СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ НАУЧНОГО РЕДАКТОРА

Книга «Компьютерная архитектура. Количественный подход» издается в России впервые. Ее авторы Дэвид А. Паттерсон и Джон Л. Хеннесси — известные ученые в области вычислительной техники, имеющие опыт разработки компьютерных систем и их узлов, профессора университетов США. Эта книга начиная с 1990 г. выдержала 5 изданий в США, и каждое ее издание корректировалось с учетом передового и коммерчески успешного текущего состояния компьютерной архитектуры. Книга обоснованно считается классическим учебником в этой области техники.

В ней представлены архитектуры современных компьютеров общего назначения и специализированных – от персональных мобильных до ориентированных на обработку массивов данных и применение в крупномасштабных системах коллективного пользования.

Компьютерная архитектура показана как средство использования параллелизма команд, данных и задач для достижения высокой производительности вычислений.

Книга адресована студентам и аспирантам, изучающим вычислительную технику и программирование, но также может быть полезна профессионалам, поскольку не только дает представление о компьютерной архитектуре, но и содержит методологию выбора и оценок ее параметров.

Основное содержание книги составляет организация подсистемы памяти и конвейера выполнения команд в процессоре, структура многопроцессорных систем с общей памятью и многомашинных систем коллективного пользования.

Подсистема памяти в компьютерах общего назначения представляет собой иерархическую систему, содержащую несколько уровней кэша и оперативную память. Описываются методы организации кэша, направленные на уменьшение доли промахов и времени доступа, а также на увеличение пропускной способности кэша и поддержку когерентности. Показано, что реализация оперативной памяти с большим количеством блоков с независимым управлением обеспечивает ее высокую пропускную способность, необходимую для векторных и многоядерных архитектур.

Конвейерная обработка является основным методом выполнения команд в современном процессоре. В книге рассматриваются почти все существующие механизмы и режимы работы процессора, повышающие пропускную способность конвейера. Прежде всего это методы, направленные на снижение влияния зависимостей между командами в конвейере, — неблокирующий кэш, ускоренная передача результатов операций через байпас, предсказание направлений передач управления, переименование регистров, условное, спекулятивное, внеочередное и суперскалярное выполнение команд, динамическое и статическое планирование работы конвейера. Также это методы, использующие различные виды параллелизма: многопотоковый режим (поочередное выполнение различных потоков одной задачи), суперскалярное выполнение (тиражирование исполняющей части конвейера для параллельного выполнения групп различных команд над разнотипными данными), векторная обработка (тиражирование исполняющей части конвейера для параллельного выполнения групп одинаковых команд над однотипными данными), многоядерная организация (тиражирование всего конвейера).

Следует отметить, что многоядерная организация хорошо использует параллелизм задач, но далеко не всегда ускоряет выполнение одной задачи, так как создание параллельных потоков управления в задаче не всегда возможно и требует усилий программиста.

Векторные архитектуры и архитектуры графических процессоров, ориентированные на обработку массивов данных, рассмотрены почти так же подробно, как и архитектуры процессоров общего назначения. Обе эти архитектуры амортизируют задержку доступа в память с помощью высокой степени расслоения памяти и конвейерного режима ее работы и выполняют параллельно одинаковые операции над блоками однотипных данных. Векторные процессоры используются, главным образом, для научно-технических вычислений, и этим объясняется их ограниченная распространенность. Графические процессоры получили массовое применение благодаря бурному развитию машинной графики в компьютерах массового спроса, но их использование в качестве ускорителей универсальных вычислений требует значительных усилий программистов.

Сравнивая эти два типа процессоров, по аналогии с процессорами общего назначения можно сказать, что векторный процессор – это одноядерный суперскалярный процессор для работы с массивами данных, тогда как графический процессор – многоядерный многопотоковый скалярный процессор обработки массивов данных.

Новый класс «облачных» компьютеров – это системы коллективного пользования, предоставляющие возможность выполнения вычислений и хранения данных как коммунальную услугу. Такое назначение определило их организацию как крупномасштабную систему, состоящую из десятков тысяч стандартных серверов, которые соединены стандартной сетью передачи данных, и ориентированную на использование параллелизма запросов (параллельное выполнение множества отдельных задач). В таких компьютерах особенно значимыми являются размещение задач в аппаратуре компьютера, время ответа на запрос, надежность и стоимость эксплуатации.

Архитектуры систем команд процессора оцениваются с позиции эффективности работы конвейера при выполнении команд. Система команд с командами обращения в память, отделенными от арифметико-логических команд, представляется предпочтительной, так как облегчает планирование работы конвейера, в частности позволяет скрывать задержки обращений к памяти за счет разнесения в программах команд считывания данных из памяти и выполнения операций над ними.

Все эти темы описаны достаточно подробно и в хорошо воспринимаемой форме. Книга оправдывает подзаголовок «Количественный подход». В ней много примеров расчетов и графиков, иллюстрирующих количественное влияние тех или иных архитектурных параметров на производительность, пропускную способность, надежность, энергоэффективность, стоимость и т.д. Приведены структурные схемы устройств современных процессоров, примеры программ и временных диаграмм их выполнения, иллюстрирующие использование различных архитектурных свойств, расчеты параметров работы узлов процессора, графики их поведения на различных пакетах тестов в зависимости от значений архитектурных параметров. В конце каждой главы представлены упражнения, позволяющие читателю оценить степень понимания рассмотренных вопросов. Весь этот материал интересен и важен для изучения компьютерной архитектуры, кроме того, он может быть полезен специалистам при проектировании и эксплуатации компьютеров.

