматривали операции на простом, неформальном уровне с множеством примеров, что способствовало лучшему понятию сути CCS.

8.3. ПОВЕДЕНИЕ ПРОЦЕССОВ

Основная идея, лежащая в основе семантики CCS – процесс проходит через различные состояния во время выполнения; процессы изменяют свои состояния, выполняя действия. Например, процесс $CS \overset{def}{=} \overline{pub}.\overline{coin}.coffe.CS$ может выполнить действие $\overline{pub}$ и превратиться в процесс, чьё поведение описывается следующим CCS выражением:

$$CS_1 \overset{def}{=} \overline{coin}.coffe.CS .$$

Затем процесс CS_1 может пустить в выходной поток монетку, тем самым превратиться в процесс, поведение которого описывается следующим CCS выражением:

$$CS_2 \overset{def}{=} coffe.CS .$$

Окончательно этот процесс может принять из входного потока кофе и вести себя также, как наш изначальный процесс *CS*. Поэтому процессы *CS*, CS_1 и CS_2 – это всего лишь возможные состояния вычисления процесса CS. Более того стоит отметить, что на самом деле нет никакой разницы между процессами и их состояниями. Выполняя действие, процесс превращается в другой процесс, который описывает действия, которые осталось запустить у изначального процесса.

В CCS процессы меняют состояния, выполняя переходы, и эти переходы обозначаются именами действий, которые произошли с процессом. Как пример перехода состояния можно рассмотреть следующий переход: $CS \xrightarrow{\overline{pub}} CS_1$, который показывает, что *CS* может выполнить действие $\overline{pub}$ и превратиться в CS_1. Поведение ученого *CS* может быть описано следующей системой переходов:

$$CS \xrightarrow{\overline{pub}} CS_1 \xrightarrow{\overline{coin}} CS_2 \xrightarrow{coffe} CS .$$

Такие схемы называются именованными системами переходов (Labelled Transition Systems – LTS). По такому же принципу мы можем описать набор состояний кофейного автомата $CM \overset{def}{=} coin.\overline{coffe}.CM$, переписав это выражение как:

$$CM \overset{def}{=} coin.CM_1$$
$$CM_1 \overset{def}{=} \overline{coffe}.CM$$

Отметим, что университетский ученый готов выдать монетку в состоянии *CS1*, как видно из перехода $CS_1 \xrightarrow{\overline{coin}} CS_2$, а кофейный автомат готов принять монетку в его исходном состоянии, потому что у него есть переход $CM \xrightarrow{coin} CM_1$. Поэтому, когда эти два процесса запущены параллельно друг другу, они могут взаимодействовать и менять состояние одновременно. Результат взаимодействия будет описываться с помощью следующего перехода:

$$CM \mid CS_1 \xrightarrow{?} CM_1 \mid CS_2.$$

Здесь мы столкнулись с важным дизайнерским решением, мы должны решить какое имя использовать вместо «?». Должны ли мы принять решение использовать стандартное имя, обозначающее вход или выход на каком-то порту, тогда третий процесс должен иметь возможность синхронизироваться в дальнейшем с кофейным автоматом и компьютерным ученым, что ведет к многопоточной синхронизации. У Милнера сделано по другому. В CCS используется взаимодействие через рукопожатие, приводящее к переходу между состояниями, который является ненаблюдаемым, а тогда он не может быть синхронизирован в дальнейшем. Такой переход между состояниями обозначается новым именем τ. Тогда вышеупомянутое выражение может быть записано в виде:

$$CM \mid CS_1 \xrightarrow{\tau} CM_1 \mid CS_2.$$

Обобщив всё это, можем написать систему переходов для процесса:

$$SmUni \overset{def}{=} (CM \mid CS) \setminus coin \setminus coffe.$$

Это представлено на рис. 8.7 в виде диаграммы

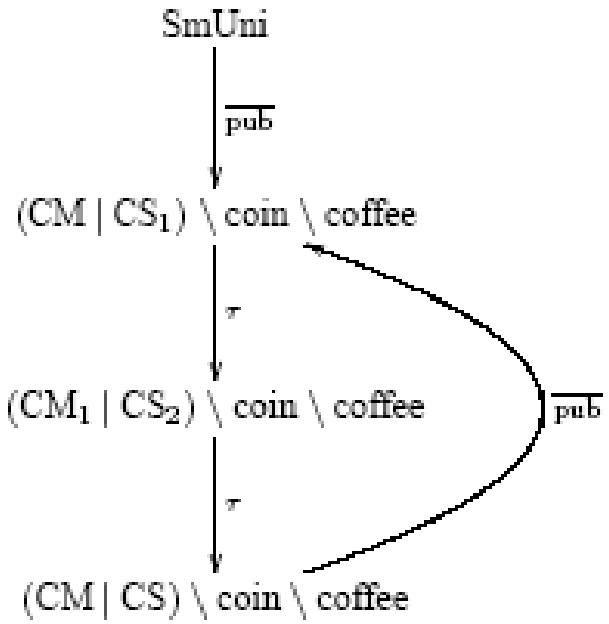

Рис. 8.7. Процесс, описывающий поведение ученого

Так как предполагается, что действие τ является ненаблюдаемым, то следующий процесс является подходящим высокоуровневым обозначением поведения процесса *SmUni*:

$$Spec \overset{def}{=} \overline{pub}.Spec\,.$$

На самом деле, мы предполагаем, что *SmUni* и *Spec* описывают наблюдаемое поведение на различных уровнях абстракции. Для того, чтобы показывать, что процессы имеют одинаковое поведение, вводится специальное понятие поведенческой эквивалентности. Эквивалентность поведения процессов обозначают с помощью символа $\cong$.

