

Министерство образования и науки РФ
Федеральное государственное бюджетное
образовательное учреждение
высшего профессионального образования
"Кемеровский государственный университет"

**К. Е. Афанасьев, С. Ю. Завозкин,
С. Н. Трофимов, А. Ю. Власенко**

**Основы высокопроизводительных вычислений
Том I**

Высокопроизводительные вычислительные системы

Учебное пособие

Кемерово 2011

УДК 004 (075.8)

ББК 397

О 75

*Печатается по решению редакционно-издательского совета
ФГБОУ ВПО "Кемеровский государственный университет"*

Рецензенты:

доктор физико-математических наук, профессор, зав. кафедрой компьютерной математики Уфимского государственного авиационного технического университета **В. П. Житников;**

доктор физико-математических наук, профессор зав. кафедрой высокопроизводительных вычислений и информационно-коммуникационных технологий Южного федерального университета, директор ЮГИНФО ЮФУ **Л. А. Крукиер**

Афанасьев, К. Е.

О 75 Основы высокопроизводительных вычислений. Том 1. Высокопроизводительные вычислительные системы: учебное пособие / К. Е. Афанасьев, С. Ю. Завозкин, С. Н. Трофимов, А. Ю. Власенко; ФГБОУ ВПО "Кемеровский государственный университет". – Кемерово, 2011. – 246 с.

ISBN 978-5-8353-1098-2

В пособии рассмотрены вопросы, касающиеся современных высокопроизводительных вычислительных систем (ВВС). Представлены архитектуры и классификация ВВС, принципы работы, современные направления развития, методы оценки производительности, общие принципы программирования, операционные системы и прикладное программное обеспечение ВВС. Значительное внимание уделяется современным процессорным разработкам ведущих фирм производителей. Приводится описание зарубежных и отечественных высокопроизводительных вычислительных систем.

Пособие рассчитано на студентов, аспирантов, преподавателей и научных работников естественнонаучных специальностей.

ISBN 978-5-8353-1098-2

ББК 397

© Афанасьев К. Е., Завозкин С. Ю.,
Трофимов С. Н., Власенко А. Ю., 2011

© ФГБОУ ВПО "Кемеровский
государственный университет", 2011

Оглавление

Сокращения	6
Введение	11
Глава 1. Краткая история развития высокопроизводительных вычислительных систем	14
1.1. Развитие зарубежных высокопроизводительных вычислительных систем	14
1.2. Развитие отечественных высокопроизводительных вычислительных систем	18
1.3. Классификация компьютеров	19
1.4. Требования, предъявляемые к ВВС	23
Глава 2. Архитектура вычислительных систем	27
2.1. Типы архитектур ЭВМ	27
2.2. Типы процессоров	30
2.3. Современные методы повышения производительности процессоров	33
2.4. Системы команд процессора	38
2.5. Основная память	41
2.6. Дисковая память	46
2.7. Подсистемы ввода/вывода	49
Глава 3. Архитектура ВВС	54
3.1. Классификация ВВС	54
3.2. Архитектуры ВВС	61
3.3. Факторы, снижающие производительность параллельных компьютеров	70

Глава 4. Оценка производительности ВВС	73
4.1. Оценка производительности вычислительных систем	73
4.2. Стандарты в области оценки производительности	75
4.3. Виды тестов	78
4.4. Критерии ранжирования ВВС	86
Глава 5. Программирование для ВВС	89
5.1. Соответствие структуры программ и архитектуры ЭВМ	89
5.2. Модели обмена данными	90
5.3. Два типа параллелизма: параллелизм данных и параллелизм задач. Парадигмы параллельного программирования	94
Глава 6. Операционные системы параллельных вычислительных систем	101
6.1. Понятие операционной системы (ОС). История развития	101
6.2. Структура ОС. Виды операционных систем	105
6.3. Принципы построения ОС	110
6.4. Операционные системы мультимикропроцессорных ЭВМ	112
6.5. Соглашения при составлении псевдокода	116
6.6. Взаимодействие и синхронизация процессов	118
6.7. Планирование процессов. Многозадачность	131
6.8. Планирование микропроцессора	134
6.9. Файловые системы	138
Глава 7. Прикладное программное обеспечение	144
7.1. Кластерное ПО	144
7.2. GRID-системы	150
7.3. Отладчики параллельных программ	155
Глава 8. Характеристики и архитектура современных процессоров	162
8.1. Intel	162
8.2. AMD	169
8.3. Sun Microsystems	175
8.4. Высокопроизводительные вычисления на графических процессорах	185
Глава 9. Вычислительные системы нетрадиционной архитектуры	192
9.1. Биологические компьютеры	192

9.2. Нейрокомпьютеры	196
9.3. Квантовые компьютеры	200
9.4. Компьютеры с реконфигурируемой архитектурой	204

Заключение	210
-------------------	------------

Приложение 1. Top50 и Top500	211
-------------------------------------	------------

Тридцать четвертая редакция Top500	211
Двенадцатая редакция Top50	212
Анализ Top5000	213

Приложение 2. Примеры ВВС	217
----------------------------------	------------

Blue Gene	217
Red Storm	222
Roadrunner	225
Cray XT5h	228
СКИФ МГУ “Чебышев”	231
Суперкомпьютер МГУ “Ломоносов”	237

Литература	240
-------------------	------------

Сокращения

ALPS – Application Level Placement Scheduler (планировщик прикладного уровня)

AMD – Advanced Micro Devices (корпорация-производитель интегрированной микроэлектроники).

ARM – Advanced RISC Machine (усовершенствованная **RISC**-машина).

Bio-Comp - Biological Computations (биологические вычисления).

BIOS – Basic Input-Output System (базовая система ввода-вывода).

BLAS – Basic Linear Algebra Subprograms (базовые подпрограммы линейной алгебры).

CC-NUMA – Coherent Cache NUMA (кэш-когерентный доступ к неоднородной оперативной памяти).

CISC – Complex Instruction Set Computing (вычисления с полным набором команд).

CORBA – Common Object Request Broker Architecture (общая архитектура брокера объектных запросов).

CPI – Clock Cycles per Instruction (количество тактов синхронизации на одну команду).

CPLD – Complex Programmable Logic Device (разновидность **ПЛИС**).

CPU – Central Processing Unit (центральный процессор).

CUDA – Compute Unified Device Architecture (унифицированная вычислительная архитектура).

DARPA – Defense Advanced Research Projects Agency (агентство передовых оборонных исследовательских проектов Министерства обороны США).

DCOM – Distributed Component Object Model (распределённая модель компонентных объектов).

DDT – Distributed Debugging Tool (инструмент распределенной отладки).

DES – Data Encryption Standard (симметричный алгоритм шифрования, в котором один ключ используется как для шифрования, так и для расшифрования данных).

DNS – Domain Name System (система доменных имен).

DRAM – Dynamic Random Access Memory (энергозависимая память с произвольным доступом, динамическая память).

DVI – Digital Visual Interface (цифровой видеоинтерфейс).

ECC – Error Checking and Correction (код коррекции ошибок).

EEPROM – Electrically Erasable Programmable Read-Only Memory (электрически стираемое перепрограммируемое постоянное запоминающее устройство).

ESCD – Extended System Configuration Data (данные расширенной системной конфигурации).

FIFO – First In First Out (первым зашел – первым вышел).

FFT – Fast Fourier Transform (быстрое преобразование Фурье).

FPGA – Field-Programmable Gate Array (разновидность ПЛИС).

FPU – Floating Point Unit (устройство для работы с плавающей точкой).

GPC – Graphics Performance Characterization Group (тесты графики).

GPU – Graphics Processing Unit (графический процессор).

GPGPU – General-Purpose computation on Graphics Processing Units (графический процессор общего назначения).

HDF – Hierarchical Data Format (иерархический формат данных).

HDMI – High-Definition Multimedia Interface (мультимедийный интерфейс высокой чёткости).

HPG – High Performance Group (тесты для высокопроизводительных вычислительных систем).

ITAC – Intel Trace Analyzer and Collector (программная система профилирования и автоматического контроля корректности MPI-программ).

LCFG – Local Configuration system (локальная конфигурационная система – система для автоматизации процесса установки UNIX).

LSF – Load Sharing Facility (одна из систем пакетной обработки).

MFLOPS – Million Floating point Operations Per Second (миллион элементарных арифметических операций над числами с плавающей точкой, выполненных в секунду).

MIMD – Multiple Instruction Stream / Multiple Data Stream (много потоков команд, много потоков данных).

MIPS – Million Instructions Per Second (миллион команд в секунду).

MISC – Minimal Instruction Set Computer (минимальный набор команд).

MISD – Multiple Instruction Stream / Single Data Stream (много потоков команд, один поток данных).

MPI – Message Passing Interface (интерфейс передачи сообщений).

MPP – Massive Parallel Processing (массивно-параллельные системы).

MRTG – Multi Router Traffic Grapher (инструмент для мониторинга загрузки сети).

MTBF – Mean Time Between Failures (среднее время наработки на отказ).

NC-NUMA – No Caching NUMA (система с неоднородным доступом к памяти без кэширования).

NFS – Network File System (сетевая файловая система).

NMOS – N-channel Metal–Oxide–Semiconductor (N-канальный металл-оксид-полупроводник).

NUMA – Non-Uniform Memory Access (система с неоднородным доступом к памяти).

OGSA – Open Grid Service Architecture (открытая архитектура **GRID**-сервисов).

ONC – Open Network Computing (стек протоколов, разработанный компанией Sun Microsystems).

OpenPBS – Open Portable Batch System (система очередей).

OPIUM – **OSCAR** Password Installer and User Management (средство для синхронизации учетных записей пользователей и конфигурирования ssh).

OSCAR – Open Source Cluster Application Resource (ресурс приложений для кластера с открытым кодом).

OSE – Open System Environment (окружение открытых систем).

OSF DCE – Open Software Foundation Distributed Computing Environment (распределенная вычислительная среда фонда открытого программного обеспечения).

OSG – Open Systems Group (основной комитет).

PBS – Portable Batch System (одна из систем пакетной обработки).

PCI – Peripheral Component Interconnect (взаимосвязь периферийных компонентов).

PGAS – Partitioned Global Address Space (параллельное глобальное адресное пространство).

PR – Performance Rating (рейтинг производительности).

PSI – Power Saving Indicator (индикатор сбережения электроэнергии).

PVFS – Parallel Virtual File System-SKIF (параллельная виртуальная файловая система).

PVM – Parallel Virtual Machine (параллельная виртуальная машина).

PVP – Parallel Vector Process (параллельные векторные системы).

PXE – Preboot Execution Environment (предзагрузочная среда исполнения).

RAID – Redundant Array of Independent/Inexpensive Disks (избыточный массив независимых/недорогих жёстких дисков).

RAM – Random-Access Memory (память с произвольным доступом, память для чтения/записи).

RAS – Reliability, Availability and Service (надежность, работоспособность и удобство эксплуатации).

RDMA – Remote Direct Memory Access (прямой доступ к удаленной памяти).

RISC – Reduced Instruction Set Computing (вычисления с сокращённым набором команд).

ROM – Read-Only Memory (постоянное запоминающее устройство, память только для чтения).

RPC – Remote Procedure Call (удаленный вызов процедур).

RR – Round Robin (циклическое планирование процессов).

RSA – Буквенная аббревиатура от фамилий Rivest, Shamir и Adleman (криптографический алгоритм с открытым ключом).

SCSI – Small Computer System Interface (интерфейс, разработанный для объединения на одной шине различных по своему назначению устройств).

SGE – Sun Grid Engine (одна из систем пакетной обработки).

SIMD – Single Instruction Stream / Multiple Data Stream (один поток команд / много потоков данных).

SIS – System Installation Suite (система для автоматизации установки).

SISD – Single Instruction Stream / Single Data Stream (один поток команд / один поток данных).

SMP – Symmetric Multi-Processing (симметричные мультипроцессорные системы).

SMT – Simultaneous Multithreading (параллельная многопоточная обработка).

SNMP – Simple Network Management Protocol (простой протокол сетевого управления).

SPARC – Scalable Processor Architecture (масштабируемая процессорная архитектура).

SPMD – Single Program Multiple Data (одна программа – множество данных).

SRAM – Static Random Access Memory (статическая оперативная память с произвольным доступом, статическая память).

SSE – Streaming **SIMD** Extensions (потокковое SIMD-расширение процессора).

SSH – Secure Shell (защищенный протокол удаленного управления).

TDP – Thermal Design Power (уровень тепла выделяемого элементом).

TLB – Translation Lookaside Buffer (буфер быстрого преобразования адреса).

TLP – Thread-Level Parallelism (параллелизм на уровне потоков).

Torque – Terascale Open-source Resource and QUEue Manager (версия **PBS**).

VDT – Virtual Data Toolkit (программная платформа для организации **GRID**).

VLIW – Very Long Instruction Word (очень длинное командное слово).

АЛУ – арифметико-логическое устройство.

БД – база данных.

ВВС – высокопроизводительные вычислительные системы.

ДНК – дезоксирибонуклеиновая кислота.

ЗУ – запоминающие устройства.

КС – критическая секция.

МГУ – Московский государственный университет.

НС – нейронная сеть.

ОЗУ – оперативное запоминающее устройство.

ОС – операционная система.

ОСВМ – оптическая сверхвысокопроизводительная вычислительная машина.

ПК – персональный компьютер.

ПЛИС – программируемая логическая интегральная схема.

ПО – программное обеспечение.

ПЭ – процессорный элементы.

РНК – рибонуклеиновая кислота.

СПО – система пакетной обработки.

УУ – устройство управления.

ФУ – функциональные устройства.

ЦП – центральный процессор.

ЭВМ – электронно-вычислительная машина.

ЭЛС – элементарные логические схемы процессора.

ЯМР спектрометр – ядерный магнитно-резонансный спектрометр.

Введение

Учебное пособие “Высокопроизводительные вычислительные системы” открывает цикл пособий под общим заглавием “Основы высокопроизводительных вычислений”.

Стремительный рост популярности тематики высокопроизводительных вычислений в последние годы связан с тем, что на данный момент практически во всех естественных науках все чаще и чаще прибегают к аппарату математического моделирования. В модели включают все больше и больше различных факторов, что, естественно, сближает их с реальными исследуемыми объектами, но приводит к тому, что получить решение можно только применив численный эксперимент. Таким образом, параллельными вычислениями сегодня занимаются уже все ученые, ставящие своей задачей не отставать от времени и поддерживать актуальность своих исследований. Но даже прикладному вычислителю изучение данной области необходимо начинать именно с вычислительных систем, поскольку для того, чтобы довести производительность своей параллельной программы хотя бы до нескольких десятков процентов от пиковой, необходимо производить оптимизацию под конкретную систему, на которой будет исполняться данная задача. Необходимо учитывать большое число факторов: характер размещения памяти (системы с общей или распределенной памятью), размер кэш-памяти процессоров, пропуская способность связи между узлами и т. д.

Несмотря на высокую цену, суперкомпьютеры и вычислительные кластеры активно приобретают научные центры, учебные заведения и предприятия промышленного сектора. Связано это с тем, что у исследователей возникает все большее число задач, которые не могут быть решены иначе, как с применением высокопроизводительной вычислительной техники. К сожалению, коэффициент загрузки высокопроизводительных вычислительных систем наукоемкими задачами остается весьма низким, что лишний раз подчеркивает актуальность изучения структуры и устройства параллельных ЭВМ.

Первые три главы учебного пособия посвящены обзору основных решений, применяемых в современных вычислительных системах. Приведена краткая

история развития высокопроизводительной вычислительной техники, дана классификация архитектур высокопроизводительных вычислительных систем, описаны основные компоненты любой вычислительной системы и технологии, применяемые при их построении. Особое внимание уделено архитектурам вычислительных систем и системам команд процессоров – устройств, наиболее влияющих на производительность компьютера.

Важнейшей характеристикой любой вычислительной системы для пользователя является её производительность. Методам оценки производительности и стандартам в этой области посвящена 4 глава данного пособия.

В зависимости от архитектуры целевой машины вычислителю приходится выбирать тип параллелизма и модель обмена данными между единицами вычислительного процесса (потоками или процессами). О необходимости учета особенностей аппаратного обеспечения при проведении вычислительных расчетов и о том, как адекватно организовать обмен сообщений между вычислительными устройствами системы излагается в 5 главе.

Программы пользователя выполняются под управлением некоторой операционной системы. В 6 главе описываются основные подходы, применяемые при создании операционных систем, использующих ресурсы параллельных вычислительных систем, которые разделяются между одновременно исполняемыми процессами и/или потоками параллельных программ. В данной главе также приведены основные программные решения, предоставляемые операционными системами для организации доступа без конфликтов конкурирующим процессам и потокам к ресурсам, а также стратегии планирования процессов, управляющих очередностью исполнения нескольких разнородных задач.

Для организации вычислительного процесса на суперкомпьютерах в центрах коллективного пользования необходим достаточно большой набор инструментальных средств. В их числе можно назвать системы автоматической установки и настройки ОС на множестве вычислительных узлов кластера, средства мониторинга аппаратуры и загрузки ресурсов суперкомпьютера, системы пакетной обработки заданий и др. Отладка параллельных программ – достаточно сложное и трудоемкое занятие, если у пользователя нет необходимых инструментов, помогающих ему в этом процессе. Об этих и некоторых других программных системах описывается в главе 7.

Дальнейшим логическим ходом за построением центров коллективного пользования вычислительными ресурсами в пределах предприятий является объединение ресурсов нескольких центров, обладающих разнородным аппаратным и программным обеспечением в единый вычислительный пул.

Возможности пользователей в этом случае сильно расширяются за счет возможности выбора платформы, подходящей для конкретной задачи. Эта идея стала именоваться GRID. О программном обеспечении, позволяющем разворачивать GRID-системы также рассказывается в главе 7.

Значительную роль при оптимизации программы играет то, насколько тщательно пользователь учитывает особенности центральных процессоров вычислительного комплекса. Поэтому четкое понимание архитектуры и принципов функционирования процессоров необходимо для эффективного использования ресурсов суперкомпьютера. В главе 8 представлена информация о наиболее распространенных в настоящее время процессорах.

Параллельно с главным направлением развития высокопроизводительной вычислительной техники также имеют место альтернативные решения. Некоторые задачи как нельзя лучше подходят для исполнения на компьютерах с нетрадиционной архитектурой и иногда дают выигрыш в несколько порядков. Некоторые типы таких архитектур и мотивации их создания описываются в главе 9.

В приложениях приведены списки Top500 и Top 50, содержащие информацию о вычислительных комплексах, обладающих максимальной производительностью в мире и странах СНГ соответственно, а также представлено несколько хорошо известных суперкомпьютеров, описания которых выступают в данном пособии в качестве примеров реально используемых высокопроизводительных вычислительных систем.

Пособие создано при поддержке гранта Министерства образования и науки Российской Федерации РНП.3.2.3.13048 “Создание системы научно-методического обеспечения образовательными ресурсами учебных заведений для подготовки специалистов по высокопроизводительным распределенным вычислениям” в рамках аналитической ведомственной целевой программы “Развитие научного потенциала высшей школы” (2007-2008 годы).

Глава 1. Краткая история развития высокопроизводительных вычислительных систем

1.1. Развитие зарубежных высокопроизводительных вычислительных систем

В настоящее время сфера применения высокопроизводительных вычислительных систем (**ВВС**) непрерывно расширяется, охватывая все новые области в различных отраслях науки, бизнеса и производства. Стремительное развитие кластерных систем создает условия для использования высокопроизводительной вычислительной техники в реальном секторе экономики.

Все современные микропроцессоры используют тот или иной вид параллельной обработки, однако идеи увеличения производительности вычислительных систем за счет параллелизма появились достаточно давно. Изначально они внедрялись в самых передовых компьютерах своего времени. После должной отработки технологии и удешевления производства эти идеи уже внедрялись в компьютеры среднего класса, и далее все это в полном объеме воплощалось в рабочих станциях и персональных компьютерах [6, 11, 24].

IBM 701. Все первые компьютеры (**EDSAC**, **EDVAC**, **UNIVAC**) имели разрядно-последовательную память, из которой слова считывались последовательно бит за битом. Одним из первых коммерческих параллельных компьютеров считается мэйнфрейм **IBM 701** (1953 г.), имеющий разрядно-параллельную память и разрядно-параллельную арифметику. Однако наибольшую популярность получила модель **IBM 704** (1955 г.), в которой была впервые применена память на ферритовых сердечниках и аппаратное арифметическое устройство с плавающей точкой.

IBM 709 (1958 г.) – независимые процессоры ввода/вывода. Процессоры первых компьютеров сами управляли вводом/выводом. Однако скорость работы самого быстрого внешнего устройства (в то время – 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 достаточно удачной - вместе с модификациями было продано около 400 экземпляров.

IBM STRETCH (1961 г.) – опережающий просмотр вперед, расслоение памяти. В 1956 году компания IBM подписывает контракт с научной лабораторией Лос-Аламос 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. Проект STRETCH руководством компании IBM первоначально был признан неудачным, но на его базе были созданы следующие, более удачные модели. К моменту начала проекта семейство мэйнфреймов первого поколения (IBM 701, 703, 704) обеспечило IBM монопольное положение (ей принадлежало 60-70 % компьютерного рынка). Однако в области высокопроизводительных вычислений у компании IBM таких успехов не было. Чтобы исправить положение, была задумана модель STRETCH, которая по производительности должна была превзойти IBM 704 на два порядка. Создатели STRETCH первыми попытались преодолеть “бутылочное горло” фон-неймановской архитектуры – они сделали упор на быструю арифметику и транзисторные регистры, что позволило минимизировать число обращений к медленной памяти. Увеличение числа регистров было абсолютно новым решением, оно позволяло выполнять целый ряд операций без обращения в память, к которой обращалось большинство машин того времени, при выполнении различных операций. Список решений, воплощенных в STRETCH, позволяет оценить, насколько этот компьютер приблизил архитектуру к современному ее пониманию:

- мультипрограммный режим работы;
- защита памяти;
- система прерываний;
- конвейерная обработка данных;
- чередование адресов в памяти;
- спекулятивное выполнение;
- просмотр вперед, совмещение операций в памяти с арифметическими операциями;
- концепция шины памяти;
- работа двух процессоров на общую память;
- использование восьми битов для кодировки символа.

Много лет спустя, разработчики первых микропроцессоров заново открывали для себя то, что впервые было изобретено создателями IBM STRETCH. Разумеется, все это было воплощено на инженерном уровне своего времени, громоздко и дорого: компьютер занимал площадь около 250 кв. метров, весил около 100 тонн, один только процессор потреблял 21 кВт электроэнергии, число использованных транзисторов – 169200 (в настоящее время – миллиарды), тактовая частота – 3,3 МГц, размер оперативной памяти около 300 Кбайт, емкость жесткого диска – 10 Мбайт.

ATLAS (1963 г.) – конвейер команд. Впервые конвейерный принцип выполнения команд был использован в 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 выполнения 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 (**ОС**), основанная на использовании виртуальной памяти и системы прерываний.

CDC 6600 (1964 г.) – независимые функциональные устройства. Фирма Control Data Corporation (CDC) при непосредственном участии одного из ее основателей, Сеймура Крэя (Seymour Cray) выпустила компьютер CDC 6600 – первый компьютер, в котором использовалось несколько независимых функциональных устройств. Машина имела громадный успех на научном рынке, активно вытесняя машины фирмы IBM. Некоторые параметры компьютера CDC 6600:

- 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 1мкс;
- 10 независимых функциональных устройств.

CDC 7600 (1969 г.) – конвейерные независимые функциональные устройства (**ФУ**). Компания CDC выпускает компьютер CDC 7600 с восемью независимыми конвейерными функциональными устройствами – сочетание параллельной и конвейерной обработки. Основные параметры:

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

ILLIAC IV (1974 г.) – матричные процессоры. ILLIAC IV был построен Иллинойским университетом (США) совместно с корпорацией Burroughs по заказу NASA. Архитектурно ILLIAC IV представлял собой мультипроцессорную систему (первоначально было задумано 256 процессорных элементов (ПЭ) = 4 квадранта по 64ПЭ с возможностью реконфигурации: 2 квадранта по 128ПЭ или 1 квадрант из 256ПЭ), такт 40 нс, производительность 1 GFLOPS (подробнее в главе 4). Работы были начаты в 1967 году и к концу 1971 изготовлена система из 1 квадранта, в 1974 г. она введена в эксплуатацию, доводка велась до 1975 года. ILLIAC IV стал первым компьютером, в котором использовалась быстрая память на микросхемах. Центральная часть ILLIAC IV – это устройство управления (УУ) + матрица из 64 ПЭ:

- УУ – это простая ЭВМ с небольшой производительностью, управляющая матрицей ПЭ; все ПЭ матрицы работали в синхронном режиме, выполняя в каждый момент времени одну и ту же команду, поступившую от УУ, но над своими данными;
- ПЭ имел собственное арифметико-логическое устройство (АЛУ) с полным набором команд, оперативную память (ОП) – 2 Кслова по 64 разряда, цикл памяти 350 нс. Каждый ПЭ имел непосредственный доступ только к своей оперативной памяти.

Данный проект оказал огромное влияние на архитектуру последующих машин, построенных по схожему принципу.

CRAY 1 (1976 г.) – векторно-конвейерные процессоры. В 1972 году С. Крэй основал компанию Cray Research, которая в 1976 г. выпустила первый векторно-конвейерный компьютер CRAY 1:

- время такта 12.5 нс;
- 12 конвейерных функциональных устройств;
- пиковая производительность 160 миллионов операций в секунду;
- оперативная память до 1 Мслова (слово - 64 разряда);
- цикл памяти 50 нс.

Главным новшеством являлось введение векторных команд, работающих с целыми массивами независимых данных и позволяющих эффективно использовать конвейерные функциональные устройства.

В настоящее время можно выделить следующие основные направления развития высокопроизводительной вычислительной техники:

- векторно-конвейерные компьютеры;
- массивно-параллельные компьютеры с распределенной памятью;
- параллельные компьютеры с общей памятью;
- комбинации перечисленных выше компьютеров.

1.2. Развитие отечественных высокопроизводительных вычислительных систем

История отечественных параллельных компьютеров начинается с компьютера “Эльбрус-1”, который был создан в 1978 году в ИТМиВТ им. С. А. Лебедева АН СССР под руководством В. С. Бурцева. Компьютер имел модульную конструкцию и мог включать от одного до десяти процессоров на базе схем средней интеграции. Быстродействие компьютера достигало 15 млн операций в секунду. Объем оперативной памяти, общей для всех процессоров, составлял до 220 машинных слов (64 Мб). Советский суперкомпьютер стал первой в мире коммерческой ЭВМ, использующей суперскалярную архитектуру. Массовое ее использование за рубежом началось лишь в 1990-х годах с появлением процессоров Intel Pentium. Для организации передачи потоков данных между оперативной памятью и периферийными устройствами в “Эльбрусе” могли использоваться специальные процессоры ввода-вывода. Таких процессоров в системе могло быть до четырех штук, они имели свою собственную память и работали параллельно с центральным процессором. В состав вычислительных комплексов “Эльбрус-1” также могли входить специализированные процессоры для трансляции команд ЭВМ БЭСМ-6, быстрого преобразования Фурье и некоторые другие аппаратные системы для решения узкоспециализированных задач [30].

М-10 (1979 г., НИИВК, рук. М. А. Карцев) – 32-разрядная многопроцессорная система М-10 в составе 16 процессоров с производительностью 5 млн инструкций в секунду каждый.

ПС-2000 (1983 г., ИПП, рук. И. В. Прангишвили) – матричная суперЭВМ, содержащая до 64 процессорных элементов, построена на микропроцессорной основе и достигала производительности 200 млн инструкций в секунду (самая быстродействующая в СССР на тот момент).

Эльбрус-2 (1985 г., ИТМиВТ, рук. В. С. Бурцев) – теговая архитектура (каждое слово памяти имеет кроме информационной части, содержащей элемент данных, ещё и управляющую часть – тег элемента, на основании которого аппаратура процессора динамически выполняет выбор нужного варианта операции и контроль типов операндов), ориентация на языки высокого уровня класса Алгол (автокод Эльбрус Эль-76, автор В. М. Пентковский). При компиляции программа на Эль-76 переводилась в безоперандные команды стековой архитектуры. Пиковая производительность до 125 млн операций в секунду, объем оперативной памяти вырос до 144 Мб, максимальная пропускная способность каналов ввода-вывода составляла 120 Мбайт/с.

МАРС-Т (1986 г., СО РАН, рук. В. Е. Котов и А. Г. Марчук) - модульная асинхронная расширяемая система, основанная на иерархии функциональных модулей, которые обеспечивают параллельную обработку. Основой суперкомпьютера МАРС-Т являлся транспьютер КРОНОС, созданный в 1986 г. и выгодно отличавшийся увеличенной длиной слова, упрощенной выборкой команд и введением функций асинхронных транспьютерных связей в ряде команд.

Электроника ССБИС ("Красный Cray") (1980-1993 гг., ИПК РАН, рук. В. А. Мельников) – векторно-конвейерный суперкомпьютер с массовой памятью на полупроводниках, которая занимала промежуточное положение между оперативной памятью и внешними дисками, и предназначалась для хранения активно используемых файлов и ликвидировала дисбаланс между низкой скоростью передачи информации с дисков и быстрой обработкой данных процессором.

Эльбрус-3 (1986-1994 гг., ИТМиВТ, рук. Б. А. Бабаян) – “постсуперскалярный” подход (архитектура широкого командного слова). Не был запущен в серийное производство.

Эльбрус-3-1 (МКП) (1993 г., ИТМиВТ, рук. А. А. Соколов) – возможность подключения процессоров с различной специализацией (радиолокационная обработка, структурная обработка, быстрые преобразования Фурье и т. д.), работа с несколькими потоками команд (обеспечивалось наличием нескольких счетчиков команд). Одновременно на едином поле памяти в процессоре выполнялось до четырех потоков команд.

ОСВМ РАН (оптическая сверхвысокопроизводительная вычислительная машина) (1993 г., ИПК, ИВВС, рук. В. С. Бурцев) - использования оптики в суперЭВМ – в системах связи и коммутации, на основе оптических принципов разработана новая архитектура, предусматривающая новую организацию вычислительного процесса, исключающего человека из распределения вычислительных ресурсов и обеспечивающего структурную надежность. Не был запущен в серийное производство.

1.3. Классификация компьютеров

Большое разнообразие вычислительных систем требует введения некоторой классификации. Эта классификация должна однозначно относить ту или иную вычислительную систему к некоторому классу, который, в свою очередь, должен достаточно полно ее характеризовать.

Одна из возможных классификаций вычислительных систем – это объединение компьютеров в группы, в соответствии с их основными характеристиками и областью применения [24].

Персональные компьютеры. Персональные компьютеры (ПК) появились в результате эволюции миникомпьютеров при переходе элементной базы машин с малой и средней степенью интеграции на большие и сверхбольшие интегральные схемы. ПК, благодаря своей низкой стоимости, очень быстро завоевали хорошие позиции на компьютерном рынке и создали предпосылки для разработки новых программных средств, ориентированных на конечного пользователя.

Серверы. Сервер – специализированный компьютер, выделенный для выполнения на нем сервисного программного обеспечения: прикладных многопользовательских коммерческих и бизнес-систем, включающих системы управления базами данных и обработки транзакций, крупных издательских систем, сетевых приложений, систем обслуживания коммуникаций и т. д.

Существует несколько типов серверов, ориентированных на разные области применения: файл-сервер, сервер базы данных, принт-сервер, вычислительный сервер, сервер приложений. Таким образом, тип сервера определяется видом ресурса, которым он владеет (файловая система, база данных, принтеры, процессоры или прикладные пакеты программ). Серверы, как правило, поставляются в однопроцессорной и мультипроцессорной конфигурациях. В системах из этой категории обычно устанавливаются более высокоскоростные сетевые интерфейсы, модули памяти и жесткие диски с большим, по сравнению с ПК, общим объемом. С точки зрения архитектуры однопроцессорный сервер не слишком отличается от персонального компьютера. Он просто работает быстрее, занимает больше места, содержит больше дискового пространства и устанавливает более скоростные сетевые соединения. Серверы работают под управлением тех же операционных систем, что и персональные компьютеры

Мейнфреймы. Мейнфрейм – это синоним понятия “большая универсальная ЭВМ”. Мейнфреймы и до сегодняшнего дня остаются наиболее мощными (не считая суперкомпьютеров) вычислительными системами общего назначения, обеспечивающими непрерывный круглосуточный режим эксплуатации. Они могут включать один или несколько процессоров, каждый из которых, в свою очередь, может оснащаться векторными сопроцессорами (ускорителями операций с суперкомпьютерной производительностью). Основным слабым местом архитектуры мейнфреймов считается ее закрытость. Все компоненты архитектуры уникальны: аппаратура, процессоры, память, шины, интерфейсы.

Суперкомпьютеры. Различие между суперкомпьютерами и минисуперкомпьютерами весьма условно и отражает только мощность машины. Вообще нет четкого понимания, как отличить суперкомпьютер от обычного. Выделяют основные признаки, характеризующие суперЭВМ, среди которых кроме высокой производительности следует отметить:

- самый современный технологический уровень;
- специфические архитектурные решения, направленные на повышение быстродействия (например, наличие операций над векторами);
- цена (обычно свыше 1-2 млн долл.).

Кластеры. Кластеры создаются из вычислительных модулей, объединенных системой связи или разделяемыми устройствами внешней памяти, например дисковыми массивами. В настоящее время для построения кластерных систем используются либо специализированные фирменные коммуникационные средства, либо универсальные локальные и глобальные сети, либо дисковые массивы с высокоскоростными контроллерами. Размер кластера варьируется от нескольких модулей до нескольких десятков и сотен модулей. Специальное программное обеспечение производит контроль работоспособности узлов. Если какой-либо из узлов кластера считается вышедшим из строя, его ресурсы, например, дисковое пространство, и программы переназначаются на другие узлы.

Кластеры характеризуются следующими признаками:

- компоновка из компонентов высокой степени готовности;
- построение на основе стандартных программно-аппаратных парадигм: Open Software Foundation Distributed Computing Environment (**OSF DCE**) и Open Network Computing (**ONC**), поддерживающих общие имена и возможности доступа;
- согласованность наборов прикладных программ, форматов данных;
- общая для всех вычислительных узлов (**ВУ**) кластера организация информационной безопасности, общий алгоритм обнаружения неисправностей и реконфигурации для обеспечения функционирования при наличии отказов;
- ограниченная масштабируемость по числу ВУ.

Метакомпьютеры. Этот термин возник вместе с развитием высокоскоростной сетевой инфраструктуры в начале 90-х годов и относился к объединению нескольких разнородных вычислительных ресурсов в локальной сети организации для решения одной задачи. Основная цель построения метакомпьютера в то время заключалась в оптимальном распределении частей

работы по вычислительным системам различной архитектуры и различной мощности.

В дальнейшем исследования в области технологий мета-компьютинга были развиты в сторону однородного доступа к вычислительным ресурсам большого числа (вплоть до нескольких тысяч) компьютеров в локальной или глобальной сети. Компонентами "метакомпьютера" могут быть как простейшие ПК, так и мощные высокопроизводительные системы. Метакомпьютер может не иметь постоянной конфигурации – отдельные компоненты могут включаться в его конфигурацию или отключаться от нее; при этом технологии метакомпьютинга обеспечивают непрерывное функционирование системы в целом. Современные исследовательские проекты в этой области направлены на обеспечение прозрачного доступа пользователей через Интернет к необходимым распределенным вычислительным ресурсам, а также прозрачного подключения простаивающих вычислительных систем к метакомпьютерам.

Очевидно, что наилучшим образом для решения на мета-компьютерах подходят задачи переборного и поискового типа, где вычислительные узлы практически не взаимодействуют друг с другом и основную часть работы производят в автономном режиме. Основная схема работы в этом случае примерно такая: специальный агент, расположенный на вычислительном узле (компьютере пользователя), определяет факт простоя этого компьютера, соединяется с управляющим узлом мета-компьютера и получает от него очередную порцию работы (область в пространстве перебора). По окончании счета по данной порции вычислительный узел передает обратно отчет о фактически проделанном переборе или сигнал о достижении цели поиска. В настоящее время идеи, заложенные в основу метакомпьютеров, воплощаются в GRID-системах.

Классификация архитектур параллельных вычислительных систем необходима для лучшего понимания исследуемой предметной области. Попытки систематизировать все множество архитектур начались после опубликования М. Флинном первого варианта классификации вычислительных систем в конце 60-х годов и непрерывно продолжают по сей день. Необходима такая классификация, которая бы помогла разобраться с тем, что представляет собой каждая архитектура, как архитектуры взаимосвязаны между собой, что должно учитываться для написания действительно эффективных программ, какой класс архитектур наиболее эффективен для решения требуемого класса задач.

К настоящему времени спроектированы и опробованы сотни различных компьютеров, использующих в своей архитектуре тот или иной вид параллельной обработки данных. В научной и технической литературе представлены более десятка различных названий, характеризующих лишь общие принципы

функционирования параллельных машин: векторно-конвейерные, массивно-параллельные, компьютеры с широким командным словом, гиперкубы, спец-процессоры, мультипроцессоры, иерархические и кластерные компьютеры, матричные ЭВМ и многие другие. Ясно, что если добавить еще и данные о таких важных параметрах, как организация памяти, топология связи между процессорами, синхронность работы отдельных устройств или способ исполнения арифметических операций, то число различных архитектур значительно вырастет.

Классификация зависит от того, для кого она создается и на решение какой задачи направлена. Деление компьютеров на персональные ЭВМ, рабочие станции, мини-ЭВМ, супер-ЭВМ, основано на стоимостном признаке, однако такое деление не подходит для классификации параллельных архитектур с учетом их эффективности для различных задач и не позволит выбрать наиболее оптимальный подход к построению ПО (язык программирования, параллельная реализация и т. д.). От большого разнообразия параллельных компьютеров страдает конечный пользователь, потому что он вынужден каждый раз подбирать наиболее эффективный алгоритм под определенную платформу, решать проблемы совместимости и переносимости программного обеспечения.

1.4. Требования, предъявляемые к ВВС

Стремительное развитие науки и техники, решение поставленных прежде проблем постоянно порождает поток вопросов и ставит новые, как правило, более сложные задачи. Высокая производительность современных суперЭВМ сегодня является явно недостаточной для многих направлений:

- научные исследования, сводящиеся к моделированию сложно воспроизводимых процессов;
- военные нужды, задачи типа **СЗИ** (command, control, communication & intelligence);
- коммерческая сфера (задачи, предполагающие интенсивную обработку транзакций для сверхбольших баз данных (**БД**) и т. п.).

В соответствии с этими проблемами к высокопроизводительным системам предъявляются как общие, так и специфические требования.

Отношение стоимости/производительность. Большая универсальная вычислительная машина (мейнфрейм) или суперкомпьютер стоят дорого. Для достижения поставленных целей при проектировании высокопроизводительных конструкций приходится игнорировать стоимостные характеристики.

Суперкомпьютеры фирмы Cray Research и высокопроизводительные мейн-фреймы компании IBM относятся именно к этой категории компьютеров. Другим крайним примером может служить низкостоймостная конструкция, где производительность принесена в жертву для достижения низкой стоимости. К этому направлению относятся персональные компьютеры различных клон-ов IBM PC. Между этими двумя крайними направлениями находятся конструкции, основанные на отношении стоимость/производительность, в которых разработчики находят баланс между стоимостными параметрами и производительностью. Типичными примерами такого рода компьютеров являются миникомпьютеры и рабочие станции. Для сравнения различных компьютеров между собой обычно используются некоторые методики измерения производительности (подробнее в главе 4). Эти методики позволяют разработчикам и пользователям использовать полученные в результате испытаний количественные показатели для оценки тех или иных технических решений, и, в конце концов, именно производительность и стоимость дают пользователю рациональную основу для решения вопроса, какой компьютер выбрать.

Надежность и отказоустойчивость. Важнейшей характеристикой вычислительных систем является *надежность*. Повышение надежности основано на принципе предотвращения неисправностей путем снижения интенсивности отказов и сбоев за счет применения электронных схем и компонентов с высокой и сверхвысокой степенью интеграции, снижения уровня помех, облегченных режимов работы схем, обеспечение тепловых режимов их работы, а также за счет совершенствования методов сборки аппаратуры. Понятие надежности включает не только аппаратные средства, но и программное обеспечение. Главной целью повышения надежности систем является целостность хранимых в них данных.

Отказоустойчивость – это свойство вычислительной системы, обеспечивающее ей, как логической машине, возможность продолжения действий, заданных программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения.

Масштабируемость представляет собой возможность наращивания числа и мощности процессоров, объемов оперативной и внешней памяти и других ресурсов вычислительной системы. Масштабируемость должна обеспечиваться архитектурой и конструкцией компьютера, а также соответствующими средствами программного обеспечения.

Возможность масштабирования системы определяется не только архитектурой аппаратных средств, но также зависит от заложенных свойств программного обеспечения. Масштабируемость программного обеспечения

затрагивает все уровни от простых механизмов передачи сообщений до работы с такими сложными объектами как мониторы транзакций и всей среды прикладной системы. В частности, программное обеспечение должно минимизировать трафик межпроцессорного обмена, который может препятствовать линейному росту производительности системы. Аппаратные средства (процессоры, шины и устройства ввода/вывода) являются только частью масштабируемой архитектуры, на которой программное обеспечение может обеспечить предсказуемый рост производительности. Однако, простой переход, например, на более мощный процессор может привести к перегрузке других компонентов системы. Это означает, что действительно масштабируемая система должна быть сбалансирована по всем параметрам.

Совместимость и мобильность программного обеспечения. В настоящее время одним из наиболее важных факторов, определяющих современные тенденции в развитии информационных технологий, является ориентация компаний-поставщиков компьютерного оборудования на рынок прикладных программных средств. Это объясняется прежде всего тем, что для конечного пользователя по большей мере важно программное обеспечение, позволяющее решить его задачи, а не выбор той или иной аппаратной платформы. Переход от однородных сетей программно совместимых компьютеров к построению неоднородных сетей, включающих компьютеры разных фирм-производителей, в корне изменил и точку зрения на саму сеть: из сравнительно простого средства обмена информацией она превратилась в средство интеграции отдельных ресурсов – мощную распределенную вычислительную систему, каждый элемент которой (сервер или рабочая станция) лучше всего соответствует требованиям конкретной прикладной задачи. В соответствии с этим появляются новые требования:

- гибкость по количеству и составу аппаратных средств и программного обеспечения;
- кроссплатформенность ПО;
- единый интерфейс пользователя.

В условиях жесткой конкуренции производителей аппаратных платформ и программного обеспечения сформировалась концепция открытых систем, представляющая собой совокупность стандартов на различные компоненты вычислительной среды, предназначенных для обеспечения мобильности программных средств в рамках неоднородной, распределенной вычислительной системы.

Одним из вариантов моделей открытой среды является модель **OSE** (Open System Environment), предложенная комитетом IEEE POSIX. На основе этой

модели национальный институт стандартов и технологии США выпустил документ “Application Portability Profile (APP). The U.S. Government’s Open System Environment Profile OSE/1 Version 2.0”, который определяет рекомендуемые для федеральных учреждений США спецификации в области информационных технологий, обеспечивающие мобильность системного и прикладного программного обеспечения. Все ведущие производители компьютеров и программного обеспечения в США в настоящее время придерживаются требований этого документа.

Вопросы для самоконтроля

1. Дайте определение архитектуры вычислительной системы.
2. Дайте характеристику основных зарубежных ВВС.
3. Дайте характеристику отечественных ВВС.
4. В чем необходимость классификации вычислительных систем?
5. В чем заключаются проблемы классификации ВВС?
6. Перечислите основные направления развития высокопроизводительной вычислительной техники в настоящее время.
7. Поясните требование к ВВС “Отношение стоимость/ производительность”? В чем оно состоит?
8. Поясните требование к ВВС “Надежность и отказоустойчивость”? В чем оно состоит?
9. Поясните требование к ВВС “Масштабируемость”? В чем оно состоит?
10. Поясните требование к ВВС “Совместимость и мобильность программного обеспечения”? В чем оно состоит?

Глава 2. Архитектура вычислительных систем

2.1. Типы архитектур ЭВМ

Понятие архитектуры вычислительной системы является достаточно широким, поскольку под архитектурой можно понимать и способ параллельной обработки данных, используемый в системе, и организацию памяти, и топологию связи между процессорами, и способ исполнения системой арифметических операций. Попытки систематизировать все множество архитектур впервые были предприняты в конце 60-х годов XX века и продолжают в настоящее время.

Фон-Неймановская архитектура

Основные архитектурно-функциональные принципы построения ЭВМ были разработаны и опубликованы в 1946 г. венгерским математиком и физиком Джоном фон Нейманом и его коллегами Г. Голдстейном и А. Берксом в работе “Предварительное обсуждение логического конструирования электронного вычислительного устройства”. Ими были выделены основополагающие принципы ЭВМ [4, 41]:

- принцип программного управления выполнением программы;
- принцип хранимой в памяти программы.

Данные принципы легли в основу понятия фон-Неймановской архитектуры (рис. 2. 1), недостатком которой является ограниченный поток команд, поступающих на исполнение.

Машины фон-Неймана хранят программу и данные в одной и той же области памяти. В машинах этого типа команды содержат указание о том, что выполнить и адрес данных, подлежащих обработке. Внутри этой машины имеются два основных функциональных блока. Первый – арифметико-логическое устройство, которое выполняет самые важные операции: умножение, сложение, вычитание и другие. На основе этих простых операций можно создать

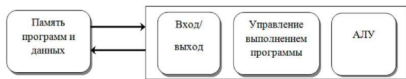


Рис. 2. 1. Фон-Неймановская архитектура

сложное программное обеспечение, например текстовый редактор. Второй блок – устройство ввода/вывода, управляющее потоком внешних данных (печатать, считывание и т. д.).

Гарвардская архитектура

Единственное отличие Гарвардской архитектуры от фон-Неймановской состоит в том, что память программ и память данных разделены и используют физически разделенные линии передачи (рис. 2. 2). Это позволяет подобной машине пересылать команды и данные одновременно (допускается одновременный анализ более одной команды), т. е. распараллеливание выполнения программы между несколькими исполнительными устройствами (процессорами).

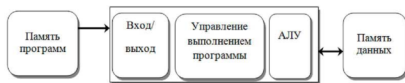


Рис. 2. 2. Гарвардская архитектура

Выполняемые действия определяются блоком управления и АЛУ, которые вместе являются основой центрального процессора. Центральный процессор выбирает и исполняет команды из памяти последовательно, адрес очередной команды задается “счетчиком адреса” в блоке управления. Этот принцип исполнения называется последовательной передачей управления.

Порядок выполнения команд определяется наличием исходной информации для выполнения каждой из них. Если несколько команд готовы к выполнению, то принципиально возможно их назначение для выполнения таким же количеством свободных процессоров. Такая конструкция может значительно увеличить производительность. Гарвардская машина, как и

фон-Неймановская, имеет арифметико-логическое устройство и устройство ввода/вывода. Однако платой за высокую скорость является высокая цена процессора (т. к. память данных и память программ разделены, поэтому на кристалле необходимо иметь в два раза больше выводов адреса и данных, что и приводит к росту цены).

Персональные компьютеры общего назначения используют процессоры, построенные согласно фон-Неймановской архитектуре. Другие семейства компьютеров также используют фон-Неймановскую архитектуру. Однако, у этой архитектуры есть узкое место – единый тракт. Гарвардская архитектура является наиболее подходящей для специализированных микропроцессоров, предназначенных для решения прикладных задач в реальном времени.

Фон-Неймановская и Гарвардская архитектуры – не единственные варианты построения ЭВМ. Существуют и другие архитектуры, которые не соответствуют указанным принципам.

Стековая архитектура

Особенностью вычислителей, построенных по стековой архитектуре, является то, что входные, промежуточные и результирующие данные хранятся в памяти данных с последовательным доступом.

Для осуществления операции необходимо записать данные на вершину стека. Вершина и следующая за вершиной позиции стека подаются на вход АЛУ, выход АЛУ, в свою очередь, может управлять вершиной стека.

Достоинства:

- простота аппаратной реализации;
- простота записи алгоритмов вычисления;
- простота мнемонического описания микроопераций (с одним операндом или вообще без операндов).

Недостатки:

- стек – запоминающее устройство с последовательным доступом обладающее низкой скоростью работы;
- немасштабируемая архитектура.

Теговая архитектура

Архитектура компьютера, в которой каждый тип данных сопровождается специальной меткой (тегом), называется теговой. Обработка данных базируется на этих метках.

При теговой организации памяти каждое, хранящееся в памяти или реестре, слово снабжается указателем (тегом), определяющим тип данных. В поле

тега обычно указывается не только тип, но и длина (формат) и некоторые другие параметры. Теги формируются компилятором. Наличие тегов придает хранящимся данным свойство самоопределяемости.

Достоинства:

- небольшой набор команд – теговая организация памяти позволяет достигнуть инвариантности команд относительно типов и форматов операндов;
- упрощение структуры процессора;
- облегчение программирования, отладки, компиляции, так как отпадает необходимость выбора типа команды в зависимости от типа данных.

Теговая организация памяти способствует реализации принципа независимости программ от данных. Использование тегов экономит память, так как в программах обычных машин имеется большая информационная избыточность на задание типов и размеров операндов при их использовании несколькими командами.

Недостатки:

- замедление работы микропроцессора - установление соответствия типа команды типу данных, выполняемое в обычных ЭВМ на этапе компиляции, в машинах с тегами переносится на этап выполнения программы.

В подавляющем большинстве современные компьютеры, включая и сложные многопроцессорные комплексы, основаны на фон-Неймановской или Гарвардской архитектурах.

2.2. Типы процессоров

Одним из важнейших компонентов ЭВМ является процессор. Двумя основными архитектурами процессоров, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники, являются архитектуры CISC и RISC.

CISC (Complex Instruction Set Computing – вычисления с полным набором команд) – архитектура процессоров, характеризующаяся следующими свойствами:

- нефиксированное значение длины команды;
- исполнение операций кодируется в одной инструкции;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию;
- большое количество методов адресации;

- большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокоуровневых языков программирования и выполняются за много тактов;
- большое количество форматов команд различной разрядности.

Основоположником CISC-архитектуры можно считать компанию IBM с базовой архитектурой System/360. Лидером в разработке микропроцессоров с полным набором команд считается компания Intel с серией x86 и Pentium. Эта архитектура является практическим стандартом для рынка микрокомпьютеров.

RISC (Reduced Instruction Set Computing – вычисления с сокращённым набором команд) – архитектура процессоров, характеризующаяся следующими свойствами:

- фиксированная длина машинных инструкций и простой формат команды;
- одна инструкция выполняет только одну операцию с памятью – чтение или запись;
- большое количество регистров общего назначения (32 и более).

Идея создания RISC процессоров появилась после того, как в 1970-х годах ученые из IBM обнаружили, что многие из функциональных особенностей традиционных процессоров игнорировались программистами. Отчасти это был побочный эффект сложности компиляторов. Поэтому многие ранние RISC-процессоры даже не имели команд умножения и деления. В то время компиляторы могли использовать лишь часть из набора команд процессора. Развитие архитектуры RISC в значительной степени определялось прогрессом в области создания оптимизирующих компиляторов. Развитие техники компиляции позволяет эффективно использовать преимущества большого регистрового файла, конвейерной организации и большей скорости выполнения команд. Поскольку некоторые сложные операции использовались редко, они, как правило, были медленнее, чем те же действия, выполняемые набором простых команд. Это происходило из-за того, что создатели процессоров тратили гораздо меньше времени на улучшение сложных команд, чем на улучшение простых.

Современные компиляторы используют преимущества другой оптимизационной техники для повышения производительности, обычно применяемой в процессорах RISC – реализацию задержанных переходов и суперскалярной обработки, позволяющей в один и тот же момент времени выдавать на выполнение несколько команд.

Недостатки RISC архитектуры:

- с увеличением быстродействия растёт разрыв между быстрым процессором и медленной памятью, для увеличения скорости доступа к памяти необходимо использовать кэш память для буферизации потоков данных и команд, однако кэш память усложняет и удорожает систему в целом;
- RISC процессоры неэффективны на операциях вызова и возврата подпрограмм. Эффективность этого механизма критична для языков высокого уровня. Многие RISC процессоры используют большой регистровый файл с окнами для облегчения вызова подпрограмм. Однако, окно должно быть достаточно большое для сохранения локальных данных. Большой регистровый файл – это потеря наиболее драгоценных ресурсов процессора и замедление при переключении контекста на его сохранение и восстановление.

Тем не менее, современные процессоры Intel и AMD являются CISC-процессорами с RISC-ядром. Они непосредственно перед исполнением преобразуют CISC-инструкции процессоров x86 в более простой набор внутренних инструкций RISC. В микропроцессор встраивается аппаратный транслятор, превращающий команды x86 в команды внутреннего RISC-процессора. При этом одна команда x86 может порождать несколько RISC-команд. Исполнение команд происходит параллельно на суперскалярном конвейере. Это потребовалось для увеличения скорости обработки CISC-команд, так как известно, что любой CISC-процессор уступает RISC-процессорам по количеству выполняемых операций в секунду. В итоге, такой подход и позволил поднять производительность процессоров x86.

MISC (англ. Minimal Instruction Set Computer – вычисления с минимальным набором команд) – архитектура процессоров, характеризующаяся следующими свойствами:

- небольшое число чаще всего встречающихся команд;
- принцип **VLIW** (Very long instruction word – очень длинное командное слово) – укладка нескольких команд в одно большое слово, позволяет обрабатывать одновременно несколько потоков данных, обеспечивает выполнение группы непротиворечивых команд за один цикл работы процессора;
- порядок выполнения команд распределяется таким образом, чтобы в максимальной степени загрузить маршруты, по которым проходят потоки данных.

По мнению авторов концепции архитектуры MISC сокращение набора

команд является эффективным шагом по повышению производительности. Был исследован вопрос достижения высокой производительности с ограничением на максимальную простоту с целью определения, насколько современная технология производства транзисторов может уменьшить стоимость построения компьютера. Авторы MISC процессора пришли к выводу, что число команд должно быть в районе 32-х, для них используется 5-и разрядное поле команды, в каждое 20-и разрядное слово помещается четыре команды, а в каждое 32-х разрядное слово может поместиться шесть команд с запасом. Система команд MISC достаточно проста и помещается на одной странице. Сокращение набора команд является эффективным шагом по повышению производительности.

Матричный процессор объединяет множество функциональных устройств, логически объединенных в матрицу и работающих в стиле “один поток команд - множество потоков данных”. Данный процессор лучше всего приспособлен для решения задач, характеризующихся параллелизмом независимых объектов или данных.

Структура матричного процессора состоит из общего управляющего устройства, генерирующего поток команд и большого числа процессорных элементов, работающих параллельно и обрабатывающих каждая свой поток данных. Таким образом, производительность системы оказывается равной сумме производительностей всех процессорных элементов. На практике, чтобы обеспечить достаточную эффективность системы при решении широкого круга задач, необходимо организовать связи между процессорными элементами с тем, чтобы наиболее полно загрузить их работой.

Векторный процессор – это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных - векторы. Отличается от обычных процессоров, которые могут работать только с одним операндом в единицу времени. Векторные процессоры были распространены в сфере научных вычислений, где являлись основой большинства суперкомпьютеров с 1980-х до 1990-х гг.

2.3. Современные методы повышения производительности процессоров

Главным препятствием высокой скорости выполнения команд является необходимость их вызова из памяти. Разработчики архитектуры компьютеров стараются применять методы проектирования, известные под общим

названием “совмещение операций”, при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Эти общие методы включают два различных подхода: параллелизм и конвейеризацию.

Конвейерная обработка основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. При этом обработка любой машинной команды разделяется на несколько этапов, организуется передача данных от одного этапа к следующему. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах. Конвейерная обработка эффективна только при загрузке конвейера близкой к полной.

При **параллелизме** совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур (производится параллельная выдача нескольких команд в каждом такте), осуществляющих решение различных частей задачи. Существуют два типа подобного рода машин: суперскалярные машины и **VLIW**-машины (Very long instruction word – очень длинное командное слово).

Преимущества суперскалярной машины:

- малое воздействие на плотность кода, поскольку машина сама определяет, может ли быть выдана следующая команда, и нет ли необходимости следить за тем, чтобы команды соответствовали возможностям выдачи;
- возможность выполнения неоптимизированных программ, или программ, откомпилированных в расчете на более старую реализацию.

Преимущества VLIW-машины:

- архитектура машин с очень длинным командным словом позволяет сократить объем оборудования, требуемого для реализации параллельной выдачи нескольких команд (потенциально – чем большее количество команд выдается параллельно, тем больше эта экономия).