Авторы не включили в книгу, но дали ссылки на 9 дополнительных приложений к книге. Их содержание описано в предисловии авторов.

В заключение стоит отметить, что в России разрабатываются и производятся компьютеры, в которых реализуется большая часть архитектурных свойств, описанных в данной книге. В частности, НИИСИ РАН, ЗАО «МЦСТ» и ПАО «ИНЭУМ им. И.С. Брука» разрабатывают семейства универсальных многоядерных микропроцессоров с суперскалярной архитектурой с динамическим планированием работы конвейера и оригинальной отечественной архитектурой со статическим планированием работы конвейера, в них поддерживаются векторный параллелизм, параллелизм потоков управления и глубокая иерархия подсистемы памяти.

В суперскалярных микропроцессорах с динамическим (аппаратным) планированием работы конвейера количество команд, выдаваемых в каждом такте для параллельного выполнения, ограничивается сложностью схемы динамического планирования и локальностью просматриваемой области – только в районе дешифрации команд.

Статическое (компиляторное) планирование работы конвейера является альтернативой динамическому планированию. В научно-технической литературе такой подход получил название архитектуры явного использования параллелизма. Статический подход к планированию работы конвейера позволяет упростить организацию процессора за счет исключения из его состава одного из «узких мест» – аппаратуры динамического планирования. Кроме того, для максимальной загрузки аппаратуры компилятор может формировать группы с большим количеством команд для параллельного выполнения за счет глобального анализа программы.

В российских микропроцессорах со статическим планированием дополнительно реализована аппаратно поддержанная технология совместимости с архитектурой x86 фирмы Intel, позволяющая эффективно выполнять коды для этой архитектуры. Методы динамической компиляции и оптимизации двоичных кодов, использующие особенности аппаратного конвейера, не нашли отражения в 5-м издании книги (даже в приложениях), хотя рассматривались в ранних изданиях. Тем не менее они сохраняют свою актуальность до сих пор и используются в коммерческих микропроцессорах, выпускаемых американскими компаниями.

В оригинальной российской архитектуре реализованы методы повышения семантической надежности программ, которые являются все более востребованными в связи с наличием множества уязвимостей в существующем программном обеспечении. В этой области разработчики, реализовав аппаратные средства контроля типов, контекстной защиты памяти и дескрипторы объектов, намного опередили западных коллег, которые для повышения надежности программ создают программные системы или, как Intel, включают в систему команд некоторые расширения исключительно в целях ускорения отладки программ.

На базе этих микропроцессоров серийно производятся отечественные компьютеры различного назначения, в том числе высокопроизводительные многопроцессорные системы с общей когерентной и распределенной оперативной памятью.

Генеральный директор ПАО «ИНЭУМ им. И.С. Брука»
к.т.н., лауреат Государственной премии
Ким Александр Киирович

Эндреа, Линде и нашим четырем сыновьям

ПРЕДИСЛОВИЕ

Луис Андре Барросо, Google Inc.

Первое издание книги Хеннесси и Паттерсона *«Компьютерная архитектура. Количественный подход»* вышло, когда я учился на первом курсе аспирантуры. Поэтому я принадлежу к той первой волне профессионалов, которые постигали вычислительную технику, используя эту книгу в качестве компаса. Поскольку перспектива является основным элементом любого полезного предисловия, в данном случае я нахожусь в невыгодном положении, если принять во внимание, какая значительная часть моих собственных представлений сформировалась под влиянием предыдущих четырех изданий этой книги. Еще одно препятствие к созданию четкой перспективы состоит в том, что меня еще не покинуло студенческое благоговение по отношению к этим двум суперзвездам вычислительной техники, несмотря на то (или, может быть, благодаря тому), что в последующие годы у меня была возможность узнать их лично. Это невыгодное положение смягчается тем, что я непрерывно занимался вычислительной техникой с момента выхода первого издания этой книги, что дало мне возможность наслаждаться ее развитием и долговременной актуальностью.

Последнее издание появилось всего через два года после того, как безудержная промышленная гонка за более высокую тактовую частоту (CPU — Central Processor Unit, центральный процессор *или* центральное процессорное устройство) достигла своего официального конца, когда фирма Intel прекратила свои разработки одноядерного процессора с тактовой частотой 4 ГГц и стала использовать многоядерные CPU. За два года у Джона и Дэвида было много времени для того, чтобы представить эту историю не как случайное обновление модельного ряда, а как переломный момент, определяющий компьютерные технологии последнего десятилетия. В четвертом издании параллелизму уровня команд (ILP — Instruction Level Parallelism) было уделено меньше внимания в пользу дополнительного материала, касающегося параллелизма уровня потоков, что в данном издании получило еще большее развитие (две главы посвящены параллелизму уровня потоков и данных, при этом обсуждение ILP ограничено одной главой. Читателям, которые знакомятся с новыми устройствами обработки графических данных, будет особенно полезна глава 4, в которой основное внимание уделяется параллелизму данных, объясняются различные, но медленно сходящиеся к одной точке решения, предлагаемые мультимедийными расширениями в универсальных процессорах и все больше используемыми программируемыми устройствами обработки графических данных. Примечательная практическая уместность: если вы когда-либо барахтались в сложной терминологии (CUDA — Compute Unified Device Architecture,

программно-аппаратная архитектура параллельных вычислений), обратитесь к рис. 4.24 (головоломка: общая память в действительности является локальной, в то время как глобальная память находится ближе к тому, что вы бы сочли общей памятью).