Это понятие является ключевым для прикладного использования этого формализма. В первую очередь это связано с задачами по верификации протоколов. Кроме того эта формальная теория служит базисом для языков распределенного программирования, например, BPEL.

8.4. ФОРМАЛЬНОЕ ОПРЕДЕЛЕНИЕ CCS

Для того чтобы дать формальное определение CCS, необходимо вначале дать определение формальной теории.

Формальная теория $\Im$ – это:

- множество символов A (алфавит);

- множество слов F в алфавите ($F \subset A^*$) – формулы;
- подмножество формул $B \subset F$ – аксиомы;
- множество отношений $R \in F^{n+1}$ (правила вывода).

Например, исчисление предикатов – формальная теория. Теперь мы можем дать формальное определение CCS.

CSS – это формальная теория, в которой алфавит: $\alpha_1, \alpha_2, \alpha_3, \ldots$ – имена действий (actions), $A, P_1, P_2, P_3, \ldots$ – идентификаторы процессов, $+, \|, ., new, (,), <, >, \to$ – служебные символы.

Множество формул порождается грамматикой:

$$P := nil \mid \alpha.P \mid P_1 + P_2 \mid P_1 \| P_2 \mid (newa)P \mid A < a_1, ..., a_n > .$$

Аксиомы: $R(.) : \alpha.P \xrightarrow{\alpha} P$.

Правила: $R(\sum), R(\overset{def}{=}), R(\mid com), R(|), R(\backslash), R([f])$.

Как уже неоднократно, упоминалось ранее, коммуникационные порты у процесса могут быть двух видов: входные и выходные. Следовательно, действия, которые производит процесс можно тоже разделять на входящие и на исходящие. Входящие действия обозначаются обычными именами (например, α), а исходящие действия обозначаются именами с надчеркиванием (например, $\overline{\alpha}$).

Теперь дадим формальные определения операторов, которые были использованы в определении CCS.

Оператор определения

$$\mathrm{R}(\overset{\text{def}}{=}) \quad \frac{P \xrightarrow{\alpha} Q}{A \xrightarrow{\alpha} Q} \quad A \overset{\text{def}}{=} P \quad .$$

Оператор +

$$\mathrm{R}(+) \quad \frac{P_1 \xrightarrow{\alpha} Q}{P_1 + P_2 \xrightarrow{\alpha} Q} \quad \frac{P_2 \xrightarrow{\alpha} Q}{P_1 + P_2 \xrightarrow{\alpha} Q} \quad .$$

Оператор $\sum$

$$\mathrm{R}(\sum) \quad \frac{P_j \xrightarrow{\alpha} Q}{\sum\{P_i \,:\, i \in I\} \xrightarrow{\alpha} Q} \quad j \in I \quad .$$

Оператор параллельной композиции

$$\mathrm{R}(|\,\mathrm{com}) \quad \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P\,|\,Q \xrightarrow{\tau} P'\,|\,Q'};$$

$$\mathrm{R}(|) \quad \frac{P \xrightarrow{\alpha} P'}{P\,|\,Q \xrightarrow{\alpha} P'\,|\,Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P\,|\,Q \xrightarrow{\alpha} P\,|\,Q'}.$$

Оператор ограничения

$$\frac{P \xrightarrow{\alpha} Q}{P\backslash K \xrightarrow{\alpha} Q\backslash K}\ \alpha \notin K \cup \overline{K},$$

где K – набор имен действий, $\overline{K} = \{\overline{a}\colon a \in K\}$.

Оператор переименования

$$\mathrm{R}([f]) \quad \frac{P \xrightarrow{\beta} Q}{P[f] \xrightarrow{\alpha} Q[f]}\ \alpha = f(\beta),$$

где f – переименовывающая функция.

В заключение описания исчисления взаимодействующих процессов еще раз подчеркнем, что CCS является формальной теорией и практическое использование ее связано в основном с описанием поведения сложного процесса на основе известного поведения составляющих процессов. При этом чаще всего решается проблема по поиску в поведении сложного процесса тупиков ("deadlock").

Контрольные вопросы для самопроверки

Что описывается с помощью CCS?

Что такое поведение процесса?

Как описывается поведение процесса в CCS?

Как пользоваться диаграммой описания поведение процесса в CCS?

Что такое оператор выбора?

Каковы особенности оператора композиции?

Каковы основные идеи, лежащие в основе семантики CCS?