Ниже приведена сравнительная таблица суперскалярной машины и VLIW-машины.

Многоядерность. Еще одним способом повышения производительности процессора является помещение двух и более вычислительных ядер на одном кристалле. Многоядерный процессор — это центральный процессор, содержащий два и более вычислительных ядра на одном кристалле.

Суперскалярность	VLIW
Возможность выдачи на выполнение в каждом такте переменного числа команд, и планирования работы конвейеров как статически с помощью компилятора, так и с помощью аппаратных средств динамической оптимизации. Использование параллелизма на уровне команд путем посылки нескольких команд из обычного потока команд в несколько функциональных устройств. Если какая-либо команда в потоке команд является логически зависимой или не удовлетворяет критериям выдачи, на выполнение будут выданы только команды, предшествующие данной. Поэтому скорость выдачи команд в суперскалярных машинах является переменной.	VLIW-машины выдают на выполнение фиксированное количество команд, которые сформатированы, либо как одна большая команда, либо как пакет команд фиксированного формата. Планирование работы VLIW-машины всегда осуществляется компилятором. В VLIW-машинах полную ответственность за формирование пакета команд, которые могут выдаваться одновременно, несет компилятор, а аппаратура в динамике не принимает никаких решений относительно выдачи нескольких команд.

В существующих на сегодняшний день многоядерных процессорах кэш-память первого уровня у каждого ядра своя, а кэш 2-го уровня существует в нескольких вариантах:

- разделяемый, когда кэш расположен на одном с обеими ядрами кристалле и доступен каждому из них в полном объеме (процессоры семейства Intel Core);
- индивидуальный, когда отдельные кэши равного объема, интегрированные в каждое из ядер (процессоры семейства AMD Athlon).

Основным преимуществом многоядерных процессоров является возможность разделения запущенных программ на несколько одновременно выполняемых потоков. Способность процессора выполнять одновременно несколько программных потоков называется параллелизмом на уровне потоков (thread-level parallelism - **TLP**). Необходимость в TLP зависит от конкретной

ситуации. Большинство программ, написанных с расчётом на работу в однопоточном режиме, не смогут задействовать вычислительные мощности дополнительных ядер, в то же время операционная система, поддерживающая многоядерные процессоры, использует вычислительные мощности для собственных системных процессов.

Внутрипроцессорная многопоточность. Для всех современных конвейеризованных процессоров характерна одна и та же проблема - если при запросе к памяти слово не обнаруживается в кэшах первого и второго уровней, на загрузку этого слова в кэш уходит длительное время, в течение которого конвейер простаивает. Одна из методик решения этой проблемы называется внутрипроцессорной многопоточностью (on-chip multithreading). Она позволяет процессору одновременно управлять несколькими программными потоками и тем самым маскировать простои.

Принцип многопоточности – если программный поток 1 блокируется, процессор может обеспечить полную загрузку аппаратуры, запустив программный поток 2.

Данный принцип реализуется несколькими способами:

- **Мелкомодульная многопоточность** (fine-grained multithreading) – применяется к процессору, способному вызывать одну команду за такт. При мелкомодульной многопоточности простой процессора маскируется путем исполнения потоков “по кругу”, то есть в смежных циклах запускаются разные потоки. Поскольку разные программные потоки никак друг с другом не связаны, каждому из них нужен свой набор регистров. Он должен быть указан для каждой вызываемой команды, и тогда аппаратное обеспечение будет знать, к какому набору регистров при необходимости нужно обращаться. Поэтому максимальное число одновременно исполняемых программных потоков определяется в период разработки микросхемы. Обращения к памяти причины простоя не ограничиваются. Иногда для исполнения следующей команды требуется результат предыдущей команды, который еще не вычислен. В других случаях команда вызвана быть не может, так как она следует за условным переходом, направление которого еще неизвестно.
- **Крупномодульная многопоточность** (coarse-grained multithreading) – в данном случае программный поток продолжает выполняться последовательно, вплоть до простоя, далее происходит переключение на первую команду следующего программного потока. Так как каждый раз при простое команды теряется один цикл, по своей эффективности крупномодульная многопоточность, казалось бы, уступает мелкомодульной, однако

у нее есть одно существенное преимущество - за счет меньшего числа программных потоков значительно сокращается расход ресурсов процессора. При недостаточном количестве активных потоков эта методика оптимальна.

- **Синхронная многопоточность** (simultaneous multithreading) – данная методика представляет собой усовершенствованный вариант крупномодульной многопоточности, где каждый программный поток может запускать по две команды за такт, однако в случае простоя с целью обеспечения полной загрузки процессора запускаются команды следующего потока. При синхронной многопоточности полностью загружаются все функциональные блоки. В случае невозможности запуска команды из-за занятости функционального блока выбирается команда из другого потока.

Мультипроцессоры. Рост быстродействия процессоров ограничен физической средой передачи информации – скорость распространения электромагнитных волн в медных проводах и света в оптико-волоконных кабелях ограничена. Кроме того, чем быстрее работает процессор, тем сильнее он нагревается, поэтому возникает задача защиты его от перегрева.

Внутрипроцессорное повышение производительности (конвейеризация, параллелизм, многоядерность, многопоточность) в определенной степени помогает, но конвейеры и суперскалярная архитектура обычно повышают скорость работы в несколько раз. Чтобы увеличить производительность в десятки и более раз, необходимо создавать компьютеры с несколькими процессорами.

В мультипроцессоре все процессоры совместно используют общую физическую память. Все процессы, работающие в мультипроцессоре совместно, могут иметь единое виртуальное адресное пространство, отображенное на общую память. Два процесса имеют возможность легко обмениваться информацией – для этого один из них записывает данные в общую память, а другой их считывает. Поскольку каждый процессор может записывать информацию в любую часть памяти и считывать информацию из любой части памяти, чтобы не допустить каких-либо нестыковок, то их работа должна согласовываться программным обеспечением. Благодаря возможности взаимодействия двух и более процессов и легкостью работы с единой общей памятью, мультипроцессоры весьма популярны.

Мультикомпьютеры. Создание больших мультипроцессоров (с количеством процессоров более 100) представляет определенные трудности. Сложность заключается в том, чтобы связать все процессоры с общей памятью. Поэтому многие разработчики стали создавать системы без общей памяти, состоящие из большого числа взаимосвязанных компьютеров, у каждого из

которых имеется собственная память. Такие системы называются мультикомпьютерами. В них процессоры являются слабо связанными, в противоположность сильно связанным процессорам в мультипроцессорных системах.

Процессоры мультикомпьютера отправляют друг другу сообщения. Каждый компьютер не обязательно соединять со всеми другими, поэтому обычно в качестве топологий используются двух- и трехмерные решетки, а также деревья и кольца. В настоящее время уже работают мультикомпьютеры, содержащие несколько десятков тысяч процессоров.

2.4. Системы команд процессора

В команде процессора четко выделяют две части – операционную и адресную. Формат команды – это описание размеров и взаимного расположения структурных частей команды. Всегда стараются сделать так, чтобы команда занимала целое число элементов хранения. У разных процессоров системы команд существенно различаются, но в основе своей они очень похожи. Количество команд у процессоров также различно. У современных мощных процессоров количество команд достигает нескольких сотен. В то же время в процессорах с сокращенным набором команд (RISC-процессорах), в которых за счет максимального сокращения количества команд достигается увеличение эффективности и скорости их выполнения, их количество невелико.

В общем случае система команд процессора включает в себя следующие основные группы команд [5, 24]:

- команды пересылки данных;
- арифметические команды;
- логические команды;
- команды переходов;
- специальные команды.

Команды пересылки данных. Команды пересылки данных не требуют выполнения никаких операций над операндами. Операнды просто пересылаются из источника в приемник. Источником и приемником могут быть внутренние регистры процессора, ячейки памяти или устройства ввода/вывода. АЛУ в данном случае не используется.

Команды пересылки выполняют следующие важнейшие функции:

- загрузка (запись) содержимого во внутренние регистры процессора;
- сохранение в памяти содержимого внутренних регистров процессора;
- копирование содержимого из одной области памяти в другую;
- запись в устройства ввода/вывода и чтение из устройств ввода/вывода.

Также к командам пересылки данных относятся команды обмена информацией. Может быть предусмотрен обмен информацией между внутренними регистрами, между двумя половинами одного регистра или между регистром и ячейкой памяти.

Арифметические команды. Арифметические команды выполняют операции сложения, вычитания, умножения, деления, увеличения на единицу (**INC**-инкремент), уменьшения на единицу (**DEC**-декремент) и т. д. Этим командам требуется один или два входных операнда. Формируют команды один выходной операнд.

Арифметические команды могут быть разделены на несколько основных групп:

- команды операций с фиксированной запятой (сложение, вычитание, умножение, деление);
- команды операций с плавающей запятой (сложение, вычитание, умножение, деление);
- команды очистки;
- команды INC и DEC;
- команда сравнения.

Команды операций с фиксированной запятой работают с кодами в регистрах процессора или в памяти как с обычными двоичными кодами, причем все эти команды могут работать как с числами со знаком, так и с числами без знака.

Команды операций с плавающей запятой (точкой) используют формат представления чисел с порядком и мантиссой. В современных мощных процессорах набор команд с плавающей запятой не ограничивается только четырьмя арифметическими действиями, а содержит и множество других более сложных команд, например, вычисление тригонометрических функций, логарифмических функций, а также сложных функций, необходимых при обработке звука и изображения.

Команды очистки предназначены для записи нулевого кода в регистр или ячейку памяти. Эти команды могут быть заменены командами пересылки нулевого кода, но специальные команды очистки обычно выполняются быстрее, чем команды пересылки. Команды очистки иногда относят к группе логических команд, но суть их от этого не меняется.

Команды инкремента и декремента также бывают очень удобны. Их можно заменить командами суммирования с единицей или вычитания единицы, но инкремент и декремент выполняются быстрее, чем суммирование и вычитание. Эти команды требуют одного входного операнда, который одновременно

является и выходным операндом.

Команда сравнения предназначена для сравнения двух входных операндов. По сути, она вычисляет разность этих двух операндов, но выходного операнда не формирует, а всего лишь изменяет биты в регистре состояния процессора по результату этого вычитания. Следующая за командой сравнения команда (обычно это команда перехода) анализирует биты в регистре состояния процессора и выполняет действия в зависимости от их значений.

Логические команды. Логические команды выполняют над операндами логические (побитовые) операции, то есть они рассматривают коды операндов не как единое число, а как набор отдельных битов. Логические команды выполняют следующие основные операции:

- логическое И, логическое ИЛИ, сложение по модулю 2 (исключающее ИЛИ);
- логические, арифметические и циклические сдвиги;
- проверка битов и операндов;
- установка и очистка битов (флагов) регистра состояния процессора.

Команды логических операций позволяют побитно вычислять основные логические функции от двух входных операндов. Команды требуют двух входных операндов и формируют один выходной операнд.

Команды сдвига позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо).

Команды проверки битов и операндов предназначены для установки или очистки битов регистра состояния процессора в зависимости от значения выбранных битов или всего операнда в целом. Выходного операнда команды не формируют. Команда проверки операнда проверяет весь код операнда в целом на равенство нулю и на знак (на значение старшего бита), она требует только одного входного операнда. Команда проверки бита проверяет только отдельные биты, для выбора которых в качестве второго операнда используется код маски. В коде маски проверяемым битам основного операнда должны соответствовать единичные разряды.

Команды перехода. Команды перехода предназначены для изменения обычного порядка последовательного выполнения команд. С их помощью организуются переходы на подпрограммы и возвраты из них, всевозможные циклы, ветвления программ, пропуски фрагментов программ и т. д. Команды

перехода всегда меняют содержимое счетчика команд. Переходы могут быть условными и безусловными.

Некоторые команды перехода предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

Команды перехода с дальнейшим возвратом в точку, из которой был произведен переход, применяются для выполнения подпрограмм, то есть вспомогательных программ. Эти команды называются также командами вызова подпрограмм. Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.

Команды перехода позволяют строить сложные алгоритмы обработки информации.

Особое место среди команд перехода с возвратом занимают команды прерываний (Interrupt, INT). Эти команды в качестве входного операнда требуют номер прерывания.

У конкретных процессоров могут быть и многие другие (специальные) команды, не относящиеся к перечисленным группам команд.

Существует несколько *принципов управления потоком команд*:

- последовательное выполнение команд (фон-Неймановская архитектура);
- по мере готовности данных (параллельная архитектура);
- по мере потребности в результатах выполнения (редукционно-программное управление).

Основные виды *адресации*: прямая, косвенная, относительная. Использование сложных методов адресации позволяет существенно сократить количество команд в программе, но при этом значительно увеличивается сложность аппаратуры.

2.5. Основная память

Память – это тот компонент компьютера, в котором хранятся программы и данные. Основной единицей хранения данных в памяти является двоичный разряд, который называется битом. Бит – наименьшая единица памяти, может принимать значения 0 или 1.

Байт – единица измерения количества информации, считается равным восьми битам. Байт в современных компьютерах – это минимально адресуемая последовательность фиксированного числа битов.

Машинное слово – упорядоченный набор цифр, букв и других символов, хранящихся в памяти ЭВМ и воспринимаемых при обработке устройствами машины как единое слово, команда. Это машиннонезависимая и платформозависимая величина, измеряемая в битах или байтах, равная разрядности регистров процессора и/или разрядности шины данных. Количество основных символов в машинном слове называется длиной слова; обычно машинное слово содержит целое число байтов.

Адреса памяти. Память состоит из ячеек, каждая из которых может хранить некоторую порцию информации. Каждая ячейка имеет номер, который называется адресом. По этому адресу программы могут ссылаться на определенную ячейку. Если память содержит n ячеек, они будут иметь адреса от 0 до $n - 1$. Все ячейки памяти содержат одинаковое число бит. Если ячейка состоит из k бит, она может содержать любую из 2^k комбинаций. Ячейка – минимальная единица памяти, к которой можно обращаться.

Виды памяти. По доступным операциям с данными память в компьютере можно разделить на следующие категории:

- **ROM** (read-only memory) – память только для чтения;
- **RAM** (random-access memory) – память для чтения/записи.

По энергозависимости память можно разделить на следующие категории:

- Энергонезависимая память (nonvolatile storage) – память, реализованная запоминающими устройствами (ЗУ), записи в которых не стираются при снятии электропитания. Энергонезависимая память в основном применяется для хранения неизменяемой (или редко изменяемой) информации – системного программного обеспечения **BIOS** (Basic Input-Output System), памяти конфигурации устройств (**ESCD** - Extended System Configuration Data), (**EEPROM** - Electrically Erasable Programmable Read-Only Memory) адаптеров.
- Энергозависимая память (volatile storage) – память, реализованная ЗУ, записи в которых стираются при снятии электропитания. К этому типу памяти относятся память на оперативных запоминающих устройствах (**ОЗУ**), кэш-память.

Энергозависимую память также можно разделить на две категории:

- Статическая память **SRAM** (Static Random Access Memory) способна хранить информацию в статическом режиме, то есть сколь угодно долго при

отсутствии обращений (но при наличии питающего напряжения). Ячейки статической памяти реализуются на триггерах – элементах с двумя устойчивыми состояниями (0 или 1). По сравнению с динамической памятью эти ячейки более сложные и занимают больше места в кристалле, однако они проще в управлении и не требуют регенерации.

- Динамическая память **DRAM** (Dynamic Random Access Memory) получила свое название от принципа действия ее запоминающих ячеек, которые выполнены в виде конденсаторов, образованных элементами полупроводниковых микросхем. При записи логической единицы в ячейку конденсатор заряжается, при записи нуля – разряжается. Благодаря относительной простоте ячейки динамической памяти, на одном кристалле удастся размещать миллионы ячеек и получать дешевую полупроводниковую память достаточно высокого быстродействия с умеренным энергопотреблением, используемую в качестве основной памяти компьютера.

По назначению память можно разделить на следующие категории:

- Буферная память – память, предназначенная для временного хранения данных при обмене ими между различными устройствами или программами.
- Временная память (temporary memory) – память для хранения промежуточных результатов обработки.
- Кэш-память (cache memory) – часть архитектуры устройства или программного обеспечения, осуществляющая хранение часто используемых данных для предоставления их в более быстрый доступ.
- Разделяемая память или память коллективного доступа (shared memory или shared access memory) – память, доступная одновременно нескольким пользователям, процессам или процессорам.

Кэш-память. Основная идея кэш-памяти состоит в том, что в ней находятся машинные слова, которые используются чаще всего. Если процессору нужно какое-нибудь слово, сначала он обращается к кэш-памяти. Только в том случае, если слова там нет, он обращается к основной памяти (рис. 2.3). Если значительная часть слова находится в кэш-памяти, среднее время доступа значительно сокращается.

Ситуация, когда при последовательных обращениях к памяти в течение некоторого промежутка времени используется только небольшая ее область, называется *принципом локальности*. Этот принцип составляет основу всех систем кэш-памяти. Идея состоит в том, что когда определенное слово вызывается из памяти, оно вместе с соседними словами переносится в кэш-память, что позволяет при очередном запросе быстро обращаться к следующим словам.

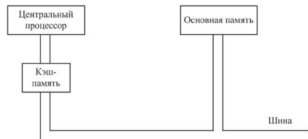


Рис. 2. 3. Кэш-память должна находиться между процессором и основной памятью

Если слово считывается или записывается k раз, компьютеру требуется сделать 1 обращение к медленной основной памяти и $k - 1$ обращений к быстрой кэш-памяти. Чем больше k , тем выше общая производительность.

Пусть c – время доступа к кэш-памяти, t – время доступа к основной памяти и h – коэффициент кэш-попаданий (hit ratio), который показывает соотношение числа обращений к кэш-памяти и общего числа всех обращений к памяти $h = \frac{k-1}{k}$. Некоторые авторы выделяют коэффициент кэш-промахов (miss ratio), равный $1 - h$

Таким образом, среднее время доступа можно вычислить по следующей формуле:

$$\text{среднее время доступа} = c + (1 - h) \cdot t.$$

Если $h \rightarrow 1$, то есть все обращения делаются только к кэш-памяти, то время доступа стремится к c . С другой стороны, если $h \rightarrow 0$, то есть каждый раз нужно обращаться к основной памяти, то время доступа стремится к $c + t$: сначала требуется время c для проверки кэш-памяти (в данном случае безуспешной), а затем – время t для обращения к основной памяти. В некоторых системах обращение к основной памяти может начинаться параллельно с исследованием кэш-памяти, чтобы в случае кэш-промаха цикл обращения к основной памяти уже начался. Однако эта стратегия требует способности останавливать процесс обращения к основной памяти в случае кэш-попадания, что усложняет разработку подобного компьютера.

Кэш-память очень важна для высокопроизводительных процессоров. При этом существует ряд проблем:

1. Объем кэш-памяти – чем больше объем, тем более эффективно она работает, но тем дороже стоит.
2. Размер строки кэша – кэш-память объемом 16 Кбайт можно разделить на 1024 строки по 16 байт, 2048 строк по 8 байт и т. д.

3. Проблема выбора слов, которые необходимо помещать в кэш-память.
4. Нахождение команд и данных в общей кэш-памяти. В объединенной кэш-памяти (unified cache) хранятся и данные и команды. В настоящее время существует тенденция к использованию разделенной кэш-памяти (split cache), когда команды хранятся в одной кэш-памяти, а данные – в другой (Гарвардская архитектура). Разделенная кэш-память позволяет осуществлять параллельный доступ, а общая – нет.
5. Количество блоков кэш-памяти – в настоящее время часто кэш-память первого уровня располагается прямо на микросхеме процессора, кэш-память второго уровня помещается не на самой микросхеме, а в корпусе процессора, кэш-память третьего уровня помещается еще дальше от процессора.

В зависимости от способа определения взаимного соответствия строки кэша и области основной памяти различают три архитектуры кэш-памяти: кэш прямого отображения (direct-mapped cache), полностью ассоциативный кэш (fully associative cache) и их комбинация - частично-ассоциативный или наборно-ассоциативный кэш (set-associative cache).

Код исправления ошибок. В памяти компьютера из-за всплесков напряжения и по другим причинам время от времени могут содержаться ошибки. Чтобы бороться с ошибками, используются специальные коды, которые умеют обнаруживать и исправлять ошибки. В этом случае к каждому слову в памяти особым образом добавляются дополнительные биты. Когда слово считывается из памяти, эти дополнительные биты проверяются, что и позволяет обнаруживать ошибки. Существуют различные алгоритмы коррекции ошибок, например Хэмминга, кодов свертки, Рида-Соломона и др [24].

Память для параллельных архитектур. Существуют различные варианты организации памяти ЭВМ с целью ускорения и обеспечения возможности параллельного доступа к ней, что особенно важно для многопроцессорных систем. Принята следующая классификация параллельных компьютеров по архитектуре подсистем оперативной памяти:

- системы с разделяемой памятью, у которых имеется одна большая виртуальная память и все процессоры имеют одинаковый доступ к данным и командам, хранящимся в этой памяти;
- системы с распределенной памятью, у которых каждый процессор имеет свою локальную оперативную память и к этой памяти у других процессоров нет доступа.

Физически память обычно делится на части, доступ к которым может быть



Рис. 2.4. Пятиуровневая организация памяти

организован независимо. Различие между разделяемой и распределенной памятью заключается в способе интерпретации адреса.

2.6. Дисковая память

Иерархическая структура памяти является традиционным решением проблемы хранения больших объемов данных. На самом верху иерархии находятся регистры процессора. Доступ к регистрам осуществляется быстрее всего. Дальше идет кэш-память, объем которой составляет от 32 Кбайт до нескольких мегабайтов. Далее следует основная память, которая в настоящее время может вмещать до десятков и сотен гигабайтов. Затем идут магнитные диски. После магнитных дисков следуют накопители на магнитной ленте и оптические диски, которые используются для хранения архивов.

По мере продвижения сверху вниз по иерархии (рис. 2.4) меняются три параметра:

- увеличивается время доступа;
- растет объем памяти;
- увеличивается количество байт устройства памяти, приобретаемое за 1 условную денежную единицу.

Для повышения эффективности организации дисковых накопителей, а также надежности и скорости доступа используются **RAID** массивы (Redundant Array of Independent Disks – избыточный массив независимых дисков).

Основная идея RAID состоит в том, что массив обычных дисков объединяется общим контроллером (RAID-контроллером), данные копируются в RAID-массив, а затем производятся обычные действия. Операционная система воспринимает RAID как один большой диск, при этом у RAID-массива выше производительность и надежность. Поскольку обычные диски обладают достаточной производительностью при довольно низкой цене и при этом один контроллер может управлять несколькими дисками (до семи дисков на 8-разрядных моделях **SCSI** (Small Computer System Interface) и до 15 на 16-разрядных), большинство RAID-устройств состоит из контроллера, предназначенного для управления RAID-массивом, и бокса дисков, которые операционная система воспринимает как один большой диск. При этом не требуется никаких изменений в программном обеспечении.

Возможность одновременной работы с несколькими дисками можно реализовать двумя способами: с использованием параллельного доступа (parallel-access array) и с использованием независимого доступа (independent-access array).

Для организации параллельного доступа рабочее пространство дисков размечается на зоны определенного размера (блоки) для размещения данных и избыточной информации. Информация, подлежащая записи на диски (запрос на обслуживание), разбивается на такие же по величине блоки, и каждый блок записывается на отдельный диск. При поступлении запроса на чтение, необходимая информация собирается из нескольких блоков. В этом случае скорость записи (и скорость чтения) увеличивается пропорционально количеству дисков, объединенных в RAID.

Для организации независимого доступа рабочее пространство дисков также размечается на зоны определенного размера (блоки). Однако, в отличие от случая параллельного доступа, каждый запрос на запись или чтение обслуживается только одним диском. В этом случае скорость записи будет не выше, чем при работе с одним диском. Однако массив с независимым доступом в каждый момент времени может обслуживать одновременно несколько запросов, при этом каждый диск обслуживает свой запрос.

Таким образом, оба архитектурных решения способствуют повышению производительности, но механизм повышения производительности у этих решений различен. Соответственно, свойства RAID существенно зависят от того, какой из этих двух механизмов в нем используется. Именно поэтому при

сравнении RAID различного уровня в первую очередь необходимо сравнивать размер логических блоков. Точнее говоря, не собственно размер, а соотношение размера блока и величины запроса на обслуживание (объем информации, подлежащей записи или считыванию).

Другим фактором, влияющим на производительность, является способ размещения избыточной информации. Избыточная информация может храниться на специально выделенном для этого диске и может распределяться по всем дискам. В RAID различного уровня применяются различные способы вычисления избыточной информации. Это также влияет на характеристики RAID. Основные способы: полное дублирование информации, применение кодов с коррекцией ошибок (применяется код с коррекцией одиночных ошибок и обнаружением двойных ошибок **ECC** (Error Checking and Correction) – код Хемминга) и вычисление четности (Parity).

Наиболее используемые уровни RAID в настоящее время:

- RAID 0 – дисковый массив из двух или более жёстких дисков с отсутствием избыточности. Информация разбивается на блоки данных и записывается на оба/несколько дисков одновременно. За счёт этого существенно повышается производительность (от количества дисков зависит кратность увеличения производительности). Однако страдает надёжность всего массива (при выходе из строя любого из входящих в RAID0 дисков полностью и безвозвратно пропадает вся информация).
- RAID 1 – зеркальные диски. Зеркальные диски представляют традиционный способ повышения надежности магнитных дисков. Все диски дублируются и при каждой записи информация записывается также и на проверочный диск. Возможно независимое параллельное считывание с основного и зеркального дисков.
- RAID 5 – четность вращения для распараллеливания записей. RAID 5 получил широкое распространение, в первую очередь, благодаря своей экономичности. Объем дискового массива рассчитывается по формуле $(n - 1) \cdot DiskSize$, где n – число дисков в массиве, а $DiskSize$ – размер одного диска. Контрольная информация распределяется между всеми дисками группы. На запись информации на том RAID 5 тратятся дополнительные ресурсы, так как требуются дополнительные вычисления, зато при чтении (по сравнению с отдельным винчестером) имеется выигрыш, потому что потоки данных с нескольких накопителей массива обрабатываются параллельно.
- RAID 6 (Advanced Data Guarding) – похож на RAID 5, но имеет более высокую степень надежности. Под контрольные суммы выделяется емкость

2-х дисков, рассчитываются 2 суммы по разным алгоритмам. Требуется более серьезный процессор контроллера, а значит более сложная и дорогая материнская плата. Обеспечивается работоспособность после выхода из строя одновременно 2-х дисков. Для организации массива требуется минимум 4 диска.

- Комбинированные уровни - помимо базовых уровней существуют комбинированные уровни RAID 1+0, RAID 0+1, RAID 5+0, RAID 1+5 и т. д. Различные производители по-своему интерпретируют каждый из комбинированных уровней.

2.7. Подсистемы ввода/вывода

В вычислительной системе, состоящей из множества подсистем, необходим механизм для их взаимодействия, обеспечивающий быстрый и эффективный обмен данными между подсистемами. Одним из простейших механизмов, позволяющих организовать взаимодействие различных подсистем, является центральная шина, к которой подсоединяются все подсистемы. Доступ к такой шине всеми подсистемами осуществляется раздельно. Подобная организация имеет два основных преимущества: низкая стоимость и универсальность. Поскольку такая шина является единственным местом подсоединения для разных устройств, новые устройства могут быть легко добавлены, и одни и те же периферийные устройства можно даже применять в разных вычислительных системах, использующих однотипную шину. Стоимость такой организации получается достаточно низкой, поскольку для реализации множества путей передачи информации используется единственный набор линий шины, разделяемый множеством устройств. Типичный пример шины **PCI** приведен ниже (рис. 2. 5).

Главным недостатком организации с единственной шиной является то, что шина создает узкое место, ограничивая, возможно, максимальную пропускную способность ввода/вывода. Если весь поток ввода/вывода должен проходить через центральную шину, такое ограничение пропускной способности весьма реально. В суперкомпьютерах, где необходимые скорости ввода/вывода очень высоки из-за высокой производительности процессоров, одним из главных вопросов разработки является создание системы нескольких шин, способной удовлетворить все запросы.

Одна из причин больших трудностей, возникающих при разработке шин, заключается в том, что максимальная скорость шины главным образом лимитируется физическими факторами: длиной шины и количеством

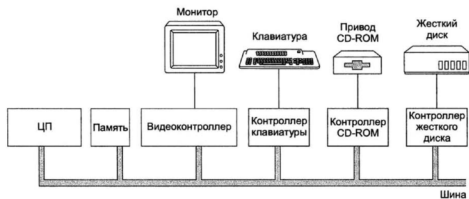


Рис. 2. 5. Архитектура компьютера с шиной PCI

подсоединяемых устройств (нагрузкой на шину). Эти физические ограничения не позволяют произвольно ускорять шины. Требования быстродействия (малой задержки) системы ввода/вывода и высокой пропускной способности являются противоречивыми. В современных крупных системах используется целый комплекс взаимосвязанных шин, каждая из которых обеспечивает упрощение взаимодействия различных подсистем, высокую пропускную способность, избыточность (для увеличения отказоустойчивости) и эффективность.

Существует несколько типов шин:

- для связи процессора с памятью – сравнительно короткие шины, обычно высокоскоростные, соответствуют организации системы памяти для обеспечения максимальной пропускной способности канала память-процессор;
- для ввода/вывода – могут иметь большую протяженность, поддерживать подключение многих типов устройств, и обычно следуют одному из шинных стандартов.

Главное устройство шины – это устройство, которое может инициировать транзакции чтения или записи. Центральный процессор (**ЦП**) всегда является главным устройством шины. Шина имеет несколько главных устройств, если имеется несколько ЦП или когда устройства ввода/вывода могут инициировать транзакции на шине. Если имеется несколько таких устройств, то требуется схема арбитража, чтобы решить, кто следующий захватит шину.

Шины можно разделить на два типа, отличающиеся способом коммутации: шины с коммутацией цепей (circuit-switched bus) (например, **Mbus**) и шины с коммутацией пакетов (packet-switched bus) (например: **xDBus**, **Rambus**),

получившие свои названия по аналогии со способами коммутации в сетях передачи данных.

Шина с коммутацией пакетов, при наличии нескольких главных устройств, обеспечивает значительно большую пропускную способность по сравнению с шиной с коммутацией цепей за счет разделения транзакции на две логические части: запроса шины и ответа. Такая методика получила название “расщепления” транзакций. Транзакция чтения разбивается на транзакцию запроса чтения, которая содержит адрес, и транзакцию ответа памяти, которая содержит данные.

Шина с коммутацией цепей не делает расщепления транзакций, любая транзакция на ней есть неделимая операция. Главное устройство запрашивает шину, далее помещает на нее адрес и блокирует шину до окончания обслуживания запроса. Большая часть времени при этом тратится не на выполнение операций на шине, а на ее обслуживание. Расщепленные транзакции делают шину доступной для других главных устройств пока память читает слово по запрошенному адресу. Это также означает, что ЦП должен бороться за шину для отправки данных, а память должна бороться за шину, чтобы вернуть данные. Таким образом, шина с расщеплением транзакций имеет более высокую пропускную способность, но обычно она имеет и большую задержку, чем шина, которая захватывается на все время выполнения транзакции. Транзакция называется расщепленной, поскольку произвольное количество других пакетов или транзакций могут использовать шину между запросом и ответом.

Шина может быть или синхронной, или асинхронной. Синхронная шина включает сигналы синхронизации, которые передаются по линиям управления шины, и фиксированный протокол, определяющий расположение сигналов адреса и данных относительно сигналов синхронизации. Поскольку практически никакой дополнительной логики не требуется для того, чтобы решить, что делать в следующий момент времени, эти шины могут быть и быстрыми, и дешевыми. Обычно шины процессор-память синхронные.

Недостаток:

- все действия на шине должны происходить с одной и той же частотой синхронизации, поэтому из-за проблемы перекоса синхросигналов, синхронные шины не могут быть длинными.

Асинхронная шина не тактируется. Вместо этого обычно используется старт-стопный режим передачи и протокол “рукопожатия” (handshaking) между источником и приемником данных на шине. Эта схема позволяет гораздо проще приспособить широкое разнообразие устройств и удлинить шину без

беспокойства о перекосе сигналов синхронизации и о системе синхронизации. Шины ввода/вывода обычно асинхронные.

Главный недостаток: построить асинхронную шину гораздо дороже, чем синхронную.

Выбор типа шины (синхронной или асинхронной) определяет не только пропускную способность, но также непосредственно влияет на емкость системы ввода/вывода в терминах физического расстояния и количества устройств, которые могут быть подсоединены к шине. Асинхронные шины по мере изменения технологии лучше масштабируются.

Вопросы для самоконтроля

1. Многоплановое толкование понятия “архитектура”.
2. В чем принципы фон-Неймановской архитектуры ЭВМ?
3. Принципы Гарвардской архитектуры ЭВМ. В чем отличие от фон-Неймановской?
4. Стековая архитектура. В чем отличие от фон-Неймановской?
5. Преимущества и недостатки архитектура системы команд CISC.
6. Преимущества и недостатки архитектура системы команд RISC.
7. Преимущества и недостатки архитектура системы команд MISC.
8. Системы команд CISC, RISC и MISC. В чем различие?
9. Понятие о матричных процессорах.
10. Принципы конвейерной обработки. Выигрыш от конвейерной обработки и основные проблемы эффективного заполнения конвейера.
11. Параллельное выполнение нескольких команд процессором. Суперскалярный процессор. VLIW машины.
12. Системы команд процессора. Типы команд.
13. Принципы управления потоком команд. Методы адресации.
14. Дайте определение понятия “память”.
15. Какие виды памяти бывают?
16. В чем основная идея “кэш памяти”?

17. Дисковая память. Варианты эффективной организации дисковых накопителей с целью повышения надежности и скорости доступа.
18. В чем различие уровней RAID?
19. В чем разница шины с коммутацией цепей и шины с коммутацией пакетов?
20. Укажите недостатки синхронной и асинхронной шин.

Глава 3. Архитектура ВВС

3.1. Классификация ВВС

Наиболее перспективным и динамично развивающимся направлением увеличения производительности вычислительных систем, а соответственно и скорости решения прикладных задач, является применение идей параллелизма к работе вычислительных систем. К настоящему времени спроектированы и опробованы сотни различных компьютеров, использующих в своей архитектуре тот или иной вид параллельной обработки данных.

Попытки систематизировать все множество архитектур начались после опубликования М. Флинном первого варианта классификации вычислительных систем в 1966 году.

Необходимость классификации архитектур параллельных вычислительных систем вызвана тем, что необходимо находить ответы на вопросы типа: “Почему параллельных архитектур так много?”, “Какие основные характеристики каждой архитектуры?”, “Что необходимо учитывать для написания действительно эффективных программ для определенной архитектуры?”, “На какой класс архитектур следует ориентироваться для решения требуемого класса задач?”, “Какова взаимосвязь различных архитектур?” и т. д. Основной вопрос – что заложить в основу классификации – может решаться по-разному, в зависимости от того, для кого данная классификация создается и на решение какой задачи направлена. Множество компьютеров и комплектовующих породили множество классификаций и интерес к данной проблеме в целом.

Классификация М. Флинна. Наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М. Флинном [45]. В основе классификации лежат понятия потоков команд и потоков данных. Поток команд соответствует счетчику команд. Система с n процессорами имеет n счетчиков команд и, следовательно, n потоков команд. Поток данных состоит из набора операндов. Потоки команд и данных в какой-то степени независимы, поэтому существует 4 комбинации таких потоков: SISD, MISD, SIMD, MIMD.

SISD (single instruction stream / single data stream) – одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины (машины фон-Неймановского типа). Компьютер фон Неймана имеет один поток команд и один поток данных, все команды обрабатываются последовательно друг за другом и в каждый момент времени может выполнять только одно действие. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка – так машина CDC 6600 со скалярными функциональными устройствами, как и CDC 7600 с конвейерными, попадают в этот класс.

SIMD (single instruction stream / multiple data stream) – одиночный поток команд и множественный поток данных. В архитектурах подобного рода имеется один блок управления, выдающий по одной команде, но при этом есть несколько АЛУ, которые могут обрабатывать несколько наборов данных одновременно. В данных системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными. Это позволяет выполнять одну арифметическую операцию сразу над многими данными, которые являются элементами вектора. Способ выполнения векторных операций не уточняется, поэтому обработка элементов вектора может производиться либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине CRAY-1 (для векторно-конвейерных машин каждый элемент вектора рассматривается как отдельный элемент потока данных). Несмотря на то, что SIMD-машины не относятся к числу широко распространенных, в некоторых обычных компьютерах для обработки мультимедийных данных используются SIMD-команды (например SSE-команды (Streaming SIMD Extensions – потоковое SIMD-расширение процессора) в процессорах Pentium относятся к категории SIMD-команд). Наиболее известная область применения вычислительных систем класса SIMD – это потоковые процессоры, которые специально разработаны для обработки мультимедийных данных.

MISD (multiple instruction stream / single data stream) – множественный поток команд и одиночный поток данных. Этот класс предполагает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных, т. е. несколько команд оперируют одним и тем же набором данных. Примером данного класса могут служить систолические массивы (массивы с непосредственными соединениями между близлежащими элементами). Такие массивы

очень эффективны, но каждый из них ориентирован на решение весьма узкого класса задач [35].

MIMD (multiple instruction stream / multiple data stream) – множественный поток команд и множественный поток данных системы. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных. В эту категорию попадают большинство параллельных процессоров. И мультипроцессоры, и мультикомпьютеры – это MIMD-машины.

Предложенная схема классификации вплоть до настоящего времени является самой применяемой при начальной характеристике того или иного компьютера. Однако у данной схемы есть недостатки:

- некоторые архитектуры четко не вписываются в данную классификацию (например, векторно-конвейерные машины);
- сильная заполненность класса MIMD.

Необходимо более подробно (точно) систематизировать архитектуры, попадающие в один класс, но совершенно различны по количеству процессоров, взаимодействию между ними, по способу организации памяти, по технологии программирования.

Дополнения К. Ванга и Ф. Бриггса к классификации М. Флинна. В работе К. Ванга и Ф. Бриггса [50] сделаны некоторые дополнения к классификации Флинна. Оставляя четыре ранее введенных базовых класса (SISD, SIMD, MISD, MIMD), авторы внесли некоторые изменения.

Класс SISD разбивается на два подкласса:

- архитектуры с единственным функциональным устройством;
- архитектуры, имеющие в своем составе несколько функциональных устройств.

В класс SIMD вводится два подкласса:

- архитектуры с пословно-последовательной обработкой;
- архитектуры с разрядно-последовательной обработкой.

В классе MIMD различаются:

- вычислительные системы со слабой связью между процессорами (системы с распределенной памятью);
- вычислительные системы с сильной связью (системы с общей памятью).

Классификация Фенга. В 1972 году Т. Фенг [44] предложил классифицировать вычислительные системы на основе двух простых характеристик:

- число бит n в машинном слове, обрабатываемых параллельно при выполнении машинных инструкций;
- число слов m , обрабатываемых одновременно данной вычислительной системой (ширина битового слоя).

Функционирование любого компьютера можно представить как параллельную обработку n битовых слоев, на каждом из которых независимо преобразуются m бит.

Если рассмотреть предельные верхние значения данных характеристик, то каждую вычислительную систему C можно описать парой чисел (n, m) и представить точкой на плоскости в системе координат длина слова - ширина битового слоя. Площадь прямоугольника со сторонами n и m определяет интегральную характеристику потенциала параллельности P архитектуры и носит название *максимальной степени параллелизма* вычислительной системы: $P(C) = nm$. Данное значение есть ничто иное, как пиковая производительность, выраженная в других единицах.

На основе введенных понятий все вычислительные системы в зависимости от способа обработки информации, заложенного в их архитектуру, можно разделить на четыре класса.

1. Разрядно-последовательные пословно-последовательные ($n = m = 1$). В каждый момент времени такие компьютеры обрабатывают только один двоичный разряд.
2. Разрядно-параллельные пословно-последовательные ($n > 1, m = 1$). Большинство классических последовательных компьютеров, так же как и многие вычислительные системы, эксплуатируемые до сих пор, принадлежит к данному классу.
3. Разрядно-последовательные пословно-параллельные ($n = 1, m > 1$). Как правило вычислительные системы данного класса состоят из большого числа одnorазрядных процессорных элементов, каждый из которых может независимо от остальных обрабатывать свои данные.
4. Разрядно-параллельные пословно-параллельные ($n > 1, m > 1$). Большая часть существующих параллельных вычислительных систем, обрабатывая одновременно $m \cdot n$ двоичных разрядов, принадлежит именно к этому классу.

Достоинства:

- введение единой числовой метрики для всех типов компьютеров, которая вместе с описанием потенциала вычислительных возможностей конкретной архитектуры позволяет сравнить любые два компьютера между собой.

Недостатки:

- нет никакого различия между процессорными матрицами, векторно-конвейерными и многопроцессорными системами;
- не делается акцент на том, за счет чего компьютер может одновременно обрабатывать более одного слова: множественности функциональных устройств, их конвейерности или же какого-то числа независимых процессоров.

Классификация Базу. Классификация строится по последовательности решений, принятых на этапе проектирования архитектуры [39].

А. Базу считает, что любую параллельную вычислительную систему можно однозначно описать последовательностью решений, принятых на этапе ее проектирования (*рис. 3. 1*), а сам процесс проектирования представить в виде дерева:

- корень дерева – это вычислительная система;
- последующие ярусы дерева – фиксируют уровень параллелизма, метод реализации алгоритма, параллелизм инструкций и способ управления.

Формирование описания системы основано на последовательном дополнении корня дерева ярусами.

Классификация Р. Хокни. Р. Хокни [48] уточнил классификацию компьютеров, попадающих в класс MIMD по систематике М. Флинна. В основе классификации лежит идея о том, что множественный поток команд может быть обработан двумя способами: либо конвейерным устройством обработки (в режиме разделения времени для отдельных потоков), либо каждый поток обрабатывается своим собственным устройством.

Архитектуры, в которых каждый поток обрабатывается своим собственным устройством далее делятся на два класса:

1. Возможна прямая связь каждого процессора с каждым, реализуемая с помощью переключателя.
2. Связь каждого процессора возможна только с ближайшими соседями по сети, а взаимодействие удаленных процессоров поддерживается специальной системой маршрутизации.

Машины с переключателем подразделяются на два класса по типу памяти:

1. С распределенной памятью между процессорами.
2. С общей памятью для всех процессоров.

Системы, имеющие как общую разделяемую память, так и распределенную локальную память, по классификации Хокни считаются гибридными MIMD с переключателем.

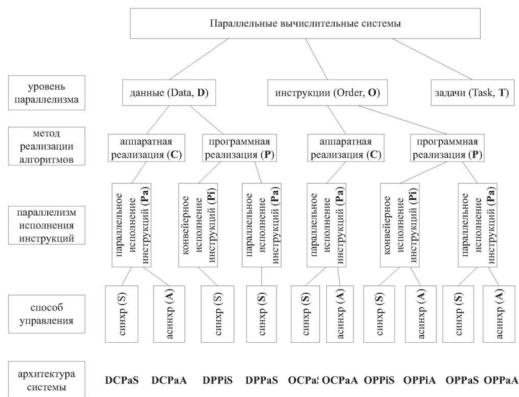


Рис. 3. 1. Классификация Базу

MIMD-машины с сетевой архитектурой считаются имеющими распределенную память, а дальнейшая классификация проводится в соответствии с топологией сети: звездообразная, гиперкубы, сети с иерархической структурой, сети с изменяющейся конфигурацией и т. д.

Классификация В. Хендлера. В основу классификации Хендлера [46] заложено явное описание возможностей параллельной и конвейерной обработки информации вычислительной системы. Предложенная классификация базируется на различиях между тремя уровнями обработки данных в процессе выполнения программ:

- уровень выполнения программы; опираясь на счетчик команд и некоторые другие регистры, устройство управления производит выборку и дешифрацию команд программы;
- уровень выполнения программы; арифметико-логическое устройство компьютера исполняет команду, выданную ему устройством управления;
- уровень битовой обработки; все элементарные логические схемы

процессора (ЭЛС) разбиваются на группы, необходимые для выполнения операций над двоичным разрядом.

Подобная схема выделения уровней предполагает, что вычислительная система содержит какое-то число процессоров, каждый со своим устройством управления. Каждое устройство управления связано с несколькими АЛУ, исполняющими одну и ту же операцию в каждый конкретный момент времени, и каждое АЛУ объединяет несколько групп элементарных логических схем, ассоциированных с обработкой одного двоичного разряда.

Классификация Л. Шнайдера. В 1988 г. Шнайдер [56] предложил классификацию, основанную на разделении потоков на адреса и их содержимое, что позволило описать “неудобные” для классификации архитектуры такие, как компьютеры с длинным командным словом и др.

Классификация Д. Скилликорна. В 1989 г. Скилликорн [55] разработал подход, в котором архитектура компьютера рассматривается как абстрактная структура, состоящая из 4-х компонент:

- процессор команд (Instruction Processor – **IP**) – функциональное устройство, работающее как интерпретатор команд;
- процессор данных (Data Processor – **DP**) – функциональное устройство, работающее как преобразователь данных в соответствии с арифметическими операциями;
- иерархия данных (Instruction Memory – **IM**, Data Memory – **DM**) – запоминающее устройство, в котором хранятся данные и команды, пересылаемые между процессорами;
- переключатель – абстрактное устройство, обеспечивающее связь между процессорами и памятью.

Функции процессора команд во многом схожи с функциями устройств управления последовательных машин. Функции процессора данных делают его во многом похожим на арифметическое устройство традиционных процессоров.

DP получает от IP команду, которую надо выполнить; получает от IP адреса операндов; выбирает операнды из DM; выполняет команду; запоминает результата в DM; возвращает в IP информацию о состоянии после выполнения команды.

Классификация Скилликорна строится на основе следующих характеристик:

- количество процессоров команд IP;
- число запоминающих устройств (модулей памяти) команд IM;

- тип переключателя между IP и IM;
- количество процессоров данных DP;
- число запоминающих устройств (модулей памяти) данных DM;
- тип переключателя между DP и DM;
- тип переключателя между IP и DP;
- тип переключателя между DP и DP.

Используя введенные характеристики и предполагая, что рассмотрение количественных характеристик можно ограничить только тремя возможными вариантами значений 0, 1 и n ($n > 1$), можно получить различные классы архитектур.

“Хорошая классификация должна показать, за счет каких структурных особенностей достигается увеличение производительности различных вычислительных систем” (Д. Скилликорн).

3.2. Архитектуры BBC

Когда два процессора или обрабатывающих элемента находятся рядом и обмениваются большими объемами данных с небольшими задержками, они называются сильно связанными (*tightly coupled*). Соответственно, когда два процессора или обрабатывающих элемента располагаются далеко друг от друга и обмениваются небольшими объемами данных с большими задержками, они называются слабо связанными (*loosely coupled*). При построении BBC важным параметром является степень связности (*рис. 3. 2*).

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. Характерным представителем данного направления является семейство векторно-конвейерных компьютеров CRAY.

Как правило, несколько специальных векторно-конвейерных процессоров работают одновременно над общей памятью в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора.

Оперативная память компьютера разделяется всеми процессорами и секцией ввода/вывода. Для увеличения скорости выборки данных память разделена на множество банков, которые могут работать одновременно. Каждый процессор имеет доступ к оперативной памяти через несколько портов, причем один

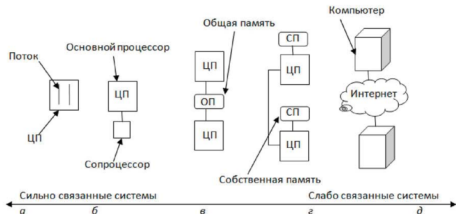


Рис. 3. 2. Внутрипроцессорный параллелизм (а); сопроцессор (б); мультипроцессор (в); мультикомпьютер (г); слабо связанная распределенная вычислительная система (метакомпьютер, Grid-система) (д)

из портов всегда связан с секций ввода/вывода и, по крайней мере, один из портов всегда выделен под операцию записи.

Межпроцессорное взаимодействие содержит разделяемые регистры и семафоры, предназначенные для передачи данных и управляющей информации между процессорами.

Функциональные устройства (ФУ) исполняют свой набор команд и могут работать одновременно друг с другом. Все ФУ конвейерные и делятся на четыре группы: адресные, скалярные, векторные и для работы с плавающей точкой. ФУ имеют различное число ступеней конвейера, но каждая ступень срабатывает за один такт, поэтому при полной загрузке все ФУ могут выдавать результат каждый такт.

Команды выбираются из оперативной памяти блоками и заносятся в буфер команд, откуда они затем выбираются для исполнения. Если необходимой для исполнения команды нет в буфере команд, то происходит выборка очередного блока. Команды имеют различный формат и могут занимать от одного до нескольких пакетов.

Все основные операции, выполняемые процессором: обращения в память, обработка команд и выполнение инструкций являются конвейерными.

Векторная обработка увеличивает скорость и эффективность обработки за счет того, что обработка целого набора (вектора) данных выполняется одной командой. Скорость выполнения операций в векторном режиме приблизительно в несколько раз выше скорости скалярной обработки. Перед тем,

как векторная операция начнет выдавать результаты, проходит некоторое время, связанное с заполнением конвейера и подкачкой аргументов. Чем больше длина векторов, тем менее заметным оказывается влияние данного начального промежутка времени на все время выполнения программы. Векторные операции, использующие различные ФУ и регистры, могут выполняться параллельно.

Зацепление функциональных устройств. Архитектура некоторых вычислительных систем позволяет использовать регистр результатов векторной операции в качестве входного регистра для последующей векторной операции, т. е. выход сразу подается на вход. Это называется зацеплением векторных операций, при этом глубина зацепления может быть любой, например, чтение векторов, выполнение операции сложения, выполнение операции умножения, запись векторов.

Современный уровень развития микроэлектронных технологий не позволяет производить однокристалльные векторные процессоры, поэтому эти компьютеры довольно громоздки и чрезвычайно дороги. В связи с этим, начиная с середины 90-х годов XX 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.

Тем не менее в марте 2002 г. корпорация NEC представила систему Earth Simulator из 5120 векторно-конвейерных процессоров. Система Earth Simulator в 2003 году занимала первое место в списке Top500 самых мощных компьютеров мира, показав производительность 35,86 Тфлопс на тесте Linpack.

Суперкомпьютеры с архитектурой симметричных многопроцессорных систем (SMP). Узел симметричной многопроцессорной системы (SMP – *Symmetric Multi-Processing*) содержит два или более одинаковых процессора, используемых равноправно. Все процессоры имеют одинаковый доступ к вычислительным ресурсам узла. Поскольку процессоры одновременно работают с данными, хранящимися в единой памяти узла, в SMP-архитектурах обязательно должен быть механизм, поддержки когерентности данных. Когерентность данных означает, что в любой момент времени для каждого элемента данных во всей памяти узла существует только одно его значение несмотря на то, что одновременно могут существовать несколько копий элементов данных, расположенных в разных видах памяти и обрабатываемых разными процессорами. Механизм когерентности должен следить за тем, чтобы операции с одним и тем же элементом данных выполнялись на разных процессорах последовательно, удаляя, в частности, устаревшие копии. В современных

SMP-архитектурах когерентность реализуется аппаратными средствами. Механизм когерентности является критичным для эффективной параллельной работы узла SMP и должен иметь малое время задержки, а это физически ограничивает возможное число подсоединенных процессоров и плат памяти. Поэтому для дальнейшего увеличения числа процессоров в узле приходится вместо аппаратно реализованной техники когерентности применять более медленную программную реализацию, что очень существенно сказывается на программируемости систем и их производительности.

SMP-узлы очень удобны для разработчиков приложений: операционная система почти автоматически масштабирует приложения, давая им возможность использовать наращиваемые ресурсы. Само приложение не должно меняться при добавлении процессоров и не обязано следить за тем, на каких ЦП оно работает. Временная задержка доступа от любого ЦП до всех частей памяти и системы ввода-вывода одна и та же. Разработчик оперирует с однородным адресным пространством. Все это приводит к тому, что SMP-архитектуры разных производителей выглядят в основном одинаково: упрощается переносимость программного обеспечения между SMP-системами. Переносимость программ - одно из основных достоинств SMP-платформ.

SMP-система состоит из нескольких однородных процессоров и массива общей памяти. Один из часто используемых в SMP-архитектурах подходов для формирования масштабируемой, общедоступной системы памяти, состоит в однородной организации доступа к памяти посредством организации масштабируемого канала память-процессоры.

Типичные SMP-архитектуры в качестве аппаратной реализации механизма поддержки когерентности используют шину слежения. Каждый процессор имеет свой собственный локальный кэш, где он хранит копию небольшой части основной памяти, доступ к которой наиболее вероятен. Для того чтобы все кэши оставались когерентными, каждый процессор "подглядывает" за шиной, осуществляя поиск тех операций считывания и записи между другими процессорами и основной памятью, которые влияют на содержимое их собственных кэшей. Если процессор "B" запрашивает ту часть памяти, которая обрабатывается процессором "A" то процессор "A" перехватывает этот запрос и помещает свои значения области памяти на шину, где процессор "B" их считывает. Когда процессор "A" записывает измененное значение обратно из своего кэша в память, то все другие процессоры видят, как эта запись проходит по шине и удаляют устаревшие значения из своих кэшей.

SMP система состоит из нескольких однородных процессоров и массива общей памяти. Один из часто используемых в SMP архитектурах подходов для

формирования масштабируемой, общедоступной системы памяти, состоит в однородной организации доступа к памяти посредством организации масштабируемого канала память-процессоры.

Каждая операция доступа к памяти интерпретируется как транзакция по шине процессоры - память. Когерентность кэшей поддерживается аппаратными средствами.

Недостатком данной архитектуры является необходимость организации канала процессоры - память с очень высокой пропускной способностью.

Системы данного типа серийно производятся многими компьютерными фирмами как многопроцессорные серверы с числом процессоров до 128 и прочно удерживают лидерство в классе малых суперкомпьютеров. Типичными представителями данного класса суперкомпьютеров являются компьютеры SGI Origin 3000, HP Superdome.

Суперкомпьютеры массивно-параллельной архитектуры (Massively Parallel Processing, MPP). Узлы в архитектуре MPP обычно состоят из одного ЦПУ, небольшой памяти и нескольких устройств ввода-вывода. В каждом узле работает своя копия операционной системы, а узлы объединяются между собой специализированным соединением. Взаимосвязи между узлами (и между копиями ОС, принадлежащими каждому узлу) не требуют аппаратно поддерживаемой когерентности, так как каждый узел имеет собственную ОС и, следовательно, свое уникальное адресное пространство физической памяти. Когерентность реализуется программными средствами, с использованием техники передачи сообщений.

Задержки, которые присущи программной поддержке когерентности на основе сообщений, обычно в сотни и тысячи раз больше, чем те, которые получаются в системах с аппаратными средствами. С другой стороны, их реализация значительно менее дорогая. В некотором смысле в MPP-узлах задержкой приходится жертвовать, чтобы подсоединить большее число процессоров - сотни и даже тысячи узлов.

MPP-архитектуры привлекательны в первую очередь для разработчиков аппаратных средств, так как в этом случае возникает меньше проблем и ниже стоимость аппаратуры. Из-за того, что нет аппаратной поддержки ни для разделенной памяти, ни для когерентности кэшей, подсоединить большое число процессоров очень просто. Такие системы обеспечивают высокий уровень производительности для приложений с большой интенсивностью вычислений, со статистически разделяемыми данными и с минимальным обменом данными между узлами. Для большинства коммерческих приложений MPP-системы подходят плохо из-за того, что структура базы данных меняется со временем

и слишком велики затраты на перераспределение данных [6].

Ключевым различием между одиночным SMP-узлом и MPP-системой является то, что внутри SMP-узла когерентность данных поддерживается исключительно аппаратными средствами. Это действительно быстро, но и дорого. В MPP-системе с таким же числом процессоров когерентность между узлами реализуется программными средствами. Поэтому происходит это более медленно, однако и цена значительно ниже.

Достоинства:

- если нужна высокая производительность, то можно добавить еще процессоров;
- если ограничены финансы или заранее известна требуемая вычислительная мощность, то легко подобрать оптимальную конфигурацию;
- отказоустойчивость – в современных MPP-машинах имеется специализированная аппаратная и программная поддержка постоянного мониторинга системы, обнаружения неполадок и их исправления.

Недостатки:

- межпроцессорное взаимодействие в компьютерах этого класса идет намного медленнее, чем происходит локальная обработка данных самими процессорами;
- написать эффективную программу для таких компьютеров очень сложно, а для некоторых алгоритмов иногда просто невозможно.