Даже несмотря на то, что мы все еще находимся посреди этого сдвига к многоядерной технологии, это издание охватывает то, что, кажется, будет следующим важным шагом: «облачные» вычисления. В данном случае возможность повсеместного подключения к Интернету и эволюция привлекательных веб-сервисов притягивают всеобщее внимание к очень маленьким устройствам (смартфонам, планшетам) и к очень большим (вычислительным системам масштаба склада). (ARM — Advanced RISC Machine, усовершенствованная RISC-машина) Cortex A8, популярный CPU для смартфонов, появляется в разделе «Соединяем все вместе» главы 3, а новая глава 6 целиком посвящена параллелизму уровня запросов и данных в контексте вычислительных систем масштаба склада. В новой главе Джон и Дэвид представляют эти новые огромные кластеры как отдельный новый класс компьютеров — это открытое приглашение компьютерным архитекторам помочь сформировать эту нарождающуюся сферу. Читатели оценят, как развивалась эта область в последнем десятилетии, сравнивая архитектуру кластера корпорации Google, описанную в третьем издании, с более современным воплощением, представленным в главе 6 данного издания.

Покупатели, уже знакомые с предыдущими изданиями этой книги, еще раз оценят работу двух выдающихся специалистов в области вычислительной техники, которые на протяжении своей многолетней деятельности совершенствуют искусство объединения принципиального академического отношения к идеям с глубоким пониманием качественно новых промышленных продуктов и технологий. Успех авторов во взаимодействии с представителями промышленности не удивит тех, кто был свидетелем того, как Дэвид проводит два раза в год проектные совещания-семинары, форумы, тщательно выстроенные для того, чтобы извлечь максимум из сотрудничества ученых и представителей промышленности. Те, кто помнит предпринимательский успех Джона с MIPS или столкнется с ним в коридоре корпорации Google (что делаю я время от времени), также не будут удивлены этим.

Эта книга стала непреходящей классикой благодаря тому, что каждое ее издание является не просто обновлением, а глубокой переработкой, содержащей самую современную информацию и отражающей суть этой увлекательной и быстро меняющейся сферы. Для меня, отдавшего этой профессии более двадцати лет жизни, это также еще одна возможность испытать то свое студенческое восхищение этими двумя выдающимися учителями.

ОТЗЫВЫ О ПЯТОМ ИЗДАНИИ КНИГИ «КОМПЬЮТЕРНАЯ АРХИТЕКТУРА. КОЛИЧЕСТВЕННЫЙ ПОДХОД»

«Пятое издание книги *«Компьютерная архитектура. Количественный подход»* продолжает предыдущие издания, предоставляя студентам, изучающим компьютерную архитектуру, самую современную информацию о текущих вычислительных платформах и понимание компьютерной архитектуры, помогающие им разрабатывать будущие системы. Ключевым моментом нового издания является значительно переработанная глава, посвященная параллелизму уровня данных, которая раскрывает тайну архитектур графических процессоров с помощью четких объяснений, используя традиционную терминологию архитектуры ЭВМ».

Krste Asanovic', Калифорнийский университет, Беркли

«*Компьютерная архитектура. Количественный подход*» — это классика, которая, как хорошее вино, со временем становится только лучше. Я приобрел свой первый экземпляр этой книги, когда заканчивал курс обучения на получение степени бакалавра, и до сих пор она остается одним из текстов, к которым я обращаюсь наиболее часто. Когда вышло четвертое издание книги, там было так много нового материала, что я, не раздумывая, приобрел его, чтобы быть в курсе происходящего в данной области. Сегодня, рецензируя пятое издание, я понимаю, что Хеннесси и Паттерсон сделали это снова. Весь текст основательно обновлен, и одна только глава 6 делает чтение данного издания книги необходимым для тех, кто на самом деле хочет понять «облачные» вычисления и компьютеры масштаба склада (WSC — Warehouse Scale-Computing). Только Хеннесси и Паттерсон имеют доступ к информации в корпорациях Google, Microsoft, Amazon и других поставщиков «облачных» вычислений и приложений масштаба Интернет, и нигде в отрасли нет лучшего освещения этой важной области».

James Hamilton, Amazon Web Services

«Хеннесси и Паттерсон написали первое издание этой книги, когда аспиранты создавали компьютеры с 50 000 транзисторов. Сегодня компьютеры масштаба склада содержат столько же серверов, каждый из которых состоит из десятков независимых процессоров и миллиардов транзисторов. Эволюция компьютерной архитектуры была быстрой и безжалостной, но «*Компьютерная архитектура. Количественный подход*» не отстает, и каждое издание этой книги точно объясняет и анализирует важные возникающие концепции, которые делают эту сферу такой захватывающей».

James Larus, Microsoft Research

«В это издание добавлена великолепная новая глава, посвященная параллелизму данных в векторных архитектурах, SIMD (Single instruction stream, Multiple data streams, один поток команд, несколько потоков данных) архитектурах и архитектурах графических процессоров. В ней объясняются основные архитектурные

концепции графических процессоров массового ассортимента, дается их сопоставление с традиционными терминами и сравнение с векторными и SIMD-архитектурами. Книга является своевременной и показывает широкомасштабный сдвиг в сторону параллельных вычислений с использованием графических процессоров. *«Компьютерная архитектура. Количественный подход»* продолжает лидировать в данной сфере, первой представив всеобъемлющее архитектурное освещение значительных новых разработок!»