Каковы формальные определения, входящие в CCS?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Антонов А. С.* Введение в параллельные вычисления: метод. пособие / А. С. Антонов. – М. : Изд–во Физического факультета МГУ, 2002. – 70 с.

2. *Антонов А. С.* Параллельное программирование с использованием технологии MPI: учеб. пособие / А. С. Антонов. – М. : Изд–во МГУ, 2004. – 71 с.

3. *Антонов А. С.* Параллельное программирование с использованием технологии OpenMP: учеб. пособие / А. С. Антонов. – М. : Изд–во МГУ, 2009. – 77 с.

4. *Барский А. Б.* Параллельные информационные технологии / А. Б. Барский. – М. : Бином, 2007. – 504 с.

5. *Богачев К. Ю.* Основы параллельного программирования / К. Ю. Богачев. – М. : Бином, 2010. – 344 с.

6. Архитектуры и топологии многопроцессорных вычислительных систем. Курс лекций / А. В. Богданов [и др.] – М. : Интернет–университет информационных технологий, 2004. – 176 с.

7. *Букатов А. А.* Программирование многопроцессорных вычислительных систем / А. А. Букатов, В. Н. Дацюк, А. И. Жегуло. – Ростов–на–Дону : Издательство ООО "ЦВВР", 2003. – 208 с.

8. *Бурова И. Г.* Алгоритмы параллельных вычислений и программирование / И. Г. Бурова, Ю. К. Демьянович. – СПб. : Изд–во СПбГУ, 2002. – 208 с.

9. *Бурова И. Г.* Лекции по параллельным вычислениям / И. Г. Бурова, Ю. К. Демьянович. – СПб. : Изд–во СПбГУ, 2003. – 132 с.

10. *Воеводин В. В.* Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – СПб. : БХВ–Петербург, 2004. – 608 с.

11. *Воеводин Вл. В.* Вычислительное дело и кластерные системы / Вл. В. Воеводин С. А. Жуматий. – М. : Изд–во МГУ, 2007. – 150 с.

12. *Гергель В. П.* Теория и практика параллельных вычислений / В. П. Гергель. – М. : Бином, 2007. – 424 с.

13. *Гергель, В. П.* Основы параллельных вычислений для многопроцессорных вычислительных систем / В. П. Гергель, Р. Г. Стронгин. – Н. Новгород : Изд–во ННГУ, 2003. – 302 с.

14. *Демьянович Ю. К.* Теория распараллеливания и синхронизация: учеб. пособие / Ю. К. Демьянович, Т. О. Евдокимова. – СПб. : Изд–во СПбГУ, 2005. – 108 с.

15. *Демьянович Ю. К.* Технология программирования для распределенных параллельных систем : курс лекций / Ю. К. Демьянович, О. Н. Иванцова. – СПб. : Изд–во СПбГУ, 2005. – 94 с.

16. *Демьянович Ю. К.* Операционная система Unix (Linux) и распараллеливание : курс лекций / Ю. К. Демьянович, Д. М. Лебединский. – СПб. : Изд–во СПбГУ, 2005. – 109 с.

17. *Корнеев В. Д.* Параллельное программирование в MPI / В. Д. Корнев. – Москва–Ижевск : Институт компьютерных исследований, 2003. – 303с.

18. *Лацис А. О.* Параллельная обработка данных: учеб. пособие / А. О. Лацис. — М. : Академия, 2010. – 336 с.

19. *Левин М. П.* Параллельное программирование с использованием OpenMP / М. П. Левин. – М. : Бином, 2008. – 120 с.

20. *Лупин С. А.* Технологии параллельного программирования / С. А. Лупин, М. А. Посыпкин. – М. : Инфра–М, 2008. – 208 с.

21. *Немнюгин С.* Параллельное программирование для многопроцессорных вычислительных систем / С. Немнюгин, О. Стесик. – СПб. : БХВ–Петербург, 2002. – 400 с.

22. *Таненбаум Э. С.* Современные операционные системы / Э. С. Таненбаум. – М. : Питер, 2010. – 1115 с.

23. *Таненбаум Э. С.* Распределенные системы. Принципы и парадигмы / Э. С. Таненбаум. – Санкт–Петербург : Питер, 2003. – 876 с.

24. *Топорков В. В.* Модели распределенных вычислений / В. В. Топорков. – М. : Физматлит, 2004. – 315 с.

25. *Milner R.* Calculus of Communicating Systems, Lecture Notes in Computer Science № 92 / R. Milner. – NY : Springer Verlag, 1982. – 260 с.

Баденко Владимир Львович

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Учебное пособие

Лицензия ЛР № 020593 от 07.08.97

Налоговая льгота – Общероссийский классификатор продукции
ОК 005–93, т. 2; 95 3005 – учебная литература

Подписано к печати _____2010-. Формат 60×84/16. Печать цифровая
Усл. печ. л. . Уч.–изд. л. . Тираж экз . Заказ

Отпечатано с готового оригинал–макета, предоставленного авторами,
в Цифровом типографском центре
Издательства Политехнического университета:
195251, Санкт–Петербург, Политехническая ул., 29.
Тел. (812) 540–40–14
Тел./факс: (812) 927–57–76