Типичными представителями данного класса суперкомпьютеров являются системы IBM BlueGene.

Системы с неоднородным доступом к памяти (NUMA). NUMA-архитектуры (*NonUniform Memory Access*) представляют собой нечто среднее между SMP и MPP. Системы с неоднородным доступом к памяти состоят из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Память физически распределена, но логически общедоступна.

NUMA-машины имеют три ключевые характеристики, которые в совокупности отличают их от других мультипроцессоров:

- существует единое адресное пространство, видимое всеми процессорами;
- аппаратно поддерживается доступ к удаленной памяти;
- доступ к удаленной памяти выполняется медленнее, чем доступ к локальной.

Если время доступа к удаленной памяти не замаскировано кэшированием

(кэш отсутствует), такая система называется **NC-NUMA** (*No Caching NUMA* - NUMA без кэширования). Если присутствуют согласованные кэши, то система называется **CC-NUMA** (*Coherent Cache NUMA* - NUMA с согласованными кэшами). Такую систему часто называют аппаратной распределенной общей памятью, поскольку она, по сути, аналогична распределенной общей памяти, реализованной программно, однако поддерживается аппаратно с использованием страниц маленького размера.

В качестве примера NUMA-машин с общей памятью можно отметить семейство мультипроцессоров Sun Fire компании Sun Microsystems.

Типичным представителем данного класса суперкомпьютеров является компьютер SUN StarFire 15K.

Параллельно-векторные системы (PVP). При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь, могут объединяться в системы с использованием общей или распределенной памяти.

Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды одностипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах. Как правило, несколько таких процессоров работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).

Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Типичным представителем данного класса суперкомпьютеров является компьютер NEC SX-6.

Кластерные системы. Изначально кластеризация использовалась для обеспечения высокой готовности вычислительных систем, и сам термин появился в 1983 г. Компания DEC первой анонсировала концепцию кластерной системы в 1983 году, определив ее как группу объединенных между собой вычислительных машин, представляющих собой единый узел обработки информации. По существу кластер представляет собой слабосвязанную многомашинную систему с общей внешней памятью, обеспечивающую единый механизм управления и администрирования.

Определение кластера, сформулированное Aberdeen Group (лидер

исследовательской и маркетинговой деятельности, работающий на рынке информационных технологий), выглядит следующим образом. Кластер – это многомашинный комплекс, который:

- выглядит с точки зрения пользователя как единая система;
- обеспечивает высокую надежность (готовность);
- имеет общую файловую систему;
- обладает свойством роста производительности при добавлении ресурсов (масштабируемостью);
- гибко перестраивается;
- управляется/администрируется как единая система.

В настоящее время под вычислительным кластером понимается совокупность компьютеров, объединенных в рамках некоторой сети для решения одной задачи. В качестве вычислительных узлов обычно используются доступные на рынке однопроцессорные компьютеры, двух- или четырехпроцессорные SMP-серверы. Каждый узел работает под управлением своей копии операционной системы. Состав и мощность узлов может меняться даже в рамках одного кластера, давая возможность создавать неоднородные системы. Выбор конкретной коммуникационной среды определяется многими факторами: особенностями класса решаемых задач, доступным финансированием, необходимостью последующего расширения кластера и т. п. Возможно включение в конфигурацию специализированных компьютеров, например, файл-сервера, и, как правило, предоставлена возможность удаленного доступа на кластер через Internet. Набор рабочих станций или персональных компьютеров общего назначения используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet) на базе шинной архитектуры или коммутатора. В отличие от отказоустойчивых систем (fault-tolerant systems), систем, построенных на основе архитектуры SMP или MPP, каждый компьютер в кластере остается относительно независимым, т. е. его можно остановить и выключить (например, для проведения профилактических работ или установки дополнительного оборудования), не нарушая работоспособность кластера в целом. В отличие от кластеров, в MPP-системах узлы не делят ресурсы для хранения. Это главное отличие между кластерными SMP-системами и традиционными MPP-системами.

Существует два подхода к созданию кластерных систем:

- объединение в единую систему полнофункциональных компьютеров, которые могут работать, в том числе, и как самостоятельные единицы, например, компьютеры учебного класса или рабочие станции лаборатории;

- целенаправленное создание мощного вычислительного ресурса, в котором роль вычислительных узлов играют промышленно выпускаемые компьютеры, и тогда нет необходимости снабжать такие компьютеры графическими картами, мониторами, дисковыми накопителями и другим периферийным оборудованием, что значительно удешевляет стоимость системы.

Производительность персональных компьютеров на базе процессоров x86 от Intel и AMD в последние годы также значительно выросла. В настоящее время создаются параллельные вычислительные системы (кластеры) из общедоступных компьютеров на базе обычных процессоров x86 и недорогих Ethernet-сетей, устанавливая на эти компьютеры ОС (например, Linux) и одну из бесплатно распространяемых коммуникационных библиотек (например, PVM, MPI) [1, 2, 24]. На многих классах задач и при достаточном числе узлов такие системы дают производительность, сравнимую с суперкомпьютерной.

Примером данного типа систем является проект Beowulf, который начался в научно-космическом центре NASA - Goddard Space Flight Center (GSFC) в 1994 году. Этот кластер состоял из 16-процессоров (486DX4/100MHz), 16MB памяти и 3 сетевых адаптеров на каждом узле, 3 "параллельных" Ethernet-кабеля по 10Mbit. Данный кластер, создавался как вычислительный ресурс проекта Earth and Space Sciences Project (ESS).

Метакомпьютинг. Сеть Интернет с ресурсами подключенных к ней компьютеров можно рассматривать как самый большой компьютер в мире и по пиковой производительности, и по объему оперативной и дисковой памяти. *Метакомпьютер* – это компьютер, состоящий из множества компьютеров, объединенных некоторой сетевой инфраструктурой. Данный термин возник вместе с развитием высокоскоростной сетевой инфраструктуры в начале 90-х годов XX века и относился к объединению нескольких разнородных вычислительных ресурсов в локальной сети организации для решения одной задачи. Основная цель построения метакомпьютера в то время заключалась в оптимальном распределении частей работы по вычислительным системам различной архитектуры и различной мощности. Некоторые системы были универсальны, некоторые ориентировались на конкретную задачу, использовались различные способы взаимодействия: и специальные протоколы передачи данных, и обычные, например HTTP. Компонентами метакомпьютера могут быть как простейшие ПК, так и мощные массивно-параллельные системы.

Важно, что метакомпьютер может не иметь постоянной конфигурации – отдельные компоненты могут включаться в его конфигурацию или отключаться от нее. При этом технологии метакомпьютинга обеспечивают

непрерывное функционирование системы в целом. Пользователь лишь выдает задание на решение определенной задачи, а поиск доступных вычислительных ресурсов, мониторинг их работоспособности, передачу данных выполняет метакомпьютер.

Метакомпьютер характеризуется рядом свойств:

- значительность ресурсов (количество процессоров, объем памяти и т. д.);
- распределенность (компоненты могут находиться на значительном расстоянии друг от друга);
- динамичность (постоянное подключение и отключение ресурсов, непредсказуемость загруженности ресурсов);
- неоднородность (различные операционные системы, глобальные и локальные каналы связи, классы компьютеров, начиная с персональных компьютеров и заканчивая суперкомпьютерами).

Основными вопросами при создании метакомпьютера являются распределение и диспетчеризация заданий, средства и модели программирования, технология доступа к метакомпьютеру и интерфейс пользователя, мониторинг работоспособности узлов, безопасность и т. д.

Современные исследовательские проекты в этой области направлены на обеспечение прозрачного доступа пользователей через Интернет к необходимым распределенным вычислительным ресурсам, а также прозрачного подключения простаивающих вычислительных систем к метакомпьютерам. Представителем данного типа систем является проект www.distributed.net, который занимается показательными “взломами” широко распространенных шифров с целью показать степень их надежности. Для расшифровки привлекаются ресурсы машин по всему миру.

3.3. Факторы, снижающие производительность параллельных компьютеров

Закон Амдала. *Время работы программы определяется ее самой медленной частью.* Пусть доля операций в программе, которые нужно выполнять последовательно, равна f , где $0 \leq f \leq 1$ (при этом доля понимается по числу операций в процессе выполнения). Крайние случаи в значениях f соответствуют полностью параллельным ($f = 0$) и полностью последовательным ($f = 1$) программам. Тогда ускорение S , которое может быть получено на компьютере

из p процессоров при данном значении f , оценивается так:

$$S \leq \frac{1}{f + (1 - f)/p}.$$

Следствие закона Амдала. Доля последовательно кода, которую нужно исследовать, чтобы получить заданное ускорение, находится по формуле:

$$f = \frac{(p - S)}{S(p - 1)}.$$

Для максимально эффективной работы параллельной вычислительной системы необходимо тщательное согласование структуры программ и алгоритмов с особенностями архитектуры вычислительной системы.

Передача сообщений по сети также влияет на производительность параллельных компьютеров:

- время инициализации отправки сообщения (латентность);
- время передачи сообщения по сети.

Максимальная скорость передачи достигается на больших сообщениях, когда латентность, возникающая лишь в начале, не столь заметна на фоне непосредственно передачи данных.

Возможность асинхронной отправки сообщений и вычислений. Если или аппаратура, или программное обеспечение не поддерживают возможност

и проведения вычислений на фоне пересылок, то возникнут неизбежные накладные расходы, связанные с ожиданием полного завершения взаимодействия параллельных процессов.

Для достижения эффективной параллельной обработки необходимо добиться **равномерной загрузки всех процессоров**. Если равномерности нет, то часть процессоров неизбежно будет простаивать, ожидая остальных, хотя в этот момент они могли бы выполнять полезную работу.

Если один процессор должен вычислить некоторые данные, которые нужны другому процессору, и если второй процесс первым дойдет до точки приема соответствующего сообщения, то он с неизбежностью будет простаивать, ожидая передачи. Для того чтобы минимизировать **время ожидания прихода сообщения** первый процесс должен отправить требуемые данные как можно раньше, отложив независимую от них работу на потом, а второй процесс должен выполнить максимум работы, не требующей ожидаемой передачи, прежде, чем выходить на точку приема сообщения.

Вопросы для самоконтроля

1. В чем проблема классификационных признаков вычислительных машин?
2. Что является основой в классификации Флинна?
3. Что является основой в классификации Фенга?
4. Что является основой в классификации Хокни?
5. Что является основой в классификации Хендлера?
6. Что является основой в классификации Шнайдера?
7. Что является основой в классификации Кришнамарфи?
8. Что является основой в классификации Базу?
9. Что означает термин “сильно связанные процессоры”?
10. В чем различие классов современных параллельных компьютеров: SMP и MPP?
11. Что значит “зацепление команд”?
12. Какие архитектуры более популярны у программистов и почему?
13. Перечислите факторы, снижающие производительность параллельных компьютеров.
14. Кластерные системы, основные задачи.
15. Закон Амдала и его следствие.

Глава 4. Оценка производительности ВВС

4.1. Оценка производительности вычислительных систем

Основу для сравнения различных типов компьютеров между собой дают стандартные методики измерения производительности. В процессе развития вычислительной техники появилось несколько таких стандартных методик [1,2]. Они позволяют разработчикам и пользователям осуществлять выбор между альтернативными решениями на основе количественных показателей, что дает возможность постоянного развития технологий в данной области.

Единицей измерения производительности компьютера является время: компьютер, выполняющий тот же объем работы за меньшее время, является более быстрым. Часто производительность измеряется как скорость появления некоторого числа событий в секунду.

Время может быть определено различными способами в зависимости от того, что мы считаем. Наиболее простой способ определения времени называется астрономическим временем, временем ответа (response time), временем выполнения (execution time) или прошедшим временем (elapsed time), это и задержки выполнения задания, включающие: работу процессора, обращения к диску, обращения к памяти, ввод/вывод и накладные расходы операционной системы. Однако при работе в мультипрограммном режиме во время ожидания ввода/вывода для одной программы процессор может выполнять другую программу, и система не обязательно будет минимизировать время выполнения данной конкретной программы.

Поэтому для измерения времени работы процессора на определенной программе используется специальный параметр – **время ЦП** (время центрального процессора, CPU time), которое не включает время ожидания ввода/вывода или время выполнения другой программы. Время ответа, видимое пользователем, является полным временем выполнения программы, а не временем ЦП. Время ЦП, как правило, делится на время, потраченное ЦП непосредственно на выполнение программы пользователя (пользовательское время ЦП), и время ЦП, затраченное операционной системой на выполнение заданий,

затребованных программой (системное время ЦП).

В ряде случаев системное время ЦП игнорируется из-за возможной неточности измерений, выполняемых самой операционной системой, а также из-за проблем, связанных со сравнением производительности машин с разными операционными системами. С другой стороны, системный код на некоторых машинах является пользовательским кодом на других и, кроме того, практически никакая программа не может работать без операционной системы. Поэтому при измерениях производительности процессора часто используется сумма пользовательского и системного времени ЦП.

В большинстве современных процессоров скорость протекания процессов взаимодействия внутренних функциональных устройств определяется не естественными задержками в этих устройствах, а задается единой системой синхросигналов, вырабатываемых некоторым генератором тактовых импульсов, как правило, работающим с постоянной скоростью. Дискретные временные события называются тактами синхронизации (clock ticks), просто тактами (ticks), периодами синхронизации (clock periods), циклами (cycles) или циклами синхронизации (clock cycles). Разработчики компьютеров обычно говорят о периоде синхронизации, который определяется либо своей длительностью (например, 10 наносекунд), либо частотой (например, 100 МГц). Длительность периода синхронизации есть величина, обратная к частоте синхронизации.

Время ЦП для некоторой программы может быть выражено двумя способами: количеством тактов синхронизации для данной программы, умноженным на длительность такта синхронизации, либо количеством тактов синхронизации для данной программы, деленным на частоту синхронизации.

Важной характеристикой, также позволяющей быстро оценить время ЦП при известном количестве выполняемых команд в программе, является среднее количество тактов синхронизации на одну команду – **CPI** (clock cycles per instruction).

Производительность ЦП, таким образом, зависит от трех параметров: такта (или частоты) синхронизации, среднего количества тактов на команду и количества выполняемых команд. Невозможно изменить ни один из указанных параметров изолированно от другого, поскольку базовые технологии, используемые для изменения каждого из этих параметров, взаимосвязаны: частота синхронизации определяется технологией аппаратных средств и функциональной организацией процессора; среднее количество тактов на команду зависит от функциональной организации и архитектуры системы команд; а количество выполняемых в программе команд определяется архитектурой

системы команд и технологией компиляторов. Когда сравниваются две машины, необходимо рассматривать все три компонента, чтобы понять относительную производительность.

4.2. Стандарты в области оценки производительности

В процессе поиска стандартной единицы измерения производительности компьютеров было принято несколько популярных единиц измерения. В действительности, наиболее подходящей и надежной единицей измерения производительности является время выполнения реальных программ, и все предлагаемые замены кроют в себе опасность и зачастую вводят в заблуждение.

Одной из наиболее популярных альтернативных единиц измерения производительности процессора (по отношению ко времени выполнения) является **MIPS** (миллион команд в секунду) [1,2]. Имеется несколько различных вариантов определения MIPS.

В общем случае MIPS есть скорость операций в единицу времени, т. е. для любой данной программы MIPS есть отношение количества команд в программе ко времени ее выполнения. Таким образом, производительность может быть определена как обратная ко времени выполнения величина, причем более быстрые машины при этом будут иметь более высокий рейтинг MIPS.

Положительной стороной MIPS является то, что эту характеристику легко понять, особенно покупателю, и что более быстрая машина характеризуется большим числом MIPS, что соответствует нашим интуитивным представлениям. Однако при использовании MIPS в качестве метрики для сравнения возникает три проблемы. Во-первых, MIPS зависит от набора команд процессора, что затрудняет сравнение по MIPS компьютеров, имеющих разные системы команд. Во-вторых, MIPS даже на одном и том же компьютере меняется от программы к программе. В-третьих, MIPS может меняться по отношению к производительности в противоположную сторону.

Классическим примером для последнего случая является рейтинг MIPS для машины, в состав которой входит сопроцессор плавающей точки. Поскольку в общем случае на каждую команду с плавающей точкой требуется большее количество тактов синхронизации, чем на целочисленную команду, то программы, используя сопроцессор плавающей точки вместо соответствующих подпрограмм из состава программного обеспечения, выполняются за меньшее время, но имеют меньший рейтинг MIPS. При отсутствии сопроцессора операции над числами с плавающей точкой реализуются с помощью подпрограмм, использующих более простые команды целочисленной арифметики и,

как следствие, такие машины имеют более высокий рейтинг MIPS, но выполняют настолько большее количество команд, что общее время выполнения значительно увеличивается. Подобные аномалии наблюдаются и при использовании оптимизирующих компиляторов, когда в результате оптимизации сокращается количество выполняемых в программе команд, рейтинг MIPS уменьшается, а производительность увеличивается.

Другое определение MIPS связано с очень популярным когда-то компьютером VAX 11/780 компании DEC. Именно этот компьютер был принят в качестве эталона для сравнения производительности различных машин. Считалось, что производительность VAX 11/780 равна 1 MIPS (одному миллиону команд в секунду). Быстродействие VAX 11/780 на тесте Dhrystone (составляло 1757 Dhrystone в секунду. Таким образом, 1 MIPS равен 1757 Dhrystone в секунду.

Третье определение MIPS связано с IBM RS/6000 MIPS. Дело в том, что ряд производителей и пользователей (последователей фирмы IBM) предпочитают сравнивать производительность своих компьютеров с производительностью современных компьютеров IBM, а не со старой машиной компании DEC. Соотношения между VAX MIPS и RS/6000 MIPS никогда широко не публиковались, но 1 RS/6000 MIPS примерно равен 1,6 VAX 11/780 MIPS.

Обычно при решении научно-технических задач, существенно использующих арифметику с плавающей точкой, производительность процессора оценивается в **MFLOPS** (миллионах элементарных арифметических операций над числами с плавающей точкой, выполненных в секунду) [1,2]. Единица измерения MFLOPS предназначена для оценки производительности только операций с плавающей точкой и поэтому не применима вне этой ограниченной области. Например, программы компиляторов имеют рейтинг MFLOPS, близкий к нулю, вне зависимости от того, насколько быстра машина, поскольку компиляторы редко используют арифметику с плавающей точкой.

Рейтинг MFLOPS зависит и от машины, и от программы. Эта единица измерения считается более объективной, чем MIPS. Обосновывается это тем, что MFLOPS базируется на количестве выполняемых операций, а не на количестве выполняемых команд. По мнению многих программистов, одна и та же программа, работающая на различных компьютерах, будет выполнять различное количество команд, но одно и то же количество операций с плавающей точкой. Поэтому рейтинг MFLOPS предназначается для справедливого сравнения различных машин между собой.

Однако MFLOPS также имеет ряд недостатков. Прежде всего, это связано с тем, что наборы операций с плавающей точкой не совместимы на различных

компьютерах. Например, в суперкомпьютерах фирмы Cray Research отсутствует команда деления (имеется, операция вычисления обратной величины числа с плавающей точкой, а операция деления может быть реализована с помощью умножения делимого на обратную величину делителя). В то же время многие современные микропроцессоры имеют команды: деления, вычисления квадратного корня, синуса и косинуса.

Другая проблема заключается в том, что рейтинг MFLOPS меняется не только на смеси целочисленных операций и операций с плавающей точкой, но и на смеси быстрых и медленных операций с плавающей точкой. Например, программа со 100 % операций сложения будет иметь более высокий рейтинг, чем программа со 100 % операций деления.

Решение обеих проблем заключается в том, чтобы взять “каноническое” или “нормализованное” число операций с плавающей точкой из исходного текста программы и затем поделить его на время выполнения.

Наиболее часто MFLOPS, как единица измерения производительности, используется при проведении контрольных испытаний на тестовых пакетах “Ливерморские циклы” и LINPACK. Далее в таблице показано, каким образом авторы тестового пакета “Ливерморские циклы” вычисляют для программы количество нормализованных операций с плавающей точкой (ПТ) в соответствии с операциями, действительно находящимися в ее исходном тексте. Таким образом, рейтинг реальных MFLOPS отличается от рейтинга нормализованных MFLOPS, который часто приводится в литературе по суперкомпьютерам.

Реальные операции с ПТ	Нормализованные операции с ПТ
Сложение, вычитание, сравнение, умножение	1
Деление, квадратный корень	4
Экспонента, синус	8

Однако, как и любая другая единица измерения, рейтинг MFLOPS для отдельной программы не может быть обобщен на все случаи жизни, чтобы представлять единственную единицу измерения производительности компьютера.

4.3. Виды тестов

Оценка эффективности вычислительных систем на специально подготовленных для этих целей задачах (benchmarks), или оценочное тестирование, изначально имело цель дать прогноз относительно возможностей исследуемой системы при решении интересующего класса задач. Это важное для пользователей назначение тестов является основным и в настоящее время.

По мере накопления тестов, опыта их использования и результатов анализа полученных данных стало ясно, что тесты, как бы их не критиковали за ограниченность и неадекватность, на практике в достаточной степени фиксируют характер и основные особенности решаемых задач. Это оказалось чрезвычайно полезным для разработчиков вычислительных систем, позволило оптимизировать разрабатываемую аппаратуру под выявленные на тестах вычислительные процессы и закономерности.

Важность применения оценочного тестирования значительно возросла в последнее время, когда стало возможным и экономически выгодным строить системы из готовых компонентов – вычислительных узлов на базе материнских плат персональных машин или рабочих станций и высокоскоростных коммутационных сетей. Правильность выбора компонентов в таких работах – основа успеха.

Таким образом, оценочное тестирование имеет по крайней мере две цели – обеспечить необходимой информацией пользователя и разработчика вычислительной системы.

Существующие тесты производительности компьютеров можно разделить на несколько категорий [11]:

- тесты производительности процессора;
- тесты производительности файловой системы;
- тесты производительности сети;
- комбинированные;
- тесты корпорации SPEC (Strandart Performance Evaluation Corporatiuon).

При этом во всех категориях отдельно выделяют **синтетические тесты** – тесты, не являющиеся частью реальной вычислительной программы (программ).

1. Тесты производительности процессора. Для работы с метриками MIPS и MFLOPS чаще всего используются системы тестов Dhrystone, LINPACK и “Ливерморские циклы”.

Синтетический тест **Dhrystone** позволяет оценивать эффективность процессоров и компиляторов с языка C для программ нечисловой

обработки [1, 11]. Он представляет собой тестовую смесь, 53 % которой составляют операторы присваивания, 32 % – операторы управления и 15 % – вызовы функций. Это короткий тест общее число команд которого равняется 100. Скорость выполнения программы измеряется в Dhrystone в секунду.

Следует отметить, что в настоящее время тест Dhrystone практически не применяется. Малый объем позволяет разместить все команды теста в кэш-памяти первого уровня современного микропроцессора, и он не позволяет даже оценить эффект наличия кэш-памяти второго уровня, хотя может хорошо отражать эффект увеличения тактовой частоты.

Пакет “Ливерморские циклы” представляет собой набор фрагментов фортран-программ, каждый из которых взят из реальных программных систем, эксплуатируемых в Ливерморской национальной лаборатории им. Лоуренса (США) [1, 11]. Обычно при проведении испытаний используется либо малый набор из 14 циклов, либо большой набор из 24 циклов.

Пакет “Ливерморские циклы” используется для оценки производительности вычислительных машин с середины 60-х годов. Он включает типичные фрагменты программ численных задач. Появление новых типов машин, в том числе векторных и параллельных, не уменьшило важности данного пакета, однако изменились значения производительности и величины разброса между разными циклами.

На векторной машине производительность зависит не только от элементной базы, но и от характера самого алгоритма, т. е. коэффициента векторизуемости. Для входящих в состав пакета “Ливерморские циклы” тестов коэффициент векторизуемости колеблется от 0 до 100 %, что еще раз подтверждает их ценность для оценки производительности векторных архитектур. Кроме характера алгоритма, на коэффициент векторизуемости влияет и качество векторизатора, встроенного в компилятор.

На параллельной машине производительность существенно зависит от соответствия между структурой аппаратных связей вычислительных элементов и структурой вычислений в алгоритме. Важно, чтобы тестовый пакет представлял алгоритмы различных структур. В пакете “Ливерморские циклы” встречаются последовательные, сеточные, конвейерные, волновые вычислительные алгоритмы, что подтверждает их пригодность и для параллельных машин. Однако обобщение результатов измерения производительности, полученных для одной параллельной машины, на другие параллельные машины или хотя бы на некоторый подкласс параллельных машин может дать неверный результат, ибо структуры аппаратных связей в таких машинах гораздо более разнообразны, чем, скажем, в векторных машинах.

LINPACK – это пакет фортран-программ для решения систем линейных алгебраических уравнений [1, 11]. Алгоритмы линейной алгебры весьма широко используются в самых разных задачах, и поэтому измерение производительности на LINPACK представляет интерес для многих пользователей.

В основе алгоритмов действующего варианта LINPACK лежит метод декомпозиции. Исходная матрица сначала представляется в виде произведения двух матриц стандартной структуры, над которыми затем выполняется собственно алгоритм нахождения решения. Подпрограммы, входящие в LINPACK, структурированы. В стандартном варианте LINPACK выделен внутренний уровень базовых подпрограмм, каждая из которых выполняет элементарную операцию над векторами. Набор базовых подпрограмм называется **BLAS** (Basic Linear Algebra Subprograms). Например, в BLAS входят две простые подпрограммы **SAXPY** (умножение вектора на скаляр и сложение векторов) и **SDOT** (скалярное произведение векторов). Все операции выполняются над числами с плавающей точкой, представленными с двойной точностью. Результат измеряется в MFLOPS.

Использование результатов работы тестового пакета LINPACK с двойной точностью как основы для демонстрации рейтинга MFLOPS стало общепринятой практикой в компьютерной промышленности. Для многопроцессорных систем также имеются параллельные версии LINPACK, и такие системы часто показывают линейное увеличение производительности с ростом числа процессоров. Также существует **C LINPACK** – тест LINPACK, переписанный на языке C.

Кроме того, в данной категории имеется множество других тестов [11], включая:

- **CPU2** – набор из 34 фортран-программ, интенсивно работающих с вещественной арифметикой;
- **Flops** – вычисляет производительность в MFLOPS на определенных последовательностях инструкций. Работает как на скалярных, так и на векторных машинах;
- **Heapsort** – целочисленная программа, сортирующая массив из целых чисел;
- **Matrix Multiply** – тест содержит 9 различных программ умножения матриц (размером 500x500). Оценивается работа кэш-памяти и уровень оптимизации компилятора;
- **NAS Parallel Benchmarks** – состоит из 8 различных программ для определения производительности параллельных компьютеров. Программы взяты из реальных аэро-космических расчетных пакетов;

- **PERFECT** – представляет собой комплект из 13 прикладных фортран-программ, представляющих четыре типа вычислительных задач – аэро-и гидродинамики, моделирования химических и физических процессов, инженерного проектирования, а также обработки сигналов. Выполняется дважды – до и после оптимизации исходных текстов;
- **SLALOM** – масштабируемый тест производительности для супер-компьютеров. Оценивает объем вычислений, который может произвести компьютер за одну минуту;
- **Stanford** – тестовый набор, состоящий из 8 целочисленных тестов (умножение матриц, сортировка 3 методами, перестановки, ханойская башня, растановка 8 ферзей, головоломка) и 2 тестов на вещественные вычисления (быстрое преобразование Фурье, перемножение матриц);
- **STREAM** – синтетический тест, оценивающий скорость работы с памятью с простой арифметикой и без. Основан на измерении времени выполнения больших векторных операций: копирование памяти, умножение на константу, сложение и умножение. Доступны версии на Фортране и Си. Результатами теста пользуются все ведущие разработчики высокопроизводительной техники;
- **Whetstone** – синтетический тест, ориентированный на численное программирование (с плавающей запятой). Не учитывает кэш. Компилятор легко можно оптимизировать под Whetstone.

2. К наиболее известным **тестам производительности файловой системы** относятся [11]:

- **Bonnie** – тестируется ряд стандартных файловых операций: вывод (посимвольно и блоками), обновление, чтение (посимвольно и блоками), перемещение по файлу. Выдается скорость (KB/sec) и степень загрузки ЦП;
- **IOZONE** – записывает на большой диск X-мегабайтный последовательный файл Y-байтными блоками, потом переворачивает его и считывает;
- **Nfsstone** – измеряет производительность файловой подсистемы NFS.

3. К наиболее известным **тестам производительности сети** [11] относятся:

- **MPI-тесты** – система тестов для определения эффективности программно-аппаратной среды выполнения параллельных приложений (на базе MPI);
- **comm** – тесты для измерения латентности и пропускной способности каналов в рамках MPI и PVM. Авторы: Jack Dongarra и Tom Dunigan;
- **Netperf** – тест производительности сети. Включает тесты скорости передачи (bandwidth) и задержки (latency) по протоколам TCP и UDP, используя BSD sockets, DLPI, Unix Domain Sockets, the Fore ATM API и HP

HiPPI Link Level Access;

- **Nettest** – тест производительности сети, разработанный в Cray Research;
- **ttcp** – определяющий скорость обменов по протоколам TCP и UDP между двумя машинами.

4. Тесты SPEC. Ввиду сложности вопроса методики и оценки производительности вычислительных систем ведущие фирмы производители компьютерных систем в 1988 году создали некоммерческую корпорацию SPEC (Standart Performance Evaluation Corporatiuon), призванную дать пользователям объективную оценку производительности вычислительных систем [11]. SPEC является разработчиком стандартных тестовых систем, являющихся де-факто стандартами для оценки производительности современных микропроцессоров, проводит тестирование и публикует результаты в специальном бюллетене "The SPEC Newsletter размещаемом на сервере www.SPEC.com.

SPEC состоит из следующих трех комитетов:

- **OSG** (Open Systems Group) – основной комитет;
- **HPG** (High Performance Group) – тесты для высокопроизводительных вычислительных систем;
- **GPC** (Graphics Performance Characterization Group) – тесты графики.

SPEC постоянно анализирует новые вычислительные системы и в соответствии с ними совершенствует свой тестовый набор. Так основным набором SPEC в период его создания был тестовый набор SPECint89 для оценки производительности процессора на операциях с целыми числами и SPECfp89 для оценки производительности процессора при работе с плавающей арифметикой. Однако появление в начале 1990-х годов нового поколения RISC-процессоров (PowerPC, PA-7200, MIPS, Rxxxx) сделало невозможным использование этого набора. Во-первых, резко уменьшилось время выполнения этого набора на новых системах, а во-вторых, использование оптимизирующих компиляторов искажало истинное состояние дел. Поэтому тестовый набор 1989 года был усовершенствован, дополнен и преобразован в смеси SPECint92 и SPECfp92.

На данный момент SPEC предлагает следующие основные продукты:

- **CPU2000** – тесты вычислительной производительности;
- **JVM98** – тест виртуальной Java-машины;
- **HPC96** – тесты для HPC-систем: приложение сейсмической обработки SPECseis96 (Seismic), приложение вычислительной химии SPECchem96 (GAMESS) и приложение моделирования климата SPECclimate (MM5);
- **SFS97** – тест производительности сетевой файловой системы;
- **WEB99** – тест для оценки производительности HTTP-серверов.

CPU2000 – это тестовый пакет, разработанный подразделением Open Systems Group (OSG) компании SPEC для оценки производительности микропроцессоров и вычислительных систем. Пакет CPU2000 из двух групп тестов – CINT2000 для оценки производительности на целочисленных операциях и CFP2000 для оценки производительности на операциях с плавающей точкой. Буква “C” в названиях тестов означает, что тесты являются “компонентными” (component-level), в отличие от тестов производительности системы в целом. Группы тестов CINT2000 и CFP2000 ориентированы на оценку работы микропроцессоров, подсистемы кэш-памяти и оперативной памяти, а также компиляторов. Эти тесты не имеют отношения к оценке производительности сети, дисков или графической подсистемы.

В набор CINT2000 входят 11 тестовых приложений, написанных на языке C, и один тест на C++. В набор CFP2000 входят 14 тестовых приложений, из которых 6 написаны на языке Fortran 77, 4 на языке Fortran 90 и 4 на C++. Результаты CPU2000 (CINT2000 и CFP2000) предоставляются в четырех вариантах (метриках): время выполнения одной итерации теста и количество итераций за фиксированное время для максимальной и стандартной оптимизации.

Тесты пакета CPU2000 оценивают производительность как собственно процессора, так и подсистемы памяти, а также уровень оптимизации компилятора.

Согласно философии SPEC, тесты должны компилироваться также, как пользователи будут компилировать свои программы. Метрики типа “base” (SPECint_base2000, SPECfp_base2000) соответствуют компиляции тестовых программ с некоторыми “базовыми” опциями оптимизации, одинаковыми для всех тестов в тестовом пакете. Метрики “non-base” (SPECint2000, SPECfp2000) являются необязательными и соответствуют наилучшей оптимизации, которую производитель может обеспечить для каждого конкретного теста. Метрики типа “rate” (SPECint_rate2000, SPECfp_rate2000) позволяют оценить суммарный объем вычислений, который компьютер может выполнить за определенное время. То есть на SMP-компьютере допускается запустить несколько копий одного теста и в качестве результата выдать суммарное количество итераций, выполненное всеми процессорами за определенное фиксированное время. Метрики “non-rate” (SPECint2000) оценивают просто “скорость” вычислений.

Разработчики тестов CPU2000 отказались от использования стандартных абсолютных единиц типа MFLOPS или MIPS. Вместо этого используются собственные относительные единицы SPEC. Результаты “нормализуются” по

отношению к аналогичным результатам на “эталонной” машине (reference system). Это рабочая станция Sun Ultra 5/10 (процессор UltraSPARC II с тактовой частотой 300 MHz). На данной машине прогон всех тестов CPU2000 занимает примерно двое суток (48 часов).

Тестовые наборы CINT2000 и CFP2000 состоят соответственно из 12 и 14 независимых тестовых программ, каждая из которых дает отдельный показатель производительности. Для усреднения результатов по отдельным тестам в данном тестовом наборе используется среднее геометрическое от нормализованных результатов всех тестов.

Для устранения влияния случайных факторов на результаты тестирования, каждый тест прогоняется нечетное число раз (как минимум, 3), результаты по запускам располагаются по возрастанию и в качестве окончательного берется результат из середины ряда (медианный).

Показатель **SPECint2000**, например, вычисляется следующим образом:

$$SPECint2000 = RefTime / MeasuredTime,$$

где *RefTime* – время исполнения теста на эталонной машине, а *MeasuredTime* – время исполнения на тестируемой машине. Таким образом, смысл этого показателя – в относительном ускорении по сравнению с эталонной машиной.

Показатель **SPECint_rate2000** вычисляется следующим образом:

$$SPECint_rate2000 = N * (RefTime / MeasuredTime) * (60*60*24/RefJobTime),$$

где *N* – число запущенных копий (итераций) теста, $60*60*24$ – это число секунд в сутках, а величина *RefJobTime* принята равной 9600.

SPEChpc96 включает в себя модельные приложения из двух прикладных областей: обработка сейсмических данных (**SPECseis96**) и вычислительная химия (**SPECchem96**). В версии 1.10 в пакет включено также приложение моделирования климата (**SPECclimate**). Приложения написаны на языке Fortran-77 с отдельными модулями, написанными на C. Приложения распараллелены в модели передачи сообщений с помощью PVM. PVM (Parallel Virtual Machine) – общедоступная библиотека, предоставляющая возможности управления процессами с помощью механизма передачи сообщений. Существуют реализации PVM для самых различных платформ.

5. Комбинированные тесты – тестовые программы и пакеты, тестирующие сразу несколько элементов архитектуры компьютера (ЦП, память, файловая система, сеть и т. д.).

Оценки SPEC важны, прежде всего, для анализа систем, основное назначение которых быть вычислителем вообще, без детального уточнения конкретной специфики – тестовые наборы дают сравнение по работе с целыми и с вещественными числами. **Методика AIM** направлена на сравнение и оценку производительности вычислительных систем, применительно к конкретному приложению и планируемому использованию [11]. Независимая компания AIM Technology, основанная в 1981 году, предложила методики оценок и тестовые смеси, предназначенные для работы с UNIX-системами и тестирующие практически все компоненты вычислительной системы. Разработанные методики позволяют получить более комплексную, чем тесты SPEC, оценку тестируемой архитектуры и основаны на сочетании нескольких критериев, представленных в Performance Report II:

- пиковая производительность (выбор рабочей станции);
- максимальная загрузка (выбор сервера);
- обработка UNIX-утилит (выбор станции для рабочего места разработчика программного обеспечения);
- пропускная способность;
- цена системы.

Основной и наиболее распространенный тестовый набор AIM Technology – System Benchmark (Suite III) предназначен для анализа вычислительной системы без учета работы X Window и сетевых задержек. Этот набор позволяет оценить производительность выполнения обменов с оперативной памятью, скорость вычислений с вещественными и целыми числами, выполнение обменов данными между процессорами, вызовы функций языка Си, быстрое действие операций чтения/записи на диск. Компания AIM Technology каждый квартал публикует обзорные отчеты по тестируемым системам. В этих отчетах приводятся показатели производительности и стоимостные оценки на каждую единицу эффективности системы.

Помимо упомянутых, компания AIM Technology разработала также специальные наборы тестовых смесей, характеризующие использование вычислительной системы в следующих прикладных областях:

- General Workstation Mix – среда разработки программного обеспечения;
- Mechanical CAD Mix – среда автоматизации проектирования в машиностроении (с использованием 3-мерной графики);
- GIS Mix – среда геоинформационных приложений;
- General Business – среда стандартных офисных приложений (электронные таблицы, почта);
- Shared/Multiuser Mix – многопользовательская среда;

- Computer Server Mix – среда центрального сервера для большого объема вычислений;
- File Server Mix – среда файлового сервера;
- RBMS Mix – среда обработки транзакций реляционной базы данных.

Byte – официальный тест журнала BYTE для UNIX-систем. Состоит из двух тестов плавающей арифметики (Dhrystone 2 с использованием и без использования регистровых переменных) и 7 тестов целочисленной арифметики (измерение накладных расходов на системные вызовы, порождение процессов, копирование файла, передача данных через pipe, переключение контекста и решение задачи Ханойских башен.)

Imbench – пакет тестов для оценки различных аспектов UNIX-систем, включает:

- тесты скорости работы с памятью, файлами, каналами и сетью (TCP/IP);
- тесты стандартных системных операций (переключение контекста, обработка сигнала, создание файла и т. п.).

MUSBUS – комплексный тест, включающий тесты арифметики, межпроцессного взаимодействия, скорости работы с памятью, файловой системой, скорость обработки системных вызовов, эмуляцию работы нескольких пользователей.

SPEC benchmarks – коммерческая система тестов для оценки производительности процессора, сети, WWW-сервера, файловой подсистемы NFS и стандартных средств разработки в UNIX.

SYSmark – тест аппаратуры ПК (DOS/Windows), разработанный журналом BYTE. Проверяет часто выполняемые задачи (обработка текстов, БД, графики и др.)

WPI Benchmark Suite – разработан Worcester Polytechnic Institute Mach Research Group для сравнения производительности различных реализаций ОС UNIX. Состоит из пяти тестов:

- компилятора gcc;
- приложения клиент-сервер, работающего с базой данных;
- резервного копирования (backup);
- передачи файлов по FTP;
- работы с X-сервером.

4.4. Критерии ранжирования ВВС

В зависимости от того, какие тесты используются для оценки производительности, существуют различные критерии ранжирования

высокопроизводительных вычислительных систем. Один из наиболее популярных критериев оценки производительности компьютеров – ранжирование по результатам теста LINPACK.

В соответствии с результатами теста LINPACK два раза в год обновляется список из 500 наиболее мощных суперкомпьютеров в мире - **Top500** [57]. Данный рейтинг ведется с 1993 г. экспертами США из Государственного научно-исследовательского вычислительного центра министерства энергетики, а также из университетов Мангейма и Теннесси. Официальная страница Top500 находится по адресу www.top500.org.

Строчки в Top500 распределяются в соответствии с тем, какую производительность продемонстрировали системы при прохождении теста LINPACK. Проект направлен на обеспечение надежной основы для выявления и отслеживания тенденций в области высокопроизводительных вычислений. Более подробно информация о Top500 на примере 34 редакции списка, вышедшей в ноябре 2009 года, представлена в приложении 1.

Список 50 наиболее мощных суперкомпьютеров по странам СНГ (**Top50**) также основан на результатах теста LINPACK [22]. Данный проект организован Межведомственным суперкомпьютерным центром РАН и Научно-исследовательским вычислительным центром МГУ им. М. В. Ломоносова в мае 2004 года при поддержке российской компании Т-Платформы. В список включаются 50 вычислительных систем, установленных на территории СНГ и показавших к моменту выхода списка наибольшую производительность на тесте LINPACK. Официальная страница Top50 находится по адресу <http://supercomputers.ru>.

Более подробно информация о Top50 на примере 12 редакции списка, вышедшей в марте 2010 года, представлена в приложении 1.

Однако рейтинги, построенные на основе теста LINPACK, не являются универсальным [8]. Недостатки использования данного подхода на примере одной из редакций списка Top500 представлены в приложении 1.

Вопросы для самоконтроля

1. Назовите наиболее распространенные способы определения времени выполнения программы на компьютере.
2. Что включает в себя время центрального процессора (время ЦП)?
3. В чем отличие времени центрального процессора и времени выполнения программы на компьютере?

4. Продолжите фразу: “В большинстве современных процессоров скорость протекания процессов взаимодействия внутренних функциональных устройств определяется не естественными задержками в этих устройствах, а ...”.
5. От каких трех параметров зависит производительность ЦП? Можно ли изменить какой-либо из этих параметров изолированно от других?
6. Назовите наиболее популярные единицы измерения производительности компьютеров и их характерные особенности.
7. Назовите положительные стороны использования MIPS.
8. В чем опасность использования MIPS?
9. Назовите положительные стороны использования MFLOPS.
10. В чем опасность использования MFLOPS?
11. Перечислите категории тестов производительности компьютеров. Назовите для каждой из категорий входящие в неё тесты.
12. Что такое синтетические тесты?
13. Опишите особенности теста Dhrystone.
14. Опишите особенности теста LINPACK.
15. Опишите особенности пакета “Ливерморские циклы”.
16. Что такое SPEC?
17. Что такое комбинированные тесты? Перечислите наиболее известные комбинированные тесты.
18. Раскройте суть методики AIM.
19. Что такое Top500 и Top50? Назовите их схожесть и отличия.

Глава 5. Программирование для ВВС

5.1. Соответствие структуры программ и архитектуры ЭВМ

В процессе решения любой задачи на параллельном компьютере можно выделить следующие этапы: формулировка задачи, выбор метода ее решения, фиксация алгоритма, выбор технологии программирования, создание программы и, наконец, выполнение ее на том или ином компьютере. Все эти этапы важны и для обычных компьютеров, но при использовании параллельных вычислительных систем они приобретают особую значимость. Параллельные компьютеры специально проектируются для того, чтобы работать с большой производительностью, но они не могут работать одинаково эффективно на любых программах. Если структура программы не соответствует особенностям их архитектуры, то производительность неизбежно падает.

Указанное несоответствие может возникнуть на любом этапе решения задачи. Если на каком-либо одном шаге не учитывались особенности целевого компьютера (его архитектура и программная среда), то большой производительности на программе заведомо не будет. В самом деле, ориентация на векторно-конвейерный компьютер или вычислительный кластер с распределенной памятью во многом определит метод решения задачи. В одном случае в программе необходимо векторизовать внутренние циклы, а в другом случае нужно думать о распараллеливании значительных фрагментов кода. И то, и другое свойство программ определяется свойствами заложенных в них методов. Не обладает выбранный метод такими свойствами, их не будет и в программе, а, значит, и не будет высокой производительности. Параллельный компьютер стоит в конце всей цепочки, и поэтому его влияние прослеживается везде. Чем аккуратнее исследователь проводит каждый этап, чем больше структура программы соответствует особенностям архитектуры компьютера, тем выше его производительность и тем ближе она к его пиковым показателям. Все понимают, что достичь пиковой производительности невозможно. Этот показатель в сравнении с реальной производительностью скорее

играет роль ориентира, показывая, насколько полно использованы возможности компьютера при выполнении той или иной реальной программы. Однако пик производительности вычисляется для случая, когда все работает с максимальной загрузкой, без конфликтов и ожиданий, т. е. в идеальных условиях. В реальности же все не так. Не так сложно построить компьютер с рекордными показателями пиковой производительности. Гораздо труднее предложить эффективный способ его использования, поскольку здесь уже нужно учитывать все предыдущие этапы в указанной выше цепочке [9].

Изящных конструкторских идей при построении параллельных вычислительных систем много. В состав компьютеров входит все больше и больше всевозможных вычислительных узлов. Порой конструкторам приходится разрабатывать уникальные решения, чтобы все это множество устройств заставить согласованно работать вместе. Это правильно, на этом держится прогресс вычислительной техники. Не нужно только забывать о главном. Погоня за внутренней согласованностью не должна становиться самоцелью. Любой параллельный компьютер является инструментом для решения реальных задач. В конечном счете, с этой точки зрения и нужно все оценивать. Если внутренняя согласованность в компьютере достигается во имя эффективного решения задач, то это оправданно. Но если нет, то сразу возникает масса вопросов.

5.2. Модели обмена данными

Модели обмена данными в распределенных системах настолько важны, что зачастую та или иная модель параллельных вычислений ассоциируется с конкретным способом взаимодействия процессов или потоков.

Процесс обычно определяют как экземпляр выполняемой программы, и он состоит из двух компонентов:

- объекта ядра, через который операционная система управляет процессом. Там же хранится статистическая информация о процессе;
- адресного пространства, в котором содержится код и данные всех исполняемых модулей и библиотек динамической компиляции. Именно в нем находятся области памяти, динамически распределяемые для стеков потоков и других нужд.

Процессы инертны. Чтобы процесс что-нибудь выполнил, в нем нужно создать поток (нить, легковесный процесс). Именно потоки отвечают за исполнение кода, содержащегося в адресном пространстве процесса. В принципе, один процесс может владеть несколькими потоками, и тогда они

“одновременно” исполняют код в адресном пространстве процесса [18].

Для этого каждый поток должен располагать собственным набором регистров процессора и собственным стеком. В каждом процессе есть минимум один поток.

Поток, в отличие от полновесного процесса, не имеет защищенной области памяти, хотя может обладать своими локальными переменными, и разделяет с другими потоками общий ресурс, например, оперативную память. Далее в этой главе понятие “процесс” будет использоваться тогда, когда деление на полновесные и легковесные процессы не является существенным. Основные способы взаимодействия процессов в масштабируемых параллельных системах: *обмен сообщениями, разделение общей памяти и прямой доступ к удаленной памяти* — **RDMA** (Remote Direct Memory Access).

Модель обмена сообщениями. Данная модель применима к различным архитектурам – без единого адресного пространства (мультикомпьютеры и кластеры) и с общей памятью (SMP, NUMA). Модели распределенной обработки традиционно использовали механизм передачи сообщений. Так, удаленный вызов процедуры **RPC** (Remote Procedure Call) является по сути одним из способов передачи сообщений. В средах **DCOM** (Distributed Component Object Model) [25] и **CORBA** (Common Object Request Broker Architecture) вызов метода представляет собой объектный вариант RPC. При этом RPC реализует схему передачи сообщений, в соответствии с которой в распределенном приложении взаимодействуют процедура-клиент и процедура-сервер. Технологии InfiniBand и RapidIO предусматривают операции передачи сообщений, хотя и предполагают реализацию концепции общей разделяемой памяти.

Разновидности модели обмена сообщениями определяются главным образом реализацией механизмов и организацией среды для взаимодействия процессов. Под *механизмом взаимодействия* понимаются, прежде всего, режимы и характер передачи/приема сообщений.

Различают неблокирующий и блокирующий режимы обмена. При неблокирующем обмене ни один из взаимодействующих процессов не приостанавливается, дожидаясь завершения приема (receive) или передачи (send) сообщения. Если обмен блокирующий, то передающий процесс (source) и/или принимающий процесс (destination) продолжают работу лишь после подтверждения получения сообщения.

Характер обмена сообщениями может быть синхронным, асинхронным либо асинхронно/синхронным. При синхронном обмене один из взаимодействующих процессов, достигая некоторого состояния, в котором он готов к обмену, переходит в режим ожидания до тех пор, пока другой процесс не достигнет

соответствующего состояния. В этом смысле такой обмен можно назвать блокирующим.

При асинхронном взаимодействии возврат из процедуры обмена осуществляется сразу же после передачи сообщения без ожидания завершения приема. Такой обмен можно охарактеризовать как неблокирующий. В случае асинхронно/синхронного обмена операция передачи сообщения является асинхронной (процесс-отправитель не блокируется), а операция приема — синхронной (процесс-получатель блокируется).

Иногда блокирующий обмен называют обменом с синхронизацией, а неблокирующий обмен — асинхронным. В данной работе понятия “синхронный” и “асинхронный” будут употребляться несколько в другом смысле. Дело в том, что для организации асинхронного обмена необходимо наличие так называемых буферов, которые упорядочивают сообщения в соответствии с некоторой дисциплиной — как правило, **FIFO** (First In First Out). Например, при асинхронной передаче сообщения оно может копироваться в буфер процесса-отправителя, который продолжает работу одновременно с пересылкой сообщения процессу-получателю. Буферизованный обмен сообщениями произвольных размеров поддерживается в ОС семейств UNIX (различные дистрибутивы Linux, Solaris, FreeBSD и др.) и Windows NT (Windows 2000, XP, Vista, Windows 7). В системах UNIX процессы взаимодействуют через каналы — специальный вид буфера, не сохраняющего границы сообщений. В ОС семейства Windows NT, помимо каналов, механизмы взаимодействия реализуются почтовыми ящиками, сокетами, RPC и общими файлами [18]. Буферные прием/передача специфицируются также средствами системы программирования MPI. Механизм рандеву не требует никакой буферизации. В дальнейшем под синхронным и асинхронным взаимодействием процессов будет пониматься соответственно безбуферный и буферизованный обмен сообщениями.

Назначение *среды взаимодействия* — обеспечение парных и коллективных обменов сообщениями между процессами, которые могут объединяться в группы. Когда говорят о парном взаимодействии процессов, иногда используют понятие канала — своего рода, виртуальной однонаправленной коммуникационной линии, связывающей процессы. Каналы можно создавать и удалять. При отправке сообщения достаточно указать идентификатор канала без указания имени принимающего процесса. В соответствии с механизмами взаимодействия процессов различают синхронные, асинхронные и асинхронно/синхронные каналы. Так, при асинхронном обмене сообщение посылается в канал, связывающий передающий и принимающий процессы, причем работа передающего процесса не блокируется. В некоторых моделях с каналами

ассоциируются очереди (потoki) сообщений.

При организации среды взаимодействия процессов необходимо учитывать опасность возникновения тупиковых ситуаций (дедлоков), заключающихся во взаимной блокировке обменивающихся сообщениями процессов. Простейшее проявление дедлока состоит в том, что один процесс не может передать сообщение другому процессу, поскольку этот другой процесс ждет, когда первый процесс примет сообщение от него.

В ОС UNIX тупики исключаются путем специальной организации взаимодействия процессов и буферизации сообщений в каналах, откуда байты считываются в порядке записи. Сообщения можно считывать порциями произвольных размеров.

Основные достоинства модели обмена сообщениями заключаются в следующем. Во-первых, снимаются проблемы разделяемой памяти, связанные с согласованным состоянием данных. Во-вторых, уделяется внимание обмену данными, который требует существенных затрат времени даже при использовании высокоскоростных сетей. В-третьих, учитываются такие аспекты, как произвольная структура (длина) сообщений и, возможно, различные последовательности (недетерминизм) формирования составных частей сообщений, что характерно для распределенных систем с коммутацией пакетов.

Модель общей памяти. Разделение общей памяти для организации взаимодействия процессов предполагает, что один процесс размещает данные в известную область памяти, а другой может их считывать. Сложность реализации такого взаимодействия обуславливается необходимостью обеспечения безопасности записи (удаления) и считывания данных [1].

Два основных подхода к решению этой проблемы — взаимное исключение критических интервалов и координация (условная синхронизация) процессов. В критическом интервале лишь один процесс имеет исключительный доступ к общей переменной. Для установки критических интервалов используются операции синхронизации, реализуемые, например, с помощью семафоров, блокирования процессов или битов “занято/свободно”, присоединяемых к разделяемому словам общей памяти. Подробнее о критических интервалах будет рассказано ниже в п.6.6.

Выбор той или иной системы синхронизации зависит от особенностей решаемой задачи. Известно, что переусложнение и избыточность затрат при реализации синхронизации процессов ведут к потере эффективности вычислений. Поэтому внимание исследователей уже давно привлекают принципы асинхронного программирования на основе событийного, потокового и динамического управления процессами. Известны также примеры

аппаратной поддержки синхронизации, в частности барьеров в компьютерах Cray T3D/3E [9].

Модель общей памяти чаще всего используется в архитектурах типа SMP и NUMA [25]. Ее достоинства обусловлены следующими факторами. Во-первых, имеется единое адресное пространство, что упрощает программирование и отладку. Во-вторых, нет необходимости явного задания операций посылки и приема сообщений, когда необходимо обеспечивать своевременную смену данных в буферах. В-третьих, программы, разработанные для SMP-систем, могут быть перенесены на DSM-системы.

Главная проблема, как уже отмечалось — поддержание непротиворечивого, согласованного состояния памяти в соответствии с принимаемой моделью.

Также стоит отметить, что в целом надстройка моделей общей памяти над аппаратными средствами передачи сообщений приводит к дополнительной нагрузке на коммуникационную сеть. Это связано с созданием копий данных в удаленных узлах, передачей модификаций и миграцией данных. Альтернативой, преодолевающей этот недостаток, может быть аппаратная реализация механизмов доступа к удаленной памяти, или RDMA.

Прямой доступ к удаленной памяти — это промежуточная форма между моделями передачи сообщений и общей памяти. По сути RDMA отражает главную особенность функционирования **RM**-систем (Remote Memory) — пространство записей в разделяемые страницы [13]. В каждом процессорном узле наблюдаются все модификации страниц, отображаемых узлом в единое адресное пространство. С помощью вектора отметок синхронизации определяются необходимые преобразования конкретной разделяемой страницы при доступе к ней какого-либо узла. Спецификация InfiniBand предусматривает операции передачи и приема сообщений, а также доступ к удаленной памяти по записи и чтению.

5.3. Два типа параллелизма: параллелизм данных и параллелизм задач. Парадигмы параллельного программирования

Тип параллелизма отражает структуру приложения и/или структуру данных. В первом случае имеет место *параллелизм задач* (функциональный параллелизм; параллелизм по управлению), во втором — *параллелизм по данным*. Функциональный параллелизм означает одновременное выполнение различных задач, которые могут взаимодействовать друг с другом.

В случае параллелизма по данным возможно совместное протекание процессов, соответствующих выполнению одного и того же программного кода, но с различными данными. 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, то будет иметь место случай обычных последовательных вычислений, и тогда 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 типы параллелизма различаются методами решения этих двух основных проблем.

По П. Хансену [47], *парадигма параллельного программирования* — это реализация того или иного типа параллелизма, характерного для некоторого класса алгоритмов. Как показывает практический опыт, в основе подавляющего числа параллельных программ лежит сравнительно немного различных парадигм программирования. Выбор той или иной из них зависит от доступных вычислительных ресурсов и типа параллелизма, присущего приложению. Существуют подходы к программированию распределенных приложений, никак не учитывающие их потенциального параллелизма, например, весьма распространенная объектная модель “клиент-сервер”, основанная на механизме RPC и используемая в средах DCOM и CORBA. Очевидно, подобные модели необходимо отличать от парадигм параллельного программирования. Рассмотрим наиболее часто используемые из них.

Парадигма распределения задачи. В основе концепции распределения задачи (task-farming) лежат два понятия — основной и зависимый процессы (другое название парадигмы — master/slave (хозяин/подчиненный)). Основной процесс (master) осуществляет декомпозицию решаемой задачи и распределяет подзадачи по зависимым процессам (slaves). Кроме этого, он собирает промежуточные результаты и формирует окончательный результат вычислений (рис. 5.1).

Как правило, зависимые процессы взаимодействуют лишь с основным. Распределение подзадач по процессорам может быть как статическим, т. е.



Рис. 5. 1. Распределение задачи

выполняться до начала вычислений, так и динамическим.