John Nickolls, NVIDIA

«Новое издание этого теперь уже классического учебника привлекает внимание к доминирующему влиянию явного параллелизма (данных, потоков, запросов), отводя по целой главе каждому из этих типов параллелизма. Глава о параллелизме данных особенно поучительна: сравнение и противопоставление векторных SIMD, SIMD уровня команд и графических процессоров выявляет сходства и различия между этими архитектурами».

Kunle Olukotun, Стэнфордский университет

«Пятое издание книги «*Компьютерная архитектура. Количественный подход*» исследует различные концепции параллелизма и соответствующие им достоинства и недостатки. Как и в предыдущих изданиях, это издание охватывает новейшие тенденции развития технологии. Два явления, к которым привлекается внимание — бурное распространение персональных мобильных устройств (PMD — Personal Mobile Devices) и компьютеров масштаба склада, где центр внимания смещается в сторону более сложного равновесия между производительностью и энергетической эффективностью по сравнению с непосредственной производительностью. Эти тенденции питают нашу потребность в большей возможности обработки, которая, в свою очередь, ведет нас еще дальше по пути параллелизма».

Andrew N. Sloss, инженер-консультант, ARM
автор «Руководства системного разработчика ARM»

КОМПЬЮТЕРНАЯ АРХИТЕКТУРА

Количественный подход

Пятое издание

Джон Л. Хеннесси — десятый ректор Стэнфордского университета, где с 1977 г. преподает на факультетах электротехники и информатики. Хеннесси является действительным членом IEEE и ACM (Association for Computing Machinery, Ассоциация вычислительной техники); членом Национальной академии инженерных наук, Национальной академии наук и Американского философского общества, а также действительным членом Американской академии наук и искусств. В числе его многочисленных наград премия Эккерта — Мочли за его вклад в RISC (Reduced Instruction Set Computer, компьютер с сокращенной системой команд) технологию в 2001 г., премия Сеймура Крея в области вычислительной техники в 2001 г. и премия Джона фон Неймана, которую он получил вместе с Дэвидом Паттерсоном в 2000 г. Он также имеет семь почетных докторских степеней.

В 1981 г. Хеннесси начал проект MIPS (Microprocessor without Interlocked Pipeline Stages, микропроцессор без блокировок между ступенями конвейера) в Стэнфорде с небольшой группой аспирантов. После завершения этого проекта в 1984 г. он покинул университет, чтобы стать одним из основателей компании MIPS Computer Systems (в настоящее время MIPS Technologies), которая разработала один из первых коммерческих RISC-микропроцессоров. По состоянию на 2006 г. компания поставила более 2 млрд микропроцессоров MIPS для устройств, начиная от видеоигр и миниатюрных ручных компьютеров и заканчивая лазерными принтерами и сетевыми коммутаторами. Затем Хеннесси возглавил проект DASH (Director Architecture for Shared Memory, архитектура справочника для разделяемой памяти), который был прототипом первого масштабируемого мультипроцессора с когерентным кэшем; многие из его основных концепций используются в современных мультипроцессорах. Помимо своей технической деятельности и обязанностей в университете, он продолжает работу с многочисленными вновь создаваемыми компаниями в качестве консультанта на начальной стадии и инвестора.

Дэвид А. Паттерсон преподает компьютерную архитектуру в Калифорнийском университете в Беркли с 1977 г., где заведует кафедрой информатики имени Парди. Его педагогическая деятельность была удостоена награды Калифорнийского университета «За выдающуюся педагогическую деятельность», премии Karlstrom от ACM, а также медали Муллигана в области образования и премии IEEE за обучение аспирантов. Паттерсон получил награду «За технические достижения» от IEEE и премию Эккерта — Мочли от ACM за вклад в RISC, он также был одним из лауреатов премии Джонсона в области хранения информации от IEEE за его вклад в RAID (Redundant Array of Independent Disks, избыточный массив независимых дисков). Он является обладателем медали Джона фон Неймана и лауреатом премии C & C вместе с Джоном Хеннесси. Как и его соавтор, Паттерсон является действительным членом Американской академии наук и искусств, музея истории компьютеров, ACM и IEEE.

Он был избран в Национальную академию инженерных наук, Национальную академию наук и Зал Славы инженеров Силиконовой долины Он работал в Консультативном комитете по информационным технологиям при Президенте США, был заведующим кафедрой информатики на факультете электротехники и информатики в Беркли, председателем Ассоциации исследований в области компьютеров (CRA — Computing Research Association) и президентом ACM. За эти достижения он был удостоен награды «За выдающиеся заслуги» ACM и CRA.

В Беркли Паттерсон возглавлял разработку и реализацию RISC I — вероятно, первого микропроцессора с сокращенным набором команд, а также заложил основу коммерческой архитектуры (SPARC — Scalable Processor ARChitecture, масштабируемая архитектура процессора). Он руководил проектом RAID, результатом чего стало появление надежных систем хранения информации от многих компаний. Он также принимал участие в проекте «Сеть рабочих станций» (NOW — Network Of Workstations), который привел к появлению технологии кластера, используемой интернет-компаниями, а позднее в «облачных» вычислениях. Эти проекты заслужили три диссертационные награды от ACM. Его текущими исследовательскими проектами являются лаборатория «Алгоритм — машина — люди» и Лаборатория параллельных вычислений, где он является директором. Цель лаборатории «Алгоритм — машина — люди» состоит в разработке масштабируемых алгоритмов машинного обучения, моделей программирования, ориентированных на компьютеры масштаба склада, и средств краудсорсинга (привлечение к выполнению работы через Интернет больших групп людей) для быстрого получения ценных знаний из больших объемов данных в «облаке». Цель Лаборатории параллельных вычислений состоит в разработке технологий создания масштабируемого, переносимого, эффективного и продуктивного программного обеспечения для параллельных персональных мобильных устройств.