Динамическая реализация парадигмы master/slave уместна тогда, когда количество подзадач превышает число процессоров, либо когда количество подзадач заранее не известно и сложно прогнозировать время их выполнения. Динамическое распределение полезно и тогда, когда необходимо сбалансировать загрузку процессоров, оптимизировать ее и учесть возможную реконфигурацию вычислительных ресурсов. Это особенно важно в кластерах в случае выхода из строя некоторых процессорных узлов.

Парадигма распределения позволяет реализовывать масштабируемые вычисления, однако “узким местом” при большом числе процессоров является централизованное управление со стороны основного процесса. Возможный путь решения этой проблемы — создание совокупности основных процессов, каждый из которых управляет своим подмножеством зависимых процессов.

Парадигма SPMD. Наиболее часто используемая парадигма — **SPMD** (Single Program Multiple Data). В рамках SPMD каждый процесс соответствует выполнению одного и того же кода с различными данными, то есть данные распределяются по доступным процессорам, и воплощается параллелизм по данным (рис. 5. 2).

Каждый из процессоров системы взаимодействует с процессорами-соседями, причем нагрузка на коммуникационную среду пропорциональна размеру промежуточных результатов вычислений. Вычислительная нагрузка на процессор зависит от размера обрабатываемого блока данных.



Рис. 5.2. Общая структура вычислений на основе парадигмы SPMD

Между различными процессорами периодически осуществляется синхронизация, причем взаимодействия процессов обычно хорошо структурированы и заранее предсказуемы. Парадигма SPMD может быть очень эффективной, если данные легко распределяются между процессами и вычислительная система является однородной, т. е. процессорные узлы идентичны.

В неоднородных системах необходима специальная схема балансировки загрузки процессоров, позволяющая перераспределять данные в ходе вычислений. SPMD весьма чувствительна к уровню готовности системы: достаточно выхода из строя одного процессора, чтобы возникла тупиковая ситуация, когда ни один из процессов не может продолжаться без общей синхронизации.

Парадигма конвейеризации данных. Парадигма конвейеризации данных основана на принципе функциональной декомпозиции: подзадачи алгоритма, которые могут выполняться одновременно, закрепляются за соответствующими процессорами, каждый из которых реализует небольшую часть (фазу) вычислений (рис. 5.3).

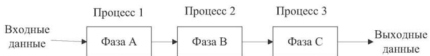


Рис. 5.3. Конвейер данных

Процессы образуют конвейер, причем каждый из них соответствует определенной стадии обработки и выполнению частной подзадачи. Модель взаимодействия процессов может быть достаточно простой, поскольку данные передаются лишь между соседними фазами вычислений. По этой причине этот тип параллелизма определяют также как параллелизм потоков данных (data flow parallelism). Взаимодействие процессов может быть синхронным и полностью асинхронным. Эффективность рассматриваемой парадигмы зависит

от возможности сбалансировать загрузку процессоров на различных стадиях конвейера. Устойчивость к возможным изменениям конфигурации (деградации) системы обеспечивается независимыми путями обмена данных в обход отдельных стадий обработки.

Парадигма “разделяй и властвуй”. Данная парадигма хорошо известна из области последовательных алгоритмов: сложная задача подразделяется на ряд подзадач, каждая из которых решается независимо, а результаты решения частных задач используются для формирования окончательного результата. Эффективность параллельной реализации этого принципа высока, если отдельные подзадачи могут быть решены приблизительно за одинаковое время. Подзадачи независимы, поэтому нет необходимости в организации взаимодействия соответствующих процессов.

Три основных типа операций, реализующих парадигму “разделяй и властвуй”, — это разбиение, вычисление и слияние. Выполнение приложения представляется в виде виртуального дерева, когда отдельные процессы создают подзадачи и комбинируют частные результаты для получения промежуточного результата (рис. 5.4).

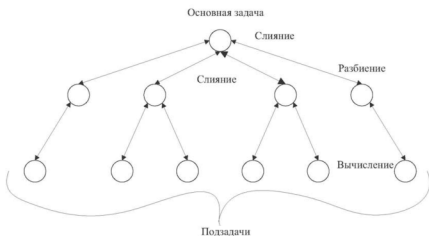


Рис. 5.4. Виртуальное дерево “разделяй и властвуй”

Таким образом, парадигма master/slave является частной формой принципа “разделяй и властвуй”, в которой операции разбиения и слияния реализуются лишь основным процессом, а другие, зависимые процессы выполняют только вычисления. В рассматриваемой парадигме задачи могут генерироваться в ходе выполнения программы и добавляться в единую очередь заданий либо распределяться по различным очередям вычислительной системы.

Спекулятивный параллелизм. Спекулятивный параллелизм представляет собой такую парадигму, когда делаются некоторые допущения, способствующие распараллеливанию программы. В частности, допускается опережающее выполнение (look-ahead execution) связанных действий при оптимистическом предположении, что параллельное выполнение таких действий не нарушает целостности процесса вычислений. Когда же случается противное, необходимо вернуться в некоторое допустимое состояние приложения, в котором этого нарушения не происходит.

Другое воплощение этой парадигмы — применение различных алгоритмов для решения одной и той же задачи. Выбирается тот из них, который первым обеспечивает окончательный результат. Такой подход полезен тогда, когда довольно трудно реализовать распараллеливание на основе рассмотренных выше парадигм. В частности, это может быть обусловлено сложными зависимостями по данным, препятствующими параллельному выполнению вычислений.

Разумеется, границы между различными парадигмами программирования могут быть весьма условными, размытыми, а для реализации сложных параллельных приложений могут потребоваться комбинации отдельных парадигм или их элементов. В такой ситуации целесообразна одновременная реализация обоих типов параллелизма (функционального и по данным), в том числе и в различных компонентах одной и той же парадигмы. В общем же парадигмы программирования отличаются двумя основными признаками: способами декомпозиции задачи и распараллеливания (распределения данных и вычислений). Например, функциональная декомпозиция и распределение конвейеризируемых данных осуществляются статически. В парадигме master/slave задача декомпозируется в статике, но распределение данных осуществляется динамически. Принцип “разделяй и властвуй” предполагает динамическую декомпозицию и распределение данных и процессов.

Вопросы для самоконтроля

1. Поясните различие понятий “нить” и “процесс”.
2. Поясните термин “механизм взаимодействия”.
3. Какие режимы обмена сообщениями существуют? Поясните их различие.
4. Какие механизмы поддерживаются в ОС Windows и UNIX для организации передачи данных между процессами?
5. Раскройте значение термина “дедлок”.

6. Какие проблемы характерны программированию в модели передачи сообщений?
7. Какие проблемы характерны программированию в модели общей памяти?
8. В чем сложность применения модели общей памяти при программировании вычислительных кластеров?
9. Чем тип параллелизма отличается от парадигмы параллельного программирования?
10. Что лежит в основе классификации типов параллелизма?
11. Какие парадигмы параллельного программирования можете назвать? Охарактеризуйте отличительные особенности каждой из них.
12. Проведите классификацию изученных парадигм параллельного программирования по типам параллелизма.

Глава 6. Операционные системы параллельных вычислительных систем

6.1. Понятие операционной системы (ОС). История развития

Под ОС понимается организованная совокупность программ, которая действует как интерфейс между аппаратурой ЭВМ и пользователем. Задачи ОС заключаются в том, чтобы:

- облегчить проектирование, программирование, отладку и сопровождение программ;
- распределить ресурсы ЭВМ таким образом, чтобы обеспечить эффективную работу всех ее компонентов (центрального процессора, устройств ввода/вывода и т. п.).

Операционная система является неотъемлемой частью любого компьютера. Главное назначение ОС — управление ресурсами ЭВМ [25]. Операционная система реализует следующие функции:

- определяет интерфейс пользователя;
- обеспечивает разделение аппаратных средств между пользователями;
- планирует доступ пользователей к общим ресурсам;
- обеспечивает эффективное выполнение операций ввода и вывода;
- осуществляет восстановление информации и вычислительного процесса в случае ошибок.

Операционная система управляет следующими основными ресурсами:

- процессорами;
- памятью;
- устройствами ввода/вывода;
- данными.

Основные этапы развития ОС можно условно разбить на несколько поколений.

Нулевое поколение (1940-е годы). Операционных систем не существовало. Все программы писались в машинных командах.

Первое поколение (1950-е годы). Основная задача ОС 1950-х годов — упрощение перехода с задачи на задачу (ОС пакетной обработки). После завершения каждой задачи (нормального или аварийного) управление ресурсами возвращалось ОС, которая приводила ЭВМ в состояние, позволяющее ввести и запустить следующую задачу, и обеспечивала ввод и запуск этой задачи. Главная цель такой ОС — сокращение времени на запуск программы и удаление ее из машины.

Характеристики ОС первого поколения [25]:

- пакетная обработка одного потока задач;
- наличие стандартных программ ввода/вывода;
- возможность автоматического перехода от программы к программе;
- наличие средств восстановления после ошибок (очистка регистров машины после аварийного завершения задачи и запуск следующей при минимальном вмешательстве оператора);
- появление языков управления заданиями (возможность подробно описывать задания и требуемые ресурсы).

В ОС первого поколения оператор составлял пакеты заданий, которые в дальнейшем без его участия последовательно запускались на выполнение управляющей программой — *монитором*. Кроме того, монитор был способен самостоятельно обрабатывать наиболее часто встречающиеся при работе пользовательских программ аварийные ситуации, такие как отсутствие исходных данных, переполнение регистров, деление на ноль, обращение к несуществующей области памяти и т. д. Пакет обычно представлял собой набор перфокарт, но для ускорения работы он мог переноситься на более удобный и емкий носитель, например на магнитную ленту или магнитный диск. Сама программа-монитор в первых реализациях также хранилась на перфокартах или перфоленте, а в более поздних — на магнитной ленте и магнитных дисках.

В числе первых программ этого типа можно назвать монитор IBSYS, управляющий выполнением заданий на компьютере IBM 7090 (1960 год), и разработанный в Советском Союзе монитор для ЭВМ БЭСМ-4 (1962 год).

Второе поколение (начало 1960-х годов). Операционная система второго поколения являлась системой коллективного пользования с мультипрограммным режимом работы или системой мультипроцессорного типа.

В мультипрограммных вычислительных системах в оперативной памяти находилось одновременно несколько программ и центральный процессор (ЦП) быстро переключался с выполнения одной программы на другую. Введение мультипрограммирования обусловлено существенным различием в

скоростных характеристиках устройств ввода/вывода и ЦП. Цель мультипрограммирования — повышение эффективности использования ЦП.

Операционные системы второго поколения можно подразделить на:

- системы разделения времени;
- системы реального времени.

Системы разделения времени функционировали в интерактивном режиме, при этом исправление ошибок в программах осуществлялось за минуты или секунды вместо часов и дней в системах пакетной обработки. Режим разделения времени способствовал повышению производительности труда программиста.

Системы реального времени использовались в системах управления технологическими процессами, бортовых вычислительных системах и т. п. Системы реального времени часто работали с недогрузкой, т. к. для них важнее быть в состоянии постоянной готовности и быстро реагировать на предусмотренные события, чем просто быть занятыми большую часть времени.

Еще одной важной тенденцией этого периода было создание семейств программно-совместимых машин и операционных систем для них. Примерами семейств программно-совместимых машин, построенных на интегральных микросхемах, являлись серии машин IBM System/360 и System/370 (аналоги этих семейств советского производства — машины серии ЕС), PDP-11 (советские аналоги — СМ-3, СМ-4, СМ-1420). Вскоре идея программно-совместимых машин стала общепризнанной.

Программная совместимость требовала и совместимости операционных систем. Однако такая совместимость подразумевала возможность работы на больших и на малых вычислительных системах, с большим и малым количеством разнообразной периферии, в коммерческой области и в области научных исследований. Операционные системы, построенные с намерением удовлетворить всем этим противоречивым требованиям, оказались чрезвычайно сложными. Они состояли из многих миллионов ассемблерных строк, написанных тысячами программистов, содержали тысячи ошибок, вызывающих нескончаемый поток исправлений. Так, объем кода системы OS/360, разработанной для серии компьютеров IBM System/360, составил 8 Мбайт, что для того времени было абсолютным рекордом.

Однако, несмотря на необозримые размеры и множество проблем, OS/360 и другие ей подобные операционные системы этого поколения действительно удовлетворяли большинству требований потребителей. За это десятилетие был сделан огромный шаг вперед и заложен прочный фундамент для создания современных операционных систем.

Третье поколение (середина 1960-х - середина 1970-х годов). Операционные системы третьего поколения являлись многорежимными, обеспечивающими обработку информации во всех известных режимах:

- пакетную обработку;
- разделение времени;
- режим реального времени;
- мультипроцессорный режим.

Универсальность этих систем обусловила их громоздкость и дороговизну. Для работы с такими системами пользователю приходилось изучать сложные языки управления заданиями, чтобы уметь описывать задания и требуемые ресурсы.

В начале 1970-х годов появились первые сетевые операционные системы, которые в отличие от многотерминальных ОС позволили не только рассредоточить пользователей, но и организовать распределенное хранение и обработку данных между несколькими компьютерами, связанными сетями. Любая сетевая операционная система, с одной стороны, несет на себе все функции локальной операционной системы, а с другой стороны, обладает некоторыми дополнительными средствами, позволяющими ей взаимодействовать по сети с операционными системами других компьютеров. Программные модули, реализующие сетевые функции, появлялись в операционных системах постепенно, по мере развития сетевых технологий, аппаратной базы компьютеров и возникновения новых задач, требующих сетевой обработки.

Четвертое поколение (с середины 1970-х годов по настоящее время). Появление четвертого поколения ОС связано с:

- распространением вычислительных сетей;
- появлением микропроцессора и персонального компьютера.

Персональные компьютеры оснащаются интерфейсными средствами приема-передачи данных и могут использоваться в качестве терминалов мощных ВС. Нет необходимости взаимодействовать с одним компьютером в режиме разделения времени — можно обращаться к территориально распределенным машинам вычислительной сети. При этом усложнились проблемы защиты информации от возможного несанкционированного доступа.

Операционные системы четвертого поколения имеют следующие особенности:

- дружественный интерфейс, ориентированный на неподготовленного пользователя;
- использование концепции виртуальных машин, благодаря которой пользователь избавлен от необходимости знать физические особенности

- машин и систем — он имеет дело с функциональным эквивалентом компьютера, создаваемым для него операционной системой и называемым виртуальной машиной;
- распределенная обработка данных: гораздо целесообразнее иметь вычислительные мощности там, где они необходимы, вместо того, чтобы передавать данные для обработки в вычислительные центры.

Важной вехой в истории миникомпьютеров и вообще в истории операционных систем явилось создание ОС Unix. Первоначально эта ОС предназначалась для поддержания режима разделения времени в миникомпьютере PDP-7. С середины 1970-х годов началось массовое использование ОС Unix. К тому времени программный код для Unix был на 90% написан на языке высокого уровня C. Широкое распространение эффективных C-компиляторов сделало Unix уникальной для того времени ОС, обладающей возможностью сравнительно легкого переноса на различные типы компьютеров: суперкомпьютеры, мэйнфреймы, миникомпьютеры, серверы и рабочие станции на базе RISC-процессоров, персональные компьютеры. Поскольку эта ОС поставлялась вместе с исходными кодами, то она стала первой открытой ОС, которую могли совершенствовать простые пользователи-энтузиасты.

Все десятилетие 1980-х годов было отмечено появлением новых, все более совершенных версий ОС Unix. Среди них были и фирменные версии Unix: SunOS, HP-UX, Irix, AIX, QNX, и многие другие, в которых производители компьютеров адаптировали коды ядра и системных утилит для своей аппаратуры. Разнообразие версий породило проблему их совместимости, которую периодически пытались решить различные организации. В результате были приняты стандарты POSIX и XPG, определяющие интерфейсы ОС для приложений, а специальное подразделение компании AT&T выпустило несколько версий Unix System III и Unix System V, призванных консолидировать разработчиков на уровне кода ядра.

6.2. Структура ОС. Виды операционных систем

Любая сложная система должна иметь понятную и рациональную структуру, то есть разделяться на части — модули, имеющие вполне законченное функциональное назначение с четко оговоренными правилами взаимодействия. Ясное понимание роли каждого отдельного модуля существенно упрощает работу по модификации и развитию системы. Напротив, сложную систему без хорошей структуры порой проще разработать заново, чем модернизировать.

Функциональная сложность операционной системы неизбежно приводит к сложности ее архитектуры, под которой понимают структурную организацию ОС на основе различных программных модулей [16]. Обычно в состав ОС входят исполняемые и объектные модули стандартных для данной ОС форматов, библиотеки разных типов, модули исходного текста программ, программные модули специального формата (например, загрузчик ОС, драйверы ввода-вывода), конфигурационные файлы, файлы документации, модули справочной системы и т. д.

Большинство современных операционных систем представляют собой хорошо структурированные модульные системы, способные к развитию, расширению и переносу на новые платформы. Какой-либо единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС.

Наиболее общим подходом к структуризации операционной системы является разделение всех ее модулей на две группы [16]:

- ядро — модули, выполняющие основные функции ОС;
- вспомогательные модули ОС.

Все операции, связанные с процессами, выполняются под управлением ядра ОС. Ядро обычно резидентно размещают в основной памяти, в то время как другие части ОС перемещаются во внешнюю память и обратно по мере необходимости.

К основным функциям ядра относятся:

- обработка прерываний;
- создание и уничтожение процессов;
- переключение процессов из состояния в состояние;
- приостановка и активизация процессов;
- синхронизация процессов;
- организация взаимодействий между процессами;
- манипулирование *векторами состояний процессов* (вектор состояния — объект, содержащий указатель на следующую команду программы, которую нужно выполнить, а также значения всех промежуточных и определяемых в программе переменных);
- поддержка операций ввода-вывода;
- поддержка распределения и перераспределения памяти;
- поддержка функций по ведению учета работы машины;
- планирование (диспетчеризация).

Одной из самых важных функций ядра является обработка прерываний.

Прерывание — событие, при котором меняется нормальная последовательность команд, выполняемых процессором [25]. Сигнал “прерывание” вырабатывается аппаратурой машины. Если произошло прерывание, то:

- управление передается ОС;
- ОС запоминает состояние прерванного процесса;
- ОС анализирует тип прерывания и передает управление соответствующей программе обработки прерываний.

Выделяют следующие типы прерываний:

- прерывания ввода-вывода;
- внешние прерывания;
- прерывания по ошибке программы;
- прерывания, вызванные сбоями аппаратуры.

Ядро обычно разрабатывается таким образом, что реализует минимально возможную предварительную обработку каждого прерывания (с запретом всех других прерываний), а затем передает это прерывание на дальнейшую обработку соответствующему системному процессу.

При проектировании ОС используется иерархический подход. В основе иерархии лежит аппаратура компьютера. На следующем уровне иерархии находится ядро ОС. В совокупности с функциями ядра ОС компьютер становится расширенной машиной, т. е. машиной, которая предоставляет для ОС и своих пользователей не только свой машинный язык, но и ряд дополнительных возможностей. Эти дополнительные функции, реализуемые при помощи ядра, часто называются примитивами.

Над ядром в иерархии находятся различные процессы ОС, которые обеспечивают поддержку процессов пользователя, например, процессы управления внешними устройствами (*рис. 6.1*).

На вершине иерархии располагаются сами процессы пользователей. Подобные иерархические системы легче отлаживать, модифицировать и тестировать. Обычно допускают только обращения сверху вниз в иерархии, т. е. средства каждого уровня могут обращаться только к тем функциям, которые находятся в нижележащих уровнях.

Сегодня существует большое количество разных типов операционных систем, отличающихся областями применения, аппаратными платформами и методами реализации. Это обуславливает значительные функциональные различия этих ОС.

ОС для автономного компьютера. Функции операционной системы автономного компьютера обычно группируются либо в соответствии с типами локальных ресурсов, которыми управляет ОС, либо в соответствии со



Рис. 6. 1. Иерархическая структура ОС

специфическими задачами, применимыми ко всем ресурсам. Иногда такие группы функций называют подсистемами [16]. Наиболее важными подсистемами управления ресурсами являются подсистемы управления процессами, памятью, файлами и внешними устройствами, а подсистемами, общими для всех ресурсов, являются подсистемы пользовательского интерфейса, защиты данных и администрирования.

Сетевые ОС. Компьютерная сеть — это набор компьютеров, связанных коммуникационной системой и снабженных соответствующим программным обеспечением, позволяющим пользователям сети получать доступ к ресурсам этого набора компьютеров [15]. Сеть могут образовывать компьютеры разных типов, в частности, небольшие микропроцессоры, рабочие станции, миникомпьютеры, персональные компьютеры или суперкомпьютеры. Коммуникационная система может включать кабели, повторители, коммутаторы, маршрутизаторы и другие устройства, обеспечивающие передачу сообщений между любой парой компьютеров сети.

Сетевая операционная система позволяет пользователю работать со своим компьютером как с автономным и добавляет к этому возможность доступа к информационным и аппаратным ресурсам других компьютеров сети.

При организации сетевой работы операционная система играет роль интерфейса, экранирующего от пользователя все детали низкоуровневых программно-аппаратных средств сети. Например, вместо числовых адресов компьютеров сети, таких как MAC-адрес и IP-адрес, операционная система компьютерной сети позволяет оперировать удобными для запоминания символическими именами. В результате в представлении пользователя сеть с ее

множеством сложных и запутанных реальных деталей превращается в достаточно понятный набор разделяемых ресурсов.

Работая в среде сетевой ОС, пользователь, хотя и может запустить задание на любой машине компьютерной сети, всегда знает, на какой машине оно выполняется. По умолчанию пользовательское задание выполняется на той машине, на которой пользователь сделал логический вход. Если же он хочет выполнить задание на другой машине, то ему нужно либо выполнить удалённый логический вход на эту машину, используя команду типа “remote login”, либо ввести специальную команду удаленного выполнения, в которой должен указать информацию, идентифицирующую удаленный компьютер.

Общий формат таких команд можно представить следующим образом:

выполни команду A на хост-машине B.

К сетевым сервисам, обеспечивающим взаимодействие между машинами посредством таких инструкций, относятся: telnet, rlogin, rsh, rdist и ssh. В настоящее время наиболее распространён протокол **ssh** (Secure Shell) — это сервис для подключения к компьютерам через сеть, выполнения команд на удаленных машинах и перемещения файлов с одного компьютера на другой. В отличие от остальных, он предоставляет строгую аутентификацию и безопасную передачу информации по незащищенным каналам.

Распределенные ОС. Магистральным направлением развития сетевых операционных систем является достижение как можно более высокой степени прозрачности сетевых ресурсов. В идеальном случае сетевая ОС должна представить пользователю сетевые ресурсы не в виде сети, а в виде ресурсов единой централизованной виртуальной машины [16]. Для такой операционной системы используют специальное название — *распределенная ОС*, или *истинно распределенная ОС*.

Распределенная ОС существует как единая операционная система в масштабах вычислительной системы. Каждый компьютер сети, работающей под управлением распределенной ОС, выполняет часть функций этой глобальной ОС. Распределенная ОС, динамически и автоматически распределяя работы по различным машинам системы для обработки, заставляет набор сетевых машин действовать как единый виртуальный вычислительный ресурс. Пользователь распределенной ОС, вообще говоря, не имеет сведений о том, на какой машине выполняется его работа. То есть в этом случае он выдаёт команды типа

выполни команду A

и агент распределённой ОС под свою ответственность выбирает хост-машину, на которой эта команда будет выполняться.

В настоящее время практически все операционные системы относятся к классу сетевых, однако они еще очень далеки от идеала истинной распределенности. Современная компьютерная сеть работает под управлением совокупности сетевых ОС, установленных на каждом из компьютеров. Эти ОС могут быть одинаковыми или разными. Например, на всех компьютерах сети может работать одна и та же ОС Unix. Более реалистичным вариантом является сеть, в которой работают разные ОС, например, часть компьютеров работает под управлением Unix FreeBSD, часть — под управлением NetWare 6.5, а остальные — под управлением Windows Vista. Все эти операционные системы функционируют независимо друг от друга в том смысле, что каждая из них принимает независимые решения о создании и завершении собственных процессов и управлении локальными ресурсами. Но в любом случае операционные системы компьютеров, работающих в сети, должны включать взаимно согласованный набор коммуникационных протоколов для организации взаимодействия процессов, выполняющихся на разных компьютерах сети, и разделения ресурсов этих компьютеров между пользователями сети.

6.3. Принципы построения ОС

Главным требованием, предъявляемым к операционной системе, является выполнение ею основных функций эффективного управления ресурсами и обеспечение удобного интерфейса для пользователя и прикладных программ [16]. Современная ОС, как правило, должна поддерживать мультипрограммную обработку, виртуальную память, свопинг, многооконный графический интерфейс пользователя, а также выполнять многие другие необходимые функции и услуги. Помимо этих требований функциональной полноты, к операционным системам предъявляются не менее важные эксплуатационные требования, которые перечислены далее.

Расширяемость. В то время как аппаратная часть компьютера устаревает за несколько лет, полезная жизнь операционных систем может измеряться десятилетиями. Примером может служить ОС Unix. Поэтому операционные системы всегда изменяются со временем эволюционно, и эти изменения более значимы, чем изменения аппаратных средств. Изменения ОС обычно заключаются в приобретении ею новых свойств, например, поддержке новых типов внешних устройств или новых сетевых технологий. Если код ОС написан таким образом, что дополнения и изменения могут вноситься без нарушения

целостности системы, то такую ОС называют расширяемой. Расширяемость достигается за счет модульной структуры ОС, при которой программы строятся из набора отдельных модулей, взаимодействующих только через функциональный интерфейс.

Переносимость. В идеале код ОС должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которые различаются не только типом процессора, но и способом организации всей аппаратуры компьютера) одного типа на аппаратную платформу другого типа. Переносимые ОС имеют несколько вариантов реализации для разных платформ, такое свойство ОС называют также *многоплатформенностью*.

Совместимость. Существует несколько “долгоживущих” популярных операционных систем (разновидности Unix/Linux, Windows, NetWare), для которых наработана широкая номенклатура популярных приложений. Для пользователя, переходящего по тем или иным причинам с одной ОС на другую, очень привлекательна возможность запуска в новой операционной системе привычного приложения. Если ОС имеет средства для выполнения прикладных программ, написанных для других операционных систем, то про нее говорят, что она обладает совместимостью с этими ОС. Следует отличать совместимость на уровне двоичных кодов от совместимости на уровне исходных текстов. Понятие совместимости включает также поддержку пользовательских интерфейсов других ОС.

Надежность и отказоустойчивость. Система должна быть защищена как от внутренних, так и от внешних ошибок, сбоев и отказов. Ее действия должны быть всегда предсказуемыми, а приложения не должны быть способны нанести вред ОС. Надежность и отказоустойчивость ОС, прежде всего, определяется архитектурными решениями, положенными в ее основу, а также качеством ее реализации (отлаженностью кода). Кроме того, важно, включает ли ОС программную поддержку аппаратных средств обеспечения отказоустойчивости, таких, например, как дисковые массивы или источники бесперебойного питания.

Безопасность. Современная ОС должна защищать данные и другие ресурсы вычислительной системы от несанкционированного доступа. Чтобы ОС обладала свойством безопасности, она должна, как минимум, иметь в своем составе средства аутентификации — определения легальности пользователей, авторизации — предоставления легальным пользователям дифференцированных прав доступа к ресурсам, аудита — фиксации всех “подозрительных” для безопасности системы событий. Свойство безопасности особенно важно для сетевых ОС. В таких ОС к задаче контроля доступа добавляется задача

защиты данных, передаваемых по сети.

Производительность. Операционная система должна обладать настолько хорошим быстродействием и временем реакции, насколько это позволяет аппаратная платформа. На производительность ОС влияет много факторов, среди которых основными являются архитектура ОС, многообразие функций, качество программного кода, возможность исполнения ОС на многопроцессорной платформе.

В случае ОС для распределенных вычислительных систем, к вышеперечисленным требованиям добавляются *масштабируемость* (возможность лёгкого добавления и удаления из состава системы новых вычислительных узлов) и *прозрачность*.

6.4. Операционные системы мультипроцессорных ЭВМ

Рассмотрим различные варианты организации операционных систем для многопроцессорных вычислительных систем.

Каждому центральному процессору — свою операционную систему. Простейший способ организации мультипроцессорных операционных систем состоит в том, чтобы статически разделить оперативную память по числу центральных процессоров и дать каждому центральному процессору свою собственную память с собственной копией операционной системы [23]. В результате n центральных процессоров будут работать как n независимых компьютеров. В качестве очевидного варианта оптимизации можно позволить всем центральным процессорам совместно использовать код операционной системы и хранить только индивидуальные копии данных (рис. 6.2). Квадраты, помеченные словом Data, означают приватные данные операционной системы для каждого центрального процессора.

Тем не менее, такая схема лучше, чем n независимых компьютеров, так как она позволяет всем машинам совместно использовать набор дисков и других устройств ввода-вывода, а также обеспечивает гибкое совместное использование памяти. Например, если требуется запустить большую программу, одному из центральных процессоров может быть выделена большая порция памяти на время выполнения этой программы. Кроме того, процессы могут эффективно общаться друг с другом, если одному процессу будет позволено писать данные в память, а другой процесс будет их считывать в этом месте. Но наличие операционной системы у каждого центрального процессора является крайне примитивным подходом.



Рис. 6. 2. Разделение памяти мультимикропроцессора между четырьмя центральными процессорами с общей копией кода операционной системы

Следует отметить следующие особенности данной схемы, возможно, не являющиеся очевидными. Во-первых, когда процесс обращается к системному вызову, системный вызов перехватывается и обрабатывается его собственным центральным процессором при помощи структур данных в таблицах операционной системы.

Во-вторых, поскольку у каждой операционной системы есть свои собственные таблицы, у нее есть также и свой набор процессов, которые она сама планирует. Совместного использования процессов нет. Если пользователь регистрируется на центральном процессоре 1, то все его процессы работают на центральном процессоре 1. В результате может случиться так, что центральный процессор 1 окажется загруженным работой, тогда как центральный процессор 2 будет простаивать.

В-третьих, совместного использования страниц также нет. Может случиться так, что у центрального процессора 2 много свободных страниц, в то время как центральный процессор 1 будет постоянно заниматься свопингом. И нет никакого способа занять свободные страницы у соседнего процессора, так как выделение памяти статически фиксировано.

В-четвертых, и это хуже всего, если операционная система поддерживает буферный кэш недавно использованных дисковых блоков, то каждая операционная система будет выполнять это независимо от остальных. Таким образом, может случиться так, что некоторый блок диска будет присутствовать в нескольких буферах одновременно, причем в нескольких буферах сразу он может оказаться модифицированным, что приведет к порче данных на диске. Единственный способ избежать этого заключается в полном отказе от блочного кэша, что значительно снизит производительность системы.

Мультимикропроцессоры типа “хозяин-подчиненный” (master-slave). По причине приведенных соображений модель “каждому процессору — свою



Рис. 6.3. Модель master-slave

операционную систему” теперь используется редко, хотя она применялась на заре эпохи мультипроцессоров, когда ставилась цель просто перенести существующие операционные системы на какой-либо новый мультипроцессор как можно быстрее. Вторая модель показана на *рис. 6.3*.

Здесь используется всего одна копия операционной системы, находящаяся на центральном процессоре 1 и отсутствующая на других центральных процессорах [23]. Все системные вызовы перенаправляются для обработки на центральный процессор 1. Центральный процессор 1 может также выполнять процессы пользователя, если у него будет оставаться для этого время. Такая схема называется “хозяин-подчиненный”, так как центральный процессор 1 является “хозяином”, то есть ведущим, а все остальные процессоры — подчиненными, или ведомыми.

Модель мультипроцессора “хозяин-подчиненный” позволяет решить большинство проблем первой модели. В этой модели используется единая структура данных (например, один общий список или набор приоритетных списков), учитывающая готовые процессы. Когда центральный процессор переходит в состояние простоя, он запрашивает у операционной системы процесс, который можно обрабатывать, и при наличии готовых процессов операционная система назначает этому процессору процесс. Поэтому при такой организации, никогда не может случиться так, что один центральный процессор будет простаивать, в то время как другой центральный процессор будет перегружен. Страницы памяти могут динамически предоставляться всем процессам. Кроме того, в такой системе есть всего один общий буферный кэш блочных устройств, поэтому дискам не грозит порча данных, как в предыдущей модели при попытке использования блочного кэша.

Недостаток этой модели состоит в том, что при большом количестве центральных процессоров хозяин может стать узким местом системы. Ведь ему



Рис. 6.4. Модель симметричного мультипроцессора

приходится обрабатывать все системные вызовы от всех центральных процессоров. Например, если обработка системных вызовов занимает 10 % времени, тогда 10 центральных процессоров завалят хозяина работой, а при 20 центральных процессорах хозяин уже не будет успевать их обрабатывать, и система начнет простаивать. Следовательно, такая модель проста и работоспособна для небольших мультипроцессоров, но на больших — она работать не может.

Симметричные мультипроцессоры. Третья модель, представляющая собой симметричные мультипроцессоры (SMP), позволяет устранить перекося предыдущей модели [23]. Как и в предыдущей схеме, в памяти находится всего одна копия операционной системы, но выполнять ее может любой процессор. При системном вызове на обратившемся к системе центральном процессоре происходит прерывание с переходом в режим ядра и обработкой данного вызова. Модель симметричного мультипроцессора показана на *рис. 6.4*.

Эта модель обеспечивает динамический баланс процессоров и памяти, поскольку в ней имеется всего один набор таблиц операционной системы. Она также позволяет избежать простоя системы, связанного с перегрузкой ведущего центрального процессора (“хозяина”), так как в ней нет ведущего центрального процессора.

И все же данная модель имеет собственные проблемы. В частности, если код операционной системы будет выполняться одновременно на двух или более центральных процессорах, произойдет катастрофа. Представьте себе два центральных процессора, одновременно берущих один и тот же процесс для запуска или запрашивающих одну и ту же свободную страницу памяти. Простейший способ разрешения подобных проблем заключается в связывании мьютекса (то есть блокировки) с операционной системой, в результате чего вся система превращается в одну большую критическую область. Когда

центральный процессор хочет выполнять код операционной системы, он должен сначала получить мьютекс. Если мьютекс заблокирован, процессор вынужден ждать. Таким образом, любой центральный процессор может выполнить код операционной системы, но в каждый момент времени только один из них будет делать это.

6.5. Соглашения при составлении псевдокода

В настоящем учебном пособии, являющимся первым в серии “Основы высокопроизводительных вычислений” будут даны правила, использующиеся при составлении псевдокода к рассматриваемым алгоритмам и тестовым программам в данном и последующих пособиях этой серии.

- Начало и конец описываемой программы начинается с ключевых слов “*Program NAME*” и “*end Program NAME*” соответственно.
- Структура блоков указывается с помощью отступов и слова *end* совместно с ключевым словом, открывающим блок.

Пример:

```
for j ← 2 to length[A] do
  summ ← A[j]
end for
```

Стиль отступов имеет силу также и в инструкциях *if-then-else*.

- Инструкции циклов *while*, *for* и *repeat*, а также общепринятая конструкция *if-then-else* интерпретируются аналогично одноименным инструкциям языка программирования Pascal. Что касается цикла *for*, то здесь есть одно различие: в языке Pascal после выхода из цикла значение переменной, выступающей в роли счетчика цикла, становится неопределенным, а в учебных пособиях данной серии счетчик сохраняет свое значение после выхода из цикла. Таким образом, сразу после завершения цикла *for* значение счетчика равно значению, на единицу превышающему верхнюю границу цикла.
- Для указания преждевременного выхода из цикла в некотором месте внутри цикла используется служебное слово *break*, а для перехода на следующую итерацию внутри цикла — слово *continue*.
- Для обозначения считывания некоторой переменной или массива с клавиатуры используется оператор *read(переменная)*, а для вывода на экран — *write(“текст”, переменная, “текст”)*.
- Символы “/” указывают, что оставшаяся часть строки – комментарий.

- Множественное присваивание, имеющее вид $i \leftarrow j \leftarrow e$, обозначает, что значение выражения e присваивается обоим переменным i и j ; оно аналогично двум последовательным инструкциям присваивания: $j \leftarrow e$ и $i \leftarrow j$.
- Переменные являются локальными по отношению к данной процедуре. Глобальные переменные не применяются, если это не указано явным образом.
- Доступ к элементам массива осуществляется путем указания имени массива, после которого в квадратных скобках следует индекс. Например, $A[i]$ — это обозначение i -го элемента массива A . С помощью обозначения “..” указывается интервал значений, которые принимает индекс массива. Таким образом, обозначение $A[1..j]$ свидетельствует о том, что данное подмножество массива A состоит из j элементов $A[1], A[2], \dots, A[j]$.
- Сложные данные обычно представляются в виде объектов, состоящих из атрибутов и полей. Доступ к определенному полю осуществляется с помощью имени поля, после которого в квадратных скобках указывается имя объекта. Например, массив трактуется как объект с атрибутом *length*, указывающим количество его элементов. Чтобы указать количество элементов массива A , нужно написать $length[A]$. Несмотря на то, что квадратные скобки используются и для индексирования элементов массива, и в атрибутах объекта, их интерпретация будет понятна из контекста.
- Параметры передаются в процедуру *по значению*: в вызывающей процедуре создается своя собственная копия параметров, и если в вызванной процедуре какому-то параметру присваивается значение, то в вызывающей процедуре не происходит никаких изменений. Если передаются объекты, то происходит копирование указателя на данные, представляющие этот объект. Например, если x — параметр вызванной процедуры, то присваивание $x \leftarrow y$, выполненное в этой процедуре, невидимо для вызывающей процедуры.
- Логические операторы “и” и “или” вычисляются *сокращенно*. Это означает, что при вычислении выражения “ x и y ” сначала вычисляется значение выражения x . Если оно ложно, то всё выражение не может быть истинным, и значение выражения y не вычисляется. Если же выражение x истинно, то для определения значения всего выражения необходимо вычислить выражение y . Аналогично, в выражении “ x или y ” величина y вычисляется только в том случае, если выражение x ложно. Сокращенные операторы позволяют составлять такие логические выражения, как “ $x \neq y$ и $f[x] \neq y$ ”, не беспокоясь о том, что произойдет при попытке вычислить выражение $f[x]$, если $x = NIL$ (значение NIL употребляется для указания того случая, когда некоторый указатель не ссылается ни на какой объект).

6.6. Взаимодействие и синхронизация процессов

Мультипрограммная ОС вместе с выполняющимися в ней заданиями пользователя может быть логически описана как набор процессов, которые взаимодействуют между собой путем пересылки сообщений и синхронизирующих сигналов и состязаются за использование ресурсов. Об этих процессах говорят, что они протекают параллельно.

Существует много определений понятия “процесс”. Например, процесс можно определить как программу в стадии выполнения. Мы определим процесс как пару $P = \langle p, S \rangle$, где p — выполняемая программа, S — вектор состояния процесса [25]. С логической точки зрения каждый процесс имеет свой собственный процессор и программу. В действительности, два различных процесса могут разделять одну и ту же программу или один и тот же процессор.

Развитие процесса может быть описано как последовательность векторов состояния $S_0, S_1, \dots, S_i, \dots$, где каждый вектор состояния S_i содержит указатель на следующую команду программы, которую нужно выполнить, а также значения всех промежуточных и определяемых в программе переменных. Взаимодействия между процессами осуществляются с помощью установки общих переменных и специальных базовых операций процессов (примитивов).

Система, управляющая процессами, должна иметь возможность выполнять следующие операции над процессами:

- создание (порождение);
- уничтожение;
- приостановка;
- возобновление;
- блокирование;
- пробуждение;
- запуск;
- изменение приоритета.

Когда несколько процессов могут асинхронно изменять содержимое общей области данных, необходимо защитить данные от одновременного доступа и изменения двумя и более процессами. Общие данные, разделяемые несколькими процессами, наиболее часто описываются как ресурс; обновление данных соответствует распределению или освобождению элементов ресурса. КС (*критической секцией*) данного процесса называется последовательность операторов его программы, выполняющих действия над ресурсом, общим для многих процессов [25].

Рассмотрим два процесса p_1 и p_2 , увеличивающих асинхронно значение

общей целочисленной переменной x , представляющей количество единиц ресурса:

$$p_1: \dots x := x + 1; \dots;$$

$$p_2: \dots x := x + 1; \dots;$$

Пусть C_1 и C_2 — центральные процессоры с внутренними регистрами R_1 и R_2 соответственно и разделяемой основной памятью. Если бы процесс p_1 выполнялся на процессоре C_1 и p_2 — на C_2 , то могла бы возникнуть любая из следующих двух последовательностей:

(1) $p_1: R_1 := x; R_1 := R_1 + 1; x := R_1; \dots$

$p_2: \dots; R_2 := x; R_2 := R_2 + 1; x := R_2; \dots$

$t_0 \xrightarrow{\hspace{10em}} t_1$

(2) $p_1: R_1 := x; R_1 := R_1 + 1; x := R_1; \dots$

$p_2: \dots; R_2 := x; R_2 := R_2 + 1; x := R_2; \dots$

Пусть x содержит значение V в момент времени t_0 . Тогда, если бы выполнение процессов p_1 и p_2 происходило как в последовательности 1, то в момент t_1 переменная x содержала бы значение $V+1$; если же — как в последовательности 2, то переменная V содержала бы значение $V+2$.

Оба значения x были бы также реализованы, если бы процессы p_1 и p_2 разделяли во времени единственный процессор с переключением управления между процессами посредством прерываний.

Решение проблемы заключается в разрешении входить в произвольный момент времени в критическую секцию, также называемую критическим интервалом " $x := x + 1$ " только одному процессу. В общем случае КС могла бы состоять из любого числа операторов.

Более точно проблема формулируется так. Пусть имеется мультипроцессорная система с общей памятью. Каждая из программ, выполняемых процессорами, содержит критическую секцию, в которой организован доступ к общим данным; программы выполняются циклически. Необходимо так запрограммировать доступ к общей памяти, чтобы в любой момент времени только один из процессоров находился в своей КС.

Относительно системы сделаны следующие предположения:

- считывание из общей памяти и запись в нее — неделимые операции, одновременные обращения (на запись или на считывание) к одной и той же ячейке памяти более чем одного процессора приведут к последовательным обращениям в произвольном порядке;
- критические секции не могут иметь связанных с ними приоритетов;
- относительные скорости процессоров неизвестны;
- программа может останавливаться вне ее КС.

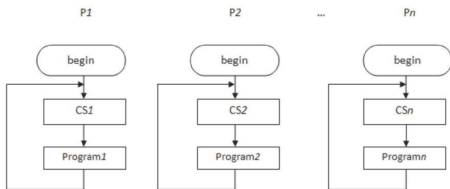


Рис. 6. 5. Система критических процессов

Проблема также может быть сформулирована в терминах нескольких процессов, асинхронно разделяющих во времени единственный процессор.

Предполагается, что система циклических процессов для проблемы КС имеет следующие программные формы (здесь и далее P_i — процесс с номером i ; $Program\ P_i$ — программа, выполняемая процессом P_i ; $program_i$ — произвольная последовательность действий в программе; Csi — критическая секция):

```

Program P1
while true
  CS1
  program1
end while
end Program P1
    
```

```

Program P2 ...
while true
  CS2
  program2
end while
end Program P2
    
```

```

Program Pn
while true
  CSn
  programn
end while
end Program Pn
    
```

На рис. 6. 5 представлена блок-схема работы процессов.

Программное решение. Пусть проблема ограничена двумя процессами. Наша цель — предохранение процессов P_1 и P_2 от одновременного вхождения в их критические секции (взаимное исключение). В то же время должны быть устранены два возможных типа блокировки:

- процесс, нормально работающий вне своей КС, не должен блокировать другой процесс при вхождении последнего в свою КС;
- два процесса, готовые войти в свои критические секции, не должны откладывать неопределенно долго решение вопроса о том, который из них действительно войдет в свою КС первым.

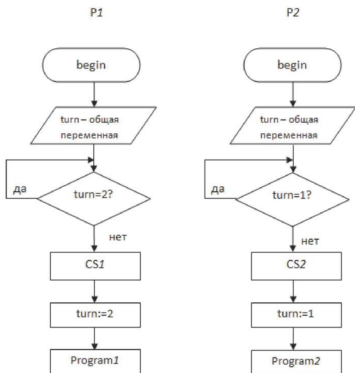


Рис. 6. 6. Решение проблемы критической секции посредством введения общей переменной

На пути разработки решения проблемы могут встретиться некоторые ловушки. Проблема легко решается, если потребовать, чтобы процессы $P1$ и $P2$ входили в свои КС попеременно: одна общая переменная может хранить указатель того, чья очередь войти в КС (рис. 6. 6):

Однако, если процессор с $P1$ гораздо медленнее, чем процессор с $P2$ и процесс $P1$ “завяз” в *program1*, то это решение нельзя считать удовлетворительным, так как процесс $P1$, находясь вне своей КС, может помешать процессу $P2$ при его вхождении в свою КС. Это возможное блокирование может быть устранено с помощью двух общих переменных $C1$ и $C2$, как флагов для указания того, находится ли процесс внутри или вовне своей КС (рис. 6. 7).

<pre> Program P1 // C1, C2 – общие переменные C1←C2←true while true while C2=false do </pre>	<pre> end while C1←false CS1 C1←true program1 </pre>
--	--


```

Program P1
// turn — общая переменная
turn←2
while true
  while turn=2 do
    end while
  CS1
  turn←2
  program1
end while
end Program P1

```

```

Program P2
// turn — общая переменная
turn←2
while true
  while turn=1 do
    end while
  CS2
  turn←1
  program2
end while
end Program P2

```

```

  end while
end Program P1

```

```

Program P2
// C1, C2 — общие переменные
C1←C2←true
while true
  while C1=false do

```

```

  end while
  C2←false
  CS2
  C2←true
  program1
end while
end Program P2

```

Когда значение $C1$ и $C2$ равно *false (true)*, то соответствующий процесс находится внутри (вне) своей критической секции. Взаимное блокирование теперь невозможно, но оба процесса могут войти в свои КС вместе; последнее может случиться, так как обе программы могут одновременно начать исполнение цикла “while true” и одновременно выполнить инструкции $C_i \leftarrow \text{false}$. Происходит так называемое взаимное выполнение, т. е. одновременное выполнение действий различными процессами, относящихся к критическим секциям каждого из процессов. Взаимное выполнение устраняется установкой значений $C1$ и $C2$ равными *false*, но теперь снова становится возможным взаимное блокирование.

Следующее решение было впервые предложено Т. Деккером [10] (рис. 6. 8):

```

Program P1
//turn, C1, C2 — общие перемен-
ные;
//exit flag — частная для P1

```

```

C1←C2←true
turn←1
while true
  exit flag←false

```

```

C1←false
while C2=false do
  if turn=1 then
    continue
  end if
  C1←true
  while turn=2 do

```

Program P2

//turn, C1, C2 — общие переменные;

//exitflag — частная для P2

C1←C2←true

turn←1

while true

```

exitflag←false
C2←false
while C1=false do
  if turn=2 then
    continue
  end if
  C2←true
  while turn=1 do
    end while
  exitflag←true
end while
if exitflag=true then
  continue
end if

```

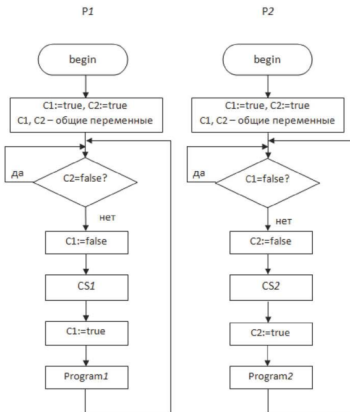


Рис. 6. 7. Решение проблемы критической секции посредством введения двух флагов

```

    CS1
    turn ← 2
    C1 ← true
    program1
end while
end Program1

```

```

    end while
    exitflag ← true
end while

```

```

if exitflag = true then
    continue
end if
CS2
turn ← 1
C2 ← true
program2
end while
end Program2

```

Переменные $C1$ и $C2$ гарантируют, что взаимное выполнение не может возникнуть; переменная $turn$ гарантирует невозможность взаимного блокирования.

Семафорные примитивы. Программное решение проблемы КС имеет следующие недостатки:

- решение запутанное (неясное);
- в то время, когда один процесс находится в своей КС, другой впустую циклится и при этом проверяет общие переменные. Чтобы сделать это, ожидающий процессор должен “красть” циклы памяти у активного процессора. Результатом является общее замедление системы процессами, которые не выполняют никакой полезной работы.

В 1965 г. Дейкстра ввел два новых примитива, которые значительно упростили взаимосвязь и синхронизацию процессов [10]. Эти примитивы, обозначаемые P и V , оперируют целыми неотрицательными переменными, называемыми *семафорами*.

Пусть S — семафор. Операция $V(S)$ увеличивает переменную S на 1 (инкремент) одним неделимым действием, т. е. выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессорам во время операции $V(S)$.

Операция $P(S)$ уменьшает S на 1 (декремент), если это возможно.

Если $S = 0$, то невозможно уменьшить S и остаться при этом в области целых неотрицательных значений S ; в этом случае процесс, вызывающий P -операцию, ждет, пока уменьшение S не станет возможным. Успешная проверка и уменьшение S образуют неделимую операцию.

Если несколько процессов одновременно запрашивают P — или V -операции над одним и тем же семафором, то эти операции будут выполняться последовательно в произвольном порядке; аналогично, если более одного процесса

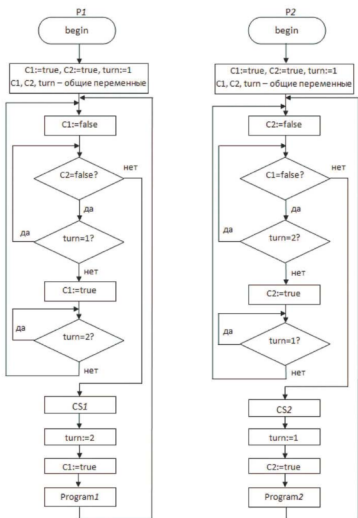


Рис. 6. 8. Алгоритм Деккера

ожидает при выполнении P -операции и изменяемый семафор становится положительным, то конкретный ожидающий процесс, который выбирается для завершения операции, произволен.

Семафорные переменные используются для синхронизации процессов. P -примитив включает в себе потенциальное ожидание вызывающих процессов, т. е. обращение к $P(S)$ может привести к блокированию вызывающего процесса, если $S = 0$. В то же время V -примитив может, возможно, активизировать некоторый ожидающий процесс. Неделимость P - и V -операций гарантирует

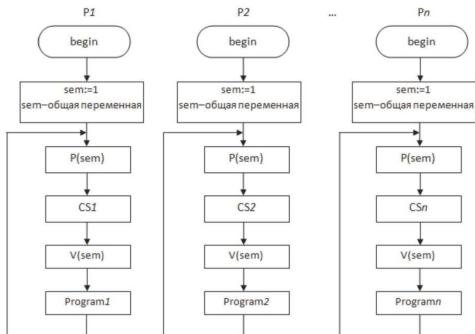


Рис. 6. 9. Решение проблемы критической секции применением семафора

целостность значений семафоров.

Семафорные операции дают простое и непосредственное решение проблемы критической секции. Пусть sem — семафор, используемый для защиты КС. Программным решением проблемы для n процессов является (рис. 6. 9):

```

Program P1
//sem — общая
//переменная
sem ← 1
while true
    P(sem)
    CS1
    V(sem)
    program1
end while
end Program P1

```

```

Program P2
//sem — общая
//переменная
sem ← 1
while true
    P(sem)
    CS2
    V(sem)
    program2
end while
end Program P2

```

```

Program Pn
//sem — общая
//переменная
sem ← 1
while true
    P(sem)
    CSn
    V(sem)
    programn
end while
end Program Pn

```

Значение sem равно 0, когда некоторый процесс находится в своей КС;

иначе $sem=1$. Взаимное исключение гарантировано, т. к. только один процесс может уменьшить sem до нуля с помощью P -операции; все другие процессы, пытающиеся войти в свои критические секции, когда $sem=0$, будут вынуждены ожидать в вызове $P(sem)$. Взаимное блокирование невозможно, т. к. одновременные попытки войти в КС, когда $sem=1$, должны, по определению, преобразоваться в последовательные P -операции.

Впоследствии для синхронизации процессов был введен примитив *мьютекс*, представляющий собой упрощенный вариант семафора. В отличие от семафора, мьютекс может принимать только 2 значения (занят/свободен).

Синхронизация в мультипроцессорах. Рассмотрим теперь более детально, как синхронизация центральных процессоров работает в мультипроцессоре [23].

Если процесс на однопроцессорной системе обращается к системному вызову, который требует доступа к некой критической таблице, программа ядра может просто запретить прерывания, прежде чем обратиться к таблице. Однако на мультипроцессоре запрет прерываний повлияет только на один центральный процессор, выполнивший данную команду. Остальные центральные процессоры продолжают свою работу и смогут получить доступ к критической таблице.

Требуется специальный мьютекс-протокол, который будет выполняться всеми центральными процессорами, чтобы гарантировать работу взаимного исключения.

Сердцем любого практического мьютекс-протокола является команда **TSL** - Test and Set Lock процессора, позволяющая исследовать и изменить слово в памяти за одну операцию. Как уже было сказано, эта команда считывает слово памяти в регистр процессора. Одновременно она записывает 1 (или другое ненулевое значение) в слово памяти. Конечно, для выполнения операций чтения и записи памяти требуется два цикла шины. На однопроцессорной машине, поскольку одна команда процессора не может быть прервана на полпути, команда TSL работает должным образом.

Теперь рассмотрим возможный вариант работы на мультипроцессоре. На рис. 6.10 показан наихудший случай совпадения по времени обращений к слову памяти по адресу 1000, начальное состояние которого 0. На шаге 1 процессор 1 считывает слово и получает 0. На шаге 2, прежде чем процессор 1 успевает изменить это слово, процессор 2 также считывает слово и тоже получает 0. На шаге 3 процессор 1 записывает в это слово 1. На шаге 4 процессор 2 также записывает в это слово 1. Оба процессора получили от команды TSL 0,

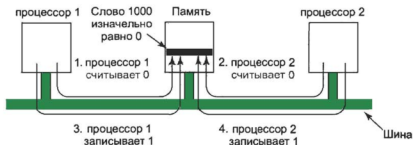


Рис. 6. 10. Неверная работа команды TSL в случае, если не заблокирована шина

поэтому оба считают себя вправе получить доступ к критической области. Таким образом, взаимное исключение не срабатывает.

Для разрешения этой проблемы команда TSL сначала должна блокировать шину, не допуская обращения к ней других центральных процессоров, затем выполнить оба обращения к памяти, после чего разблокировать шину. Как правило, для блокировки шины сначала выполняется обычный запрос шины по стандартному протоколу, затем устанавливается в 1 некая специальная линия шины. Пока эта линия шины установлена в 1, никакой другой центральный процессор не может получить к ней доступ. Такая команда может быть выполнена только на шине, у которой есть необходимые специальные линии и (аппаратный) протокол для их использования. Современные шины обладают подобными свойствами.

Если команда TSL корректно реализована и применяется, она гарантирует правильную работу взаимного исключения. Однако подобный способ реализации взаимного исключения использует *спин-блокировку*, так как запрашивающий процессор находится в цикле опроса блокировки с максимальной скоростью. Этот метод является не только тратой времени запрашивающего процессора (или процессоров), но он также может накладывать значительную нагрузку на шину или память, существенно снижая скорость работы всех остальных процессоров, пытающихся выполнять свою обычную работу.

На первый взгляд может показаться, что наличие кэширования должно устранить проблему конкуренции за шину, но это не так. Теоретически, как только запрашивающий центральный процессор прочитал слово блокировки, он должен получить его копию в свой кэш. Пока ни один другой центральный процессор не предпринимает попыток использовать это слово, запрашивающий центральный процессор может работать с ним в своем кэше. Когда

центральный процессор, владеющий словом блокировки, записывает в него 1, протокол кэша автоматически помечает как недействительные все копии этого слова в удаленных кэшах, требуя получения правильных значений.

Проблема заключается в том, что кэш оперирует блоками по 32 или 64 байта.

Обычно слова, окружающие слово блокировки, нужны центральному процессору, удерживающему это слово. Поскольку команда TSL представляет собой запись (так как она модифицирует слово блокировки), ей требуется монопольный доступ к блоку кэша, содержащему слово блокировки. Таким образом, каждая команда TSL помечает блок кэша владельца блокировки как недействительный и получает приватную, эксклюзивную копию для запрашивающего центрального процессора. Как только владелец блокировки изменит слово, соседнее с блокировкой, блок кэша перемещается на его машину. В результате весь блок кэша, содержащий слово блокировки, постоянно перемещается от центрального процессора, удерживающего блокировку, к центральному процессору, пытающемуся ее получить. Это создает довольно значительный и совершенно излишний трафик шины.

Проблему можно было бы решить, если бы удалось избавиться от всех операций записи, вызванных командой TSL запрашивающей стороны. Это можно сделать, если запрашивающая сторона сначала будет выполнять простую операцию чтения, чтобы убедиться, что мьютекс свободен. Только убедившись, что он свободен, центральный процессор выполняет команду TSL, чтобы захватить его. В результате большинство операций опроса представляют собой операции чтения, а не операции записи. Когда мьютекс считан, владелец выполняет операцию записи, для чего требуется монопольный доступ. При этом все остальные копии этого блока кэша объявляются недействительными. При следующей операции чтения запрашивающий центральный процессор перезагрузит этот блок кэша. Обратите внимание, что при одновременном споре двух центральных процессоров за один и тот же мьютекс может случиться, что они одновременно увидят, что мьютекс свободен, и одновременно выполнят команду TSL. Ничего страшного в этом случае не произойдет, так как выполнена будет только одна команда TSL и такая ситуация не приведет к состязанию. Если вы видите, что мьютекс свободен, и тут же пытаетесь его схватить командой TSL, то нет никакой гарантии успеха данного действия. Мьютекс может успеть захватить кто-либо другой.

Другой способ снижения шинного трафика заключается в использовании алгоритма двоичного экспоненциального отката, применяемого в сети Ethernet.

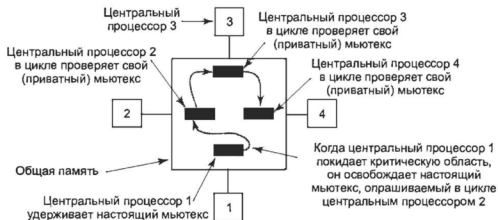


Рис. 6. 11. Использование нескольких мьютексов во избежание "пробуксовки" кэша

Вместо постоянного опроса, между опросами может быть вставлен цикл задержки. Вначале задержка представляет собой одну команду. Если мьютекс занят, задержка удваивается, учетверяется и т. д. до некоторого максимального уровня. При низком уровне мы получим быструю реакцию при освобождении мьютекса, зато потребуются больше обращений к шине для чтения блока кэша. Высокий максимальный уровень позволит уменьшить число лишних обращений к шине за счет более медленной реакции программы на освобождение мьютекса. Использование алгоритма двоичного экспоненциального отката не зависит от применения операций чистого чтения перед командой TSL.

Еще более удачная идея заключается в том, чтобы каждому центральному процессору, желающему заполучить мьютекс, позволить опрашивать свою собственную переменную блокировки. Во избежание конфликтов переменная должна располагаться в неиспользуемом для других целей блоке кэша. Работа алгоритма состоит в том, что центральный процессор, которому не удастся получить мьютекс, захватывает переменную блокировки и присоединяется к концу списка центральных процессоров, ожидающих освобождения мьютекса. Когда центральный процессор, удерживающий мьютекс, покидает критическую область, он освобождает приватную переменную, проверяемую первым центральным процессором в списке (в его собственном кэше). Тем самым следующий центральный процессор получает возможность войти в критическую область. Покинув критическую область, этот процессор освобождает переменную блокировки, тестируемую следующим процессором и т. д. (рис. 6. 11). Хотя такой протокол несколько сложен (это необходимо, чтобы

не допустить одновременного присоединения двух центральных процессоров к концу списка), он достаточно эффективен.

6.7. Планирование процессов. Многозадачность

Мультипрограммирование, или *многозадачность* — это такой вариант организации вычислительного процесса, при котором на одном процессоре попеременно выполняются сразу несколько программ [16].

Попеременно выполняемые программы совместно используют не только процессор, но и другие ресурсы компьютера: оперативную и внешнюю память, устройства ввода-вывода, данные. Многозадачность призвана повысить эффективность использования вычислительной системы.

Работа по определению того, в какой момент необходимо прервать выполнение текущего активного процесса и какому процессу предоставить возможность выполняться, называется *планированием*.

Рассмотрим проблемы, связанные с определением того, когда следует выделять процессоры и каким именно процессам.

Уровни планирования. Планирование на верхнем уровне (*планирование заданий*): определяется, какие задания будут допущены в систему. Вошедшие в систему задания становятся процессами или группами процессов.

Планирование на промежуточном уровне (*планирование нагрузки*): определяется, каким процессам будет разрешено состязаться за захват центрального процессора, отслеживаются колебания системной нагрузки (кратковременно приостанавливаются и вновь активизируются процессы).

Планирование на нижнем уровне (*диспетчирование процессов*): определяет, какому из готовых к выполнению процессов будет предоставлен освободившийся ЦП, и выделяет этот ЦП (запуск процесса). Планирование на нижнем уровне производится диспетчером, который обязательно располагается в основной памяти.

Дисциплина планирования должна:

- относиться ко всем процессам одинаково (не должна возникать проблема “ливлока”, то есть ни один процесс не должен пострадать из-за бесконечного откладывания);
- обеспечить максимальную пропускную способность системы;
- обеспечить для максимального числа пользователей приемлемые времена ответа;
- быть предсказуемой. Время выполнения задания не должно зависеть от нагрузки на систему;

- сбалансировать использование ресурсов (предпочтение отдается тем процессам, которые используют недогруженные ресурсы);
- учитывать приоритеты.

Механизм планирования должен учитывать следующие факторы:

- лимитируется ли процесс вводом-выводом;
- лимитируется ли процесс центральным процессором;
- является ли процесс пакетным или диалоговым;
- насколько часто при выполнении процесса возникают прерывания по отсутствию в оперативной памяти нужных страниц;
- сколько времени уже использовал данный процесс;
- сколько еще времени требуется данному процессу для завершения.

Вытесняющая и не вытесняющая многозадачности. С самых общих позиций все множество алгоритмов планирования (и, соответственно, типов многозадачности) можно разделить на два класса: вытесняющие и не вытесняющие.

- *Не вытесняющие* (non-preemptive) алгоритмы основаны на том, что активному потоку позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление операционной системе, чтобы та выбрала из очереди другой готовый к выполнению поток.
- *Вытесняющие* (preemptive) алгоритмы — это такие алгоритмы планирования потоков, в которых решение о переключении процессора с выполнения одного потока на выполнение другого принимается операционной системой, а не активной задачей.

Основным различием между вытесняющими и не вытесняющими алгоритмами является степень централизации механизма планирования потоков. При вытесняющем мультипрограммировании функции планирования потоков целиком сосредоточены в операционной системе, и программист пишет свое приложение, не заботясь о том, что оно будет выполняться одновременно с другими задачами. При этом операционная система выполняет следующие функции: определяет момент снятия с выполнения активного потока, запоминает его контекст, выбирает из очереди готовых потоков следующий, запускает новый поток на выполнение, загружая его контекст.

При не вытесняющем мультипрограммировании механизм планирования распределен между операционной системой и прикладными программами. Прикладная программа, получив управление от операционной системы, сама определяет момент завершения очередного цикла своего выполнения и только затем передает управление ОС с помощью какого-либо системного вызова. ОС формирует очереди потоков и выбирает в соответствии с некоторым правилом

(например, с учетом приоритетов) следующий поток на выполнение. Такой механизм создает проблемы, как для пользователей, так и для разработчиков приложений.

Интервальный таймер. Чтобы не допускать монополизации системы пользователями, в ОС предусмотрены механизмы, позволяющие отбирать ЦП у пользователя. Операционная система устанавливает интервальный таймер с целью генерации сигнала прерывания в некоторый конкретный момент времени в будущем. После прерывания процессор передается в распоряжение следующего процесса.

Временные прерывания помогают гарантировать приемлемые времена ответа для пользователей, работающих в диалоговом режиме, предотвращают зависание системы из-за заикливания какой-то программы пользователя, а также позволяют процессам соответствующим образом реагировать на события, зависящие от времени.

Приоритеты. Статические приоритеты не изменяются. Их легко реализовать, они сопряжены с небольшими издержками, но не реагируют на изменение обстановки.

Механизмы динамических приоритетов реагируют на изменение ситуации, но они сложны в реализации и связаны с большими издержками по сравнению со статическими схемами. Издержки оправдываются повышением реактивности системы.

Планирование по сроку завершения. Определенные задания должны завершаться к определенному сроку, иначе теряется их смысл.

Планирование по сроку завершения сложно организовать по многим причинам:

- редко имеется информация о ресурсах, необходимых для выполнения задания;
- не должен серьезно измениться уровень обслуживания для других пользователей;
- трудно распределять загрузку ресурсов в условиях непредсказуемых требований к системе со стороны заданий;
- если требуется соблюсти сроки для нескольких заданий, то требуются специальные методы оптимизации, гарантирующие соблюдение всех сроков;
- существенные накладные расходы, ухудшение обслуживания других пользователей.

Планирование по принципу FIFO. Процессы планируются по принципу обычной очереди — FIFO.

Длинные задания заставляют ждать короткие задания. Невелики колебания времен ответа, большая предсказуемость, чем у других дисциплин обслуживания. Не рекомендуется для интерактивной работы, т. к. не гарантирует приемлемые времена ответов.

Во многих схемах планирования предусматривается диспетчирование процессов в соответствии с их приоритетами, однако, процессы одного уровня приоритета диспетчируются по принципу FIFO.

Циклическое планирование. Диспетчирование процессов осуществляется по FIFO, но каждому процессу предоставляется ограниченное количество времени ЦП, называемое временным квантом. Процесс, у которого перехватили ЦП, переходит в конец списка готовых к выполнению процессов.

Данная дисциплина, именуемая **RR** (Round Robin), хороша для интерактивной работы. Накладные расходы снижаются за счет эффективных механизмов контекстного переключения и благодаря предоставлению достаточного объема основной памяти, чтобы процессы могли размещаться в ней одновременно.

Если квант велик, то RR вырождается в FIFO. Если квант очень мал, то ОС занимается только переключениями (велики накладные расходы). Чем характеризуется оптимальное значение кванта? Он достаточно велик, так что подавляющее большинство интерактивных запросов требует для своего обслуживания меньшего времени, чем длительность кванта. Величина оптимального кванта меняется от системы к системе, от процесса к процессу и зависит от нагрузки.

Существуют также и другие стратегии планирования процессов. К таковым относятся: стратегии “кратчайшее задание — первым”, “по наименьшему остающемуся времени”, “по наименьшему относительному времени реакции” и т. д.

6.8. Планирование мультипроцессора

На однопроцессорной системе планирование одномерно. Единственный вопрос, на который должен быть каждый раз получен ответ: “какой процесс должен быть запущен следующим?” На мультипроцессоре планирование двумерно. Планировщик должен решить, какой процесс и на каком процессоре запустить.

Это дополнительное измерение существенно усложняет планирование на мультипроцессорах [23].

Другой усложняющий фактор состоит в том, что в некоторых системах все процессы являются независимыми, тогда как в других системах они

формируют группы. Примером первой ситуации является система реального времени, в которой независимые пользователи запускают независимые процессы. Эти процессы не связаны друг с другом, и планирование каждого из них не зависит от остальных процессов.

Пример второй ситуации часто встречается в среде разработки программ. Большие системы, как правило, состоят из некоторого количества заголовочных файлов, содержащих макросы, определения типов и объявления переменных, используемых в собственно файлах программы. При изменении заголовочного файла все программные файлы, в которые он включен, должны быть перекомпилированы. Для этого обычно используется программа `make`. При вызове программа `make` начинает компиляцию только тех файлов, которые должны быть перекомпилированы из-за изменений самих файлов или заголовочных файлов, включенных в эти файлы кода. Все еще имеющие силу объектные файлы заново не пересоздаются.

Оригинальная версия программы `make` выполняла свою работу последовательно, но новые версии, созданные для мультипроцессоров, могут начинать всю компиляцию одновременно. Если требуется откомпилировать 10 файлов, то нет смысла выполнить компиляцию 9 файлов быстро, а последний файл отложить на потом, так как пользователь будет считать работу выполненной только по завершении компиляции последнего файла. В этом случае есть смысл рассматривать все эти процессы как группу и учитывать это при их планировании.

Ниже описывается планирование независимых, а затем — зависимых процессов [23]. Простейший алгоритм планирования независимых процессов (или потоков) состоит в поддержании единой структуры данных для готовых процессов, возможно, просто списка, но скорее всего множества списков для процессов с различными приоритетами (рис. 6.12). Здесь все 16 ЦП в данный момент заняты, а 14 процессов с различными приоритетами ожидают запуска. Первым заканчивает работу (или его процесс блокируется) процессор 4. При этом он блокирует очередь планирования и выбирает из нее процесс с наименьшим приоритетом, то есть процесс А. Затем освобождается центральный процессор 12 и выбирается процесс В. Пока эти процессы независимы, подобное планирование представляет собой разумный выбор.

Наличие единой структуры данных планирования, используемой всеми ЦП, обеспечивает процессорам режим разделения времени, подобно тому, как это выполняется на однопроцессорной системе. Кроме того, такая организация позволяет автоматически балансировать нагрузку, то есть она исключает ситуацию, при которой один ЦП простаивает, в то время как другие процессоры

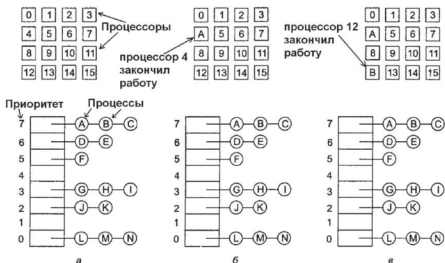


Рис. 6. 12. Использование единой структуры данных для планирования мультипроцессора

перегружены. Два недостатка такой схемы представляют собой потенциальный рост конкуренции за структуру данных планирования по мере увеличения числа процессоров и обычные накладные расходы на выполнение переключения контекста, когда процесс блокируется, ожидая выполнения операции ввода-вывода.

Переключение контекста также может случиться, когда истекает квант процесса. Мультипроцессор обладает определенными свойствами, которыми не обладают однопроцессорные системы. Предположим, что процесс удерживает спин-блокировку, что часто встречается на мультипроцессорах. Другие ЦП, ждущие освобождения блокировки, просто теряют время в циклах ожидания, пока этот процесс не будет запущен снова и не отпустит блокировку. На однопроцессорных системах спин-блокировка применяется редко. Поэтому если процесс, удерживающий мьютекс, блокируется и запускается другой процесс, то при попытке получить мьютекс второй процесс будет тут же блокирован, и много времени потеряно не будет.

Чтобы решить данную проблему, в некоторых системах применяется умное планирование, в котором процесс, захватывающий спин-блокировку, устанавливает флаг, демонстрирующий, что он в данный момент обладает спин-блокировкой. Когда процесс освобождает блокировку, он также очищает и флаг. Таким образом, планировщик не останавливает процесс, удерживающий спин-блокировку, а, напротив, дает ему еще немного времени, чтобы тот

завершил выполнение критической области и отпустил мьютекс.

Другая проблема, играющая важную роль в планировании, заключается в следующем [23]. Когда процесс A достаточно долго работает на процессоре k , кэш процессора k будет полон блоками процесса A . Если процесс A должен быть снова вскоре запущен, его производительность может оказаться выше, если он будет запущен снова на процессоре k , так как кэш k может все еще содержать некоторые блоки процесса A .

Наличие блоков в кэше увеличит частоту кэш-попаданий и, таким образом, увеличит скорость выполнения процесса. Кроме того, TLB (Translation Lookaside Buffer — буфер сверхоперативной памяти процессора, используемый для ускорения трансляции адреса виртуальной памяти в адрес физической памяти) также может содержать правильные страницы, что снизит количество прерываний из-за ошибок TLB.

Некоторые мультипроцессоры учитывают данные соображения и используют так называемое родственное планирование. Основная идея данного метода заключается в приложении серьезных усилий для того, чтобы процесс был запущен на том же центральном процессоре, что и в прошлый раз. Один из способов реализации этого метода состоит в использовании двухуровневого алгоритма планирования. В момент создания процесс назначается конкретному центральному процессору, например, наименее загруженному в данный момент. Это назначение процессов процессорам представляет собой верхний уровень алгоритма. В результате, каждый центральный процессор получает свой набор процессов.

Действительное планирование процессов находится на нижнем уровне алгоритма. Оно выполняется отдельно каждым ЦП при помощи приоритетов или других средств. Старания удерживать процессы на одном и том же центральном процессоре максимизируют родственность кэша. Однако если у какого-либо процессора нет работы, у загруженного работой процессора отнимается процесс и отдается ему.

Двухуровневое планирование обладает тремя преимуществами. Во-первых, оно достаточно равномерно распределяет нагрузку среди имеющихся ЦП. Во-вторых, двухуровневое планирование по возможности использует преимущество родственности кэша. В-третьих, поскольку у каждого процессора при таком варианте планирования есть свой собственный список свободных процессов, конкуренция за списки свободных процессов минимизируется, так как попытки использования списка другого процессора происходят относительно нечасто.

6.9. Файловые системы

Всем компьютерным приложениям нужно хранить и получать информацию. Во время работы процесс может хранить ограниченное количество данных в собственном адресном пространстве. Однако емкость такого хранилища ограничена размерами виртуального адресного пространства. Для некоторых приложений такого размера вполне достаточно, но для других, например систем резервирования авиабилетов, систем банковского или корпоративного учета, одного только виртуального адресного пространства будет недостаточно.

Кроме того, после завершения работы процесса, информация, хранящаяся в его адресном пространстве, теряется. Для большинства приложений (например, баз данных) эта информация должна храниться неделями, месяцами или даже вообще никогда не должна быть потеряна.

Исчезновение данных после завершения работы процесса для таких приложений неприемлемо. Информация должна сохраняться даже при аварийном завершении процесса в случае сбоя компьютера.

Третья проблема состоит в том, что часто возникает необходимость нескольким процессам одновременно получить доступ к одним и тем же данным (или части данных). Если интерактивный телефонный справочник будет храниться в адресном пространстве одного процесса, то доступ к нему будет только у этого процесса. Для решения этой проблемы необходимо отделить информацию от процесса.

Таким образом, к долговременным устройствам хранения информации предъявляются три следующих важных требования:

- устройства должны позволять хранить гигантские объемы данных;
- информация должна сохраняться после прекращения работы процесса, использующего ее;
- несколько процессов должны иметь возможность получения одновременного доступа к информации.

Обычное решение всех этих проблем состоит в хранении информации на дисках и других внешних хранилищах в модулях, называемых файлами. Процессы могут по мере надобности читать их и создавать новые файлы. Информация, хранящаяся в файлах, должна обладать устойчивостью, то есть на нее не должны оказывать влияния создание или прекращение работы какого-либо процесса. Файл должен исчезать только тогда, когда его владелец дает команду удаления файла. Файлами управляет операционная система. Их структура, именование, использование, защита, реализация и доступ к ним являются

важными пунктами устройства операционной системы. Часть операционной системы, работающая с файлами, называется *файловой системой*.

Файловая система является важной компонентой ОС и содержит [25]:

- методы доступа, которые определяют конкретную организацию доступа к данным, хранящимся в файлах;
- средства управления файлами, которые обеспечивают хранение файлов, обращение к ним, коллективное использование и защиту;
- средства управления внешней памятью, которые обеспечивают распределение пространства внешней памяти для размещения файлов;
- средства обеспечения целостности файлов, которые гарантируют сохранность информации файла.

Рассмотрим задачи файловой системы и конкретные методы их решения.

Проблема выделения и освобождения места в дисковой памяти напоминает проблему распределения основной памяти при мультипрограммировании с переменными разделами (фрагментация, необходимость сборки мусора). Сборка мусора обычно производится в нерабочее время, когда систему можно закрыть для пользователей. В некоторых системах динамическая сборка мусора производится без прекращения работы (реорганизуются либо файлы, которые в данный момент не работают, либо файлы, к которым длительное время не было обращений).

При выделении памяти для хранения файла используется концепция локальности: данные файлы должны храниться как можно более компактно, чтобы обеспечить минимум времени доступа к его записям. Желательно хранить логически смежные страницы виртуальной памяти пользователя как физически смежные во внешней памяти.

При *связном распределении памяти* каждому файлу выделяется непрерывная область памяти. Пользователь заранее указывает размер области, необходимой для размещения создаваемого файла. Если нужной непрерывной области нет, то файл создать невозможно.

Достоинства связного распределения памяти:

- последовательные логические записи располагаются, как правило, рядом — это позволяет повысить скорость доступа;
- просто реализуются директории: для каждого файла достаточно задать начальный адрес и длину файла.

Недостатки:

- фрагментация, возникающая при уничтожении файлов; может потребоваться периодическое уплотнение памяти;

- если размер файла со временем меняется, то связанное распределение может оказаться нерациональным. Для обеспечения расширения файла приходится завышать размеры требуемых областей. Если файл перестает помещаться в выделенную область, его необходимо переписать в новую достаточно большую область.

В настоящее время больше распространены системы с несвязным распределением внешней памяти.

При распределении при помощи *списка секторов* дисковая память рассматривается как набор секторов [25]. Файлы состоят из секторов, которые могут находиться в различных местах дисковой памяти. Секторы, принадлежащие одному файлу, содержат указатели друг на друга и образуют список. В списке свободного пространства находятся все свободные секторы дисковой памяти. При увеличении размера файла процесс запрашивает дополнительное количество секторов из числа свободных. Освободившиеся секторы возвращаются в список свободного пространства. Проблемы уплотнения свободной памяти не возникает.

Недостатки:

- выборка логически смежных записей может быть сопряжена с длительными поисками;
- велики накладные расходы времени на обработку указателей;
- хранение указателей приводит к уменьшению объема полезной памяти.

Вариант *поблочного распределения* предполагает, что память распределяется блоками смежных секторов (экстендами или кластерами). При выделении новых блоков система стремится выбирать новые блоки как можно ближе к уже существующим блокам файла. При обращении к файлу вначале определяется блок, а затем — сектор в рамках этого блока. Существует несколько способов реализации систем поблочного распределения. Рассмотрим два из них:

- при помощи цепочек блоков;
- при помощи индексных блоков.

Схема с цепочкой блоков. Обычно размер блока равен размеру полной дорожки дискового накопителя. Процесс просмотра цепочки блоков может занимать много времени, затрачиваемого на перемещение головок. Однако, легко производить вставки и исключения блоков (модификации указателей). В некоторых списках применяется двойное связывание.

Схема с цепочкой индексов. Указатели записей помещаются в отдельные индексные блоки. Каждый индексный блок содержит фиксированное количество

элементов. Каждая строка-статья индексного блока содержит идентификатор записи и указатель на эту запись. Если для описания файла требуется более, чем один индексный блок, то организуется цепочка индексных блоков. Преимущество данного способа в том, что для поиска записи достаточно просмотреть индексные блоки. Индексные блоки размещаются рядом друг с другом во внешней памяти, иногда размещаются в оперативной памяти. Недостаток: для вставки дополнительных записей может понадобиться полная перестройка структуры индексных блоков.

Каждый файл обладает своим *дескриптором* (блоком управления) — это блок, содержащий информацию для выполнения различных операций с файлом (сильно зависит от конкретной системы).

Типичный дескриптор содержит следующую информацию:

- имя файла;
- размещение его на внешнем устройстве;
- тип организации файла (последовательная, индексно-последовательная и т. п.);
- тип устройства;
- данные для управления доступом;
- тип файла (данные, объектная программа, исходная программа и т. п.);
- дата и время создания;
- дата и время последней модификации;
- счетчики активности доступа (например, количество обращений по чтению).

Дескриптор файла передается с внешнего устройства в оперативную память только после открытия соответствующего файла. Он управляется файловой системой (пользователь не может к нему обращаться).

В случае многопользовательской ОС необходимо решать задачу управления доступом к файлам. Для этой цели может быть использована матрица управления доступом. Каждая строка этой матрицы соответствует некоторому пользователю, а каждый столбец — некоторому файлу. Пользователю с номером i разрешен доступ к файлу с номером j , если элемент матрицы управления доступом, находящийся на пересечении строки i и столбца j , равен единице, в противном случае этот элемент равен нулю.

В системе с большим количеством пользователей и файлов матрица управления доступом велика и очень разрежена, поэтому ее использование в этом случае не является эффективным.

В таких системах для управления доступом все пользователи подразделяются на классы, каждый из которых имеет определенные права доступа. Наивысшими правами на доступ к файлу обладает его создатель (класс

“Владелец”). Владелец файла определяет класс пользователей, которым разрешено использование файла (но не его модификации!). Такой класс называется классом “допущенных пользователей”.

Для работы над общим проектом создается класс пользователей, получивший наименование “Группа”. Пользователи “Группы” обладают одинаковыми правами доступа к некоторому общему множеству файлов.

Если файл доступен любому пользователю, то он характеризуется как “Общедоступный”. Обычно это означает, что пользователям разрешается читать и выполнять файл, а производить запись запрещается.

Вопросы для самоконтроля

1. Дайте определение понятию “операционная система”.
2. Какие основные задачи решает ОС?
3. Какие основные этапы развития операционных систем можно выделить?
4. Что такое “ОС пакетной обработки”?
5. На какие классы делились многозадачные ОС 2-го поколения?
6. Какие уровни иерархии можно выделить при описании структуры современных ОС?
7. Поясните значение термина “ядро ОС”.
8. Назовите наиболее важные подсистемы в ОС автономного компьютера.
9. Какие дополнительные функции несёт на себе сетевая ОС по сравнению с ОС автономного компьютера?
10. В чем состоит идея построения распределенной ОС?
11. Какие требования предъявляются к современным ОС?
12. Назовите основные модели, используемые при построении мультипроцессорных ОС.
13. Что такое ОС мультипроцессорных ЭВМ типа “master-slave”?
14. Какой принцип работы положен в основу ОС симметричных мультипроцессоров?
15. Какие механизмы синхронизации процессов и потоков в ОС можете назвать?

16. Как программным методом можно запретить одновременный доступ к критической секции нескольким процессам?
17. Какими преимуществами обладает механизм синхронизации процессов и потоков при помощи семафоров по сравнению с программным решением при помощи общих переменных?
18. Что такое “вытесняющая” и “невывтесняющая” многозадачности?
19. Поясните значение термина “планирование процессов”.
20. Какие уровни планирования процессов применяются в современных ОС?
21. Поясните термин “файловая система”.

Глава 7. Прикладное программное обеспечение

7.1. Кластерное ПО

Кластер — слабо интегрированная архитектура, как на уровне аппаратуры, так и на уровне операционной системы. Из-за отсутствия общей памяти на каждый процессорный узел должна быть установлена своя копия ОС, поэтому для работы с кластерными комплексами нужно решить проблемы обслуживания и управления, которые уже при нескольких десятках процессоров становятся критическими. Перечисленные ниже программные средства позволяют справиться с проблемой “большой системы”.

Средства для установки ПО на кластер. Клонирование образа является одним из самых распространенных способов установки ПО на набор вычислительных машин (как однородный, так и гетерогенный) [63]. Сначала вручную создается (инсталлируется) прообраз машины, называемый “золотым клиентом”, после этого клонируется тем или иным способом на вычислительные узлы (к примеру — файловыми системами командой `dd`). Так как готовые файловые системы могут быть скопированы на диск быстрее, чем с помощью пофайлового копирования, то этот метод является самым быстрым вариантом инсталляции. Тем не менее, подход обладает несколькими недостатками: во-первых, каждый узел вычислительного комплекса должен обладать приводом дисков, а во-вторых, настройка установленного ПО все равно должна производиться вручную.

В связи с этим все большее распространение приобретают программные системы, производящие клонирование образов по сети и настройку необходимого ПО при помощи сценариев, выполняемых до или после инсталляции.

При помощи данных программных средств решаются следующие задачи:

- создание репозитория (набора ПО для кластера), обновление его версий;
- распределение ПО по узлам кластера;

- автоматическая установка ПО на кластерных узлах и их конфигурирование.

В качестве характерного примера таких средств можно назвать SystemImager, обеспечивающий установку ОС, конфигурацию и обслуживание для больших сетей, предпочтительно с подобными или одинаковыми аппаратными средствами, для дистрибутивов GNU/Linux. Целевая станция может загружаться с гибкого диска, CD-ROM или **PXE** (Preboot Execution Environment — стандарт, предложенный для обеспечения возможности сетевой загрузки ОС). “Золотой клиент” должен быть вручную установлен и сконфигурирован. Его файловая система после этого переносится на сервер, и именно с него устанавливаются целевые машины. Если “золотой клиент” модифицирован, эти изменения размножаются клиентам при помощи rsync (программа для UNIX-подобных систем, которая выполняет синхронизацию файлов и каталогов в двух местах с минимизированием трафика, используя кодировку данных при необходимости). SystemImager работает с устанавливаемым ПО на уровне файлов, а не побитово.

Другим примером может служить **LCFG** (Local ConFiGuration system)— свободно распространяемая система для автоматизации установки и конфигурирования разных версий Unix. LCFG обладает хорошей производительностью и подходит для кластеров с разнообразными и часто меняющимися конфигурациями узлов.

Средства регулярного администрирования кластеров. Вычислительный кластер зачастую является ресурсом, который использует большое множество пользователей из нескольких организаций, научно-исследовательских институтов и др. Поэтому в большинстве случаев кластер загружен задачами круглосуточно. В связи с этим крайне важно постоянно следить за состоянием аппаратной и программной составляющих вычислительного ресурса. Для этого существуют программные средства, выполняющие рассмотренные ниже задачи.

- *Мониторинг аппаратуры* (температура процессора, работа вентилятора, состояние блока питания, и т.п.).
- SNMP-агенты от производителей (**SNMP** — Simple Network Management Protocol — технология, призванная обеспечить управление и контроль за устройствами и приложениями в сети связи путём обмена управляющей информацией между агентами, располагающимися на сетевых устройствах, и менеджерами, расположенными на станциях управления).

- **LMSENSORS** — программные модули (сенсоры) мониторинга.
- *Мониторинг кластеров с автоматизацией реакций на критические события.*
- **Big Brother** — инструмент для мониторинга в реальном времени на базе Web-технологий;
- **Mon** — система для мониторинга программных служб и генерации сигнала тревоги при их неработоспособности. Mon легко расширяется и дополняется любыми средствами мониторинга через общий интерфейс.
- *Мониторинг производительности кластерной системы* (визуализация загрузки всех узлов системы в реальном времени).
- **Ganglia** — свободно распространяемая система, предназначенная для мониторинга вычислительных комплексов, распределенных как в локальной, так и в глобальной сетях. Использует оригинальный сетевой протокол и компактный способ хранения данных мониторинга, что делает ее эффективной, масштабируемой и надежной.
- **MRTG** (Multi Router Traffic Grapher) — свободно распространяемый инструмент для мониторинга загрузки сети. MRTG генерирует HTML страницы с визуализацией сетевого трафика. MRTG создан на базе Perl и Си, и работает в средах UNIX и Windows NT.
- **Возможность выполнения административных команд параллельно на группах кластерных узлов.**
- **C3** (Cluster Command Control) — свободно распространяемый продукт, представляющий собой набор инструментов для администрирования и поддержки приложений в кластерах с ОС Linux (разработка Oak Ridge National Laboratory). В его состав входят средства для параллельного запуска команд на все или группу кластерных узлов, сбор и распределение файлов, завершение процессов, удаленное отключение или перезапуск машин.
- **WebMin** — свободно распространяемый инструмент для системного администрирования в среде Unix на базе Web технологий. WebMin обеспечивает унифицированный интерфейс для выполнения административных действий на различных подмножествах узлов кластера (регистрация пользователей, настройка Apache, **DNS** (Domain Name System), **NFS** (Network File System) и т.д.).

Системы пакетной обработки. Вычислительные кластеры в большинстве своем используются в режиме пакетной обработки заданий [63]. Для этой цели разработаны программные продукты, называемые менеджерами ресурсов

(Resource Manager), кластерными системами управления (Cluster Management System) или **СПО** (системами пакетной обработки).

Известно более двадцати СПО, из которых наиболее популярны свободно распространяемые PBS, SGE и Condor, а также коммерческие LoadLeveler и LSF.

Использование СПО в кластере позволяет перейти от работы с индивидуальными компьютерами, распределенными в локальной сети, к работе с единым многопроцессорным вычислительным комплексом в режиме пакетной обработки заданий. Пользователь может помещать задания в общую для комплекса очередь, используя единый интерфейс для запуска, модификации, снятия и получения информации о заданиях. СПО автоматически распределяет задания по узлам с учетом их загрузки, выполняет и доставляет результаты пользователю. Все СПО имеют богатый настраиваемый набор средств для управления процессом обработки заданий.

Все СПО имеют примерно одинаковые наборы функций, но отличаются по отдельным характеристикам. Системы PBS и SGE примерно равны по своим возможностям и предполагают полное выделение машин под пакетную обработку. Ими поддерживаются сетевые кластеры с однопроцессорными машинами, а также мультипроцессоры SMP и MPP. Отдельные процессоры могут использоваться в монопольном режиме (только одна задача на процессор) или в режиме разделения времени (несколько заданий могут разделять процессор). В последнем варианте при распределении заданий контролируется уровень процессорной загрузки. PBS и SGE имеют аппарат запуска многопроцессорных заданий. В отличие от PBS в последних версиях системы SGE реализован механизм поддержки контрольных точек.

Ниже приведено краткое описание наиболее распространенных систем пакетной обработки [63]:

- **PBS** (Portable Batch System) обеспечивает управление заданиями в пакетном режиме в среде компьютеров с ОС Unix. Сегодня предлагается OpenPBS — свободно распространяемая версия и PBSPRO — расширенная коммерческая версия. К достоинствам системы PBS следует отнести наличие удобных средств оперативного контроля состояния очередей, состояния вычислительных узлов, а также наличие системы регистрации выполненных заданий [6].
- **Torque** (Terascale Open-source Resource and QUEue Manager) — новая версия PBS, разработанная компанией Cluster Resources, Inc. на основе OpenPBS. Система обладает рядом дополнительных усовершенствований: улучшена масштабируемость (работа в среде до 2500 узлов);

- повышена устойчивость к сбоям (внесены дополнительные проверки); усовершенствован интерфейс планировщика с целью его обеспечения дополнительной и более точной информацией; усовершенствована система записей в log-файлах.
- **SGE (Sun Grid Engine)**. Семейство из нескольких различных вариантов СПО. Sun Grid Engine — свободно распространяемая версия, предназначенная для управления ресурсами одного проекта или подразделения. Основана на полной централизации обслуживания ресурсов и пользователей. Sun Grid Engine реализуется открытым сообществом разработчиков и спонсируется Sun Microsystems.
 - **Condor**. Свободно распространяемый продукт. Может быть использован в кластере машин с управляющим узлом, который берет на себя часть вычислительной работы на время, пока он простаивает. В связи с этим Condor имеет несколько важных механизмов поддержки контрольных точек, рестарта и миграции заданий. В Condor развит весьма мощный язык описания ресурсов, позволяющий формально описать как требования к типам и объемам ресурсов со стороны задания, так и ограничения на доступ к ресурсам со стороны владельцев.
 - **LoadLeveler** — коммерческий программный продукт компании IBM, предназначенный для пакетной обработки последовательных и параллельных (многопроцессорных) заданий на вычислительных кластерах. Система обеспечивает средства для подготовки, запуска и слежения за заданиями в режиме пакетной обработки в гетерогенной сети компьютеров. Текущая версия LoadLeveler Version 3.1 дополнена рядом новых характеристик, таких как новые планировщики; механизм поддержки контрольных точек; интеграция с AIX WorkLoad Manager; включает примеры сценариев.
 - **LSF (Load Sharing Facility)** — коммерческая кластерная система компании Platform Computing Corporation для управления пакетной обработкой. Представляет собой полномасштабную систему, обеспечивающую все существенные требования к СПО: поддержку очередей заданий, сбор информации о наличии и занятости ресурсов в кластере, нахождение компьютеров с подходящими ресурсами для выполнения заданий, поддержку режима контрольных точек (checkpointing), миграцию заданий и др.
 - **MAUI** — внешний планировщик, который может использоваться взамен штатных планировщиков для нескольких СПО: PBS, SGE, Loadleveler, LSF, Wiki. MAUI — открытый продукт, который отличается большим набором режимов (политик) планирования и наличием механизма предварительного резервирования. MAUI представляет большой интерес в связи

с тем, что это единственная из свободно распространяемых СПО, способная обеспечивать автоматический запуск многопроцессорных заданий, избегая при этом неоправданного простоя ресурсов. Реализация основана на мощном алгоритме планирования Backfill.

Интегрированные программные средства для кластеров. Из вышесказанного можно сделать вывод, что для стабильной и эффективной работы вычислительного кластера необходимо множество программных систем, решающих различные задачи. Поэтому существуют пакеты программ, содержащие все необходимые средства для организации настройки и использования кластера.

OSCAR (Open Source Cluster Application Resource) — свободно распространяемый пакет инструментов для кластерных систем с ОС Linux. OSCAR включает в свой состав пакеты:

- **SIS** (System Installation Suite) — инструмент для автоматизации установки и конфигурирования ОС Linux на узлы сетевых кластеров. OSCAR использует этот инструмент для установки своего ПО.
- **C3**.
- **HDF5** (Hierarchical Data Format) — продукт, включающий спецификацию форматов данных и поддержку библиотек.
- **MPI-LAM** — свободно-распространяемая реализация стандарта MPI.
- **OPIUM** (OSCAR Password Installer and User Management) — средства для синхронизации учетных записей пользователей и конфигурирования ssh.
- **PBS**.
- **Pfilter** — пакет для настройки параметров, влияющих на коммуникации между кластерными машинами и внешней сетью.
- **PVM** (Parallel Virtual Machine) — программный пакет для поддержки параллельных вычислений в гетерогенных кластерах. Является альтернативой MPI.
- **Switcher** — пакет, позволяющий при запуске заданий устанавливать переменные среды исполнения.

ROCKS — свободно распространяемый пакет программ аналогичный OSCAR. Основная задача ROCKS — максимально упростить управление большими кластерными системами в среде Red Hat Linux. Эта задача решается путем:

- полной автоматизации процесса установки и конфигурирования гетерогенных узлов кластера;
- использования только пакетов RPM;
- использования масштабируемых сервисов (HTTP, DNS, NIS-подобных, и т. д.);

- механизма мониторинга производительности с использованием системы Ganglia.

7.2. GRID-системы

Понятие “GRID”. Несмотря на очень интенсивное увеличение производительности кластерных вычислительных систем и суперкомпьютеров, перед учеными стоят некоторые задачи моделирования природных явлений и процессов, для которых недостаточно быстродействия даже лидирующих в списке Top500 кластеров. Это, например, задачи моделирования взаимодействия тысяч различных молекул с молекулами белков для производства новых лекарственных препаратов; обработка данных, получаемых от ускорителей (например, Большого Адронного Коллайдера); расшифровка генома человека и т. д. К тому же количество операций с числами, выполняемых в единицу времени, — не единственная характеристика, которой недостаточно ученым в современных вычислительных кластерах. Огромные массивы данных для многих научных задач иногда находятся в разных организациях (возможно, размещенных в разных частях света). Например, физика высоких энергий скоро будет ежегодно производить около 10 Пбайт данных и тысячи физиков во многих университетах мира захотят анализировать эти данные.

В связи с этим у ученых появилась идея объединять ресурсы, территориально разбросанные по всему миру, в большие вычислительные среды. Такая среда может дать каждому клиенту, близкое к бесконечному, пространство памяти, вычислительной мощности, дискового объема — в общем, всех ресурсов, которые будут требоваться ученым по мере возникновения потребности в них. Эта идея стала именоваться “GRID” (грид).

Более формально цель создания GRID можно сформулировать следующим образом: “интеграция определенного множества пространственно распределенных ресурсов для того, чтобы обеспечить возможность выполнения широкого класса приложений на любой совокупности этих ресурсов, независимо от места их расположения”.

Разумеется, сегодня уже существует универсальное средство для обмена информацией — это Интернет. Но GRID призвана предоставлять возможность обмена не только информацией, а также вычислительными ресурсами и такими ресурсами, как базы данных, хранилища, различные приложения. Грид-среда должна представляться пользователю в виде “единого виртуального суперкомпьютера”, подключившись к которому, пользователю уже не нужно задумываться о том, где на самом деле хранятся и обрабатываются данные для

его задачи. Следует заметить, что термин GRID в переводе с английского означает “электрическая сеть”. Причина этого заключается в следующей аналогии: GRID должна позволять пользователям получать вычислительные мощности через сеть Интернет также просто и прозрачно, как мы получаем электрическую энергию из розетки.

Окончательно понятие GRID сформулировал Ян Фостер [27]. По мнению Фостера, это система, удовлетворяющая трем критериям:

- координирует использование ресурсов при отсутствии централизованного управления этими ресурсами (если это не так, мы имеем дело с локальной системой управления);
- использует стандартные, открытые, универсальные протоколы и интерфейсы (в основе многоцелевые протоколы и интерфейсы, позволяющие решать такие задачи как аутентификация, авторизация, обнаружение ресурсов и доступ к ресурсам);
- нетривиальным образом обеспечивает высококачественное обслуживание.

GRID сегодня. Те системы, которые используются в настоящее время и носят название “GRID” представляют собой так называемые “локальные GRID-системы”, в которых круг пользователей и доступные для использования ресурсы строго ограничены. Такие GRID могут образовываться, например, на базе набора ресурсов, принадлежащих различным коммерческим компаниям, организациям и университетам. Подобные системы хорошо подходят для задач, которые можно разделить на множество более мелких подзадач, причем отдельные подзадачи должны быть независимы между собой.

Некоторые характеристики локальных GRID:

- локальные вычислительные GRID являются масштабируемыми — при необходимости к ним возможно добавлять новые ресурсы. Обычно отдельные компьютеры соединяются между собой посредством локальной сети, хотя возможно и соединение через Интернет. В большинстве случаев компьютеры географически близки друг к другу и могут относиться к одному административному домену;
- круг пользователей локальных GRID строго определен и ограничен членами организации или нескольких организаций, хотя в ряде случаев и могут допускаться некоторые “сторонние” пользователи;
- локальные GRID состоят из специально выделенных для этого ресурсов, а основные ресурсы недоступны для распределенного компьютеринга.

Принцип работы большинства функционирующих сегодня GRID-систем описывается следующим алгоритмом:

- пользователь подключается к GRID с использованием специального программного интерфейса, который запускается на локальном компьютере;
- при помощи механизма сертификатов осуществляется процедура проверки безопасности;
- после подключения к GRID пользователь получает доступ к важнейшей компоненте любой GRID — “брокеру ресурсов” (также называемому диспетчером);
- брокер ресурсов находит в GRID подходящие для задачи ресурсы;
- брокер ресурсов обращается к “информационному сервису” для того, чтобы выяснить, какое программное и аппаратное обеспечение доступно в данный момент;
- также брокер ресурсов обращается к “каталогу реплик” для поиска нужной копии реплицированных данных;
- задание начинает выполняться;
- после завершения программы брокер ресурсов возвращает результат вычислений обратно пользователю.

Используемые при построении локальных GRID подходы можно разделить на 2 способа организации ресурсов [12]. Первый можно назвать двухуровневым (или горизонтально интегрированным). В этой форме GRID образуется из совокупности комплексных, то есть содержащих множество компьютеров, узлов. Ресурсы отдельного узла находятся в автономном административном домене, связаны локальной сетью и обычно управляются системой пакетной обработки, которая играет роль локального менеджера ресурсов. Узел включается в GRID через одну или несколько машин-шлюзов, на которые устанавливаются GRID-службы, и, таким образом, ресурсы узла становятся доступны повсеместно. На такой способ организации ресурсов ориентирован пакет Globus Toolkit, в котором поддерживаются интерфейсы PBS, Condor, LSF, SGE и др. Обратим внимание на то, что виртуализация ресурсов здесь происходит на уровне отдельных узлов GRID, а для обеспечения прозрачного доступа ко всем ресурсам GRID-системы предполагается наличие службы глобальной виртуализации — брокера или диспетчера (*рис. 7.1*).

Описанный подход хорош для GRID, которые опираются на развитую локальную инфраструктуру ресурсов. Однако представляют интерес и ситуации, когда владельцы ресурсов не используют системы пакетной обработки, не могут позволить себе усложнять обслуживание имеющегося парка машин или создают GRID на короткое время для совместного выполнения конкретного проекта. В таких обстоятельствах более подходящим может быть второй способ организации GRID — одноуровневый (или с вертикальной интеграцией).



Рис. 7. 1. Двухуровневый горизонтально-интегрированный грид

В одноуровневой архитектуре ресурсы — пространственно распределенные компьютеры — интегрируются через управляющий центр, который, с одной стороны, представляет собой точку доступа ко всем ресурсам, а, с другой стороны, выполняет функции менеджера ресурсов, управляя ресурсами и виртуализируя их.

Одноуровневый подход был предложен в проектах SETI@home и Distributed.net, но использовался для решения достаточно специальной по сравнению с целями GRID задачи — организации счета отдельных приложений на глобально распределенных ресурсах. Дальнейшее развитие одноуровневого подхода, выразившееся в появлении средств запуска и управления заданиями в продуктах нескольких компаний: Entropia, DataSynapse, United Devices, Parabon Computing, показало возможность его применения в GRID, то есть в среде для выполнения множества разных приложений. Следует отметить, что до недавнего времени существовало препятствие для создания GRID с помощью коммерческих систем: поскольку все они опирались на частные протоколы дистанционного взаимодействия, область действия построенных на их базе инфраструктур была ограничена корпоративным уровнем. Но положение меняется — перечисленные выше компании в той или иной степени принимали участие в разработке протоколов для архитектуры OGSA (Open Grid Service Architecture) и имеют конкретные планы перехода от частных решений к стандартам, утверждаемым GRID-сообществом.

В перспективе двухуровневый и одноуровневый подходы могут стать взаимодополняющими способами построения GRID, а некоторые технологии, разработанные под определенный контекст, представляют ценность и в более широком плане [12].

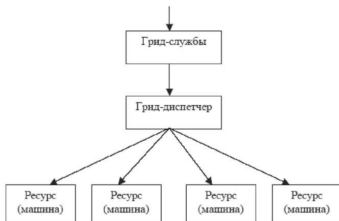


Рис. 7.2. Одноуровневый локально-интегрированный грид

Программное обеспечение GRID. К базовым средствам GRID относятся инструментальные пакеты, содержащие средства разработки систем для GRID и минимальный набор служб, поддерживающих дистанционные операции.

Прежде всего, в этом классе ПО следует назвать Globus Toolkit [37] — стандарт де-факто для GRID, признанный ведущими производителями программного обеспечения. Этот набор программного обеспечения и служб, созданный партнерством Globus Alliance, является базисом для построения GRID и приложений для него. Последняя версия этого пакета Globus Toolkit 4 (GT4) включает средства разработки служб в соответствии с группой стандартов WS, среду их функционирования, а также базовый набор служб: управления заданиями, передачи файлов, информационного обслуживания и безопасности. В GT4 имеется три основных функции:

- управление заданиями (Execution Management) для управления, мониторинга и координации удаленного выполнения заданий;
- информационные службы (Information Services) для описания служб и ресурсов;
- управление данными (Data Management), позволяющее пользователям иметь доступ, передавать и управлять распределенными данными.

Кроме того, следует назвать Legion — пакет, во многом разделяющий с Globus Toolkit базовые принципы построения GRID, но в его основу положена

объектно-ориентированная модель, в отличие от Globus Toolkit, в котором акцент делается на стандартизацию удаленного взаимодействия.

Программные платформы представляют собой взаимосогласованные наборы программных средств, которые, исходя из сложившегося на сегодня представления о формах компьютеринга в GRID, поддерживают развертывание и обслуживание GRID, а также организуют его функционирование как единой операционной среды.

Характерные отличия платформ проявляются в том, что в них, в рамках единой архитектуры начинают реально решаться задачи обеспечения надежности, детерминированности, качества обслуживания, управляемости GRID с помощью механизмов планирования, мониторинга заданий и устройств, учета и протоколирования.

К открытым широко распространенным платформам относятся: EU DataGrid, NorduGrid, Unicore, VDT (Virtual Data Toolkit).

7.3. Отладчики параллельных программ

При программировании кластеров — вычислительных комплексов самой распространенной в данное время архитектуры, наиболее часто применяются различные реализации стандарта **MPI** (Message Passing Interface), использующего модель передачи сообщений между совместно протекающими процессами [1]. Спецификация MPI содержит множество коммуникационных функций различного характера: отправки типа точка-точка, широковещательные рассылки, редукционные операции и др. Кроме того, из практики разработки программного обеспечения хорошо известно, что примерно 2/3 времени, затрачиваемого на создание системы, приходится на отладку [36]. Ситуация еще более усложняется тем, что при распараллеливании программы для исполнения на кластерной вычислительной системе, состоящей из n узлов, число ошибок в общем случае следует умножать на n . К тому же известно, что стандартные техники тестирования и отладки неэффективны применительно к параллельным программам, потому как из-за взаимодействия параллельно работающих процессов на разных узлах кластерной системы возникает большое множество ошибок, которые не имеют место в последовательных программах. Среди этих ошибок наиболее опасны потенциальные, возникающие из-за большого количества источников недетерминированного поведения таких программ во время работы [54]. Это множество дополняется еще и специфическими ошибками, источником которых является применение интерфейса MPI, обладающего своими типами данных и правилами использования внутренних объектов.

В связи с вышесказанным необходимо осветить существующие в данное время разработки, касающиеся вопроса отладки параллельных программ.

Можно выделить несколько подходов к построению программных средств, служащих в помощь пользователю при отладке параллельного приложения: классические отладчики (диалоговые), сравнительная отладка, контроль корректности во время исполнения и анализ по трассе (так называемый “посмертный анализ”).

1. К классу диалоговых отладчиков относятся такие известные коммерческие средства как TotalView и **DDT** (Distributed Debugging Tool). Они основаны на последовательных отладчиках подобно gdb и обеспечивают обычные интерактивные функциональные возможности отладчиков, типа выполнения в пошаговом режиме, установки контрольных точек, оценки переменных и т. д., но дополнительно позволяют пользователю контролировать и воздействовать на группы процессов в отдельном сеансе отладки. Однако многие типы ошибок, в первую очередь, связанных с логикой параллельного поведения процессов, при помощи диалогового отладчика обнаружить трудно или невозможно (например, возможность зависания процессов).
2. Сравнительная отладка предполагает наличие двух разных программ (или двух версий одной программы), одна из них считается эталонной, а другая — отлаживаемой, и заключается в сравнении соответствующих переменных в соответствующие моменты выполнения этих программ. Сравнительная отладка позволяет, в частности, обнаруживать ошибки, проявляющиеся в нестабильном поведении программы. Метод сравнительной отладки впервые был предложен для последовательных программ в отладчике Guard, а для параллельных программ — в DVM-системе.
3. Другая возможность состоит в том, чтобы разработать специализированные инструментальные средства по обнаружению ошибок в приложениях MPI во время исполнения. Основная идея систем подобного типа заключается в том, чтобы проверять пользовательские вызовы MPI-функций на соответствие MPI-стандарту. В настоящее время, три различных проверяющих сообщения инструментальных средства находятся в более или менее активном развитии: MPI-CHECK, Umpire и MARMOT. MPI-CHECK поддерживает только Fortran, и выполняет контроль соответствия типов параметров или находит проблемы подобные взаимным блокировкам. MARMOT и Umpire используют интерфейс профилирования MPI.
4. Последний подход заключается в том, чтобы во время работы

параллельной программы собирать всю информацию о вызовах MPI-функций (времени вызова, параметрах функции, вызвавшем процессе) процессами в файле трассы. После работы программы этот файл анализируется отдельной утилитой (“посмертный анализ”). Пример такого средства - **ITAC** (Intel Trace Analyzer and Collector). Этот подход также используется многими утилитами для анализа производительности.

Далее рассмотрены некоторые распространённые средства автоматического контроля корректности выполнения MPI-программы. К этому классу относятся системы динамического анализа приложения и анализа по трассе.

MARMOT. Система MARMOT разрабатывается в центре высокопроизводительных вычислений Штуттгартского университета в Германии для операционных систем Linux [51].

MARMOT использует так называемый интерфейс профилирования для перехвата вызовов MPI и анализа их во время выполнения параллельной программы. Он не требует какой-либо модификации ни исходного кода приложения, ни используемой реализации MPI — это просто библиотека, которая должна быть статически скомпонована с приложением в дополнение к основной MPI-библиотеке.

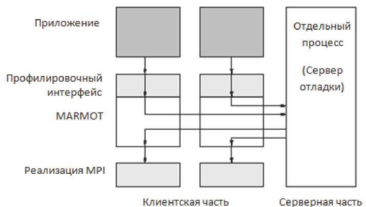


Рис. 7.3. Архитектура системы MARMOT

На *рис. 7.3* представлена архитектура MARMOT. Для всех ошибок, отлавливание которых требует глобального анализа, например обнаружение блокировок или управление потоком выполнения, MARMOT использует дополнительный процесс — так называемый сервер отладки. Каждый клиент (клиентами являются MPI-процессы) регистрируется в сервере отладки, который в свою очередь дает своим клиентам разрешение для работы циклическим способом. Чтобы гарантировать, что этот дополнительный процесс

отладки является прозрачным по отношению к приложению, в коммуникатор MPI_COMM_WORLD входят только прикладные процессы. Так как все другие коммуникаторы получаются из MPI_COMM_WORLD, они автоматически исключают процесс сервера отладки. Все ошибки, которые могут быть выявлены локально, например проверка параметров типа тэгов, коммуникаторов или рангов, выполняются на стороне клиента.

Ниже приведены алгоритмы обнаружения утилитой MARMOT некоторых типичных ошибок в параллельных программах.

Дедлоки (тупики). MARMOT содержит механизм автоматического обнаружения тупиков и уведомления пользователя о том, где и почему они произошли. Дедлоки могут быть вызваны, например, несоответствием операций send/receive или несоответствием коллективных операций в различных процессах. В настоящее время механизм обнаружения тупиков в MARMOT основан на времени ожидания и поэтому находит все реальные тупики. Сервер отладки MARMOT производит подсчёт времени, которое каждый процесс ждёт в MPI-вызове. Если это время превышает некоторый определяемый пользователем предел на всех процессах в один и тот же момент, процесс отладки выдаёт предупреждение о дедлоке.

Гонки данных. В настоящее время MARMOT указывает на использование макросов MPI_ANY_TAG и MPI_ANY_SOURCE, но он не строит графы зависимости, чтобы можно было рассмотреть различное возможное исполнение, и не сравнивает различные запуски.

Обработка ресурса. Это та область в MPI, где неправильное использование может привести к фатальным ошибкам без какой-либо очевидной ссылки на реальную причину. Так как такие ошибки очень трудны для поиска, то разработчики MARMOT уделяют особое внимание их обнаружению. MARMOT способен следить за надлежащим созданием, использованием и удалением всех MPI-ресурсов, таких как коммуникаторы, группы, типы данных и т. д. Поскольку эти ресурсы являются скрытыми объектами и поэтому, целиком и полностью зависят от реализации, то MARMOT имеет свой собственный учёт этих ресурсов и, таким образом, управление ими дублируется. MARMOT также проверяет корректное использование переменных типа MPI_Request и других параметров (тэги, ранги, и т. д.), например, отлавливает попытки повторного использования активных request-ov.

Исчерпывание ресурсов памяти и других ресурсов. Неблокирующие функции типа MPI_Isend и т. д. могут завершиться без проведения соответствующих тестов или вызовов функций ожидания. Однако число дескрипторов объектов типа MPI_Request ограничено (и определено реализацией MPI).

Поэтому request-ы всегда необходимо освобождать. MARMOT выдает предупреждение, когда request используется повторно, и также когда есть активные или неосвобожденные запросы на момент вызова функции MPI_Finalize. Другая проблема — повторное использование памяти, работа с которой ещё не закончена, например чтение/запись из/в буфер незаконченной операцией неблокирующей отправки/приёма. MARMOT в настоящее время не производит никаких проверок на то, может ли быть буфер безопасно повторно использован.

Intel Trace Analyzer and Collector. ITAC — это еще одна система, выполняющая автоматический контроль корректности MPI-приложений. Но, в отличие от рассмотренной выше системы MARMOT, это программное средство использует подход анализа трассы, собранной по ходу работы программы (посмертный анализ). ITAC состоит из 3-х компонентов:

1. Intel Trace Collector — используется расширенная версия программного продукта компании Intel. Это приложение собирает информацию о каждом вызове MPI-функции в процессах параллельной программы.
2. Анализатор — способен анализировать один файл трассы или сравнивать 2 различных файла. После работы записывает диагностическую информацию назад в трассу для третьего компонента — Визуализатора.
3. Визуализатор — утилита, строящая графическое изображение трассы работы процессов.