Джон Л. Хеннесси
Стэнфордский университет

Дэвид А. Паттерсон
Калифорнийский университет, Беркли

С участием

Крсте Асанович
Калифорнийский университет, Беркли
Джейсона Д. Бейкоса
Университет штата Южная Каролина
Роберта П. Колуэлла
R&E Collwell & Assoc. Inc.
Томаса М. Конте
Университет штата Северная Каролина
Хозе Дуато
Политехнический университет Валенсии и Симула
Дианы Франклин
Калифорнийский университет, Санта-Барбара
Дэвида Голдберга
Исследовательский институт Скриппс
Нормана П. Джуппи
Лаборатории «Хьюлетт-Паккард»
Шенг Ли
Лаборатории «Хьюлетт-Паккард»
Навин Муралиманохар
Лаборатории «Хьюлетт-Паккард»
Грегори Д. Петерсона
Университет штата Теннесси
Тимоти М. Пинкстона
Южно-Калифорнийский университет
Партасарати Ранганатан
Лаборатории «Хьюлетт-Паккард»
Дэвида А. Вуда
Висконсинский университет в Мэдисоне
Амр Заки
Университет Санта Клары

ОТ АВТОРОВ

Почему мы написали эту книгу

На протяжении пяти изданий этой книги наша цель заключалась в описании основных принципов, лежащих в основе того, что станет завтрашним днем в развитии технологий. Наша взволнованность относительно возможностей в компьютерной архитектуре не ослабла, и мы, как эхо, повторяем то, что говорили об этой сфере в первом издании: «Это не нудная наука о бумажных машинах, которые никогда не будут работать. Нет! Это — дисциплина, вызывающая острый интеллектуальный интерес, требующая равновесия между рыночными силами и стоимостью — производительностью — мощностью, ведущая к поражениям в красивой борьбе и нескольким выдающимся успехам».

Основная цель при написании нашей первой книги заключалась в том, чтобы изменить то, как люди изучают компьютерную архитектуру и думают о ней. Мы считаем, что эта цель по-прежнему важна и остается в силе. Эта область меняется ежедневно, и ее следует изучать на реальных примерах и с измерениями на реальных компьютерах, а не просто как набор определений и проектов, которые никогда не придется реализовывать. Мы от души приветствуем любого, кому было с нами по пути в прошлом, и тех, кто присоединится к нам сейчас. В любом случае мы можем пообещать тот же количественный подход к реальным системам и их анализ.

Как и в предыдущих изданиях, мы старались, чтобы новое издание было попрежнему столь же актуальным как для профессиональных инженеров и архитекторов, так и для тех, кто связан с преподаванием и изучением курсов современной архитектуры и проектирования компьютеров. Это данное издание, как и первое, четко сфокусировано на новых платформах — персональных мобильных устройствах и компьютерах масштаба склада, и на новых архитектурах — многоядерных и графических процессорах. Это издание имеет своей целью раскрыть тайну компьютерной архитектуры, придавая особое значение компромиссам между стоимостью — производительностью — энергопотреблением и хорошим инженерным проектом. Мы верим в то, что данная область продолжает совершенствоваться и двигаться по направлению к строгому количественному фундаменту давно устоявшихся научных и технических дисциплин.

Данное издание

Мы говорили, что четвертое издание книги *«Компьютерная архитектура. Количественный подход»*, возможно, является наиболее значимым после выхода первого издания из-за перехода к многоядерным микропроцессорам. Отзывы, которые мы получили на это издание, показали, что книга потеряла четкий фокус первого издания, освещая все в равной степени, ничего не выделяя и в отрыве от контекста. Мы уверены, что этого нельзя будет сказать в отношении пятого издания.

Мы считаем, что наибольшее впечатление производят колоссальные различия в размерах средств вычислительной техники, от персональных мобильных

устройств, таких, как сотовые телефоны и планшеты в качестве клиентов, до компьютеров масштаба склада, предлагающих «облачные» вычисления, в качестве сервера. (Наблюдательные читатели могли заметить намек на «облачные» вычисления на обложке.) Нас поражает общее этих двух крайностей в стоимости, производительности и энергетической эффективности, несмотря на их разные размеры. В результате в каждой главе непременно рассматривается вычислительная техника для персональных мобильных устройств и для компьютеров масштаба склада, а глава 6 является абсолютно новой и посвящена компьютерам масштаба склада.