Первые 2 компонента системы предназначены для работы на вычислительном кластере, а Визуализатор — на персональном компьютере пользователя. Trase Collector использует профилировочный интерфейс для сбора информации обо всех входных и выходных параметрах в MPI-вызовах и некоторой дополнительной информации, как-то например контрольная сумма буфера сообщения. Анализатор ищет ошибки в файле трассы и производит вывод в файл с информацией об ошибках и предупреждениях с указанием на соответствующие строки в файле исходного кода.

Одним из преимуществ системы является графический интерфейс.

ITAC обнаруживает порядка 40 типов ошибок. Данные ошибки, сгруппированные по типам, перечислены ниже.

Общие ошибки:

- аварийное или ненормальное завершение процесса;
- незавершённый вызов функции;
- ошибочная операция;
- неизвестный тип данных;

- несоответствующий тип данных в операции.

Ошибки в операциях точка-точка:

- неосвобожденный объект типа MPI_Request / неверное освобождение объекта;
- не вызвана функция ожидания неблокирующей операции;
- разная контрольная сумма в начале и конце неблокирующей функции;
- непарная операция типа точка-точка;
- разная контрольная сумма в функции отправки и соответствующей ей функции приёма;
- несоответствие размеров отправленного и полученного сообщения.

Ошибки в коллективных операциях:

- не все процессы, входящие в коммуникатор, вызвали операцию;
- не все процессы, входящие в коммуникатор, закончили операцию;
- несоответствие размеров посылающегося/принимающегося сообщения в коллективной операции;
- ошибочный корневой процесс.

Потенциальные и реальные дедлоки.

Вопросы для самоконтроля

1. Какие программные средства для автоматизации процессов установки и настройки вычислительных узлов кластерной системы можете назвать?
2. На какие типы можно разделить программные системы мониторинга вычислительных кластеров? Назовите средства, относящиеся к каждому из типов.
3. Какие возможности предоставляют инструментальные средства для задач администрирования узлов кластерной системы?
4. Для чего служат системы пакетной обработки заданий?
5. Какие системы пакетной обработки заданий вам известны? Назовите их отличия.
6. Какие инструментальные средства включает в свой состав пакет OSCAR?
7. Дайте определение понятию GRID согласно Яну Фостеру.
8. В чем цель создания GRID-систем?
9. Приведите характеристики локальных GRID-систем, действующих в настоящее время.

10. Поясните понятия горизонтально-интегрированный и вертикально-интегрированный грид.
11. Какие подходы построения программных систем отладки параллельных программ вы можете назвать? Приведите примеры систем, использующих каждый из подходов.

Глава 8. Характеристики и архитектура современных процессоров

8.1. Intel

Intel Corporation – американская корпорация, производящая широкий спектр электронных устройств и компьютерных компонентов, включая полупроводники, микропроцессоры, наборы системной логики (чипсеты) и др.

Компанию Intel основали Роберт Нойс и Гордон Мур в 1968 году. Успех к компании пришёл в 1971, когда Intel начала сотрудничество с японской компанией Busicom, от которой получила заказ на двенадцать специализированных микросхем, но в итоге разработала один универсальный микропроцессор Intel 4004. Производительность этого процессора была сравнима с производительностью мощнейших компьютеров того времени. Следующим был разработан Intel 8008.

1971 год: микропроцессор 4004 – выпущен первый процессор корпорации Intel, разработанный для калькуляторов Busicom. 4-битный микропроцессор Intel 4004 считается первым в мире однокристальным микропроцессором. Этот процессор стал изобретением, открывшим путь к созданию искусственных интеллектуальных систем вообще и ПК в частности.

1972 год: микропроцессор 8008 – первый 8-битный центральный процессор, позиционировался как процессор для продвинутых калькуляторов общего назначения и терминалов ввода-вывода и автоматов бутылочного розлива.

1974 год: микропроцессор 8080 – 8-битный микропроцессор, представляет собой усовершенствованную версию процессора **Intel 8008**. Новый процессор выпускался по новейшей в то время 6-микронной NMOS (N-channel metal-oxide-semiconductor) технологии, что позволило разместить на кристалле 4500 транзисторов. В новом процессоре была очень развита система команд – 16 команд передачи данных, 31 команда для их обработки, 28 команд для перехода (с прямой адресацией), 5 команд управления. Благодаря наличию 16-разрядной адресной шины, мог адресовать 64 Кбайт памяти, которая теперь не

разделялась на память команд и данных. На базе микропроцессора Intel 8080 фирмой **MITS** был выпущен “первый в мире миникомпьютерный комплект, который может соперничать с промышленными образцами” (персональный компьютер) Altair-8800, который пользовался большой по тем временам популярностью.

1978 год: микропроцессоры 8086 - 8088 – первый 16-битный процессор, имел набор команд, который применяется и в современных процессорах. Именно от этого процессора берёт своё начало известная на сегодня архитектура x86. В персональных компьютерах процессор i8086 практически не использовался из-за дороговизны специализированных микросхем, которые были необходимы для работы процессора. Поэтому был выпущен процессор i8088, у которого шина данных была 8-битной. Успех новинки поставил Intel в число 500 крупнейших американских промышленных компаний. Также Intel выпустила в комплекте с процессором 8086 математические сопроцессоры 8087 и 8089. В 1983 году компания IBM представила ПК IBM 5160 (IBM PC/XT) на базе шестнадцатибитного (с восьмибитной шиной данных) процессора Intel 8088.

1982 год: микропроцессор 286, известный также под наименованием 80286, стал первым процессором Intel, способным выполнять любые программы, написанные для его предшественников, т. е. был x86 совместимым процессором. В 1984 году фирма IBM представила свой ПК, основанный на процессоре i286 с частотой 6 МГц - IBM PC AT, который вызвал большой интерес к архитектуре x86 вообще и к ПК IBM PC в частности. Спустя 6 лет с момента выпуска 286-го, количество ПК на базе этого процессора было оценено в 15 миллионов штук по всему миру.

1985 год: микропроцессор 386 – 32-битный x86-совместимый “многозадачный” процессор, был первым 32-разрядным процессором для IBM PC-совместимых ПК. Применялся, преимущественно, в настольных и портативных компьютерах.

1989 год: процессор 486 – 32-битный скалярный x86-совместимый процессор, построенный на гибридном CISC-RISC ядре. Это был первый микропроцессор со встроенным математическим сопроцессором, который существенно ускорил обработку данных, выполняя сложные математические действия вместо центрального процессора. Применялся, преимущественно, в настольных ПК, в высокопроизводительных рабочих станциях, в серверах и портативных ПК (ноутбуки и лэптопы).

1993 год: процессор **Pentium** – торговая марка нескольких поколений микропроцессоров семейства x86, выпускаемых корпорацией Intel с 22 марта 1993

года. Pentium является процессором Intel пятого поколения и пришел на смену процессору Intel 80486. Благодаря использованию суперскалярной архитектуры процессор мог выполнять 2 команды за 1 такт. Pentium является первым CISC процессором использующим многоконвейерную архитектуру, а 64-битная шина данных позволяла процессору Pentium обмениваться вдвое большим объемом данных с оперативной памятью, чем 486 за один шинный цикл (при одинаковой тактовой частоте). Позже к стандартному набору инструкций x86 был добавлен дополнительный набор **MMX** (Multi Media eXtension – технология, которая была разработана с целью ускорения работы мультимедийных и коммуникационных программных приложений), и процессор с этими инструкциями стал называться Pentium MMX. С этого времени компьютеры начали работать с атрибутами реального мира – звуком, голосовой и письменной речью, фотоизображениями.

1995 год: процессор **Pentium Pro** – процессор Pentium Pro, выпущенный осенью 1995 года, разрабатывался как средство наращивания быстродействия 32-разрядных приложений для серверов и рабочих станций, программных пакетов, используемых в машиностроении и научной работе. Все процессоры Pentium Pro оснащались второй микросхемой кэш-памяти, еще больше увеличивающей быстродействие. Кроме того, процессор мог быть использован при сборке многопроцессорной конфигурации (до 4-х процессоров).

1997 год: процессор **Pentium II** – насчитывающий 7,5 миллионов транзисторов, этот процессор использовал технологию Intel MMX, обеспечивающую эффективную обработку аудио- и графических данных. Кристалл и микросхема кэш-памяти помещались в корпус с односторонним контактом, который устанавливался на системной плате с помощью одностороннего разъема – в отличие от прежних процессоров, имевших множество контактов.

1998 год: процессор **Pentium II Xeon** был специально разработан для использования в серверах среднего и высшего уровней, а также в автоматизированных рабочих станциях. Системы, основанные на этом процессоре, могли быть сконфигурированы из 4, 8 и более процессоров.

1999 год: процессор **Celeron** – процессор создавался специально для рынка недорогих ПК. За небольшие деньги потребители получали возможность приобретения довольно производительного компьютера на базе Intel Celeron. Изначально он был вообще лишен кэш-памяти второго уровня, но затем в него были добавлены сначала 128 кбайт, а затем и 256 кбайт кэша.

1999 год: процессор **Pentium III** – по сравнению с предшественником (Pentium II) расширен набор команд (добавлен набор инструкций SSE, Streaming SIMD Extensions) и оптимизирована работа с памятью. Это

позволило повысить производительность как в новых приложениях, использующих расширения SSE (работа с трехмерной графикой, потоковым видео и аудио, распознавании речи), так и в существующих (за счёт возросшей скорости работы с памятью). Также был введён 64-битный серийный номер, уникальный для каждого процессора. Процессор включал 9,5 миллионов транзисторов и был выполнен по 0,25 мкм технологическому процессу.

2000 год: процессор **Pentium 4** – одноядерный x86-совместимый микропроцессор, в основе которого лежала принципиально новая, по сравнению с предшественниками, архитектура **NetBurst** (суперскалярная гиперконвейерная архитектура). Архитектура NetBurst разрабатывалась в первую очередь, с целью достижения высоких тактовых частот процессоров. Характерными особенностями архитектуры NetBurst являются гиперконвейеризация и применение кэша последовательностей микроопераций вместо традиционного кэша инструкций. АЛУ процессоров архитектуры NetBurst также имеет существенные отличия от АЛУ процессоров других архитектур. Первые процессоры Pentium 4 содержали 42 млн. транзисторов, а ширина проводников составляла всего 0,18 микрон. Этот процессор целиком и полностью был ориентирован на многозадачную работу с мультимедийными приложениями.

2001 год: процессор **Intel Xeon** – линейка серверных микропроцессоров, отличается увеличенным кэшем и поддержкой больших многопроцессорных систем. Процессор Intel Xeon обеспечивал высокую производительность при работе многопоточных приложений в многозадачной среде и предназначался для приложений, требующих большого объема вычислений с плавающей запятой.

2001 год: процессор **Intel Itanium** был первым 64-разрядным процессором Intel, в основе которого лежала новая архитектура **EPIC** (Explicitly Parallel Instruction Computing – обработка команд с явным параллелизмом), при этом компилятор должен выстраивать инструкции для параллельного исполнения. Ожидалось, что разрабатываемый процессор будет доминировать на рынке серверов, рабочих станций и возможно даже настольных ПК, вытеснив архитектуру x86.

2003 год: процессор **Intel Pentium M** с низким энергопотреблением, набор логики Intel 855 и сетевой чип Intel PRO/Wireless 2100 – эти три компонента стали составляющими технологии Intel Centrino, разработанной специально для мобильных ПК. Ноутбуки на базе технологии Centrino смогли работать до 6 часов без подзарядки батареи и обеспечивать быстрый доступ к беспроводным сетям стандарта IEEE 802.11b.

2006 год: **Intel Core, Core 2** – название, используемое для процессоров с

кодовым именем Yonah. Это название является частью операции по ребрендингу, запущенной компанией Intel в январе 2006 года, для замены торговой марки Pentium. Core, Core 2 – поколение микропроцессоров архитектуры x86, основанное на совершенно новой процессорной архитектуре, которая называется Intel Core. Процессоры Intel Core делятся на модели **Solo** (одноядерные), **Duo** (двухъядерные), **Quad** (четырёхъядерные) и **Extreme** (двух- или четырёхъядерные с высочайшей скоростью). В отличие от процессоров архитектуры NetBurst, в архитектуре Core 2 ставка делается не на повышение тактовой частоты, а на улучшение других параметров процессоров, таких как кэш-память, эффективность и количество ядер. Рассеиваемая мощность этих процессоров значительно ниже, чем у настольной линейки Pentium. С параметром **TDP** (thermal design power), равным 65 Вт, процессор Core 2 имеет наименьшую рассеиваемую мощность из всех доступных в продаже настольных чипов.

2008 год: **Intel Atom** – микропроцессор для ультрамобильных систем класса Netbook и Nettop с поддержкой симметричной многопроцессорности (SMP), ограниченной двумя микропроцессорами. Atom является CISC-процессором с архитектурой x86. В настоящее время платформа Atom имеет слишком большое потребление для применения в смартфонах, и по этому параметру не может соревноваться с процессорами с архитектурой **ARM** (Advanced RISC Machine), которые основаны на архитектуре RISC и широко применяются в современных мобильных устройствах. Однако, запланированная новая платформа будет использовать дизайн система-на-кристалле и потреблять вдвое меньше электроэнергии, что позволит использовать данную платформу для построения смартфонов.

Восьмидесятиядерные процессоры. Инженеры Intel предложили новый способ организации многоядерных процессоров – устанавливать один кристалл на другой, чтобы сам многоядерный процессор стал лишь одним из нескольких кристаллов в “бутерброде”, включающем в себя встроенную динамическую и флэш-память. В этом случае, поскольку площадь кристалла не увеличится, не будет расти и процент производственного брака в таких процессорах. По оценкам инженеров Intel, в один слой можно поместить 256 Мб памяти, поэтому всё-таки нужда в основной оперативной памяти не отпадёт, однако благодаря нескольким слоям и расширению шины системной памяти появится реальная возможность создания микропроцессоров с десятками вычислительных ядер.

Для увеличения пропускной способности системной шины в Intel рассчитывают использовать наработки в области оптико-кремниевых технологий –

гибридный кремниевый лазер, при помощи которого можно реализовать оптические линии передачи данных, обеспечивающие существенно большую скорость при меньших энергозатратах, чем традиционные электрические проводники. Intel рассматривает оптические шины передачи данных в качестве идеальной системной шины для связи микропроцессоров с десятками ядер с другими компонентами системы.

Данные технологии тестируются в восьмидесятиядерном процессоре, получившем название **Teraflops Research Chip** [58], который фактически представляет собой не законченный продукт, а исследовательский стенд. Производительность данного процессора выросла с номинального 1 терафлопа (на момент представления процессора) до 1,8 терафлопа в настоящее время. Коммерческий продукт из этого процессора в Intel делать не собираются, однако реализованные в нём технологии будут использоваться в других чипах нового поколения.

80-ядерный процессор изготовлен по 65-нанометровой технологии, состоит из восьми металлических слоев с медными соединениями и обладает небольшим для такого чипа числом транзисторов – на 275 мм^2 кристалла размещены всего 100 миллионов транзисторов. Причина того, что на такой большой площади размещено так мало транзисторов заключается в том, что в кристалле экспериментального чипа очень мало памяти, но много цепей ввода-вывода, которые не поддаются значительной миниатюризации. Все 80 независимых процессорных ядер размещены на одном кристалле в виде прямоугольника 8×10 ядер.

Чип может работать на нескольких скоростях, в зависимости от напряжения, при этом минимальная тактовая частота, позволяющая добиться производительности в один терафлоп, составляет 3,31 ГГц и достигается она при напряжении в один вольт. Энергопотребление процессора в этом режиме составляет 98 Вт. Если повысить подаваемое напряжение до 1,2 В, тактовая частота вырастет до 4 ГГц, производительность – до 1,28 терафлопс, а энергопотребление – до 181 Вт. Если же поднять частоту до 5,7 ГГц, то производительность достигнет 1,8 терафлоп, но энергопотребление вырастет уже до 265 Вт. Минимальная рабочая тактовая частота чипа составляет 1 ГГц, при этом процессор потребляет всего 11 Вт электроэнергии, а его производительность достигает 310 миллиардов операций с плавающей запятой в секунду. В ходе публичной демонстрации процессор работал на частоте 3,16 ГГц при напряжении 0,95 В и потреблял 62 Вт электроэнергии.

Особенностью архитектуры Teraflops Research Chip является использование абсолютно одинаковых и чрезвычайно простых ядер, которые не являются

x86-совместимыми. Каждое такое ядро состоит из вычислительного модуля и пятипортового маршрутизатора, отвечающего за обмен данными с другими ядрами.

В целях максимального упрощения ядер они построены на основе 96-разрядной архитектуры **VLIW**. При этом в одной команде VLIW может содержаться до восьми простых операций. Архитектура VLIW, применяемая Intel также в серверных процессорах Itanium, позволяет упростить конструкцию процессора за счёт использования компилятора для определения очередности выполнения инструкций. При этом отсутствует необходимость в сложных динамических алгоритмах расчёта последовательных и параллельных вычислений. Архитектура VLIW неплохо себя проявляет в однотипных и специализированных приложениях, однако, для персональных компьютеров она мало пригодна.

Идентичность всех вычислительных ядер позволяет упростить не только конструкцию процессора и его производство, но и управление всеми этими ядрами. Встроенный в каждое из ядер пятипортовый маршрутизатор используется для обмена данными между ядрами, которые, тем самым, объединяются в сеть. При этом любое ядро может использоваться исключительно для передачи данных, что позволяет динамически отключать питание вычислительных модулей таких ядер и экономить электроэнергию.

При тактовой частоте процессора в 4 ГГц пять 39-разрядных портов встроенного маршрутизатора в сумме обеспечивают пропускную способность в 80 Гб/с, при этом роутер занимает всего 0,34 мм² от общей площади ядра 3 мм². Четыре из пяти портов используются для связи с другими ядрами, а пятый – для подключения к встроенной в “слоёный” чип памяти. В целях оптимизации охлаждения процессора сам многоядерный чип будет находиться сверху “бутерброда”, а слои с памятью – под ним.

Одно из главных преимуществ реализованной в этом процессоре сети ядер заключается в том, что эту сеть, работающую с 80 ядрами, можно заставить работать как с меньшим, так и с большим числом ядер. Благодаря “черепичной” структуре чипа вычислительный модуль ядра может быть заменён любым другим, включая ядро с набором инструкций x86, специализированным цифровым сигнальным процессором или системой аппаратного шифрования. Вместо 80 ядер можно использовать, например, 12, шесть из которых будут представлять собой универсальные x86-ядра, а прочие будут специализированными ядрами, к примеру, для обработки трёхмерных изображений, поддержки TCP/IP или декодирования видео и звука.

С повышением тактовой частоты и площади процессора обеспечение

одновременного доступа тактового сигнала во все части чипа становится нетривиальной задачей, однако без этого невозможно добиться нормальной работы чипа. По данным Intel, в современных микропроцессорах на распределение тактового сигнала расходуется порядка 30 % всей энергии, используемой чипом, поэтому любое снижение энергопотребления этой части конструкции будет сразу заметно.

Поскольку каждое из ядер Teraflops Research Chip является независимым, тактовый сигнал должен одновременно доходить до каждого элемента отдельного ядра, а не всего процессора в целом. Поэтому инженеры Intel рассинхронизировали подачу тактового сигнала на ядра. Несмотря на то, что в результате этого обмен данными между ядрами несколько замедляется, однако выигрыш в энергосбережении более существенен. По данным Intel, в 80-ядерном чипе при работе на частоте 4 ГГц на распределение тактового сигнала расходуется всего 2,2 Вт энергии, что составляет 1,2 % от общего энергопотребления процессора под нагрузкой.

В новом чипе реализована и ещё одна энергосберегающая технология: каждое ядро поделено на 21 область, питание каждой из которых может отключаться отдельно от других, в зависимости от типа выполняемых команд, при этом и любое ядро может отключаться независимо от других. Вычислительный модуль и маршрутизатор также имеют независимое питание. Каждый из пяти портов маршрутизатора могут выключаться независимо друг от друга, а модули “умножители-аккумуляторы” (FPMAC) вычислительного блока пребывают в состоянии “сна” до тех пор, пока они не потребовались. Около 90 % FPMAC и 74 % вычислительного блока состоят из транзисторов, поддерживающих “спящий режим”. По информации Intel, такие транзисторы занимают на 5,4 % больше площади чипа, чем обычные.

В результате практически все мельчайшие компоненты процессора оснащены независимыми линиями электропитания, за что отвечает большая часть из 1248 контактов микросхемы. Такая конструкция позволяет гибко распределять нагрузку и даже регулировать нагрев процессора: если какие-то его части в процессе работы чересчур нагрелись, им на смену придут “отдыхавшие” до этого элементы.

8.2. AMD

Advanced Micro Devices, Inc. (AMD) — компания-производитель интегрированной электроники. Второй крупнейший поставщик x86-совместимых процессоров и крупный поставщик флеш-памяти [33]. Её финансовые показатели

имеют стабильную тенденцию к росту. При этом AMD, не скупясь, финансирует свои научные исследования (до 20 % от объёма продаж) и расширяет свои производственные мощности. У AMD одни из наиболее современных в отрасли производственные мощности, расположенные в США, Юго-Восточной Азии и Германии.

История компании AMD начинается с 1981 года, когда Intel была выбрана IBM для производства процессоров первого персонального компьютера. У IBM была в планах поставка процессоров для ПК как минимум двумя компаниями, поэтому технология процессоров Intel была лицензирована. Таким образом, AMD стала одной из первых компаний, которые производили клоны процессоров 8086. Первый процессор AMD поступил в продажу в 1982 г. Поскольку произведен он был по лицензии, AMD 8086 (и 8088) был идентичен модели Intel.

AMD Am286, клон процессора Intel 80286, производимый по лицензии начиная с 1983 г., был идентичен чипу от Intel, но отличался большим преимуществом: более высокой тактовой частотой. Если процессор Intel 286 выпускался максимум на 12,5 МГц, AMD продавала 20-МГц версии. В этих процессорах была увеличена шина адреса до 24 бит, максимальный поддерживаемый объём памяти – до 16 Мбайт, а максимальная тактовая частота для процессоров составила 20 МГц.

В 1991 г. AMD выпустила свой 386-й процессор. Подобно предшествующим моделям, этот процессор был идентичен версии Intel. AMD лицензировала производство клонов продуктов Intel вплоть до микрокода (“прошивки” CPU). Процессор отличался двумя интересными функциями: он был быстрее, чем модель Intel (40 МГц против частоты 33 МГц у Intel), и он стал первым, получившим на упаковку логотип Windows Compatible. Архитектура данного процессора была уже 32-битной.

В 1996 г. компания AMD выпустила процессор пятого поколения K5. По сравнению с Intel Pentium, K5 был технически более совершенен, хотя и у него было несколько недостатков. Процессор особенно интересен из-за внутренней архитектуры на основе RISC, которая декодировала инструкции x86 в микроинструкции перед их выполнением. У K5 возникли трудности с достижением высоких тактовых частот. К тому же FPU (Floating Point Unit) было не самым лучшим. Впрочем, в обычных условиях, K5 работал лучше, чем Pentium, и рейтинг PR (Performance Rating) был близок к реальности: K5 на частоте 100 МГц продавался как чип PR133, то есть AMD считала его эквивалентом по производительности 133-МГц Pentium. В отличие от предшественников, этот процессор обладал уже кэшем первого уровня объёмом 16 Кбайт.

В 1997 г. AMD выпустила новый процессор K6. Этот процессор был совместим с материнскими платами Socket 7 (Pentium) и обеспечивал очень хороший уровень производительности по сравнению с процессорами Intel Pentium II, по гораздо меньшей цене. Однако FPU у K6 по-прежнему было чуть слабее, чем у Intel. 250-нм версия K6, названная Little Foot, вышла в 1998 г.

Кроме того, в 1998 г. AMD анонсировала K6-2 - процессор, который использовал более скоростную шину (100 МГц) и обеспечивал улучшенную производительность SIMD. Он получил на один блок MMX больше, чем у K6, а также новый набор инструкций "3DNow!" для вычислений с плавающей запятой (набор инструкций MMX поддерживал только целочисленные вычисления). Процессор K6-2 оказался очень успешным, поскольку он стал хорошим решением модернизации для владельцев платформы Pentium MMX - используя множитель 2x на материнской плате с шиной 66 МГц, процессор работал на самом деле на 6x (400 МГц), что давало существенный прирост производительности при меньшей стоимости апгрейда.

Наконец, в 1999 г. AMD выпустила третью версию K6, а именно K6-III. Основным отличием от K6-2 стал интегрированный на кристалл кэш объемом 256 Кбайт. K6-III был довольно быстрым, но затратным в производстве, и ему на смену быстро пришёл Athlon (K7). Выпуск этого процессора также датируется 1999 годом. У чипа были устранены недостатки предыдущих моделей, а FPU, наконец, стал очень достойным - фактически, он оказался даже лучше, чем у Intel. Athlon стал самым быстрым процессором x86 и отличался многими сильными моментами, включая быструю шину FSB - EV6, которая использовалась в первых процессорах Alpha, и высокую производительность. Единственная проблема появилась со стороны не процессоров, а чипсетов: ни модели AMD, ни VIA не могли конкурировать с чипсетами Intel. K7 использовал Slot A (конкурировал с Intel Slot 1) и поддерживал кэш L2 с разными делителями (1/2, 2/5 или 1/3).

Следует заметить, что AMD первой объявила (и продавала) 1-ГГц процессор в линейке Athlon.

Затем были созданы следующие ядра для настольных ПК: Thunderbird (2000 г.), Palomino (2001 г.), Thoroughbred (2002 г.), Barton (2003 г.).

Следует отметить, что AMD также производила версии для серверов (Athlon MP) и ноутбуков (Athlon 4, Athlon XP Mobile), а также Geode NX (130 нм с кэшем 256 Кбайт). AMD продавала Thorton (130 нм, 512 Кбайт, 256 Кбайт из которого было выключено) и планировала Trinidad - Athlon на основе 90-нм техпроцесса.

K8 (Athlon 64) стал первым процессором x86, поддерживающим 64-битную

адресацию. У архитектуры появились другие улучшения, такие как интегрированный контроллер памяти.

В 2005 г. AMD изменила свою архитектуру, предложив версию K8 с двумя ядрами, так и появился Athlon 64 X2. Хотя процессор был создан из двух ядер K8, архитектура, использующая интерфейс HyperTransport, обеспечила высокую производительность в отличие от решения Intel, где за связь между ядрами отвечала FSB. Athlon 64 X2 вышел для разных сокетов.

В 2007 г. AMD представила новую платформу для настольных ПК под кодовым названием “AMD Spider” [19], состоящую из процессоров линейки AMD Phenom (K10) и чипсетов семейства AMD 7-Series. Основной инновацией, реализованной в платформе AMD Spider, можно назвать значительное повышение параметра “производительность на ватт”, главным образом, за счёт энергоэффективного дизайна 65 нм процессоров AMD Phenom. Наряду с этим, платформа AMD Spider обладает поддержкой ряда специфических технологий экономии энергии: ATI PowerPlay, Cool’n’Quiet 2.0, Microsoft DirectX 10.1, HyperTransport 3.0 и PCI Express 2.0. В частности, технология Cool’n’Quiet 2.0 позволяет снижать энергопотребление процессоров AMD Phenom, обладающих TDP 95 Вт, до средних 32 Вт в бытовых и средних 29 Вт в коммерческих приложениях. В то же время технология AMD CoolCore, реализованная в чипсетах AMD 7-Series, обеспечивает работу ядер процессора на разных частотах и, соответственно, снижение энергопотребления. При этом TDP чипсетов в среднем составляет порядка 10-12 Вт.

Во время своего выхода Phenom столкнулся с серьезными проблемами: он был не такой быстрый, как конкурирующие модели Intel, в первых версиях были ошибки (в TLB), к тому же он плохо поддавался разгону. Впрочем, Phenom остался по-прежнему хорошим процессором. Его архитектура хорошо проработана и неплохо подходит, например, для серверов. Впрочем, сегодня Phenom более ориентирован в качестве “бюджетного” процессора, но он, по крайней мере, свободен от ранних ошибок, а с помощью технологии AMD ACC его можно разгонять существенно сильнее, чем раньше.

Трёхъядерный Phenom X3 по сути является Phenom X4 с отключённым или дефектным четвёртым ядром.

Среди ключевых характеристик, присущих новым 4-ядерным процессорам Phenom, следует отметить следующие ключевые функциональные возможности.

Основным отличием процессоров поколения K10 от своих предшественников на базе K8 является объединение четырёх ядер на одном кристалле, обновления протокола Hyper-Transport до версии 3.0, общий для всех ядер

кэш L3, а также перспективная поддержка контроллером памяти DDR3. Сами ядра также были модернизированы по сравнению с ядрами K8.

Архитектура *Direct Connect Architecture* позволяет увеличить производительность и эффективность путём прямого соединения контроллера памяти и канала ввода/вывода с ядром. Разработана для одновременного выполнения как 32-битных, так и 64-битных вычислений. Преимущества:

- увеличение производительности приложений путём сокращения задержек при обращении к памяти;
- распределение полосы пропускания памяти в зависимости от запросов;
- обеспечение технологией Hyper-Transport соединения на пиковой скорости до 16.0 Гб/сек для предотвращения задержек;
- до 33.1 Гб/сек суммарной пропускной способности между процессором и системой (с учетом шины Hyper-Transport и контроллера памяти).

AMD Balanced Smart Cache - общий для всех ядер кэш L3 объёмом 2 Мб в дополнение к 512 Кб кэша L2 для каждого ядра [19]. Преимущества:

- сокращение задержек при обращении к часто используемым данным для увеличения производительности.

AMD Wide Floating Point Accelerator – 128-битный FPU для каждого ядра. Преимущества:

- ускорение выборки и обработки данных в вычислениях с плавающей запятой.

HyperTransport technology – один 16-битный канал со скоростью 4000 Гб/сек, соединение Hyper-Transport с пиковой скоростью до 8.0 Гб/сек и до 16.0 Гб/сек при работе в режиме Hyper-Transport 3.0. Выдает до 33.1 Гб/сек суммарной пропускной способности между процессором и системой (с учетом шины Hyper-Transport и контроллера памяти). Преимущества:

- быстрый доступ к системным ресурсам для увеличения производительности.

Integrated DDR2 DRAM Controller with AMD Memory Optimizer Technology – интегрированный контроллер памяти с высокой пропускной способностью и низкими задержками. Поддержка от PC2-3200 (DDR2-400) до PC2-8500 (DDR2-1066) небуферизованных модулей памяти, поддержка 64-битной DDR2 SDRAM. Пропускная способность до 17.1 Гб/сек. Преимущества:

- быстрый доступ к системным ресурсам для увеличения производительности.

AMD Virtualization (AMD-V) With Rapid Virtualization Indexing – аппаратный набор функций, разработанных для увеличения производительности, надёжности и безопасности в существующих и будущих средах виртуализации, позволяющий виртуальным машинам напрямую обращаться к выделенной памяти. Преимущества:

- позволяет программному обеспечению создавать более защищенные и эффективные виртуальные машины.

В дополнение также необходимо отметить появление в процессорах Phenom поддержки системы динамического управления тактовой частотой по каждому ядру. Поскольку архитектура не позволяет регулировать напряжение питания каждого ядра в независимом режиме, дополнительное энергосбережение обеспечивается снижением тактовой частоты каждого ядра в режиме простоя.

Помимо этого, дополнительное энергосбережение обеспечивает технология AMD CoolCore, позволяющая отключать неиспользуемые функциональные модули ядра. Имеется также технология Dual Dynamic Power Management, которая позволяет задавать независимые уровни напряжения для ядра процессора и интегрированного контроллера памяти. Дополнительная функция AMD Wideband Frequency Control обеспечивает наиболее эффективный с точки зрения потребления энергии режим с сохранением производительности при изменении режимов производительности. Наконец, технология Multi-Point Thermal Control, функционирующая за счёт интегрированных в ядро термодатчиков, обеспечивает термозащиту чипа, следя за уровнем температуры и автоматически снижая производительность до разумных температурных пределов.

Среди прочих энергосберегающих технологий также следует отметить появление режима питания C1E, активирующегося в момент перехода всех ядер чипа в ждущее состояние [19]. При этом отключается шина HyperTransport, память переходит в режим пониженного питания и значительным образом снижаются внутренние тактовые частоты процессора. Также любопытна функция Processor PSI (Power Saving Indicator), “сообщающая” регулятору напряжения о снижении нагрузки. Вместе с сохранением обратной совместимости процессоров AMD Phenom с VID-управлением плат под разъём AM2, новая последовательная система VID-управления обеспечивает 7-битную точность регулировки напряжения (с шагом 12,5 мВ).

На сегодняшний день существует 3 модели K10: Phenom 9500, Phenom 9600, Phenom 9700.

В целом предполагается, что платформа AMD Spider в перспективе станет

основой самых разнообразных систем AMD LIVE! – настольных и портативных ПК, домашних серверов и семейных цифровых развлекательных центров и пр. Одна из главных маркетинговых идей, под которой подаётся платформа AMD Spider, заключается в полноценной поддержке HD видео.

8.3. Sun Microsystems

Sun Microsystems – американская компания, производящая программное и аппаратное обеспечение, была основана 24 февраля 1982 года. SUN – аббревиатура Stanford University Networks.

Sun Microsystems традиционно является одним из крупнейших производителей серверов и рабочих станций на базе RISC-процессоров **SPARC** (Scalable Processor Architecture) собственной разработки, а также на базе микропроцессоров **Opteron** (AMD) и микропроцессоров Xeon (Intel); известна как разработчик таких технологий как **NFS** и **Java**, а также поддерживает программное обеспечение с открытым исходным кодом.

В середине 1990-х годов компания концентрируется прежде всего на поставке высокомасштабируемых серверов, использующих симметричное мультипроцессирование. Sun разрабатывает и производит 8-процессорный **SPARCserver 1000** и 20-процессорный **SPARCserver 2000**. Затем сменилось несколько поколений высокопроизводительных микропроцессоров SPARC, среди них **SPARC**, **SuperSPARC**, **UltraSPARC I**, **UltraSPARC II**, **UltraSPARC III**, **UltraSPARC IV** и новейшие **UltraSPARC IV+** и **UltraSPARC T1**. Для нижней ценовой категории разработаны процессоры **microSPARC I**, **microSPARC II**, **UltraSPARC IIe**, **UltraSPARC Iii**, и **UltraSPARC IIIi**.

Новейший микропроцессор UltraSPARC T1 [61] примечателен прежде всего тем, что использует 8 ядер (и по 4 потока на ядро). Это дань тенденции использовать горизонтальное масштабирование вместо вертикального, то есть использовать большее количество микропроцессоров с меньшими тактовыми частотами. В условиях центров обработки данных это приводит к пониженному тепловыделению и энергопотреблению.

SPARC. Процессоры SPARC – это линейка RISC-процессоров, разработанных компанией Sun Microsystems в 1985 году. Первоначально архитектура SPARC была разработана с целью упрощения реализации 32-битового процессора. Впоследствии, по мере улучшения технологии изготовления интегральных схем, она постепенно развивалась, и сегодня существует 64-битовая

версия этой архитектуры (SPARC-V9) [62], которая положена в основу семейства новых микропроцессоров, получивших название UltraSPARC [60].

Процессорная архитектура SPARC компании Sun Microsystems является одной из распространенных среди RISC-систем.

Архитектура SPARC является открытой, что означает:

- архитектура системы команд SPARC опубликована как стандарт IEEE 1754-1994;
- спецификации SPARC доступны для лицензирования любой компанией или частным лицом и дают возможность разрабатывать свои собственные решения;
- развитием архитектуры SPARC занимается независимая, некоммерческая организация SPARC International Inc., основанная в 1989 году. Членство в SPARC International открыто для всех желающих.

Для производства процессоров с архитектурой SPARC достаточно закупить у SPARC International Inc. лицензию на архитектуру системы команд и разработать свою реализацию архитектуры, либо закупить готовую реализацию.

Процессоры с архитектурой SPARC лицензированы и изготавливаются по спецификациям компании несколькими производителями: **Texas Instruments, Fujitsu, LSI Logic, Bipolar International Technology, Philips, Cypress Semiconductor и RossTechnologies.**

Семейство процессоров UltraSPARC. Современные бизнес-задачи требуют качественно новых подходов к обработке информации и организации процесса принятия решений. Объем данных, требующих обработки, растет высокими темпами. При этом все большую роль играет временной фактор, когда задержка в принятии решения может заметно сказаться на финансовых результатах деятельности компании. Такие тенденции предъявляют новые требования к организации вычислительного процесса и влекут за собой новые решения и технологии.

Основные критерии разработки. Производительность любого процессора при выполнении заданной программы зависит от такта синхронизации, среднего количества команд, выполняемых за один такт, и общего количества выполняемых в программе команд.

UltraSPARC-1. Системы UltraSPARC-1 обеспечивали высокую производительность при достаточно умеренной тактовой частоте (до 200 МГц) путем оптимизации среднего количества команд, выполняемых за один такт. При таком подходе основным вопросом является эффективность управления конвейером команд и иерархией памяти системы. Для увеличения производительности необходимо по возможности уменьшить среднее время доступа к памяти и

увеличить среднее количество команд, выдаваемых для выполнения в каждом такте, не превышая при этом разумного уровня сложности процессора.

При разработке суперскалярного процессора необходимо “проработать” ряд узких мест, ограничивающих выдачу для выполнения нескольких команд в каждом такте. Такими узкими местами являются наличие в программном коде зависимостей по управлению и данным, аппаратные ограничения на количество портов в регистровых файлах процессора и устройствах, реализующих иерархию памяти, а также количество целочисленных конвейеров и конвейеров выполнения операций с плавающей точкой.

При создании процессора UltraSPARC-1 компания Sun решила добиться увеличения производительности процессора в тех направлениях, где это не противоречило экономическим соображениям. Чтобы сократить число потенциальных проблем, было принято несколько конструкторских решений, которые определили основные характеристики UltraSPARC-1:

- реализация на кристалле отдельной кэш-памяти команд и данных;
- организация широкой выборки команд (128 бит);
- создание эффективных средств динамического прогнозирования направления переходов;
- реализация девятиступенчатого конвейера, обеспечивающего выдачу для выполнения до четырех команд в каждом такте;
- оптимизация конвейерных операций обращения к памяти;
- реализация команд обмена данными между памятью и регистрами плавающей точки, позволяющая не приостанавливать диспетчеризацию команд обработки;
- реализация на кристалле устройства управления памятью (MMU);
- расширение набора команд для поддержки графики и обработки изображений;
- реализация новой архитектуры шины UPA.

Устройство предварительной выборки и диспетчеризации команд.

Устройство предварительной выборки и диспетчеризации команд процессора UltraSPARC-1 (PDU) обеспечивает выборку команд в буфер команд, окончательную их дешифрацию, группировку и распределение для параллельного выполнения в конвейерных функциональных устройствах процессора. Буфер команд позволяет согласовать скорость работы памяти со скоростью обработки исполнительных устройств процессора. Команды могут быть предварительно выбраны из любого уровня иерархии памяти, например, из кэш-памяти команд (I-кэша), внешней кэш-памяти (E-кэша) или из основной памяти системы.

В процессоре реализована схема динамического прогнозирования направления ветвлений программы, основанная на двухбитовой истории переходов и обеспечивающая ускоренную обработку команд условного перехода. Для реализации этой схемы с каждым двумя командами в I-кэше связано специальное поле, хранящее двухбитовое значение прогноза. Таким образом, UltraSPARC-1 позволяет хранить информацию о направлении 2048 переходов, что превышает потребности большинства прикладных программ. Поскольку направление перехода может меняться каждый раз, когда обрабатывается соответствующая команда, состояние двух бит прогноза должно каждый раз модифицироваться для отражения реального исхода перехода. Эта схема особенно эффективна при обработке циклов.

Кроме того, в процессоре UltraSPARC-1 с каждым четырьмя командами в I-кэше связано специальное поле, указывающее на следующую строку кэш-памяти, которая должна выбираться вслед за данной. Использование этого поля позволяет осуществлять выборку командных строк в соответствии с выполняемыми переходами, что обеспечивает для программ с большим числом ветвлений практически ту же самую пропускную способность команд, что и на линейном участке программы. Способность быстро выбрать команды по прогнозируемому целевому адресу команды перехода является очень важной для оптимизации производительности суперскалярного процессора и позволяет UltraSPARC-1 эффективно выполнять “по предположению” (speculative) достаточно сложные последовательности условных переходов.

Используемые в UltraSPARC-1 механизмы динамического прогнозирования направления и свертки переходов сравнительно просты в реализации и обеспечивают высокую производительность. По результатам контрольных испытаний UltraSPARC-1 88 % переходов по условиям целочисленных операций и 94 % переходов по условиям операций с плавающей точкой предсказываются успешно.

Кэш-память команд. Кэш-память команд (I-кэш) представляет собой двухканальную множественно-ассоциативную кэш-память. С каждой строкой кэш-памяти связан соответствующий адресный тег. Команды, поступающие для записи в I-кэш, проходят предварительное декодирование и записываются в кэш-память вместе с соответствующими признаками, облегчающими их последующую обработку. Окончательное декодирование команд происходит перед их записью в буфер команд.

Организация конвейера. В процессоре UltraSPARC-1 реализован девятиступенчатый конвейер. Это означает, что задержка (время от начала до конца выполнения) большинства команд составляет девять тактов. Однако в любой

данный момент времени в процессе обработки могут одновременно находиться до девяти команд, обеспечивая во многих случаях завершение выполнения команд в каждом такте. В действительности эта скорость может быть ниже в связи с природой самих команд, промахами кэш-памяти или другими конфликтами по ресурсам. Первая ступень конвейера – выборка из кэш-памяти команд. На второй ступени команды декодируются и помещаются в буфер команд. Третья ступень, осуществляет группировку и распределение команд по функциональным исполнительным устройствам.

В каждом такте на выполнение в исполнительные устройства процессора могут выдаваться по 4 команды (не более двух целочисленных команд или команд плавающей точки/графических команд, одной команды загрузки/записи и одной команды перехода). На следующей ступени происходит выполнение целочисленных команд или вычисляется виртуальный адрес для обращения к памяти, а также осуществляются окончательное декодирование команд плавающей точки (ПТ) и обращение к регистрам ПТ. На пятой ступени происходит обращение к кэш-памяти данных. Определяются попадания и промахи кэш-памяти и разрешаются переходы. При обнаружении промаха кэш-памяти, соответствующая команда загрузки поступает в буфер загрузки. С этого момента целочисленный конвейер ожидает завершения работы конвейеров плавающей точки/графики, которые начинают выполнение соответствующих команд. Затем производится анализ возникновения исключительных ситуаций. На последней ступени все результаты записываются в регистровые файлы и команды изымаются из обработки.

Целочисленное исполнительное устройство. Главной задачей при разработке целочисленного исполнительного устройства (IEU) является обеспечение максимальной производительности при поддержке полной программной совместимости с существующим системным и прикладным ПО. Целочисленное исполнительное устройство UltraSPARC-1 объединяет в себе несколько важных особенностей:

- 2 АЛУ для выполнения арифметических и логических операций, а также операций сдвига;
- многотактные целочисленные устройства умножения и деления;
- регистровый файл с восемью окнами и четырьмя наборами глобальных регистров;
- реализация цепей ускоренной пересылки результатов;
- реализация устройства завершения команд, которое обеспечивает минимальное количество цепей обхода (ускоренной пересылки данных) при построении девятиступенчатого конвейера;

- устройство загрузки/записи (LSU).

LSU отвечает за формирование виртуального адреса для всех команд загрузки и записи (включая атомарные операции), за доступ к кэш-памяти данных, а также за буферизацию команд загрузки в случае промаха D-кэша (в буфере загрузки) и буферизацию команд записи (в буфере записи). В каждом такте может выдаваться для выполнения одна команда загрузки и одна команда записи.

Устройство плавающей точки (FPU). Конвейерное устройство плавающей точки процессора UltraSPARC построено в соответствии со спецификациями архитектуры SPARC-V9 и стандарта IEEE 754. Оно состоит из пяти отдельных функциональных устройств и обеспечивает выполнение операций с плавающей точкой и графических операций. Реализация отдельных исполнительных устройств позволяет UltraSPARC-1 выдавать и выполнять две операции в каждом такте. Большинство команд полностью конвейеризованы (имеют пропускную способность 1 такт), задержку в 3 такта и не зависят от точности операндов (имеют одну и ту же задержку для одинарной и двойной точности).

Устройство управления памятью (MMU). Высокая суперскалярная производительность процессора поддерживается высокой скоростью поступления для обработки команд и данных. Обычно эта задача ложится на иерархию памяти системы. Устройство управления памятью процессора UltraSPARC-1 выполняет все операции обращения к памяти, реализуя необходимые средства поддержки виртуальной памяти

Управление интерфейсом памяти (MIU). В процессоре UltraSPARC-1 применяется специальная подсистема ввода/вывода (MIU), которая обеспечивает управление всеми операциями ввода и вывода, которые осуществляются между локальными ресурсами: процессором, основной памятью, схемами управления и всеми внешними ресурсами системы. В частности, все системные транзакции, связанные с обработкой промахов кэш-памяти, прерываниями, наблюдением за когерентным состоянием кэш-памяти, операциями обратной записи и т. д., обрабатываются MIU.

Управление внешней кэш-памятью (Е-кэшем). Одной из важнейших проблем построения системы является согласование производительности процессора со скоростью основной памяти. Основными методами решения этой проблемы (помимо различных способов организации основной памяти и системы межсоединений) являются увеличение размеров и многоуровневая организация кэш-памяти. Устройство управления внешней кэш-памятью (ECU) процессора UltraSPARC-1 позволяет эффективно обрабатывать промахи кэш-памяти данных (D-кэша) и команд (Е-кэша). Все обращения к внешней кэш-

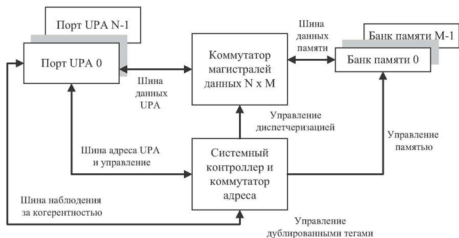


Рис. 8. 1. Масштабируемая архитектура UPA

памяти (Е-кэшу) конвейеризованы и выполняются за 3 такта. Такая организация дает возможность эффективно планировать конвейерное выполнение программного кода, содержащего большой объем обрабатываемых данных, и минимизировать потери производительности, связанные с обработкой промахов в D-кэше.

Типовой процессорный модуль UltraSPARC-1. Типовой процессорный модуль UltraSPARC-1 состоит из процессора UltraSPARC-1, микросхем синхронной статической памяти (SRAM), используемых для построения памяти тегов и данных внешнего кэша и двух кристаллов буферов системных данных (UDB). UDB изолируют внешний кэш процессора от остальной части системы и обеспечивают буферизацию данных для входящих и исходящих системных транзакций, а также формирование, проверку контрольных разрядов и автоматическую коррекцию данных (с помощью ECC-кодов). Таким образом, UDB позволяет интерфейсу работать на тактовой частоте процессора (за счет снижения емкостной нагрузки).

Архитектура системной шины UPA. Высокая производительность процессора UltraSPARC-1 потребовала создания гибкой масштабируемой архитектуры межсоединений, позволяющей достаточно просто строить системы для широкого круга приложений от небольших настольных систем индивидуального пользования до больших многопроцессорных серверов масштаба предприятия. Архитектура UPA (Ultra Port Architecture) (рис. 8. 1) определяет возможности построения целого семейства тесно связанных многопроцессорных систем с общей памятью.

UPA представляет собой спецификацию, описывающую логические и физические интерфейсы порта системной шины и требования, накладываемые на организацию межсоединений. К этим портам подключаются все устройства системы. Спецификация UPA включает также описание поведения системного контроллера и интерфейс ввода/вывода системы межсоединений.

В отличие от традиционных мультипроцессорных систем, которые поддерживают когерентное состояние кэш-памяти и разделяют глобально наблюдаемую адресную шину, архитектура межсоединений UPA основана на пакетной коммутации сообщений по принципу точка-точка. Поддержка когерентного состояния кэш-памяти системы для настольных рабочих станций осуществляется централизованным системным контроллером, а для больших серверов – распределенным системным контроллером. UPA может поддерживать дублирование наборов тегов всех кэш-системы и позволяет для каждой когерентной транзакции выполнять параллельно просмотр дублированных тегов и обращение к основной памяти.

Отход от традиционных методов построения мультипроцессорных систем, основанных на наблюдаемой шине или на справочнике, позволяет существенно минимизировать задержки доступа к данным благодаря сокращению потерь на обработку промахов кэш-памяти. В итоге архитектура межсоединений UPA позволяет полностью использовать высокую пропускную способность процессора UltraSPARC-1.

Разработчики архитектуры UPA многое сделали с целью минимизации задержек доступа к данным. Например, UPA поддерживает отдельные шины адреса и данных. Именно эти широкие шины (адресная шина имеет ширину 64 бит (в соответствии со спецификацией 64-битовой архитектуры V9), а шина данных – 144 бит (128 бит данных и 16 бит для контроля ошибок)) обеспечивают пиковую пропускную способность системы. Наличие отдельных шин позволяет устранить задержки, возникающие при переключении разделяемой шины между данными и адресом, а также возможные конфликты доступа к общей шине.

UPA не только поддерживает отдельные шины адреса и данных, но позволяет также иметь несколько шин с организацией соединений точка-точка. Обычно в большинстве систем имеются несколько интерфейсов для обеспечения работы подсистемы ввода/вывода, графической подсистемы и процессора. В мультипроцессорных системах требуются также дополнительные интерфейсы для организации связи между несколькими ЦП. Вместо одного набора шин данных и адреса для всех этих интерфейсов UPA допускает создание неограниченного количества шин.

Подобная организация имеет ряд достоинств. Наличие нескольких наборов шин позволяет минимизировать количество циклов арбитража и уменьшает вероятность конфликтов. Системный контроллер несет ответственность за работу и взаимодействие различных шин и может параллельно обрабатывать запросы нескольких шин. Он позволяет также уменьшить задержки, связанные с захватом шины. По существу, наличие нескольких шин адреса и данных означает меньшее число потенциальных главных устройств на каждом наборе шин. Для обеспечения наименьшей возможной задержки захвата шины используется распределенный конвейеризованный протокол арбитража. Каждый порт UPA имеет собственные схемы арбитража, при этом каждый порт в системе видит запросы шины всех других портов. Такая схема также позволяет уменьшить задержку доступа и обеспечивает увеличение общей производительности системы.

Архитектура UPA легко адаптируется для работы почти с любой конфигурацией системы (от однопроцессорной до массивно-параллельной). Разработчиками были предприняты специальные усилия с целью ее оптимизации для систем, содержащих от 1 до 4 процессоров. В результате до четырех тесно связанных процессоров и системный контроллер могут разделять доступ к одной и той же системной адресной шине. Однако на базе богатого набора транзакций и протокола когерентности, которые поддерживаются устройством интерфейса памяти процессора UltraSPARC-1 могут быть построены мультипроцессорные системы с большим количеством процессоров. В архитектуре UPA применяется протокол когерентности, построенный на основе операций записи с аннулированием соответствующих копий блока в кэш-памяти других процессоров системы и использующий для наблюдения дублированные теги. Процессор UltraSPARC поддерживает переходы состояний блоков кэш-памяти, соответствующие протоколам MOESI, MOSI и MSI.

Следует отметить, что в основу архитектуры UPA заложены гибкие принципы, которые позволяют иметь в системе не только несколько шин (мультиплексированных или отдельных), но и в широких пределах варьировать разрядность шины данных для удовлетворения различных требований к отношению стоимость/производительность. При этом в различных частях системы в зависимости от конкретных требований может использоваться разная скорость передачи данных. Например, разрядность шины данных системы ввода/вывода вполне может быть ограничена 64 битами, но для согласования с интерфейсом процессора более предпочтительна разрядность в 128 бит. С другой стороны, разрядность шины данных оперативной памяти системы может быть еще более увеличена для обеспечения высокой пропускной способности

при использовании более медленных, но более дешевых микросхем памяти (в младших моделях компьютеров на базе микропроцессора UltraSPARC-1 используется 256-битовая шина данных памяти, а в старших моделях – 512-битовая).

Процессор Sun UltraSparc T1. UltraSPARC T1 – многоядерный микропроцессор с аппаратной поддержкой многопоточности (до анонса 14 ноября 2005 г. известен как Niagara) [61].

Работы над процессором, оптимизированным для многопоточных приложений, начались в Sun Microsystems во второй половине 1990-х годов в рамках проекта MAJC (Microprocessor Architecture for Java Computing) по разработке аппаратной платформы Java-станций. Первоначально предполагалось, что применимость Java будет ограничена устройствами бытовой электроники, поэтому, несмотря на готовность в 1999 г. двухъядерного процессора MAJC-5200 с интегрированным контроллером памяти и графическим препроцессором, проект MAJC был переориентирован в разработку многопоточных процессоров для серверных систем.

Вторым фактором явился проект Hydra Стэнфордского университета (Stanford Hydra Single-Chip Multiprocessor) по разработке однокристалльного многоядерного суперскалярного процессора с разделяемым кэшем. Результатом проекта в 1998 г. стал однокристалльный процессор с 4-мя процессорными ядрами MIPS R10000.

Аппаратная архитектура. UltraSPARC T1 представляет собой кристалл, на котором размещаются до 8 ядер SPARC V9 с 16 КБайт L1 кэша инструкций и 8 КБайт L1 кэша данных, и блок операций с плавающей точкой (FPU), объединяемых внутрипроцессорным коммутатором (crossbar) с пропускной способностью 132 ГБайт/сек. К коммутатору присоединены 4 банка L2 кэша суммарной ёмкостью 3 Мбайт, разделяемого всеми процессорными ядрами. Каждый из банков обслуживается контроллером памяти DDR-II DRAM, используются 144-битные интерфейсы, агрегированная пиковая пропускная способность контроллеров – 25 ГБайт/сек.

Логическая архитектура. Процессор базируется на архитектуре RISC-архитектуре UltraSPARC Architecture 2005 specification с поддержкой набора команд SPARC v9 и выпускается в различных модификациях, отличающихся тактовыми частотами (1 - 1.4 ГГц) и количеством ядер (4, 6 и 8 ядер) с аппаратной поддержкой четырёх потоков на ядро. Одновременная обработка 32 потоков команд процессором с низким энергопотреблением и низкой теплоотдачей, обеспечивает необходимую вычислительную мощность и позволяет сэкономить значительные средства на электроэнергии и охлаждении.

Компания Sun Microsystems объявила чип UltraSparc T1 (кодовое название Niagara) открытым процессором. В данном случае Sun предлагает разработчикам не только спецификации, но и элементы конструкции процессора Niagara на языке описания аппаратной базы Verilog. Подробная информация о чипе обнародована в рамках проекта OpenSparc (OpenSparc T1). Инициатива OpenSparc привлекла большой интерес к новому процессору (аппаратная спецификация процессора загружена более 10000 раз), что, в свою очередь, приводит к более широкому распространению операционной системы Solaris. Таким образом, Sun Microsystems старается укрепить свои позиции по отношению к процессорам IBM Power, а также чипам AMD и Intel архитектуры x86.

8.4. Высокопроизводительные вычисления на графических процессорах

NVIDIA CUDA. Существуют задачи, для которых не требуется случайного доступа к памяти, и наибольшего ускорения можно добиться, когда одна инструкция выполняется над несколькими потоками. Это задачи, где используется параллельная обработка данных, когда одну и ту же последовательность математических операций применяют к большому объёму данных. Стремление уменьшить время решения таких задач и послужило толчком к появлению неграфических расчетов общего назначения **GPGPU** (General-Purpose computation on Graphics Processing Units – GPU общего назначения), суть которых сводится к использованию графических интерфейсов прикладного программирования (OpenGL, Direct3D) для задач, не связанных с графическими вычислениями. Данные к видеочипу передаются в виде текстур, а расчётные программы загружаются в виде шейдеров. Недостатками такого метода являются сравнительно высокая сложность программирования, низкая скорость обмена данными между CPU и **GPU** (Graphics Processing Unit – графический процессор, ГП), невозможность использования специфических возможностей видеокарт таких, как разделяемая память.

Компания NVIDIA разработала программно-аппаратную вычислительную архитектуру **CUDA** (Compute Unified Device Architecture – унифицированная вычислительная архитектура), основанную на расширении языка Си, которая даёт возможность организации доступа к набору инструкций графического ускорителя и управления его памятью при организации параллельных вычислений [34, 43].