Другой темой является параллелизм во всех его формах. Сначала мы определяем два вида параллелизма уровня приложений в главе 1: *параллелизм уровня данных* (*DLP — Data-Level Parallelism*), возникающий из-за наличия большого количества элементов данных, которые можно обрабатывать одновременно, и *параллелизм уровня задач* (*TLP — Task-Level Parallelism*), возникающий благодаря тому, что создаются задачи, которые могут выполняться независимо и в значительной степени параллельно. Затем мы объясняем четыре архитектурных стиля, которые используют DLP и TLP: *параллелизм уровня команд* (*ILP*) в главе 3, *векторные архитектуры и графические процессоры* (*GPU — Graphic Processor Units*) в главе 4, которая является новой, *параллелизм уровня потоков* в главе 5 и *параллелизм уровня запросов* (*RLP — Request-Level Parallelism*) посредством компьютеров масштаба склада в главе 6, которая также является новой. Мы переместили иерархию памяти в начало книги в главу 2, а главу о системах хранения данных — в приложение D. Мы особенно гордимся главой 4, в которой содержится самое подробное и четкое на сегодняшний день объяснение GPU, и главой 6, которая является первой публикацией самых последних данных о компьютере масштаба склада корпорации Google.

Как и раньше, в первых трех приложениях книги даются основы системы команд, иерархии памяти и конвейерной обработки в MIPS для читателей, которые не читали книгу, подобную книге *«Организация и проектирование компьютеров»*. Чтобы не увеличивать стоимость, но предоставить дополнительный материал, который может оказаться интересным для некоторых читателей, по адресу *http://booksite.mkp.com/9780123838728/* доступны онлайн еще девять приложений. В этих приложениях больше страниц, чем в этой книге!

В данном издании мы продолжаем традицию использования реальных примеров для демонстрации концепций, а разделы «Соединяем все вместе» являются новыми. Они содержат описания организаций конвейеров и иерархий памяти процессора ARM Cortex A8, процессора Intel Core i7, графических процессоров NVIDIA GTX-280 и GTX-480 и одного из компьютеров масштаба склада корпорации Google.

Выбор тем и организация

Как и раньше, мы консервативно подошли к выбору тем, так как в данной области имеется гораздо больше интересных идей, чем можно охватить в разумных пределах, рассказывая об основных принципах. Мы избегаем всеобъемлющего обзора каждой архитектуры, с которой может встретиться читатель. Вместо этого наша презентация сосредоточена на ключевых концепциях, которые, вероятно, можно найти в любой новой машине. Основным критерием остается критерий от-

бора идей, которые были изучены и применены настолько успешно, что имеется возможность их количественного обсуждения.

Мы всегда старались сосредоточиться на материале, который нельзя получить в такой же форме из других источников, поэтому всегда, когда это возможно, мы продолжаем делать акцент на современном содержании. Разумеется, здесь есть несколько систем, описание которых нельзя найти в литературе. (Читателям, заинтересованным исключительно в более фудаментальном введении в компьютерную архитектуру, следует прочитать книгу *«Организация и проектирование компьютеров: Аппаратный/программный интерфейс»*.)

Обзор содержания

В этом издании глава 1 была расширена. Она содержит формулы для расчета энергии, статической мощности, динамической мощности, стоимости интегральных схем, надежности и доступности. (Эти формулы также можно найти в приложении M.) Мы надеемся, что эти темы могут быть полезны читателям на протяжении остальной части книги. В дополнение к классическим количественным принципам проектирования компьютеров и измерения производительности раздел «Соединяем все вместе» был расширен с тем, чтобы использовать новый тест SPECPower.

Мы считаем, что сегодня архитектура системы команд играет меньшую роль, чем в 1990-х гг., поэтому переместили этот материал в приложение A. В приложении A по-прежнему используется архитектура MIPS64. (Сводную информацию об архитектуре системы команд MIPS для быстрого просмотра можно найти в 1 главе на с. 43—44 рис. 1.5.) Для любителей архитектур систем команд в приложении K описаны 10 RISC-архитектур, 80x86, DEC VAX и IBM 360/370.

Затем в главе 2 мы переходим к иерархии памяти, поскольку к этому материалу легко применять принципы стоимости — производительности — энергопотребления, а память является крайне важным ресурсом для понимания остальных глав. Как и в прошлом издании, в приложение B содержится вводный обзор принципов организации кэшей на случай, если это вам нужно. В главе 2 обсуждаются 10 современных оптимизаций кэшей. В эту главу включена информация о виртуальных машинах, которые обладают преимуществами в плане защиты, управления программным обеспечением, управления аппаратурой и играют важную роль в «облачных» вычислениях. Помимо описания технологий памяти (SRAM — Static Random Access Memory, статическая оперативная память с произвольным доступом) и (DRAM — Dynamic Random Access Memory, динамическая память с произвольным доступом), эта глава содержит новый материал по флэш-памяти. Примерами раздела «Соединяем все вместе» являются ARM Cortex A8, который используется в персональных мобильных устройствах, и Intel Core i7, используемый в серверах.

В главе 3 описано использование параллелизма уровня команд в высокопроизводительных процессорах, в том числе суперскалярное выполнение, предсказание передач управления, спекулятивное выполнение, динамическое планирование и многопотоковая обработка. Как упоминалось выше, приложение C содержит обзор конвейерной обработки на случай, если это вам нужно. В главе 3 также рассматриваются ограничения ILP. Как и в главе 2, примерами раздела «Соединяем все вместе» снова являются ARM Cortex A8 и Intel Core i7. В то время как третье издание содержало много

информации по архитектуре Itanium и VLIW (Very Long Instruction Word, сверхдлинное командное слово), теперь этот материал содержится в приложении H, так как мы считаем, что эта архитектура не оправдала более ранних заявлений.

Возрастающая важность мультимедийных приложений, таких как игры и обработка видеоинформации, также повысила важность архитектур, которые могут использовать параллелизм уровня данных. В частности, растет интерес к вычислениям с использованием графических процессоров (GPU), но лишь немногие архитекторы понимают, как реально работают GPU. Мы решили написать новую главу во многом для того, чтобы открыть тайны этого нового стиля компьютерной архитектуры. Глава 4 начинается с введения в векторные архитектуры, которое служит фундаментом для объяснения мультимедийных расширений системы команд SIMD и GPU. (В приложении G векторные архитектуры рассматриваются более подробно.) В этой книге труднее всего было писать раздел о GPU; на то, чтобы получить точное и легко понимаемое описание, понадобилось немало усилий. Весьма сложной проблемой была терминология. Мы решили использовать наши собственные термины, а затем дать сопоставление наших терминов с официальными терминами компании NVIDIA. (Эту таблицу можно найти в приложении M.) В этой главе представлена модель производительности Roofline, которая затем используется для сравнения Intel Core i7 и графических процессоров NVIDIA GTX 280 и GTX 480. Здесь также содержится описание графического процессора Tegra 2 для персональных мобильных устройств.

В главе 5 описываются многоядерные процессоры. В ней исследуются симметричные архитектуры и архитектуры с распределенной памятью, рассматриваются принципы их организации и производительность. Освещаются темы, связанные с синхронизацией и моделями согласованности памяти. Примером является Intel Core i7. Читателям, интересующимся сетями на кристалле, следует прочитать приложение F, а тем, кто интересуется крупномасштабными мультипроцессорами, — приложение I.

Как упоминалось ранее, в главе 6 рассматривается новейшая тема в компьютерной архитектуре — компьютеры масштаба склада (WSC). В этой главе на основе данных, предоставленных инженерами Amazon Web Services и Google, мы собрали сведения об устройстве, стоимости и производительности компьютеров WSC, известные лишь немногим архитекторам. Она начинается с популярной модели программирования MapReduce, за ней следует описание архитектуры и физической реализации компьютеров WSC, включая затраты. Затраты помогают нам объяснить появление «облачных» вычислений, благодаря которым можно дешевле выполнять вычисления, используя компьютеры WSC в «облаке», чем в вашем местном центре обработки данных. Примером раздела «Соединяем все вместе» является описание компьютера WSC корпорации Google, содержащее информацию, опубликованную впервые в этой книге.

Перейдем теперь к приложениям с A по L[1] включительно. В приложении A рассматриваются принципы архитектур систем команд, включая MIPS64, а в приложении K описываются 64-разрядные версии архитектур систем команд Alpha,

[1] В приложение M включена информация с форзацев оригинала книги.

MIPS, PowerPC и SPARC и их мультимедийные расширения. Сюда также включены несколько классических архитектур (80x86, VAX и IBM 360/370) и популярные встраиваемые системы команд (ARM, Thumb, SuperH, MIPS16 и Mitsubishi M32R). К этой теме относится приложение H, так как в нем рассматриваются архитектуры и компиляторы для архитектур систем команд VLIW.

Как упоминалось ранее, приложения B и C являются учебными пособиями по основным концепциям кэширования и конвейерной обработки. Читателям, для которых кэширование относительно ново, следует прочитать приложение B раньше главы 2, а тем, для кого относительно новой является конвейерная обработка, следует прочитать приложение C раньше главы 3.

Приложение D «Системы хранения данных» содержит расширенное обсуждение надежности и доступности, учебное пособие по RAID с описанием схем RAID 6 и редко встречающуюся статистику отказов реальных систем. Затем следуют введение в теорию очередей и тесты оценки производительности подсистемы ввода/вывода. Мы оцениваем стоимость, производительность и надежность реального кластера Internet Archive. Примером для раздела «Соединяем все вместе» является файловая система NetApp FAS6000.

Приложение E, написанное Томасом М. Конте, соединяет в одном месте материал по встроенным системам.

Приложение F по сетям было переработано Тимоти М. Пинкстоном и Хозе Дуато. Приложение G, первоначально написанное Крсте Асанович, содержит описание векторных процессоров. Мы считаем, что эти два приложения являются одним из лучших известных нам материалов по каждой из данных тем.

В приложении H описываются архитектуры VLIW и EPIC (Explicitly Parallel Instruction Computing, микропроцессорная архитектура с явным параллелизмом команд) Itanium.

В приложении I описываются приложения параллельной обработки и протоколы когерентности для крупномасштабной многопроцессорной обработки с общей памятью. В приложении J, написанном Дэвидом Голдбергом, описывается компьютерная арифметика.

В приложении L собраны «Исторические обзоры и ссылки» из каждой главы в одном месте. В нем сделана попытка отдать должное концепциям каждой главы и дать ощущение исторической обстановки, окружавшей данные изобретения. Мы хотим думать об этом как о человеческой драме (трагедии) проектирования ЭВМ. Там также имеются ссылки, которыми может захотеть воспользоваться тот, кто изучает архитектуру. Если у вас есть время, мы рекомендуем прочитать некоторые классические статьи в данной области, упомянутые в этих разделах. Узнать идеи от их создателей и приятно, и полезно для обучения. Раздел «Исторические обзоры» был одним из самых популярных разделов предыдущих изданий.

Порядок чтения книги

Единого, самого лучшего порядка изучения этих глав и приложений нет, за исключением того, что всем читателям следует начать с главы 1. Если вы не хотите читать все, можете воспользоваться предлагаемыми ниже последовательностями.