Основные характеристики CUDA:

- унифицированное программно-аппаратное решение для параллельных вычислений на видеочипах NVIDIA;
- большой набор поддерживаемых решений (от мобильных до мультимедийных);
- стандартный язык программирования Си;
- стандартные библиотеки численного анализа **FFT** (быстрое преобразование Фурье) и **BLAS** (линейная алгебра);
- оптимизированный обмен данными между ЦП и ГП;
- поддержка 32- и 64-битных операционных систем.

Несмотря на то, что CUDA – это C-подобный язык программирования со своим компилятором и библиотеками для вычислений на ГП, программирование на CUDA и на GPGPU не одно и то же.

Преимущества CUDA перед традиционным подходом к GPGPU вычислениям:

- интерфейс программирования приложений CUDA основан на стандартном языке программирования C с расширениями, что упрощает процесс изучения и внедрения архитектуры CUDA;
- CUDA обеспечивает доступ к разделяемой между потоками памяти размером в 16 Кб на мультипроцессор, которая может быть использована для организации кэша с широкой полосой пропускания по сравнению с текстурными выборками;
- более эффективная передача данных между системной и видеопамятью;
- отсутствие необходимости в графических интерфейсах прикладного программирования с избыточностью и накладными расходами;
- линейная адресация памяти, возможность записи по произвольным адресам;
- аппаратная поддержка целочисленных и битовых операций.

Основные ограничения CUDA:

- отсутствие поддержки рекурсии для выполняемых функций;
- минимальная ширина блока в 32 потока.

Так как видеочип может поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук, то основой для эффективного использования производительности ГП в научных и иных неграфических расчётах является распараллеливание алгоритмов на сотни исполнительных блоков. При этом лучшие результаты достигаются, если отношение числа арифметических инструкций к числу обращений к памяти достаточно велико.

В настоящее время существует множество приложений и алгоритмов, которые показывают огромные приросты производительности

по сравнению с кодом на ЦП.

AMD Fusion – это гетерогенная многоядерная микропроцессорная архитектура, которая комбинирует процессорные ядра общего назначения с последовательной обработкой данных и многопоточные графические ядра с параллельной обработкой данных в одном процессорном кристалле [31, 33].

Центральный и графический процессоры реализованы отдельными ядрами на одном кристалле, при этом у ЦП – собственная кэш-память, а у ГП – собственный набор буферной памяти. Контроллер памяти – общий для обоих процессоров. Графическое ядро поддерживает различные интерфейсы (**HDMI** (High-Definition Multimedia Interface – мультимедийный интерфейс высокой чёткости), **DVI** (Digital Visual Interface – цифровой видеоинтерфейс) и др.). Процессор может поддерживать до 16 устройств стандарта PCI Express. Ядра имеют различную тактовую частоту и выпускаются по разным технологическим процессам. Вычислительное ядро начинается с 45-нм, а графическое – с 55-нм процесса (планируется переход на 32-нм и 22-нм процессы). В дальнейшем планируется объединение обоих ядер в одно, то есть и центральный и графический процессоры будут единым целым.

Преимущества Fusion:

- лучшее отношение производительности к тепловыделению, то есть при равной производительности процессор Fusion будет расходовать меньше энергии, чем связка процессор – видео;
- большая производительность из-за отсутствия медленной шины, по которой передаются данные, существующей в современных компьютерах;
- цена Fusion будет ниже, чем цена той же связки процессора с отдельно стоящим видео.

Intel Larrabee. Компания Intel провела исследование и сделала следующий вывод: у самых требовательных к вычислительной мощности приложений поток инструкций, как правило, очень линейен. Там немного операций ветвления и зависимостей между инструкциями, следовательно, современные процессоры, где большая часть кристалла отведена под управляющую логику (предсказание ветвлений и динамическое планирование инструкций), не очень хорошо подходят для подобных задач, для них с тем же количеством транзисторов будет намного полезнее использовать большое число маленьких, намного более простых процессоров.

Процессор Larrabee – это гибридное решение, сочетающее графический и центральный процессоры, построенное на базе усовершенствованной платформы x86 и может быть представлено в мультиядерном варианте, при котором данные могут делиться на неограниченное число потоков. Larrabee

отличается от графических процессоров других фирм тем, что использует набор инструкций, схожих с x86, что делает этот ГП более “гибким”, чем ГП конкурентов. В дополнение к обработке графики в компьютерных играх Larrabee способен проводить физические расчеты и трассировку лучей в реальном времени. Первые версии чипа насчитывают от 16 до 24 ядер и работают на тактовой частоте около 2 ГГц [52].

Заявленная производительность превышает один триллион операций с плавающей запятой в секунду. В качестве сферы применения многоядерного процессора Larrabee можно выделить два примера: анализ видеоизображений и компьютерные игры, в которых будет возможность создавать трёхмерные изображения методом трассировки лучей в реальном времени. Для этого требуются колоссальные вычислительные ресурсы, и заявленный потенциал Larrabee будет востребован.

Процессоры типа Larrabee могут экономить электроэнергию путём автоматического отключения невостребованной части ядер – процессор сам оценивает нагрузку, и использует только те аппаратные ресурсы, которые необходимы и достаточны для решения конкретной задачи. Еще одной отличительной особенностью архитектуры Larrabee является способность обрабатывать до 16 инструкций за такт, это в 4 раза больше, чем в любом существующем процессоре.

Larrabee представляет собой набор очень простых процессоров x86, выполняющих инструкции упорядоченно, которые дополнены векторным блоком математического сопроцессора, работающим с 512-битными векторами (вектор состоит из 16 чисел с плавающей запятой одинарной точности или целых чисел, либо из восьми чисел с плавающей запятой двойной точности). Каждое ядро снабжено 256 кбайт кэш-памяти, а связь между процессором и когерентность кэш-обеспечивается кольцевой шиной (рис. 8. 2).

В процессор Larrabee также добавлена поддержка технологии одновременной мультипоточности (SMT, Simultaneous Multithreading). Современные ЦП могут изменять порядок выполнения инструкций, чтобы максимально нагрузить вычислительные блоки, а ядра Larrabee этого делать не могут, поэтому некоторые участки кода будут слишком слабо использовать доступные ресурсы, а сочетая одновременное выполнение нескольких потоков, можно увеличить использование ресурсов. Инженеры обеспечили выполнение четырёх потоков на одном ядре с разными регистрами для каждого потока. Каждое ядро оснащено векторным блоком, одновременно работающим с 16 элементами (целыми числами, числами с плавающей запятой одинарной точности и двойной точности). При этом инструкции в наборе будут поддерживать до трёх



Рис. 8. 2. Архитектура процессора Larrabee

операндов, что позволяет реализовать инструкции “умножения-и-сложения” (**MAD**, multiply + add), а также выполнение неразрушающих инструкций, в которых один из исходных регистров перезаписывается значением результата выполнения инструкции.

Добавлен регистр маски векторного блока, содержащий 16 булевых значений, который указывает на то, нужно или нет записывать получившиеся данные в регистр назначения. Маска позволяет использовать технологию под названием предикация. В случае условного перехода “если-то” вместо попытки предсказания результата этого условия для продолжения выполнения программы без потери производительности, обе ветки выполняются параллельно, но только правильная сохраняется (с помощью регистра маски). Если код необходимо выполнять в относительно коротких ветвлениях, то такой подход эффективнее, поскольку он предотвращает риск ошибочных предсказаний ветвления.

Одно из основных ограничений ГП, даже при использовании интерфейса прикладного программирования подобного CUDA, заключается в доступе к памяти. Работа с памятью сильно усложнена, и чтобы получить оптимальную производительность, нужно минимизировать число используемых регистров, а также приходится использовать небольшие участки памяти в несколько килобайт.

Несмотря на гибкость, которую получили современные ГП, их функции остаются, в значительной мере, ориентированными на чистые расчёты (отказ от операций ввода/вывода на ГП). При этом Larrabee способен с этим справиться. Кроме того, можно использовать рекурсивные и виртуальные функции, что в случае ЦП невозможно. Возможность выполнения подобного кода без использования ЦП открывает ряд возможностей, одна из которых заключается в использовании компилятора "just-in-time" (JIT) в драйверах, которые адаптируют шейдеры к специфическим деталям архитектуры "на лету" во время выполнения задачи. При этом компилятор работает напрямую на процессоре Larrabee, вместо интеграции данного компилятора с центральным процессором.

Проводить отладку программного кода можно напрямую на Larrabee, вместе с тем отладка кода CUDA, как правило, невозможна без использования эмуляции на ЦП. В подобных случаях ЦП эмулирует ГП, но не в точности повторяет его поведение, поэтому могут возникнуть различные ошибки, "выловить" которые не всегда просто. Благодаря тому, что в основу Larrabee положена обычная архитектура x86, возможно выполнение написанных для центральных процессоров программ без изменения кода, что может пригодиться для ускорения работы специфических многопоточных приложений.

В связи с тем, что Larrabee будет обрабатывать шейдеры, используя универсальные ядра ЦП, а не разработанные исключительно в рамках спецификаций какого-либо интерфейса прикладного программирования, потоковые процессоры, их совместимость с новыми версиями интерфейсов прикладного программирования зависит лишь от их драйверов.

Современные видеокарты имеют большую производительность для трехмерных приложений (читай, компьютерных игр), но недостаточную скорость и точность для других, неспециализированных расчетов для компьютерных приложений. Практически все они построены на 32-битном процессе, точности которого не хватает в параллельных вычислениях. Если неправильно отображенный пиксель в игре – это не такая уж и проблема, то в других приложениях, например, при обсчете каких-то сложных физико-математических моделей, недостаточная точность может привести к большим проблемам. Таким образом, ядро ГП в процессоре требуется переводить на 64-битность. Вслед за этим под новый процессор потребуются новые 64-битные приложения, в создании которых должны быть заинтересованы разработчики программного обеспечения.

Вопросы для самоконтроля

1. Как образовалась компания Intel?
2. На смену какому процессору пришел процессор Pentium и чем он принципиально отличался от предшественника?
3. Какие существуют специальные линейки процессоров у компании Intel?
4. Будет ли процессор Teraflops Research Chip коммерческим и почему?
5. В чем отличие процессора Larrabee от других графических процессоров?
6. В чем особенность архитектуры системной шины UPA?
7. Как образовалась компания AMD?
8. Опишите архитектуру первого процессора AMD. Чем он отличался от Intel 8086?
9. Какие компоненты входят в платформу AMD Spider?
10. Дайте характеристику процессорам AMD Phenom.
11. Какие новые конструкторские решения были использованы при создании процессора UltraSPARC-1?
12. Реализованы ли в процессоре UltraSPARC-1 графические команды, и если да, то какие?
13. На чем основана архитектура межсоединений UPA?
14. В чем отличия процессора Larrabee от других графических процессоров?
15. В чем преимущество параллельных вычислений на графических процессорах?

Глава 9. Вычислительные системы нетрадиционной архитектуры

9.1. Биологические компьютеры

В настоящее время в поисках альтернативы полупроводниковым технологиям создания новых вычислительных систем ученые обращают все большее внимание на биотехнологии (биокомпьютинг), представляющий собой гибрид информационных, молекулярных технологий и биохимии. Биокомпьютинг позволяет решать сложные вычислительные задачи, пользуясь методами, принятыми в биохимии и молекулярной биологии, организуя вычисления при помощи живых тканей, клеток, вирусов и биомолекул.

ДНК-компьютеры. Наибольшее распространение получил подход, где в качестве основного элемента (процессора) используются молекулы ДНК (дезоксирибонуклеиновой кислоты). Центральное место в этом подходе занимает **ДНК-процессор**. Кроме ДНК в качестве биопроцессора могут быть использованы также белковые молекулы и биологические мембраны.

Как и любой другой процессор, ДНК-процессор характеризуется структурой (структура молекулы ДНК) и набором команд (перечень биохимических операций с молекулами). Принцип устройства компьютерной ДНК-памяти основан на последовательном соединении четырех нуклеотидов (основных кирпичиков ДНК-цепи). Три нуклеотида, соединяясь в любой последовательности, образуют элементарную ячейку памяти – кодон, которые затем формируют цепь ДНК. Основная трудность в разработке ДНК-компьютеров связана с проведением избирательных однокодонных реакций (взаимодействий) внутри цепи ДНК. Однако прогресс есть и в этом направлении. Уже есть экспериментальное оборудование, позволяющее работать с одним из 1020 кодонов или молекул ДНК. Другой проблемой является самосборка ДНК, приводящая к потере информации. Ее преодолевают введением в клетку специальных ингибиторов – веществ, предотвращающих химическую реакцию самосшивки.

Теоретическое обоснование возможности использования молекул ДНК для

организации вычислений было сделано еще в 50-х годах прошлого века Р. П. Фейнманом. В деталях эта теория была проработана в 1970-х годах Ч. Бенеттом и в 1980-х годах – М. Конрадом.

Первый компьютер на базе ДНК был создан в 1994 г. американским ученым Леонардом Адлеманом. Он смешал в пробирке молекулу ДНК, в которой были закодированы исходные данные, и специальным образом подобранные ферменты. В результате химической реакции структура ДНК изменилась таким образом, что в ней в закодированном виде был представлен ответ задачи. Поскольку вычисления проводились в ходе химической реакции с участием ферментов, на них было затрачено очень мало времени.

Вслед за работой Адлемана последовали другие исследования. Ллойд Смит из Университета Висконсин решил с помощью ДНК задачу доставки четырех сортов пищи по четырем адресам, которая подразумевала 16 вариантов ответа. Ученые из Принстонского университета решили комбинаторную шахматную задачу: при помощи **РНК** (рибонуклеиновой кислоты) нашли правильный ход шахматного коня на доске из девяти клеток (всего их 512 вариантов).

Ричард Липтон из Принстона первым показал, как, используя ДНК, кодировать двоичные числа и решать проблему удовлетворения логического выражения. Суть ее в том, что, имея некоторое логическое выражение, включающее n логических переменных, нужно найти все комбинации значений переменных, делающих выражение истинным. Задачу можно решить только перебором 2^n комбинаций. Все эти комбинации легко закодировать с помощью ДНК, а дальше действовать по методике Адлемана. Липтон предложил также способ взлома шифра **DES** (Data Encryption Standard), трактуемого как своеобразное логическое выражение.

Первую модель биокомпьютера в виде механизма из пластмассы высотой 30 см, в 1999 г. создал Ихуд Шапиро из Вейцмановского института естественных наук. Она имитировала работу “молекулярной машины” в живой клетке, собирающей белковые молекулы по информации с ДНК, используя РНК в качестве посредника между ДНК и белком.

А в 2001 г. Шапиро удалось реализовать вычислительное устройство на основе ДНК, которое может работать почти без вмешательства человека. Система имитирует машину Тьюринга – одну из фундаментальных концепций вычислительной техники. Машина Тьюринга шаг за шагом считает данные и в зависимости от их значений принимает решения о дальнейших действиях. Теоретически она может решить любую вычислительную задачу. По своей природе молекулы ДНК работают аналогичным образом, распадаясь и рекомбинируя в соответствии с информацией, закодированной в цепочках

химических соединений.

Разработанная в Вейцмановском институте установка кодирует входные данные и программы в состоящих из двух цепей молекулах ДНК и смешивает их с двумя ферментами. Молекулы фермента выполняют роль аппаратного, а молекулы ДНК – программного обеспечения. Один фермент расщепляет молекулу ДНК с входными данными на отрезки разной длины в зависимости от содержащегося в ней кода. Другой рекомбинирует эти отрезки в соответствии с их кодом и кодом молекулы ДНК с программой. Процесс продолжается вдоль входной цепи, и, когда доходит до конца, получается выходная молекула, соответствующая конечному состоянию системы.

Этот механизм может использоваться для решения самых разных задач. Хотя на уровне отдельных молекул обработка ДНК происходит медленно – с типичной скоростью от 500 до 1000 бит/с, что во много миллионов раз медленнее современных кремниевых процессоров, но по своей природе она допускает массовый параллелизм. По оценкам Шапиро и его коллег, в одной пробирке может одновременно происходить триллион процессов, так что при потребляемой мощности в единицы нановатт может выполняться миллиард операций в секунду.

В конце февраля 2002 г. фирма Olympus Optical в сотрудничестве с доцентом Токийского университета Акирой Тояма впервые создали коммерческую версию ДНК-компьютера, предназначенного для генетического анализа. Компьютер имеет молекулярную и электронную составляющие. Первая осуществляет химические реакции между молекулами ДНК, обеспечивает поиск и выделение результата вычислений. Вторая – обрабатывает информацию и анализирует полученные результаты.

Возможностями биокомпьютеров заинтересовались и военные. Агентство передовых оборонных исследовательских проектов Министерства обороны США **DARPA** (Defense Advanced Research Projects Agency) выполнило проект, получивший название **Bio-Comp** (Biological Computations, биологические вычисления). Его цель – создание мощных вычислительных систем на основе ДНК.

Несмотря на ряд успехов, до практического применения компьютеров на базе ДНК еще очень далеко. Однако в будущем их смогут использовать не только для вычислений, но и для многих других задач, например, для создания устройств хранения информации и медицинских нанороботов.

Еще один распространенный подход биокомпьютинга – **клеточные компьютеры**. Клеточные компьютеры представляют собой самоорганизующиеся колонии различных "умных" микроорганизмов, в геном которых удалось

включить некую логическую схему, которая могла бы активизироваться в присутствии определенного вещества. Для этой цели идеально подошли бы бактерии, стакан с которыми и представлял бы собой компьютер. Такие компьютеры очень дешевы в производстве. Им не нужна столь стерильная атмосфера, как при производстве полупроводников.

Главным свойством компьютера такого рода является то, что каждая их клетка представляет собой миниатюрную химическую лабораторию. Если биоорганизм запрограммирован, то он просто производит нужные вещества. Достаточно вырастить одну клетку, обладающую заданными качествами, и можно легко и быстро вырастить тысячи клеток с такой же программой.

Основная проблема, с которой сталкиваются создатели клеточных биокомпьютеров, – организация всех клеток в единую работающую систему. На сегодняшний день практические достижения в области клеточных компьютеров напоминают достижения 1920-х годов в области ламповых и полупроводниковых компьютеров. Например, в Лаборатории искусственного интеллекта Массачусетского технологического университета создана клетка, способная хранить на генетическом уровне 1 бит информации. Также разрабатываются технологии, позволяющие единичной бактерии отыскивать своих соседей, образовывать с ними упорядоченную структуру и осуществлять массив параллельных операций.

В 2001 г. американские ученые создали трансгенные микроорганизмы (т. е. микроорганизмы с искусственно измененными генами), клетки которых могут выполнять логические операции И и ИЛИ.

Специалисты лаборатории Оук-Ридж, штат Теннесси, использовали способность генов синтезировать тот или иной белок под воздействием определенной группы химических раздражителей. Ученые изменили генетический код бактерий *Pseudomonas putida* таким образом, что их клетки обрели способность выполнять простые логические операции. Например, при выполнении операции И в клетку подаются два вещества (по сути – входные операнды), под влиянием которых ген вырабатывает определенный белок. Теперь ученые пытаются создать на базе этих клеток более сложные логические элементы, а также подумывают о возможности создания клетки, выполняющей параллельно несколько логических операций.

Потенциал биокомпьютеров велик. К достоинствам, выгодно отличающим их от компьютеров, основанных на кремниевых технологиях, относятся:

1. Более простая технология изготовления, не требующая для своей реализации столь жестких условий, как при производстве полупроводников.

2. Использование не бинарного, а тернарного кода (информация кодируется тройками нуклеотидов), что позволит при меньшем количестве шагов перебрать большее число вариантов при анализе сложных систем.
3. Потенциально исключительно высокая производительность, которая может составлять до 1014 операций в секунду за счет одновременного вступления в реакцию триллионов молекул ДНК.
4. Возможность хранить данные с плотностью, в триллионы раз превышающей показатели оптических дисков.
5. Исключительно низкое энергопотребление.

Однако, наряду с очевидными достоинствами, биокомпьютеры имеют и существенные недостатки, такие как:

1. Сложность со считыванием результатов – современные способы определения кодирующей последовательности не совершенны, сложны, трудоемки и дороги.
2. Низкая точность вычислений, связанная с возникновением мутаций, прилипанием молекул к стенкам сосудов и т. д.
3. Невозможность длительного хранения результатов вычислений в связи с распадом ДНК в течение времени.

Хотя до практического использования биокомпьютеров еще очень далеко, и они вряд ли будут рассчитаны на широкие массы пользователей, предполагается, что, они найдут достойное применение в медицине и фармакологии, а также с их помощью станет возможным объединение информационных и биотехнологий.

9.2. Нейрокомпьютеры

Одно из наиболее перспективных направлений разработки принципиально новых архитектур вычислительных систем тесно связано с созданием компьютеров нового поколения на основе принципов обработки информации, заложенных в искусственных **НС** (нейронных сетях).

Под нейронной сетью обычно понимают совокупность элементарных преобразователей информации, называемых "нейронами", которые определенным образом соединены друг с другом каналами обмена информации – "синаптическими связями" [29]. Нейрон, по сути, представляет собой элементарный

процессор, характеризующийся входным и выходным состояниями, передающей функцией (функция активации) и локальной памятью.

Состояния нейронов изменяются в процессе функционирования и составляют кратковременную память нейросети. Каждый нейрон вычисляет взвешенную сумму пришедших к нему по синапсам сигналов и производит над ней нелинейное преобразование. При пересылке по синапсам сигналы умножаются на некоторый весовой коэффициент. В распределении весовых коэффициентов заключается информация, хранящаяся в ассоциативной памяти НС. Основным элементом проектирования сети является ее обучение. При обучении и переобучении НС ее весовые коэффициенты изменяются. Однако они остаются постоянными при функционировании нейросети, формируя долговременную память.

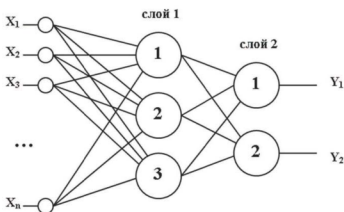


Рис. 9. 1. Двухслойная НС

НС может состоять из одного слоя, из двух, из трех и большего числа слоев (рис. 9. 1). Число входов НС определяет размерность гиперпространства, в котором входные сигналы могут быть представлены точками или гиперобластями из близко расположенных точек. Количество нейронов в слое сети определяет число гиперплоскостей в гиперпространстве. Вычисление взвешенных сумм и выполнение нелинейного преобразования позволяют определить, с какой стороны от той или иной гиперплоскости находится точка входного сигнала в гиперпространстве.

Искусственные нейронные сети обладают рядом полезных свойств. Они не требуют детализированной разработки программного обеспечения и открывают возможности решения задач, для которых отсутствуют теоретические

модели или эвристические правила, определяющие алгоритм решения. Такие сети обладают способностью адаптироваться к изменениям условий функционирования, в том числе к возникновению заранее непредусмотренных факторов. По своей природе НС являются системами с очень высоким уровнем параллелизма.

В отличие от цифровых систем, представляющих собой комбинации процессорных и запоминающих блоков, нейропроцессоры содержат память, распределенную в связях между очень простыми процессорами, которые часто могут быть описаны как формальные нейроны или блоки из однотипных формальных нейронов. Тем самым основная нагрузка на выполнение конкретных функций процессора ложится на архитектуру системы, детали которой, в свою очередь, определяются межнейронными связями. Таким образом, **нейрокомпьютер** – устройство переработки информации на основе принципов работы естественных нейронных систем [14].

В нейрокомпьютерах используются принципы обработки информации, осуществляемые в реальных нейронных сетях. Эти принципиально новые вычислительные средства с нетрадиционной архитектурой позволяют выполнять высокопроизводительную обработку информационных массивов большой размерности. В отличие от традиционных вычислительных систем, нейросетевые вычислители, аналогично нейронным сетям, дают возможность с большей скоростью обрабатывать информационные потоки дискретных и непрерывных сигналов, содержат простые вычислительные элементы и с высокой степенью надежности позволяют решать информационные задачи обработки данных, обеспечивая при этом режим самоперестройки вычислительной среды в зависимости от полученных решений.

Преимущества нейрокомпьютеров:

1. Все алгоритмы нейроинформатики высокопараллельны, что само по себе является залогом высокого быстродействия.
2. Нейросистемы можно сделать устойчивыми к помехам и разрушениям.
3. Устойчивые и надежные нейросистемы могут создаваться и из ненадежных элементов, имеющих значительный разброс параметров.

Общие принципы построения нейронных сетей были заложены в начале второй половины 20 века в работах таких ученых, как Д. Хебб, М. Минский, Ф. Розенблат.

Первыми нейрокомпьютерами были перцептроны Розенблатта Марк-1 (1958) и Тобермори (1964), а также Адалин, разработанный Уидроу (Widrow) и Хоффом (1960) на основе дельта-правила (формулы Уидроу). В настоящее

время Адалин (адаптивный сумматор, обучающийся по формуле Уидроу) является стандартным элементом многих систем обработки сигналов и связи.

Идея нейробионики (создания технических средств на нейро-принципах) стала интенсивно реализовываться в начале 1980-х годов. Импульсом было следующее противоречие: размеры элементарных деталей компьютеров сравнялись с размерами элементарных “преобразователей информации” в нервной системе, было достигнуто быстроедействие отдельных электронных элементов в миллионы раз большее, чем у биологических систем, а эффективность решения задач, особенно связанных с задачами ориентировки и принятия решений в естественной среде, у живых систем пока недостижимо выше.

Другой импульс развитию нейрокомпьютеров дали теоретические разработки 1980-х годов по теории нейронных сетей (сети Хопфилда, сети Кохонена, метод обратного распространения ошибки).

В настоящее время постепенно складывается рынок нейрокомпьютеров. Примерами являются нейрокомпьютер Synapse (Siemens, Германия), нейрокомпьютер “Силиконовый мозг” (созданный в США по программе “Электронный мозг” и предназначенный для обработки аэрокосмических изображений). Широко распространены высокопараллельные нейроускорители (сопроцессоры) для различных задач.

Длительное время считалось, что нейрокомпьютеры эффективны для решения так называемых неформализуемых и плохо формализуемых задач, связанных с необходимостью включения в алгоритм решения задачи процесса обучения на реальном экспериментальном материале. В первую очередь к таким задачам относилась задача аппроксимации частного вида функций, принимающих дискретное множество значений, т. е. задача распознавания образов.

В настоящее время к этому классу задач добавляется класс задач, иногда не требующий обучения на экспериментальном материале, но хорошо представимый в нейросетевом логическом базисе. К ним относятся задачи с ярко выраженным естественным параллелизмом обработки сигналов, обработка изображений и др. Подтверждением точки зрения, что в будущем нейрокомпьютеры будут более эффективными, чем прочие архитектуры, может, в частности, служить резкое расширение в последние годы класса общематематических задач, решаемых в нейросетевом логическом базисе. К ним, кроме перечисленных выше, можно отнести задачи решения линейных и нелинейных алгебраических уравнений и неравенств большой размерности; систем нелинейных дифференциальных уравнений; уравнений в частных производных; задач оптимизации и других задач.

Основные области применения нейροкомпьютеров:

1. Управление в реальном времени.
2. Распознавание образов.
3. Прогнозирование в реальном времени.
4. Оптимизация – поиск наилучших вариантов.
5. Обработка сигналов при наличии больших шумов.
6. Протезирование (“умные протезы”) и усиление естественных функций, в том числе – за счёт прямого подключения нервной системы человека к компьютерам.
7. Психодиагностика.

9.3. Квантовые компьютеры

На данный момент имеется ряд задач, для эффективного решения которых не достаточно возможностей, предлагаемых классическими компьютерами, созданными с помощью полупроводниковых технологий. Примерами таких задач являются поиск в неструктурированной базе данных, моделирование эволюции квантовых систем (например, ядерные реакции) и факторизация больших чисел (основа практически всех современных шифров).

Одним из вариантов решения перечисленных задач стала разработка квантовых компьютеров. **Квантовый компьютер** – вычислительное устройство, которое путем выполнения квантовых алгоритмов существенно использует при работе квантовомеханические эффекты, такие как квантовый параллелизм и квантовая запутанность (в физике это называется принцип “двойственности” или неопределенности Гейзенберга) [7].

Квантовый параллелизм можно объяснить как то, что данные в процессе вычислений представляют собой квантовую информацию, которая по окончании процесса преобразуется в классическую путём измерения конечного состояния квантового регистра [7, 21]. Выигрыш в квантовых алгоритмах достигается за счет того, что при применении одной квантовой операции большое число коэффициентов суперпозиции квантовых состояний, которые в виртуальной форме содержат классическую информацию, преобразуется одновременно.

Под **квантовой запутанностью** обычно понимается следующее: “Вообразите атом, который мог бы подвергнуться радиоактивному распаду в определенный промежуток времени. Или не мог бы. Мы можем ожидать, что у этого атома есть только два возможных состояния: “распад” и “не распад”, но в квантовой механике у атома может быть некое объединенное состояние – “распада – не распада”, то есть ни то, ни другое, а как бы между. Вот это состояние и называется “суперпозицией” [42].

Таким образом, квантовый компьютер – это вычислительное устройство, работа которого строится на квантовомеханических эффектах, в частности, на принципе квантовой запутанности, позволяющем реализовать параллелизм вычислений.

При построении квантового компьютера вместо микросхем используются элементарные частицы, что со временем должно резко уменьшить размеры и многократно поднять производительность. Ученые охлаждают атомы до сверхнизкой температуры, чтобы те не улетали, а лазерный луч передает им задачу и снимает решение.

Собрать вместе и заставить работать даже не тысячу, а хотя бы полсотни атомов, крайне сложно технически. Например, у канадцев из “Ди-Вейв” (DiWave), представивших публике на конференции в Калифорнии в 2008 году квантовый компьютер, вычисления совершали 16 частиц, что на сегодняшний день является рекордом.

При этом ни о какой миниатюрности и тем более квантовых ноутбуках речи пока нет. Сам процессор получается миниатюрным, но вокруг него присутствуют огромные емкости с охлаждающим газом и мощные лазеры. Например, на свою презентацию представители фирмы “Ди-Вейв” не смогли привезти сам компьютер, так как ему нужны слишком большие емкости с жидким гелием, а установили связь с компьютером по Интернету.

Идея квантовых вычислений впервые была высказана Ю. И. Маниным в 1980 году, но активно эта проблема стала обсуждаться после появления в 1982 году статьи американского физика-теоретика Р. Фейнмана. В этих работах было предложено использовать для вычислений операции с состояниями квантовой системы. Основная идея квантового вычисления состоит в том, чтобы хранить данные в ядрах атомов, изменяя их ориентацию в пространстве. Элементарная ячейка такого компьютера получила название квантовый бит (Q-бит = quantum bit). В отличие от привычной единицы информации – бита (binary digits = bits), который может принимать только два значения или “0”, или “1”, квантовый бит в соответствии с принципом неопределенности, постулируемым квантовой механикой, может находиться одновременно в

состоянии и “0”, и “1”. Таким образом, для квантового устройства размером N кубит количество выполняемых параллельно операций будет равно 2^N .

Первый алгоритм для квантовых компьютеров (разложение числа на простые множители за полиномиальное от $\log(n)$ время) был разработан в 1994 году Питером Шором. Алгоритм Шора имеет большое значение для современной криптографии. Если удастся создать достаточно мощные квантовые компьютеры, то часть использующихся систем шифрования с открытым ключом (например, криптографический алгоритм с открытым ключом **RSA** – буквенная аббревиатура от фамилий Rivest, Shamir и Adleman) станет уязвима для взлома: для подбора тайного ключа необходимо разложить открытый ключ на простые множители. При достаточно длинном ключе даже современным суперкомпьютерам на это нужны сотни лет, а вот перед квантовыми он не устоит.

Также разрабатывается применение квантовых компьютеров для противоположной задачи: не взлома, а усиления защиты информации. Применение идей квантовой механики уже открыли новую эпоху в области криптографии, так как методы квантовой криптографии открывают новые возможности в области передачи сообщений.

Еще одним известным алгоритмом для квантовых компьютеров является алгоритм Гровера осуществляющий поиск в неструктурированной базе данных.

Теоретических моделей квантового компьютера множество. Проблема состоит, скорее, в том, чтобы найти разумные пути создания реального прибора. Существует несколько подходов к осуществлению идеи такого устройства.

Компьютер на ядерно-магнитном резонансе. Ученые, сами того не предполагая, уже создали квантовый компьютер. Его первый “опытный образец” – это импульсный ядерный магнитно-резонансный (ЯМР) спектрометр высокого разрешения. Спины ядер, входящих в состав атомов, в свою очередь образующих исследуемую в ЯМР-спектрометре молекулу, – это Q-биты, единицы измерения квантовой информации. Каждое ядро имеет свою частоту резонанса в данном магнитном поле. При воздействии импульсом на резонансной частоте одного из ядер оно начинает эволюционировать, остальные же ядра “молчат”. Для того чтобы заставить эволюционировать второй атом, надо взять другую частоту и дать импульс на ней. Иными словами, процесс вычислений управляется импульсами переменного магнитного поля, – нужно только написать алгоритм поставленной задачи. Например, 1000 в степени 3 (то есть миллиард) операций в алгоритме Шора для 1000-разрядного числа – это миллиард воздействий на отдельные спины и на их пары. При этом в молекуле

есть прямая связь между спинами, и поэтому она является идеальной заготовкой для квантового компьютера, а сам спектрометр – практически готовый “процессор” для этого компьютера. Однако в настоящее время удается работать с системами с общим числом спинов не более пяти-семи, в то время как для решения полномасштабных задач их необходимо порядка 1000. Подобного рода работы в России не ведутся, так как российские ученые считают, что принципиально невозможно увеличить количество спинов до требуемого числа.

Компьютер на ионных ловушках. Другой подход основан на использовании ионных ловушек, или “подвешенных” в вакууме ионов. За разработку метода удержания одиночных ионов, лежащего в основе ионных ловушек, ученому Боннского университета Паулю в 1989 году была присуждена Нобелевская премия. Эти ионные ловушки удалось “растянуть” и получить одномерный ионный кристалл, удерживаемый и в осевом, и в радиальном направлении внешними полями. У каждого иона кристалла берутся два уровня энергии – это один Q-бит; между собой эти ионы связаны через колебания внутри одномерного кристалла, который имеет набор резонансных частот. Больше всего экспериментов по квантовым вычислениям с использованием таких кристаллов предложили ученые Инсбрукского университета в Австрии, а провели больше всего такого рода экспериментов ученые из Лос-Аламосской лаборатории в США. Больших кристаллов получить пока не удастся – на сегодняшний день получена цепочка из 30 ионов. Но дальнейший прогресс в увеличении числа ионов связывают с созданием трехмерной лазерной стоячей волны – трехмерной совокупности точек с минимумами потенциальной энергии для поляризованных атомов. Иными словами, это трехмерная решетка, которая уже хорошо изучена; изучена также и методология лазерного охлаждения, и поэтому сейчас стоит задача в каждый минимум поместить атом и охладить его. В данном направлении предстоит сделать еще очень много работы, но оно выглядит весьма перспективным.

Квантовый компьютер на твердом теле. Это могут быть сверхпроводники, как предлагают ученые из Института Ландау. Другой подход высказал в 2006 году австралийский физик Кейн: делать квантовый компьютер точно на том кремнии, на котором сегодня работает традиционная микроэлектроника. В нужных местах на расстояниях порядка 10 нм располагают атомы фосфора – обычную примесь в кремнии, которая прекрасно изучена. Если на таком расстоянии расположить два атома фосфора, то облака внешних электронов немного пересекутся, что необходимо для их взаимодействия, и атомы смогут обмениваться состояниями. Один атом управляет электронами другого.

Над этими атомами делаются 50-ангстремные электродики, и с помощью напряжения на этом электроде меняют резонансную частоту спина ядра атома фосфора. Очень похоже на полевой транзистор – как бы те же затворы, только вместо тока – состояния атома. Мы предложили работать не на одном атоме, а на серии атомов; под этими электродами должна быть последовательность атомов, чтобы они действовали параллельно, тогда сформируется относительно большой сигнал, который легче регистрировать.

В последнее время интерес к квантовому компьютеру резко увеличился. Некоторые из причин этого:

1. Разработаны квантовые алгоритмы для решения ряда трудных задач (например, для факторизации больших чисел и для поиска в неструктурированной базе данных).
2. Разработана процедура коррекции квантовых ошибок, без которой практически невозможны попытки создавать квантовый компьютер.
3. Экспериментально продемонстрирована возможность квантовых вычислений на основе алгоритма Гровера и других алгоритмов.

Пока что одним из наиболее удачных экспериментов можно считать 7-ми Q-битный “ядерный” квантовый компьютер, созданный группой учёных из IBM и Стенфордского университета, который позволил разложить число 15 на множители 5 и 3. Несмотря на простоту выполненной математической операции, и колоссальность затрат на разработку (более 100 млн долларов США), разработчикам удалось добиться значительного сокращения числа операций (1 действие и 1 проверка, вместо 4 действий и 4 проверок).

Один из последних удачных проектов – представленный канадцами из “Ди-Вейв” на конференции в Калифорнии в 2008 году 16-ти Q-битный “ядерный” квантовый компьютер.

Как показывают теоретические расчеты, при использовании большего количества Q-битов, квантовые компьютеры позволят за минуты решать задачи, которые современными компьютерами должны решаться веками.

9.4. Компьютеры с реконфигурируемой архитектурой

Компьютеры с реконфигурируемой архитектурой, называемые по-другому **ПЛИС-компьютерами** (ПЛИС – программируемая логическая интегральная схема), широко используются для построения различных по сложности и возможностям цифровых устройств [11].

Среди множества областей применения реконфигурируемых вычислительных систем можно особо выделить следующие:

1. Символьная обработка информации.
2. Защита компьютерных сетей.
3. Создание принципиально новых лекарственных препаратов и материалов нового поколения.
4. Управление в реальном масштабе времени объектами ядерной энергетики, летательными и космическими аппаратами.
5. Оптимизация эксплуатации нефтяных месторождений.
6. Томографические исследования приповерхностных слоев Земли акустическими и электромагнитными волнами.
7. Нейроматематика.

Основой такого рода компьютеров является программируемая логическая интегральная схема (**ПЛИС**). ПЛИС – электронный компонент, используемый для создания цифровых интегральных схем. В отличие от обычных цифровых микросхем, логика работы ПЛИС не определяется при изготовлении, а задается посредством программирования. Для программирования ПЛИС используются отладочные среды, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках (например, Verilog, VHDL).

ПЛИС широко используется для построения различных по сложности и возможностям цифровых устройств. К ним можно отнести приложения, где необходимо большое количество портов ввода-вывода (бывают ПЛИС с более чем 1000 выводов (“пинов”)), цифровую обработку сигнала (ЦОС), цифровую видео-аудио аппаратуру, высокоскоростную передачу данных, криптографию, проектирование ASIC, в качестве мостов (коммутаторов) между системами с различной логикой и напряжением питания.

Альтернативой ПЛИС являются заказные **БИС** (Большие Интегральные Схемы), которые существенно дороже, а также компьютеры, которые из-за программного способа реализации алгоритмов медленнее ПЛИС.

Некоторые производители предлагают для своих ПЛИС процессоры, которые могут быть модифицированы под конкретную задачу, и затем встроены в ПЛИС, тем самым уменьшив место на печатной плате и упростив разработку для самой ПЛИС.

Основные современные типы ПЛИС: CPLD и FPGA.

CPLD (*complex programmable logic device*) содержат относительно крупные программируемые логические блоки (макроячейки), соединённые с внешними выводами и внутренними шинами. Функциональность CPLD кодируется в энергонезависимой памяти, поэтому нет необходимости их перепрограммировать при включении.

FPGA (*Field-Programmable Gate Array*) – разновидность ПЛИС, содержащая блоки умножения – суммирования (DSP), которые широко применяются при обработке сигналов, а также логические вентили и их блоки коммутации.

FPGA обычно используются для обработки сигналов, имеют больше логических элементов и более гибкую архитектуру, чем CPLD. Программа для FPGA хранится в распределённой памяти, которая может быть выполнена как на основе энергонезависимых ячеек статического ОЗУ (в этом случае программа не сохраняется при исчезновении электропитания микросхемы), так и на основе энергонезависимых ячеек Flash-памяти или специальных перемычек antifuse (в этих случаях программа сохраняется при исчезновении электропитания). Если программа хранится в энергонезависимой памяти, то при каждом включении питания микросхемы необходимо заново конфигурировать ее при помощи начального загрузчика, который может быть встроен и в саму FPGA. FPGA применяются также, как ускорители универсальных процессоров в суперкомпьютерах (например: Cray -XD1, SGI - Проект RASC).

Для использования всех преимуществ систем, строящихся на основе ПЛИС, должны существовать приложения, требующие соответствующей программно-аппаратной базы для своего выполнения. Сам процесс адаптации научных и производственных приложений для ПЛИС является очень трудоёмкой задачей.

На сегодняшний день относительно небольшое количество компаний занимается производством микросхем ПЛИС. Одним из лидеров в области производства ПЛИС-микросхем является американская компания **Xilinx**, основанная в 1984 году. На данный момент у неё существует несколько серий выпускаемой аппаратуры для разного рода вычислений:

1. **Virtex**. Высокопроизводительные ПЛИС на основе FPGA, призванные заменить специализированные интегральные схемы при решениях различных ресурсоемких задач.
2. **Spartan**. Более дешевые и менее производительные ПЛИС FPGA, разработанные для использования в устройствах, рассчитанных на большие тиражи и невысокую стоимость комплектующих.
3. **CoolRunner** и **XC9500**. Серии ПЛИС типа CPLD, предназначенные для

использования в портативных устройствах – мобильных телефонах, GPS-навигаторах, КПК и т. д. Для микросхем данного типа главными критериями является минимизация размеров и потребляемой мощности.

Основным конкурентом компании Xilinx по всем основным направлениям является компания **Altera**. Главное из них – это производство ПЛИС как типа FPGA, так и типа CPLD.

В мае 2008 года Altera представила новое семейство из серии Stratix высокопроизводительных микросхем типа FPGA – Stratix IV. Для менее ресурсоемких задач компания Altera предлагает серию ПЛИС FPGA Cyclone, а в качестве компромисса между производительными Stratix и недорогими Cyclone – серию Arria. Для мобильных устройств выпускается серия Max на основе ПЛИС типа CPLD. Также в дополнение к этим микросхемам компания выпускает серию ASIC микросхем HardCopy, разработанных в качестве специализированных аналогов соответствующих FPGA Stratix.

Микросхемы компании Altera активно применяются во многих областях, например, на рынке беспроводных и проводных коммуникаций, в военных технологиях, в области телевидения, а также в различных мобильных устройствах.

Также на рынке имеется ряд других фирм, специализирующихся в каком-либо одном или нескольких направлениях:

1. **Lattice Semiconductor** на рынке FPGA-микросхем занимает всего порядка нескольких процентов. Однако она является одним из лидеров в области производства CPLD и SPLD (simple PLD – более простые по сравнению с CPLD программируемые устройства) микросхем.
2. Компания **Actel** делает ставку на производство небольших и недорогих микросхем типа FPGA, ориентируясь на надежность таких ПЛИС.
3. Компания **Atmel** начала заниматься направлением программируемых микросхем в 1999 году, поэтому набор предоставляемой продукции не очень велик. Также Atmel занимается производством SPLD- и CPLD-микросхем, последние из которых характерны тем, что являются совместимыми с аналогичными устройствами основных производителей ПЛИС аппаратуры, таких как Altera и Xilinx.
4. Шотландская компания **Nallatech** является одним из лидеров в области создания готовых вычислительных решений на основе ПЛИС. Она занимается разработкой системных плат и вычислительных модулей для персональных компьютеров, используя в качестве базы

микросхемы компании Xilinx. Данные готовые решения применяются в оборонной промышленности для увеличения производительности.

5. Компания **Sun** в 2005 году запустила проект OpenSparc, в рамках которого она предоставляет открытые коды процессора UltraSparc T1 и предлагает разработчикам участвовать в процессе его усовершенствования. А для того, чтобы процесс разработки был более доступным, Sun предлагает реализацию на ПЛИС некоторых элементов OpenSparc – собственно ядра процессора Sparc, математического сопроцессора и коммутатора между центральным процессором и кэш-памятью.
6. Компьютерный гигант компания **Cray** поддерживает программу Adaptive Supercomputing, идея которой состоит в создании гибридных суперкомпьютеров, объединяющих различные архитектуры. Необходимость таких компьютеров они объясняют тем, что суперкомпьютеры с обычной архитектурой зачастую не предоставляют нужной производительности, а главное – гибкости. В 2004 году Cray построила гибридный суперкомпьютер Cray XD1, в котором для большего ускорения совместно с основными процессорами AMD Opteron используются FPGA-микросхемы компании Xilinx. А в 2007 году был построен суперкомпьютер Cray XT5h, объединяющий в себе скалярные и векторные процессоры, а также процессоры на основе ПЛИС.

Вопросы для самоконтроля

1. Что такое – биологический компьютер?
2. Что может быть использовано в качестве био-процессора?
3. Кем и в каком году был сделан первый шаг в использовании молекул ДНК для организации вычислений?
4. Расскажите про первый компьютер на базе ДНК.
5. Чем известна фирма Olympus Optical?
6. Назовите преимуществ и недостатки биокомпьютеров.
7. Что такое – нейрокомпьютер?
8. Преимущества и недостатки нейрокомпьютеров.
9. Что послужило импульсом к развитию нейро-бионики?

10. Приведите примеры нейрокомпьютеров.
11. Назовите основные области применения нейрокомпьютеров.
12. Что такое – квантовый компьютер?
13. Какие важные задачи призваны решить квантовые компьютеры?
14. В чем основная идея квантового вычисления?
15. Какие задачи были решены с использованием квантовых компьютеров?
16. Назовите причины возросшего в последнее время интереса к квантовым компьютерам.
17. Что вам известно про алгоритм Шора?
18. Что такое ПЛИС?
19. Назовите примеры областей применения реконфигурируемых вычислительных систем.
20. Назовите альтернативы ПЛИС.
21. Что такое CPLD?
22. Что такое FPGA?
23. Назовите компании, занимающиеся производством микросхем ПЛИС.

Заключение

Учебное пособие “Высокопроизводительные вычислительные системы” открывает цикл пособий под общим заглавием “Основы высокопроизводительных вычислений”. В пособии дана краткая история развития высокопроизводительной вычислительной техники, классификация архитектур, описаны основные компоненты и технологии, применяемые при их построении. Приведены методы оценки производительности, проведен обзор современных процессоров, применяемых при построении ВВС. Описаны нетрадиционные архитектуры ВВС, применяемые на определенных классах задач. Приведены основные программные решения, предоставляемые операционными системами для организации доступа без конфликтов конкурирующим процессам и потокам к ресурсам, а также стратегии планирования процессов, управляющих очередностью исполнения нескольких разнородных задач. Рассмотрены системы автоматической установки и настройки ОС на множестве вычислительных узлов кластера, средства мониторинга аппаратуры и загрузки ресурсов суперкомпьютера, системы пакетной обработки заданий, системы отладки параллельных программ, используемых при организации вычислительного процесса на суперкомпьютерах в центрах коллективного пользования. Уделено внимание программному обеспечению, позволяющем развертывать GRID-системы, для объединения ресурсов нескольких центров в единый вычислительный пул. Подробно изложены типы параллелизма и модели обмена данными между единицами вычислительного процесса, описываются основные подходы, применяемые при создании операционных систем, использующих ресурсы параллельных вычислительных систем.

Более подробно технологии параллельного программирования на многопроцессорных вычислительных системах с распределенной (разделяемой) оперативной памятью или с использованием комбинированного подхода, вопросы проектирования, реализации и анализа параллельных алгоритмов обработки данных описаны во 2-м и 3-м пособиях из цикла “Основы высокопроизводительных вычислений” – “Технологии параллельного программирования” и “Параллельные вычислительные алгоритмы”.

Приложение 1. Top50 и Top500

Тридцать четвертая редакция Top500

В ноябре 2009 года был обнародован очередной тридцать четвертый список самых производительных суперкомпьютеров, входящих в Top500 [57].

Тройку лидеров представили:

1. **Jaguar** – Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc. – максимальная производительность 1759.00 TFlop/s, пиковая – 2331.00 TFlop/s, установлен в Oak Ridge National Laboratory, США (подробнее в приложении 2);
2. **Roadrunner** – BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband IBM – максимальная производительность 1042.00 TFlop/s, пиковая – 1375.78 TFlop/s, установлен в DOE/NNSA/LANL, США (подробнее в приложении 2);
3. **Kraken XT5** – Cray XT5-HE Opteron Six Core 2.6 GHz Cray Inc. – максимальная производительность 831.70 TFlop/s, пиковая – 1028.85 TFlop/s, установлен в National Institute for Computational Sciences/University of Tennessee, США (подробнее в приложении 2).

Безусловным лидером списка стал компьютер производства Cray Inc., который обошел IBM Roadrunner с практически удвоенным преимуществом в производительности. Для сравнения, в тридцать втором списке Top500 от ноября 2008 года первое место занял IBM Roadrunner с максимальной производительностью в 1,105 PFlop/s, а второе место – Cray XT5 Jaguar с максимальной производительностью в 1,059 PFlop/s.

Важно отметить, что на 12 месте в списке расположился российский кластер “Ломоносов”, работающий в научно-исследовательском вычислительном центре МГУ (максимальная производительность 350.10 TFlop/s, пиковая – 414.42 TFlop/s).

По количеству компьютеров в списке страны распределились так:

1. США – 277
2. Великобритания – 45

3. Германия – 27
4. Франция – 26
5. Китай – 21
6. Япония – 16
7. Канада – 9

Россия имеет 8 компьютеров и делит с Австрией и Новой Зеландией 8 место, однако Российские компьютеры обладают большей суммарной производительностью.

Операционные системы, под управлением которых работают суперкомпьютеры:

1. Linux – 446 (89.20 %)
2. Unix – 25 (5.00 %)
3. Windows – 5 (1.00 %)
4. BSD Based – 1 (0.20 %)
5. Смешанные – 23 (4.60 %)

Двенадцатая редакция Top50

30 марта 2010 года научно-исследовательский вычислительный центр МГУ имени М. В. Ломоносова и Межведомственный Суперкомпьютерный Центр РАН объявили о выпуске двенадцатой редакции списка Top50 самых мощных компьютеров СНГ [22]. Объявление состоялось на Международной научной конференции "Параллельные вычислительные технологии"(ПАВТ'2010), проводимой Суперкомпьютерным консорциумом университетов России и Российской академией наук при поддержке Российского фонда фундаментальных исследований.

Тройку лидеров представили:

1. Суперкомпьютер МГУ "Ломоносов" производства компании "Т – Платформы" с производительностью 350.1 TFlop/s, установленный в научно-исследовательском вычислительном центре МГУ (подробнее в приложении 2).
2. Суперкомпьютер MBC-100K производства Hewlett-Packard производительностью 107.4 TFlop/s, установленный в Межведомственном Суперкомпьютерном Центре РАН.

3. Суперкомпьютер СКИФ МГУ "ЧЕБЫШЁВ", являющийся совместной разработкой Московского государственного университета имени М. В. Ломоносова, Института программных систем РАН и компании "Т-Платформы", выполненной при поддержке компании Intel в рамках суперкомпьютерной программы "СКИФ-ГРИД" Союзного государства России и Беларуси и национального приоритетного проекта "Образование" производительностью 47.3 TFlop/s (подробнее в приложении 2). Компьютер установлен в Научно-исследовательском вычислительном центре МГУ.

Все системы списка теперь принадлежат "терафлопному диапазону" – для попадания в текущую редакцию потребовалась производительность 1.47 TFlop/s, а нижняя граница первой десятки по производительности поднялась с 12.36 до 14.6 TFlop/s.

Увеличилось количество систем в списке, построенных на процессорах Intel (39); на процессорах AMD построено 6 систем, на процессорах IBM – 4 системы и 1 система построена на базе процессоров HP. Продолжается постоянный рост количества процессорных ядер в системе – в данной редакции списка оно составило не менее 176, при том, что уже 20 систем стали более чем 1024-ядерными.

По количеству систем, входящих в список, лидирующие позиции занимает компания "Т-Платформы" (14 суперкомпьютеров), на одну систему опережая компанию Hewlett-Packard (13 систем), увеличила свою долю и компания IBM (11 систем).

Лидирует количество систем, используемых в науке и образовании (28), несколько увеличилось число систем, ориентированных на конкретные прикладные исследования (10) и систем, используемых в финансовой сфере (5), уменьшилось число систем, используемых в промышленности (5).

Анализ Top500

В определенный момент массовость скалярных процессоров создала иллюзию исключительной универсальности выбранного направления. Однако, рейтинг Top500 не является универсальным [8]. Рассмотрим это на примере 25 редакции Top500, вышедшей в июне 2005 года.

Первые два места в 25 редакции списка Top500 занял IBM BlueGene/L – производительностью 136,8 TFlop/s и 91,3 TFlop/s соответственно (подробнее см. приложение 2), затем шел SGI Altix производительностью 51,9 TFlop/s и NEC Vector SX6 производительностью 35,9 TFlop/s.

Как уже было сказано, Top500 содержит рейтинги компьютеров, которые определяются с помощью теста LINPACK (подробнее в главе 4). Пиковая производительность R_{peak} сопровождается описанием каждой системы, но места распределяются по реальной производительности R_{max} , достигнутой при выполнении теста LINPACK. Дополнительно по каждой системе представляется еще ряд характеристик: число процессоров N_{Proc} , размер матрицы N_{max} , при котором достигнуто значение R_{max} , размер матрицы N_{half} , при котором получена половина максимальной производительности $R_{max}/2$, и др.

Найдем для каждой строки Top500 списка значение $U_{eff} = (R_{max}/R_{peak}) * 100 \%$. Смысл данной величины в том, что чем её значение больше, тем эффективнее работает вычислительная система.

Если отсортировать строки списка Top500 не по R_{max} , а по U_{eff} , то список принципиально изменится. Первые места с наибольшим показателем эффективности займут векторные компьютеры, а лидеры Top500 в новом списке следуют далеко не в первых рядах.

Рассмотрим другую характеристику – отношение размера задачи, при котором достигается половина максимально возможной реальной производительности компьютера $R_{max}/2$, к размеру задачи, при котором тот же компьютер показывает максимальную производительность R_{max} . Для каждой строки вычислим значение $TN = (N_{half}/N_{max}) * 100 \%$. Ясно, что для всех современных компьютеров $N_{half} < N_{max}$, причем N_{max} – это наибольший размер задачи, при котором соответствующая программа еще помещается в оперативной памяти. Дальнейшее увеличение N_{max} неизбежно вызывает свопинг и обращение к диску для подкачки страниц, что сразу снижает производительность.

Предположим, что для некоторого компьютера значение $TN1$ достаточно мало. Это означает, что уже при относительно небольшом размере задачи N_{half1} производительность компьютера достигает половины максимального значения (рис. 9. 2).

Такой компьютер быстро набирает “скорость”, и значительная доля задач с размерами от N_{half1} до N_{max} выполняется с высокой производительностью. Цифрой 1 на рисунке обозначен примерный график роста производительности R такого компьютера в зависимости от размера задачи N .

Допустим, что для второго компьютера значение $TN2$ близко к 100% – цифрой 2 на рисунке отмечен график изменения производительности этого компьютера. В таком случае доля задач размером от минимальных до N_{half2} будет выполняться с крайне низкой производительностью. Чем меньше характеристика TN компьютера, тем быстрее он выходит на максимальное значение производительности. Причем если TN определяет скорость роста

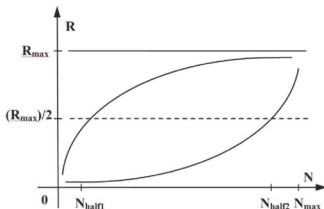


Рис. 9.2. Изменение производительности компьютера в зависимости от соотношения между N_{half1} и N_{max}

производительности R при увеличении N , то площадь фигуры, находящейся под кривой $R(N)$, показывает общий потенциал компьютера при решении исходной задачи. Итак, можно с уверенностью утверждать, что первый компьютер имеет явное преимущество перед вторым.

Если вычислить значение TN для каждой строки рассматриваемого списка Top500, и отсортировать список по возрастанию TN , то список опять же существенно изменится. Снова первые места в получившемся списке займут векторные компьютеры, а не лидеры Top500.

Рассмотрим для каждого компьютера списка Top500 отношение максимального размера задачи к общему числу процессоров в системе, т. е. N_{max}/N_{Proc} . Данная величина показывает, какая часть задачи приходится на каждый процессор. Несмотря на внешнюю простоту, это сложная характеристика – на нее прямо или косвенно оказывают влияние сразу много факторов: объем памяти на каждом процессоре, сбалансированность скорости работы процессора и памяти, взаимодействие процессора, качество коммуникационной сети и др. С одной стороны, такая величина нормирована по числу процессоров, но N_{max} – это размер задачи, при котором достигается максимум производительности R_{max} уже всей вычислительной системы. Чем выше значение данной характеристики, тем более масштабную задачу способен решить процессор системы и тем лучшие свойства имеет система в целом.

Если отсортировать Top500 по характеристике N_{max}/N_{Proc} , то можно увидеть, что в первую десятку опять же войдет большинство векторных компьютеров, занимающих в основном списке Top500 далеко не самые первые места.

Схожие результаты можно получить при анализе большинства других

редакций Top500. Приведенный анализ показывает, что критерий Top500 не является универсальным и нельзя слепо доверяться сложившимся стереотипам. В суперкомпьютерном мире все неоднозначно, в нем есть отработанные годами и опытом решения, а есть – и отголоски текущих технологических успехов. Первые обеспечат преемственность, а вторые могут быстро потерять актуальность.

Данный вывод не призывает отказаться от использования рейтинга Top500, он лишь подчеркивает, что это лишь один из способов ранжирования суперкомпьютеров. Рейтинг Top500 оправдан, по крайней мере, по двум причинам. Во-первых, внимание общественности к этому списку заставляет всех производителей высокопроизводительных вычислительных систем представлять себя в наилучшем свете. Во-вторых, открытость списка не позволяет производителям публиковать явно завышенные данные. Кроме того, такой подход позволяет добиться объективности оценки, поскольку все заинтересованные стороны находятся в одинаковых условиях. А в случае каких-либо сомнений проверить выводы не составит труда.

Приложение 2. Примеры ВВС

Основным параметром классификации параллельных компьютеров является наличие общей (SMP) или распределенной памяти (MPP). Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Кластерные системы являются более дешевым вариантом MPP. При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь, могут объединяться в PVP-системы с использованием общей или распределенной памяти. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем (подробнее в главе 2).

Рассмотрим реализацию некоторых из перечисленных технологий на примере суперкомпьютеров, занимавших в момент их выпуска передовые места в Top500 (например, Blue Gene/L – 1 место в 26-ой редакции, Roadrunner – 1 место в 32-ой редакции, Cray XT5-HE Jaguar – 1 место в 34-ой редакции), а также суперкомпьютеров, разработанных в России и занимавших в момент их выпуска первые места в Top50 (например, "Ломоносов" – 1 место в 12-ой редакции).

Blue Gene

В качестве примера системы MPP рассмотрим **BlueGene** [11, 24]. Blue Gene – проект компьютерной архитектуры, разработанный для нескольких суперкомпьютеров. Проект был запущен фирмой IBM в 1999 году. Его целью являлось построение суперкомпьютера с производительностью в 1 PFlop/s, в первую очередь предназначенного для решения задач бимолекулярного моделирования. Была объявлена предварительная стоимость реализации проекта в \$100 млн, в дальнейшем значительно превышенная.

Целью проекта BlueGene было построение MPP-машины, которая не только была бы самой быстрой, но и самой эффективной в отношении показателей терафлоп/доллар, терафлоп/ватт и терафлоп/м³. По этой причине в

IBM отказались от принципов, которые были положены в основу разработки предыдущих MPP-машин и согласно которым применялись самые быстрые компоненты независимо от их цены. Вместо этого было решено выпустить собственный однокристалльный компонент, работающий с умеренной скоростью и обладающий низким энергопотреблением, чтобы на его основе построить большую машину с эффективным расположением компонентов.

На начало 2006 года была реализована предварительная часть проекта, **IBM Blue Gene/L**. В 26-ю редакцию списка Top500 (ноябрь 2005 года) вошли 19 систем Blue Gene/L, в том числе на 1-ом и 2-ом местах. В максимальной конфигурации суперкомпьютер Blue Gene/L (занявший 1-е место) объединял 131072 процессора, что позволило получить пиковую производительность 360 TFlop/s.

Blue Gene/L является массивно-параллельным компьютером и имеет масштабируемую сотовую архитектуру – необходимый компьютер собирается из однотипных стоек. Одна стойка Blue Gene, занимающая менее одного квадратного метра площади, объединяет до 1024 вычислительных узлов и обладает быстродействием в 5.7 TFlop/s (*рис. 9. 3*). Основные характеристики компьютера Blue Gene/L:

- вычислительный узел: PowerPC 440 700MHz, двухпроцессорный (в штатном режиме один процессор – вычислительный, второй занимается коммуникациями; при отсутствии пересылок оба процессора могут работать как вычислительные);
- пиковая производительность вычислительного узла: 5.6 GFlop/s (2.8 GFlop/s на процессор);
- оперативная память: 512 MB SDRAM-DDR на узел (максимально до 2 Гбайт), пропускная способность канала процессор-память 5.5 Гбайт/сек;
- коммуникационная сеть: 3-мерный тор (в максимальной конфигурации 64x32x32 узла), пропускная способность 175 Мбайт/с;
- транспортные и сервисные сети: Gigabit Ethernet для мониторинга и управления, Gigabit Ethernet для ввода-вывода, специальная сеть синхронизации, древовидная сеть для редукционных операций и бродкаста (время выполнения бродкаста или глобального суммирования около 2.5 мкс., пропускная способность каждой ветки 2.8 Гбайт/с.);
- комплектация стойки: до 1024 двухпроцессорных вычислительных узлов, 16-128 двухпроцессорных узлов ввода-вывода;
- операционная система: вычислительные узлы – специальное легковесное ядро, узлы ввода-вывода – встраиваемый Linux, управляющие узлы – SuSE SLES 9 Linux;

- производительность на стойку: пиковая - 5.73 TFlop/s, на тесте LINPACK – 4.71 TFlop/s;
- производительность в полной конфигурации: пиковая – 360 TFlop/s, на тесте LINPACK – 280.6 TFlop/s;
- энергопотребление: на стойку – до 27.6 КВт, в полной конфигурации вместе с системами охлаждения – около 2 МВт.

BlueGene/L

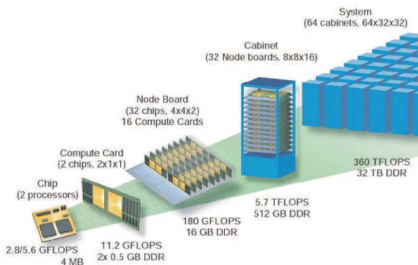


Рис. 9.3. Структура суперкомпьютера Blue Gene/L

Программное обеспечение компьютера Blue Gene/L включает: MPI (на основе MPICH2) для C, C++, Fortran, Co-Array FORTRAN (CAF), Unified Parallel C (UPC), Engineering and Scientific Subroutine Library (ESSL), Mathematical Acceleration Subsystem (MASS), Fast Fourier Transforms (FFT) и 3D-FFT, General Parallel File System (GPFS), LoadLeveler. ESSL предоставляет более 150 специально оптимизированных математических функций, MASS – набор оптимизированных математических библиотек для вычисления дробей и квадратных корней с одинарной и двойной точностью, Fast Fourier Transforms (FFT) и 3D-FFT – реализации быстрого преобразования Фурье, GPFS – высокопроизводительная кластерная файловая система, LoadLeveler – менеджер ресурсов.

В 32-ой редакции списка Top500 максимальная конфигурация суперкомпьютера Blue Gene/L, включающая 212992 процессора, заняла 4-е место,

показав пиковую производительность 596 TFlop/s. Данный суперкомпьютер установлен в Ливерморской национальной лаборатории имени Лоуренса.

Следующее после Blue Gene/L семейство компьютеров архитектуры Blue Gene получило название **Blue Gene/P**. С 2008 года на факультете вычислительной математики и кибернетики МГУ имени М. В. Ломоносова работает суперкомпьютер IBM Blue Gene/P, являющийся одной из первых систем данной серии среди установленных в мире [40].

IBM Blue Gene/P – массивно-параллельная вычислительная система, состоящая из двух стоек, включающих 8192 процессорных ядра (2×1024 четырехъядерных вычислительных узлов), с пиковой производительностью 27,9 TFlop/s. Она принадлежит к новому семейству суперкомпьютеров, обладающих высокой производительностью, масштабируемостью, возможностью обрабатывать данные большего объема, потребляя при этом значительно меньше энергии и занимая меньшую площадь по сравнению с предыдущими системами.

Архитектура Blue Gene / P спроектирована для вычислительных кодов, которые хорошо масштабируются до сотен и тысяч процессоров. Несмотря на то, что индивидуальные процессорные ядра системы Blue Gene / P работают на относительно низкой частоте, для приложений, способных эффективно использовать большое число процессорных элементов, удастся достигнуть значительной производительности.

Характеристики системы:

- комплектация стойки: 1024 четырехъядерных вычислительных узла в каждой из стоек, 16 узлов ввода-вывода в стойке;
- энергопотребление: высокая энергоэффективность 372 MFlop/s/W;
- система воздушного охлаждения;
- вычислительный узел: четыре микропроцессорных ядра PowerPC 450 (4-way SMP, 850 MHz);
- пиковая производительность вычислительного узла: $4 \text{ cores} \times 3,4 \text{ GFlop/s per core} = 13,6 \text{ GFlop/s}$;
- оперативная память: 2 Гб на узел, пропускная память 13,6 Гбайт/с;
- коммуникационная сеть: 3-мерный тор (в максимальной конфигурации $64 \times 32 \times 32$ узла), пропускная способность 175 Мбайт/с;
- коммуникационная сеть: 3-мерный тор (three-dimensional torus), пропускная способность 425 Мбайт/с;
- операционная система: Вычислительные узлы – легковесное ядро, представляющее собой Linux-подобную операционную систему, поддерживающую значительное подмножество Linux-совместимых системных вызовов.

Программное обеспечение:

– Компиляторы:

- IBM XL C/C++ Advanced Edition for Blue Gene/P V9.0;
- IBM XL Fortran Advanced Edition for Blue Gene/P V11.1;
- GNU Compiler Collection 4.1.2.

– Системное ПО:

- IBM Tivoli Workload Scheduler LoadLeveler Version 3 Release 4.2 PTF 1.

Более крупные суперкомпьютеры серии Blue Gene/P в момент выпуска занимали лидирующие позиции в Top500. Например, в 32-ой редакции списка Top500 максимальная конфигурация суперкомпьютера Blue Gene/P (163840 процессора, максимальная производительность 450.3 TFlop/s) заняла 5-е место, а в 34-ой редакции максимальная конфигурация суперкомпьютера Blue Gene/P заняла 4-ое место (294912 процессора, максимальная производительность 825.5 TFlop/s).

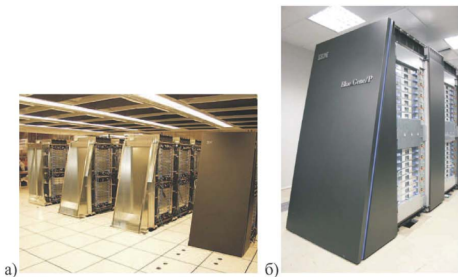


Рис. 9.4. а) Суперкомпьютер Blue Gene/L б) Суперкомпьютер Blue Gene/P

Red Storm

В качестве второго примера систем MPP рассмотрим разработанный в национальной лаборатории Sandia суперкомпьютер Red Storm [11, 24]. Лаборатория Sandia выполняет секретные и несекретные задания департамента энергетики США. Среди секретных работ можно назвать моделирование ядерных взрывов, требующее очень интенсивных вычислений. В лаборатории Sandia в течение десятилетий отдавалось предпочтение векторным суперкомпьютерам, но в определенный момент, благодаря развитию технологии и изменениям в экономике, на смену им стали приходиться MPP-машины.

В 2002 году в Sandia решили заменить ASCI Red, выбрав в качестве долгосрочного поставщика суперкомпьютеров компанию Cray Research. Новая система была поставлена в августе 2004 года, что очень быстро для разработки и реализации столь большого суперкомпьютера. Причина такой оперативности состоит в том, что Red Storm построен почти исключительно из обычных имеющихся в продаже компонентов. Исключение составляет только специализированная микросхема, используемая для маршрутизации.

При построении BlueGene IBM использовала подход, заключающийся в построении из существующих ядер, хотя и специализированной, но низкоскоростной и дешевой в массовом производстве микросхемы, а затем объединении не слишком скоростной сетью очень большого количества этих микросхем. В Sandia выбрали другой подход, заключающийся в том, чтобы взять наиболее мощный из имеющихся в продаже 64-разрядный процессор, снабдить его высокоскоростным специализированным маршрутизатором и большим объемом памяти, получив гораздо более мощные вычислительные узлы, чем узлы BlueGene. Поскольку таких узлов требуется гораздо меньше, то и обмен информацией между ними будет происходить, соответственно, быстрее.

Для Red Storm был выбран процессор Opteron производства компании AMD. Этот выбор обусловили несколько его ключевых характеристик. Первая – поддержание трех режимов работы. В унаследованном режиме на этом процессоре без всякой модификации можно выполнять обычные программы, рассчитанные на Pentium. В режиме совместимости операционная система работает как 64-разрядная и может адресовать до 2^{64} байт памяти, в то время как прикладные программы являются 32-разрядными. Наконец, в 64-разрядном режиме машина целиком становится 64-разрядной и может адресовать все 64-разрядное адресное пространство. Причем в 64-разрядном режиме одновременно могут работать и 32-разрядные, и 64-разрядные программы, что

упрощает обновление системы.

Еще одной ключевой характеристикой Opteron является тщательная проработка вопросов пропускной способности памяти. В последние годы процессоры становились все быстрее и быстрее, заметно опережая в этой гонке память. В результате, в случае кэш-промаха в кэше второго уровня время обращения к памяти значительно возрастает. Инженеры AMD установили в процессор Opteron контроллер памяти, работающий на частоте процессора, а не на частоте шины памяти, что повышает производительность памяти. Контроллер может работать с восемью модулями DIMM по 4 Гбайт каждый, что дает максимальный объем памяти в 32 Гбайт. В системе Red Storm для каждого процессора Opteron устанавливается 2-4 Гбайт, но со временем, по мере удешевления памяти, это значение планируется увеличивать. Другая возможность повышения производительности системы – замена процессоров Opteron двухъядерными моделями, что теоретически должно удвоить вычислительную мощность.

Каждому процессору Opteron выделяется собственный специализированный сетевой процессор под названием Seastar производства IBM. Это критически важный элемент системы, так как практически весь обмен информацией между процессорами происходит через сеть Seastar. Без высокоскоростной коммуникационной сети, функционирование которой поддерживают эти микросхемы, система быстро бы “утонула” в данных.

Хотя процессоры Opteron – это обычные процессоры, имеющиеся в продаже, в Red Storm они устанавливаются в собственные специализированные платы. На каждой такой плате располагаются 4 процессора Opteron, операоеративная память на 4 Гбайт, 4 процессора Seastar, процессор RAS (Reliability, Availability and Service – надежность, работоспособность и удобство эксплуатации), микросхема Ethernet на 100 Мбит/с.

Набор из восьми плат устанавливается в панель и вставляется в кассету. В каждой стойке есть 3 таких кассеты, что позволяет установить 96 процессоров Opteron, а также необходимые источники питания и систему охлаждения. Вся система состоит из 108 стоек, что дает 10 368 процессоров с модулями памяти SDRAM объемом 10 Тбайт. Каждый процессор имеет доступ только к своему модулю SDRAM, общей памяти нет. Теоретическая вычислительная мощность системы равна 41 терафлоп/с.

Для взаимосвязи отдельных центральных процессоров Opteron служат специализированные маршрутизаторы Seastar, по одному маршрутизатору на процессор. Они соединены друг с другом в трехмерный тор размером 27 x 16 x 24, в каждом узле которого находится один маршрутизатор. У каждого

маршрутизатора 7 двусторонних высокоскоростных (24 Гбит/с) линий связи. Шесть из них ведут к соседям: на север, восток, юг, запад, вверх и вниз, а еще одна соединяет маршрутизатор с процессором Opteron. Время передачи между соседними узлами решетки составляет 2 мкс. Для прохождения всего набора вычислительных узлов требуется 5 мкс. Еще одна сеть построена на основе технологии Ethernet со скоростью 100 Мбит/с и служит для обслуживания и поддержки системы.

В дополнение к 108 вычислительным стойкам в системе имеется 16 стоек для процессоров ввода-вывода и обслуживания. В каждом из них установлено 32 процессора Opteron. Из этих 512 процессоров 256 отвечают за ввод-вывод и 256 – за обслуживание. Остальное место занимают диски, организованные в RAID-массивы уровня 3 и 5, каждый с диском четности и диском “горячей” замены. Общий объем дискового пространства составляет 240 Тбайт. Объединенная производительность дисковой системы равна 50 Гбайт/с.

Система при помощи механических переключателей разбивается на две части, секретную и несекретную, которые при необходимости можно разъединять. Из общего числа процессоров 2688 всегда находятся в секретной секции, еще 2688 – всегда в несекретной. Остальные 4992 вычислительных процессора можно переключать между секциями. Все процессоры Opteron из секретной секции имеют по 4 Гбайт памяти, все остальные – по 2 Гбайт. Процессоры ввода-вывода и обслуживания поделены между секциями.

Вся система располагалась в отдельном здании площадью 2000 м², спроектированном и построенном так, чтобы в будущем при необходимости можно было бы расширить систему до 30 000 процессоров. Энергопотребление вычислительных узлов составило 1,6 МВт, еще 1 МВт потребляют диски. Вместе с системой вентиляции и кондиционирования воздуха общее энергопотребление равнялось 3,5 МВт.

Стоимость аппаратного и программного обеспечения компьютера равнялась 90 млн долларов, здание и вентиляция еще 9 млн, что в общем обошлось немногим меньше 100 млн долларов.

Вычислительные узлы работают под управлением облегченного ядра, названного Catamount (“дикая кошка”). Узлы ввода-вывода управляются обычной ОС Linux с небольшими дополнениями, связанными с поддержкой интерфейса MPI. В RAS-узлах работает урезанная версия Linux. На Red Storm можно запускать имеющиеся в изобилии программы для ASCI Red, среди которых есть: программы выделения процессоров, планировщики, MPI-библиотеки, математические библиотеки, прикладные программы.

Для такой большой системы вопросы надежности выходят на первый план.

На каждой плате имеется RAS-процессор, предназначенный для обслуживания системы, а также специализированные аппаратные средства. Целью разработчиков было обеспечить показатель **MTBF** (Mean Time Between Failures – среднее время наработки на отказ), равный 50 часам. У аппаратного обеспечения ASCII Red этот показатель был равен 900 часам, но все портила операционная система – полный отказ системы случался каждые 40 часов. И хотя новая аппаратура намного надежнее старой, слабым местом по-прежнему остаются программы.

Дальнейшее развитие проекта Red Storm привело к созданию **суперкомпьютера Red Storm Cray XT3**, занявшего 9 место в 32-ой редакции списка Top500.

Основные характеристики суперкомпьютера:

- число процессоров: 38208;
- процессоры: двухъядерный AMD Opteron™ 2.4 GHz и четырехъядерный AMD Opteron™ 2.2 GHz;
- память: 75 Тбайт DDR памяти;
- дисковое пространство: 1753 Тбайт;
- коммуникационная сеть: 3-мерный топ;
- системное ПО: Linux/Catamount Operating Systems;
- энергопотребление: 2,5 МВт;
- пиковая производительность на тесте LINPACK - 284.16 TFlop/s.

Дальнейшим развитием системы Cray XT3, является суперкомпьютер Cray XT4, отличаясь от неё в основном использованием четырехъядерных процессоров AMD Opteron. Согласно 32-ой редакции списка Top500 Cray XT4 Franklin, содержащий 38642 процессора QuadCore 2.3 GHz, занимает 7 место, показав пиковую производительность 355.51 TFlop/s.

Roadrunner

Согласно 32-ой редакции списка Top500 вышедшей в ноябре 2008 года суперкомпьютер Roadrunner [11, 53], созданный компанией IBM для Министерства энергетики США и установленный в Лос-Аламосской национальной лаборатории в Нью-Мексико (США) являлся самым производительным суперкомпьютером в мире. В 34-ой редакции списка Top500 он занимает уже второе место.

Министерство энергетики США планирует использовать RoadRunner для расчёта старения ядерных материалов и анализа безопасности и надёжности

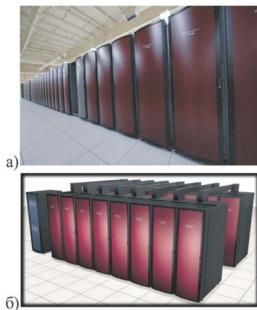


Рис. 9.5. а) Суперкомпьютер Red Storm б) Суперкомпьютер Red Storm Cray XT3

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



Рис. 9.6. Суперкомпьютер IBM Roadrunner

Суперкомпьютер Roadrunner построен по гибридной схеме из процессоров AMD Opteron, работающих на частоте 1.8 ГГц, и процессоров IBM PowerXCell 8i, работающих на частоте 3.2 ГГц, в специальных блэйд-модулях TriBlades, соединенных с помощью коммуникационной сети Infiniband.

В Roadrunner задействовано 6912 двухъядерных процессоров AMD Opteron

2210: 6120 вычислительных и 442 процессора для системных функций (13124 процессорных ядра: 12240 и 884 соответственно), также 12240 процессоров IBM PowerXCell 8i, что составляет 12240 ядер PPE и 97920 ядер SPE (всего 110160 ядер). Каждый процессор IBM PowerXCell 8i включает в себя одно универсальное ядро Power (PPE) и 8 специальных ядер для операций с плавающей точкой (SPE).

TriBlade состоит из четырех ядер Opteron, четырех PowerXCell 8i процессоров, 16 GB Opteron'овой и 16 GB Cell'овой RAM. Блэйд-модули входят в объединенные модули, каждый из которых – это 60 BladeCenter H с установленными TriBlades, всего 180 TriBlades. Все TriBlades подсоединены к 288-портовому Voltaire ISR2012 Infiniband маршрутизатору. Каждый модуль также подсоединён к файловой системе Panasas через 12 машин System x3655. Системная информация по модулю:

- 360 двухядерных Opteron с 2.88 Тбайт RAM;
- 720 PowerXCell ядер с 2.88 Тбайт RAM;
- 12 System x3655 с двумя 10ГBit Ethernet каждый;
- 288-port Voltaire ISR2012 маршрутизатор с 192 Infiniband 4x DDR (180 TriBlades и 12 I/O узлов).

Кластер Roadrunner собран из 18 объединённых модулей, соединённых через восемь дополнительных ISR2012 маршрутизаторов. Каждый модуль подсоединён через 12 каналов к каждому маршрутизатору – суммарно 96 соединений. Суммарная информация:

- 6,480 двухъядерных Opteron с 51.8 Тбайт RAM (на 3,240 LS21);
- 12,960 Cell процессоров с 51.8 Тбайт RAM (на 6,480 QS22);
- 216 System x3655 I/O узлов;
- 26 288-port ISR2012 Infiniband 4x DDR маршрутизаторов;
- 296 корпусов.

Roadrunner работает под управлением Red Hat Enterprise Linux и управляется при помощи программного обеспечения xCAT.

Установка занимает площадь 560 квадратных метров и весит 226 тонн. Общее энергопотребление установки – 2.35 МВт, энергоэффективность составляет 437 MFlop/s/Вт. Стоимость IBM Roadrunner – 133 миллиона долларов. Пиковая производительность 1.456 PFlop/s, производительность на тесте LINPACK – 1.105 PFlop/s.

Cray XT5h

Cray XT5h является одним из первых суперкомпьютеров на основе гибридной архитектуры [11]. Он объединяет в единую систему скалярные и векторные процессоры, а также процессоры на основе программируемых логических интегральных схем (подробнее в главе 9). Создание такого суперкомпьютера компания Cray обуславливает тем, что скалярные процессоры хорошо подходят для вычислений, не требующих большого числа обращений к памяти, а ПЛИС и векторные элементы предназначены для использования в тех задачах, для которых характерны высокая загруженность памяти и большие объемы обрабатываемых данных. Поэтому их совместное применение может позволить эффективнее решать современные сложные вычислительные задачи.

В качестве скалярных процессоров в системе Cray XT5h используются двух- и четырехъядерные процессоры AMD Opteron; векторные вычисления и вычисления на основе ПЛИС производятся на блэйд-серверах Cray X2 и Cray XR1 соответственно. Все вычислительные элементы объединены общей сетью SeaStar2+.

Векторный блэйд-сервер **Cray X2** состоит из двух независимых вычислительных узлов, каждый из которых, в свою очередь, состоит из 4 векторных процессоров, работающих над общей памятью размером от 32 до 64 Гбайт. В пределах узла поддерживается однородный доступ к памяти, а на уровне одного блэйд-сервера обеспечивается когерентность кэшей. Производительность каждого вычислительного узла достигает 100 GFlop/s и более на 64-разрядной архитектуре. Система масштабируется до 32000 процессоров, при этом их работа обеспечивается в едином адресном пространстве. Между собой различные серверы Cray X2 объединены высокоскоростной сетью с топологией "толстого дерева".

Наличие специализированной среды программирования позволяет писать программы для векторных процессоров на стандартных языках Си и Фортран, которые затем будут скомпилированы в соответствующий векторным вычислениям код.

Суммарная информация:

- Данные по процессору:

- одноядерный 64-разрядный векторный процессор Cray XT2 с частотой 1.6 ГГц, каждые 4 процессора объединяются в вычислительный узел;
- 8 векторных каналов;
- кэш L2 – 512 Кбайт, кэш L3 – 8 Мбайт.

- Число процессоров:
 - 128 процессоров в стойке (максимально), 32768 процессоров в системе (максимально);
 - изначально предоставляется 1024 процессора, дальнейшая масштабируемость по требованию.
- Пиковая производительность: 25.6 GFlop/s на каждом процессоре.
- Глобальная адресация:
 - внутри одного узла для всех процессоров поддерживается однородный доступ к общей локальной памяти на основе SMP-модели;
 - внутри всей системы между различными вычислительными узлами поддерживается неоднородный доступ к памяти;
 - обеспечивается глобальная когерентность кэшей.
- Основная память: 32, 64 или 128 (планируется) Гбайт DRAM на узел, 16 Тбайт на систему (при 1024 процессорах).
- Скорость доступа к памяти: 28.5 Гбайт/сек для каждого процессора.
- Коммуникации: используется топология "толстого дерева" скорость передачи данных от одного узла к другому - 15 Гбайт/сек.
- Системные программные средства:
 - ОС Linux на всех узлах;
 - планировщик ALPS, поддерживающий такие системы как PBSPro и LSF;
 - программа для системного администрирования Mazama;
 - параллельная файловая система Lustre, работающая на встроенном RAID-хранилище, напрямую подключенном к общей системе Cray XT5h;
 - система детального учета процессов, проектов и задач.

1. Программное обеспечение:

- компиляторы C/C++ и Fortran;
- gcc;
- Unified Parallel C (UPC);
- Co-Array Fortran (CAF);
- поддержка OpenMP внутри узла Cray X2;
- инструменты анализа производительности Cray PAT и Cray Apprentice2;
- MPI на основе MPICH2;

- Shmem;
- отладчик TotalView;
- gdb для однопроцессорных приложений;
- различные научные библиотеки.

2. Физические характеристики:

- 2045 мм * 1080 мм * 1607 мм (В*Ш*Г);
- максимальный вес одной стойки – 1134 кг;
- энергопотребление 45 кВт на стойку;
- уровень шума – 75 dba на высоте 1 метр.

Реконфигурируемый блэйд-сервер Cray XR1 состоит из двух узлов, в каждый из которых входит процессор AMD Opteron, связанный с помощью высокоскоростной шины HyperTransport с двумя ПЛИС-процессорами Xilinx Virtex-4. Масштабируемость системы позволяет объединять до 30000 ПЛИС-процессоров в одной системе. Для работы с ПЛИС-процессорами предлагается набор функций API, которые могут быть вызваны пользователем при написании исходного кода программы. Изменение архитектуры ПЛИС-процессоров осуществляется программно с помощью прикладных пакетов компаний Celoxica, Synplicity, Mitronics и др., при этом различные ПЛИС-процессоры могут быть настроены для решения разных задач.

Суммарная информация:

- Модель ПЛИС: Xilinx Virtex-4 LX200.
- Количество логических ячеек на ПЛИС: 200448.
- Сокет: Socket 940.
- Потребляемая мощность: 12 кВт на стойку.
- Версия интерфейса HT: HyperTransport 1.0.
- Скорость обмена с ПЛИС по HT: 400 МГц * 16 бит или 3.2 Гбайт/сек.
- Память процессора Opteron: 2-8 Гбайт, 128-бит DDR 400, 6.4 Гбайт/сек.
- Объем памяти RDRAM на ПЛИС: 256 Мбайт.
- Пропускная способность шины HT, на одно соединение: 400 МГц * 16 бит или 3.2 Гбайт/сек.
- Программное обеспечение: набор API, совместимых с различными инструментами для программирования архитектуры ПЛИС.

Система Cray XT5h позволяет варьировать число используемых серверов с разной архитектурой, поэтому пользователь может конструировать наиболее подходящую систему с учетом собственных требований.

Программное обеспечение. Суперкомпьютер Cray XT5h поддерживает языки, работающие с моделью PGAS (partitioned global address space, параллельное глобальное адресное пространство), Co-Array Fortran (CAF) и

Unified Parallel C (UPC). Это позволяет программистам при распределении данных и проводимых вычислениях руководствоваться моделью программирования ОКМД (один поток команд, множество потоков данных). В системах с общим адресным пространством эти языки могут использоваться в качестве альтернативы основанным на передаче сообщений MPI и SHMEM.

В состав суперкомпьютера Cray XT5h входит различное программное обеспечение, которое призвано помочь пользователю использовать ресурсы суперкомпьютера более эффективно – компиляторы для векторных процессоров, различные прикладные библиотеки, отладчики и инструменты анализа производительности.

Cray XT5h работает на ОС Linux и обеспечивает единую точку доступа ко всем ресурсам и блэйд-серверам, а также поддерживает распределенную файловую систему и коммуникацию на основе сокетов. Для распределения заданий между скалярными, векторными и ПЛИС-процессорами существует планировщик **ALPS** (Application Level Placement Scheduler), который может быть интегрирован с системами обработки заданий, что гарантирует доступ к ресурсам для различных запросов.

Файловая система Lustre, используемая в Cray XT5h, предоставляет приложениям и вычислительным ресурсам совместный доступ к общим файлам, при этом результат работы одного приложения может сразу поступать на вход другого приложения в рамках данного потока.

Дальнейшее развитие данного направления привело к появлению суперкомпьютера Cray XT5 Jaguar, занявшего второе место в Top500 от ноября 2008 года с производительностью 1,059 PFlop/s.

Одна из следующих версий суперкомпьютера Cray XT5-HE Jaguar (максимальная производительность 1759 TFlop/s) заняла первое место в 34 списке Top500 от ноября 2009 года, став на тот момент самым производительным суперкомпьютером в мире.

СКИФ МГУ “Чебышев”

Рассматривая список суперкомпьютеров, являющихся лидерами Top50, несомненно, стоит уделить особое внимание суперкомпьютеру СКИФ МГУ “Чебышев”, представляющему собой совместную разработку МГУ им. М. В. Ломоносова, ИПС РАН и компании Т-Платформы, выполненной при поддержке компании Intel в рамках суперкомпьютерной программы “СКИФ-ГРИД” Союзного государства России и Беларуси и национального приоритетного проекта “Образование”. На момент завершения строительства в марте 2008

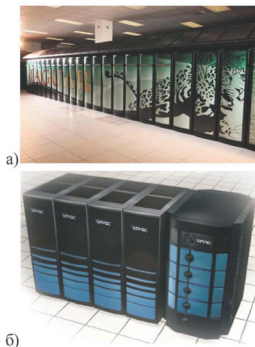


Рис. 9. 7. а) Суперкомпьютер Cray XT5 Jaguar б) Суперкомпьютер Cray XT5h

года СКИФ МГУ являлся самым мощным суперкомпьютером в России, странах СНГ и Восточной Европы (1 место в Top50 от 27 марта 2008 года). В редакции от 23 сентября 2008 года он занимал второе место. Что касается Top500, то в редакции от июня 2008 года “СКИФ МГУ” занимал 36 место, в редакции от ноября 2008 года – 54.

Суперкомпьютерная программа “СКИФ”

Суперкомпьютерная программа “СКИФ” Союзного государства России и Беларуси “Разработка и освоение в серийном производстве семейства высокопроизводительных вычислительных систем с параллельной архитектурой (суперкомпьютеров) и создание прикладных программно-аппаратных комплексов на их основе” реализована в 2000 – 2004 гг. на основании Постановления Исполкома Союза Беларуси и России N 43 от 22 ноября 1999 года [17, 20]. Цель программы: возрождение компьютерной отрасли двух стран, промышленное производство ряда программно-совместимых моделей суперкомпьютеров с широким спектром производительности – до триллионов операций в секунду.

В программе принимало участие по десять предприятий из России и Беларуси. Главными исполнителями программы были от Российской Федерации – Институт программных систем (ИПС) Российской академии наук, от Республики Беларусь – Объединенный институт проблем информатики Национальной академии наук Беларуси (ОИПИ НАН).

За время исполнения программы было создано семейство суперкомпьютеров “СКИФ” (Ряд 1 и Ряд 2) и разработана соответствующая конструкторская документация. Выпущено 16 образцов ЭВМ семейства “СКИФ”, причем производительность моделей “СКИФ” за пять лет выросла в 185 раз (с 11 до 2032 GFlop/s). В ноябре 2004 года суперкомпьютерные установки семейства “СКИФ” заняли 14 % мест в суперкомпьютерном рейтинге Top50 (СНГ) и обеспечили четверть суммарной пиковой производительности этого рейтинга. Две установки, разработанные в рамках программы “СКИФ”, вошли в мировой рейтинг суперкомпьютеров TOP500: “СКИФ К-500” (пиковая производительность 716.8 GFlops, LINPACK-производительность 425.2 GFlops) – 407 место в ноябре 2003 года и “СКИФ К-1000” (пиковая производительность 2,534 GFlops, LINPACK-производительность 2,032 GFlops) – 98 место в ноябре 2004 года.

Создана система, реализующая функции удаленного низкоуровневого управления вычислительными узлами кластера, – управляющая сеть суперкомпьютеров семейства “СКИФ” ServNet. Она реализует такие функции удаленного управления вычислительными узлами, как селективное включение и отключение питания на любом узле; селективное выполнение аппаратного сброса любого вычислительного узла; осуществление взаимодействия с узлом кластера в консольном режиме “Linux serial console”, а также возможность удаленной работы с узлом в режиме BIOS Setup.

В рамках программы “СКИФ” было разработано базовое программное обеспечение кластерного уровня семейства суперкомпьютеров “СКИФ”, в том числе: ядро ОС **Linux-SKIF**; **PVFS-SKIF** (Parallel Virtual File System – SKIF) – параллельная файловая система; **OpenPBS-SKIF** (Open Portable Batch System – SKIF) – система очередей; **FLAME-SKIF** – система мониторинга и управления (reset, power on/off) установками семейства “СКИФ”; **TDB** – распределенный интерактивный отладчик MPI-приложений. В комплект базового программного обеспечения вошла также система поддержки автоматического распараллеливания программ **OpenTS** – оригинальная разработка ИПС РАН и МГУ.

Суперкомпьютерная программа “СКИФ-ГРИД”

Суперкомпьютерная программа Союзного государства Российской Федерации и Беларуси “Разработка и использование программно-аппаратных средств Грид-технологий перспективных высокопроизводительных (суперкомпьютерных) вычислительных систем семейства “СКИФ” (шифр программы “СКИФ-ГРИД”), утверждена в апреле 2007 года со сроком исполнения в 2010 году [20]. Программа выполняется кооперацией исполнителей. Главным исполнителем от Российской Федерации является ИСП РАН, от Республики Беларусь – ОИПИ НАН Беларуси.

Программа имеет целью развитие и поддержку оригинальных отечественных технологий высокопроизводительных вычислений, в том числе, оригинальных средств для создания сервис-ориентированных грид-систем; разработку суперкомпьютеров семейства “СКИФ” (ряд 3 и ряд 4) и их программного обеспечения; развертывание экспериментальной грид-системы “СКИФ-Полигон”, объединяющей ресурсы нескольких отечественных суперкомпьютерных центров; выполнение ряда пилотных проектов.

Основные направления научно-технической программы “СКИФ-ГРИД”:

- **Направление 1. ГРИД-технологии.** Развитие, исследование и внедрение средств высокопроизводительных вычислений на основе ГРИД-технологий с поддержкой гетерогенных, территориально-распределённых вычислительных комплексов.
- **Направление 2. Суперкомпьютеры “СКИФ” ряда 3 и 4.**
Создание суперкомпьютеров “СКИФ” нового поколения (Ряд 3 и 4) на основе исследования и освоения передового мирового опыта, с использованием новых технических решений для соединений узлов (интерконнект), управления кластерами, создания гибридных узлов, применения различного набора реконфигурируемых и специализированных вычислителей. Это направление включает в себя:
 - разработку аппаратных средств;
 - разработку базового программного обеспечения;
 - разработку опытных образцов суперкомпьютеров “СКИФ” ряда 3 и 4.
- **Направление 3. Информационная безопасность.** Создание средств защиты информации в вычислительных комплексах, включая программные и аппаратные средства защиты информации. Разработка системы активного аудита безопасности в ГРИД-сети.
- **Направление 4. Пилотные проекты.**

- Пилотный проект по созданию системы "СКИФ-Полигон" предусматривает развертывание и обеспечение эксплуатации пилотного сегмента распределенной вычислительной ГРИД-системы "СКИФ".
- Научно-исследовательские работы по перспективным областям применения создаваемых вычислительных установок, в том числе: разработка пилотных образцов различных систем; проведение проектов, демонстрирующих возможности создаваемых вычислительных средств, а также исследований, посвященных решению актуальных задач науки и техники на основе использования суперкомпьютеров.

За время, прошедшее с начала исполнения программы "СКИФ-ГРИД", созданы суперкомпьютерные установки, вошедшие в TOP500:

- "СКИФ Cyberia" (пиковая производительность 12 TFlops, LINPACK-производительность 9.019 TFlops) 105 место в редакции от июня 2007 года;
- СКИФ МГУ "Чебышев" – (пиковая производительность 60 TFlops, LINPACK-производительность 47.17 TFlops) 36 место в редакции от июня 2008 года;
- "СКИФ Урал" (пиковая производительность 15.936 TFlops, LINPACK-производительность 12.2 TFlops) 282 место редакции от июня 2008 года.

Описание суперкомпьютера СКИФ МГУ "Чебышев"

СКИФ МГУ "Чебышев" построен на базе новейших технологических решений отрасли и использует целый ряд российских разработок, созданных в рамках суперкомпьютерных программ "СКИФ" и "СКИФ-ГРИД" Союзного государства России и Белоруси [11,20].



Рис. 9. 8. Суперкомпьютер "СКИФ МГУ"

Основу суперкомпьютера составляют blade-модули T-Blade производства “Т-Платформы”, позволяющие разместить 20 четырехъядерных процессоров Intel Xeon с частотой 3.0 ГГц в шасси. Используемый в blade-модулях чипсет – Intel 5400. Общая характеристика:

- пиковая производительность – 60 TFlop/s;
- производительность на LINPACK – 47.04 TFlop/s (78.4 % от пиковой);
- число процессоров/ядер в системе – 1250 / 5000;
- модель процессора – Intel Xeon E5472 3.0 ГГц;
- объём оперативной памяти – 5.5 Тбайт;
- дисковая память узлов – 15 Тбайт;
- число стоек всего/вычислительных – 42 / 14;
- число блэйд-шасси/вычислительных узлов – 63 / 625;
- общая занимаемая площадь – 96 м²;
- общий вес – 30 т;
- номинальное энергопотребление компьютера – 330 кВт;
- общее энергопотребление комплекса – 720 кВт;
- ОС: AltLinux HPC.

Группы вычислительных узлов:

- S – 8 узлов, 2 процессора, 32 Гбайт, HDD 160 Гбайт;
- A – 32 узла, 2 процессора, 16 Гбайт, HDD 160 Гбайт;
- B – 64 узла, 2 процессора, 8 Гбайт, HDD 160 Гбайт;
- C – 529 узлов, 2 процессора, 8 Гбайт.

Все узлы в СКИФ МГУ “Чебышев” связаны тремя независимыми сетями:

- системная сеть: InfiniBand DDR (Mellanox ConnectX)
(Fat Tree: 6x144 порта; латентность на уровне MPI: 1.3-1.95 мкс; скорость обмена на уровне MPI: 1540 Мбайт/с);
- вспомогательная сеть: Gigabit Ethernet
(2x(336 портов+4x10G) + 2x(48 портов+4x10G) + 24x10G);
- управляющая сеть: СКИФ-ServNet+IPMI.

Программное обеспечение:

- ОС: AltLinux HPC;
- параллельная среда: mvapich;
- компиляторы: Intel 10.1, gcc 4.1.2, PathScale, PGI;
- система управления задачами: Cleo-5.12;
- средства разработки: будут установлены Intel Cluster Toolkit, Intel TBB, Intel VTune.

СКИФ МГУ “Чебышев” впервые использует российские программные средства для кластерных систем, разработанные в рамках программы “СКИФ-ГРИД” и включающие отечественный кластерный дистрибутив ОС Linux,

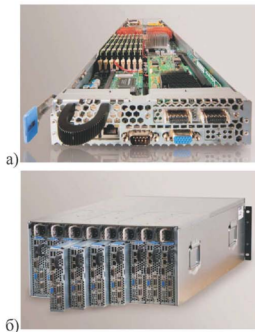


Рис. 9.9. а) Вычислительный модуль T-Blade на базе двух процессоров Intel Xeon E5472 3.0 ГГц б) Blade-масси, включающее 10 модулей T-Blade

интегрирующий весь необходимый для работы кластера набор системного ПО, а также систему мониторинга и управления. Полнофункциональная программно-аппаратная система мониторинга разработки ИПС РАН позволяет удаленно управлять системой на всех уровнях через единый иерархический веб-интерфейс – от распределения процессорного времени для различных задач и управления нагрузкой системы хранения на верхнем уровне до контроля температуры, состояния блоков питания и автоматического изменения скорости вращения вентиляторов на вычислительных узлах.

Суперкомпьютер МГУ “Ломоносов”

По окончании строительства суперкомпьютер “Ломоносов” производства компании “Т-Платформы”, установленный в научно-исследовательском вычислительном центре МГУ, занял первое место в 12-ом списке Top50, став самым производительным суперкомпьютером СНГ. В 34-ой версии Top500 суперкомпьютер МГУ “Ломоносов” занял 12-ое место.



Рис. 9. 10. Суперкомпьютер "Ломоносов"

Максимальная производительность суперкомпьютера МГУ "Ломоносов" оставила 350.10 TFlop/s, а пиковая – 414.42 TFlop/s. Другие параметры суперкомпьютера представлены далее:

- число вычислительных узлов – 4446;
- число процессоров – 8892;
- число процессорных ядер – 35776;
- число типов вычислительных узлов – 3 (T-Blade 2, T-Blade 1.1, платформа на базе процессора PowerXCell 8i);
- основной тип вычислительных узлов – T-Blade 2;
- процессор основного типа вычислительных узлов – Intel© Xeon X5570;
- оперативная память – 55 576 ГБайт;
- общий объем дисковой памяти вычислителя – 166 400 ГБайт;
- занимаемая площадь – 252 кв. м.;
- энергопотребление вычислителя – 1.5 мВт;
- интерконнект – QDR Infiniband;
- система хранения данных – трехуровневая с параллельной файловой системой;

- объем системы хранения данных – до 1350 Тбайт;
- операционная система – Linux Clustrx T-Platforms Edition;
- параллельная среда: openmpi;
- компиляторы: Intel 11, gcc 4.1.2;
- система управления задачами: Cleo-5.12;
- средства разработки: Intel Cluster Toolkit.

Литература

- [1] АФАНАСЬЕВ, К. Е. Многопроцессорные вычислительные системы и параллельное программирование [Текст] / К. Е. Афанасьев, С. В. Стуколов. - Кемерово: Кузбассвуиздат, 2003. - 182 с.
- [2] АФАНАСЬЕВ, К. Е. Многопроцессорные системы: построение, развитие, обучение [Текст] / К. Е. Афанасьев, В. Г. Домрачев, И. В. Ретинская, А. К. Скуратов, С. В. Стуколов / под ред. А. Н. Тихонова – М.: КУДИЦ-ОБРАЗ, 2005. - 224 с.
- [3] БЕРИЛЛО, А. NVIDIA CUDA – неграфические вычисления на графических процессорах [Электронный ресурс] / А. Берилло // <http://www.ixbt.com/video3/cuda-1.shtml>. – 23.09.2008. – 18.02.2010
- [4] БЕРКС, А. Предварительное рассмотрение логической конструкции электронного вычислительного устройства [Текст] / А. Беркс, Г. Голдстейн, Дж. Нейман // Кибернетический сборник. – М.: Мир. – 1964. - Вып. 9.
- [5] БОГДАНОВ, А. Архитектура и топология вычислительных систем [Электронный ресурс] / А. Богданов, В. Мареев, Е. Станнова, В. Корхов // <http://www.informika.ru/text/teach/topolog/index.htm>. – 10.09.2008
- [6] БУКАТОВ, А.А. Программирование многопроцессорных вычислительных систем [Текст] / А. А. Букатов, В. Н. Дацюк, А. И. Жегуло // Ростов-на-Дону. Издательство ООО "ЦВВР 2003. – 208 с.
- [7] ВАЛИЕВ, К. А. Квантовые компьютеры: надежды и реальность [Текст] / К. А. Валиев, А. А. Кокин// Регулярная и хаотическая динамика (РХД) М.- Ижевск. 2001. – 350 с.
- [8] ВОЕВОДИН, В. В. Топ500: числом или уменьем? [Электронный ресурс] / В. В. Воеводин// Открытые системы, <http://www.osp.ru/os/2005/10/380430>. – 20.10.2005. – 25.11.2008.
- [9] ВОЕВОДИН, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин// СПб: БХВ-Петербург, 2002. – 608 с.

- [10] ДЕЙКСТРА, Э. Взаимодействие последовательных процессов [Текст] / Э. Дейкстра // Языки программирования.– М.: Мир, 1972.
- [11] Информационно-аналитический центр по параллельным вычислениям [Электронный ресурс] / <http://parallel.ru>. – 22.05.2008
- [12] КОВАЛЕНКО, В. Н. Организация ресурсов Grid [Электронный ресурс] /В. Н. Коваленко, Д. А. Корягин // http://www.keldysh.ru/papers/2004/prep63/prep2004_63.html. – 25.09.2009
- [13] КОРНЕЕВ, В. А. Архитектуры с распределённой разделяемой памятью [Текст] / В. А. Корнеев // Открытые системы. – 2001. – № 3. – С. 15-23.
- [14] Нейрокомпьютеры и их применение. Книга 5. [Текст] / Под общей ред. А.И. Галушкина, Я.З. Цыпкина // М. ИПРЖР, 2001. – 840 с.
- [15] ОЛИФЕР, В. Г. Компьютерные сети. Принципы, технологии, протоколы. 3-е издание [Текст] / В. Г. Олифер, Н. А. Олифер.– СПб.: Питер, 2006. – 753 с.
- [16] ОЛИФЕР, В.Г. Сетевые операционные системы [Текст] / В. Г. Олифер, Н. А. Олифер.–СПб.: Питер, 2008. – 669 с.
- [17] Программа союзного государства “СКИФ” [Электронный ресурс] / <http://www.skif.bas-net.by/index.html>. – 12.04.2008
- [18] РИХТЕР, Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows [Текст] / Дж. Рихтер; пер. с англ. – 4-е изд.–СПб: Питер, 2004. – 749 с.
- [19] РОМАНЧЕНКО В. AMD Spider: процессоры Phenom, чипсеты 7-Series и не только [Электронный ресурс] / В. Романченко // http://www.3dnews.ru/cpu/amd_phenom. – 20.11.2007. – 12.06.2008
- [20] Суперкомпьютерные программы Союзного государства "СКИФ"и "СКИФ-ГРИД"[Электронный ресурс] / <http://skif.pereslavl.ru/psi-info/rcms-skif/index.ru.html>. – 26.11.2008.
- [21] СТИН, Э. Квантовые вычисления [Текст] / Э. Стин // РХД. Москва-Ижевск, – 2000. – 111 с.
- [22] Суперкомпьютеры Top50 [Электронный ресурс] / <http://supercomputers.ru>. – 25.11.2008.

- [23] ТАНИЕНБАУМ, Э. С. Современные операционные системы [Текст]/ Э. С. Таненбаум; пер. с англ. – 2-е изд. СПб.: Питер, 2002. – 1040 с.
- [24] ТАНИЕНБАУМ, Э. С. Архитектура компьютера [Текст]/ Э. С. Таненбаум // СПб.: Питер, 2007. – 844 с.
- [25] ТАРКОВ, М. С. Введение в операционные системы [Текст] / М. С. Тарков // Новосибирск: ЗАО “АЛГрупп”, 1999. – 95 с.
- [26] ТОПОРКОВ, В. В. Модели распределённых вычислений [Текст] / В. В. Топорков // М.:ФИЗМАТЛИТ, 2004. – 320 с.
- [27] ФОСТЕР, Я. What is the Grid? A Three Point Checklist [Электронный ресурс] / Я. Фостер // <http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf> – 7.08.2009.
- [28] ФРОЛОВ, А. В. Программирование для Windows NT [Текст] / А. В. Фролов, Г. В. Фролов // М.: Диалог-МИФИ, 1996. – 272 с.
- [29] ХАЙКИН, С. Нейронные сети: полный курс [Текст] / С. Хайкин // М.: Вильямс, 2006. – 1104 с.
- [30] ЧАСТНИКОВ, А. П. История компьютера [Текст] / А. П. Частников // М.: Информатика и образование, 1996. – 128 с.
- [31] Слияние двух лун: технология AMD Fusion [Текст] // <http://www.xard.ru/post/12548/default.asp>. – 07.12.2007. – 17.02.2010.
- [32] СуперЭВМ в России [Электронный ресурс] // Электроника: Наука, Технология, Бизнес. – 2000. № 4. – С. 5-9.
- [33] ЧЕКАНОВ, Д. История AMD в процессорах [Электронный ресурс] / Д. Чеканов // www.thg.ru/cpu/amd_cpu_history/index.html. – 4.09.2008. – 2.10.2008.
- [34] ЧЕКАНОВ, Д. nVidia CUDA: вычисления на видеокарте или смерть CPU? [Электронный ресурс] / Д. Чеканов // http://www.thg.ru/graphic/nvidia_cuda/onepage.html. – 22.06.2008. – 27.01.2010
- [35] ШПАКОВСКИЙ, Г. И. Организация параллельных ЭВМ и суперскалярных процессоров: учеб. пособие. [Текст] // Г. И. Шпаковский. Мн.: Белгосуниверситет, 1996. – 284 с.

- [36] ЭДМУНД, М. “Верификация моделей программ: Model Checking” [Текст] / Эдмунд М. Кларк мл., О. Грамберг, Д. Пелед. – Москва: издательство московского центра непрерывного математического образования, 2002. – 416 с.
- [37] About the Globus Toolkit [Электронный ресурс] / <http://www.globus.org/toolkit/about.html>. – 7.08.2009.
- [38] AHUJA, S. Linda and friends [Текст] / S. Ahuja, N. Carreriro, D. Gelernter // Computer. – 1986.- V. 19, № 8 . – P. 26-34.
- [39] BASU, A. Parallel Processing Systems: a Nomenclature based on their Characteristics [Текст] / A. Basu // Proc. IEE(UK). – № 134.– 1987.– P.143-147.
- [40] Blue Gene/P BMK МГУ [Электронный ресурс] / <http://hpc.cmc.msu.ru/bgp>. – 15.05.2008
- [41] BURKS, A. W. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument [Текст]/ A. W. Burks, H. H. Goldstine, J. von Neumann // 1946.
- [42] Centre for Quantum Technologies in Singapore [Электронная публикация] / <http://www.quantumlah.org> - 12.02.2008
- [43] CUDA Zone [Электронный ресурс] // http://www.nvidia.ru/object/cuda_home_new_ru.html
- [44] FENG, T. Some Characteristics of Assotiative Parallel Processing [Текст]/ T. Feng // Proc. 1972 Sagamore Computing Conf. – 1972. – P. 5-16.
- [45] FLYNN M. Some Computer Organizations and Their Effectiveness [Текст]/ M. Flynn // IEEE Trans. Comput. – Vol. C-21. – P. 948. – 1972.
- [46] HANDLER, W. The Impact Classification Schemes on Computer Architecture [Текст]/ W. Handler // Proc. Int'l Conf. on Parallel Processing. – 1977. – P. 7-15.
- [47] HANSEN, P. B. Model programs for computational science: a programming methodology for multicomputers [Текст]/ P. B. Hansen // Concurrency: Practice and Experience. – 1993. – V. 5. – P. 407-423.
- [48] HOCKNEY, R. Classification and Evaluation of ParallelComputer Systems [Текст]/ R. Hockney // Lecture Notes in Computer Science. 1987. – N 295. – P. 13-25.

- [49] HOCKNEY, R. Parallel Computers: Architecture and Performance [Текст]/ R. Hockney // Proc. of Int. Conf. Parallel Computing'85. – 1986. – P.33-69.
- [50] HWANG K. Computer Architecture and Parallel Processing [Текст]/ K. Hwang, F. A. Briggs. – 1984. – P.32-40.
- [51] KRAMMER B. Runtime Checking of MPI Applications with MARMOT [Электронный ресурс]/ B. Krammer, M. Mueller, M. Resch // <http://www.hlr.de/organization/amt/projects/marmot/pubs/parco2005-marmot.pdf>. – 2005.
- [52] Larrabee: A Many-Core x86 Architecture for Visual Computing [Текст] //ACM Transactions on Graphics, 2008. – Vol. 27. – № 3. – Article 18.
- [53] Roadrunner installed at Los Almos national laboratory [Электронный ресурс] / <http://www.lanl.gov/roadrunner>. – 15.05.2008
- [54] SIEGEL, S. Verifying Parallel Programs with MPI-Spin [Текст]/ S. Siegel // Proceedings of the 14th European PVM/MPI Users' Group Meeting, Paris, France, September/October 2007. – P. 13-14.
- [55] SKILLICORN, D. A Taxonomy for Computer Architectures [Текст]/ D. Skillicorn // Computer. – 1988. – Vol. 21. – N 11. – P. 46-57.
- [56] SNYDER, L. A Taxonomy of Synchronous Parallel Machines [Текст]/ L. Snyder // University Park. Penn. – 1988. – P. 281-289.
- [57] Supercomputers sites Top500 [Электронный ресурс] / <http://top500.org>. – 25.11.2008.
- [58] Teraflops Research Chip [Электронный ресурс] / <http://techresearch.intel.com/articles/Tera-Scale/1449.htm> . – 17.06.2008
- [59] The Industry-Changing Impact of Accelerated Computing [Электронный ресурс] / http://sites.amd.com/us/Documents/AMD_fusion_Whitepaper.pdf. – 4.12.2009
- [60] UltraSPARC Architecture 2005 Draft D0.8.3 [Электронный ресурс] / <http://opensparc-t1.sunsource.net/specs/UA2005-current-draft-HP-EXT.pdf>. – 7.02.2006. – 12.06.2008
- [61] UltraSPARC T1 processor [Электронный ресурс] / <http://www.sun.com/processors/UltraSPARC-T1>. – 12.08.2008

- [62] WEAVER, D. L. The SPARC Architecture Manual Version 9 [Текст] / D. L. Weaver, T. Germond // PTR Prentice Hall, 1994. – 399 с.
- [63] Кластерные системы [Электронный ресурс] / <http://www.gridclub.ru/software/cluster.html>. – 5.08.2009

Афанасьев Константин Евгеньевич
Завозкин Сергей Юрьевич
Трофимов Сергей Николаевич
Власенко Андрей Юрьевич

Основы высокопроизводительных вычислений

Том I

Высокопроизводительные вычислительные системы

Учебное пособие

Редактор Л. М. Борискина

Подписано в печать 10.02.2011 г. Формат 60x84 $\frac{1}{16}$.

Печать офсетная. Бумага офсетная № 1. Уч. изд. л. 15,4.

Тираж 100 экз. Заказ № 98.

ФГБОУ ВПО «Кемеровский государственный университет».

650043, г. Кемерово, ул. Красная, 6.

Отпечатано в типографии «СТИВЭС», 644053, г. Омск,

ул. Нефтезаводская, д. 38, кор. 2, тел. 8(3812)64-75-77.