- *Иерархия памяти*: приложение B, глава 2 и приложение D.
- *Параллелизм уровня команд*: приложение C, глава 3 и приложение H.
- *Параллелизм уровня данных*: главы 4 и 6, приложение G.
- *Параллелизм уровня потоков*: глава 5, приложения F и I.
- *Параллелизм уровня запросов*: глава 6.
- *Архитектура системы команд*: приложения A и K.

Приложение E можно читать в любое время, но лучше всего его читать после последовательностей «*Архитектура системы команд*» и «*Кэши*». Приложение J можно читать в любое время, когда вас волнует компьютерная арифметика. После завершения каждой главы вам следует прочитать соответствующую часть приложения L.

Структура главы

Выбранный нами материал был расположен согласно последовательной структуре, которая соблюдается в каждой главе. Мы начинаем с объяснений концепций данной главы. За ними следует раздел «Смежные вопросы», в котором показывается, как концепции, изложенные в одной главе, взаимодействуют с концепциями, содержащимися в других. За ним следует раздел «Соединяем все вместе», который соединяет эти концепции между собой, показывая, как они используются в реальной машине.

Следующим является раздел «Заблуждения и просчеты», предоставляющий читателям возможность поучиться на чужих ошибках. Мы приводим примеры распространенных неправильных пониманий и архитектурных ловушек, которых трудно избежать, даже если вы знаете, что они вас поджидают. Разделы «Заблуждения и просчеты» являются одними из самых популярных разделов этой книги. Каждая глава заканчивается разделом «Заключение».

Учебные примеры с упражнениями

Каждая глава заканчивается разбором учебных примеров и сопровождающими их упражнениями. Их авторами являются промышленные и научные эксперты; в этих примерах исследуются ключевые концепции данной главы и проверяется их понимание с помощью упражнений с возрастающей сложностью. Преподаватели найдут эти примеры достаточно подробными и продуманными, дающими им возможность создавать свои собственные дополнительные упражнения.

Скобки в каждом упражнении (<глава.раздел>) указывают разделы текста, имеющие наибольшее отношение к выполнению данного упражнения. Мы надеемся, что это поможет читателям избежать упражнений, соответствующий раздел для которых они не прочитали, а также предоставит источник для повторения прочитанного. Чтобы дать читателю чувство времени, необходимого для выполнения упражнения, упражнения классифицированы:

[10] менее 5 минут (чтобы прочитать и понять);

[15] 5—15 минут для полного ответа;

[20] 15—20 минут для полного ответа;

[25] 1 час для полного письменного ответа;

[30] короткий программный проект: меньше одного полного дня программирования;

[40] значительный программный проект: две недели затраченного времени;

[обсуждение] тема для обсуждения с другими.

Решения конкретных примеров и упражнения доступны для преподавателей, которые зарегистрируются по адресу *textbooks.elsevier.com*.

Дополнительные материалы

Разнообразные ресурсы доступны онлайн по адресу: http://booksite.mkp.com/9780123838728/, в том числе:

- справочные приложения, освещающие ряд современных проблем, — некоторые из них написаны приглашенными экспертами в данной области;
- материал «Исторические перспективы», исследующий развитие ключевых концепций, представленных в каждой главе текста;
- слайды для преподавателей в PowerPoint;
- рисунки из книги в форматах PDF, PDS и PPT;
- ссылки на материал в сети;
- список замеченных опечаток.

Будут регулярно добавляться новые материалы и ссылки на другие ресурсы, имеющиеся в сети.

Помощь в улучшении этой книги

И, наконец, можно сделать деньги, читая эту книгу. (Поговорим о стоимости — производительности!) Если вы прочтете приведенные далее выражения признательности, вы увидите, что мы приложили значительные усилия, чтобы исправить ошибки. Поскольку книга перепечатывается много раз, у нас есть возможность сделать еще больше исправлений. Если вы обнаружите какие-либо оставшиеся упрямые ошибки, пожалуйста, обратитесь к издателю по электронной почте (*ca5bugs@mkp.com*).

Мы приветствуем общие комментарии к тексту и приглашаем вас посылать их по отдельному адресу электронной почты: *ca5comments@mkp.com*.

Заключение

И опять эта книга была результатом подлинного соавторства, где каждый из нас написал половину глав и одинаковую часть приложений. Мы не можем представить себе, сколько времени потребовалось бы, если бы кто-то еще не делал половину работы, вдохновляя, когда задача казалась безнадежной, помогая обрести ключевое понимание для объяснения сложной концепции, за выходные дни предоставляя разбор глав и соболезнуя, когда из-за груза наших других обязанностей было трудно взяться за перо. (Эти обязанности росли в геометрической прогрессии с числом изданий, о чем свидетельствуют биографии.) Таким образом, мы еще раз поровну делим вину за то, что вы собираетесь прочитать.

Джон Хеннесси
Дэвид Паттерсон

ВЫРАЖЕНИЯ БЛАГОДАРНОСТИ

Хотя данное издание является всего лишь пятым, фактически мы создали десять различных вариантов текста: три варианта первого издания (альфа, бета и окончательный) и по два варианта второго, третьего и четвертого изданий (бета и окончательный). За это время мы получили помощь от сотен экспертов и пользователей. Каждый из этих людей помог сделать книгу лучше. Поэтому мы решили перечислить всех, кто внес свой вклад в тот или иной вариант книги.

Вклад в пятое издание

Подобно предыдущим изданиям, данное издание является коллективным творчеством десятков добровольцев. Без их помощи данное издание было бы не столь безукоризненным.

Эксперты

Jason D. Bakos, University of South Carolina; Diana Franklin, The University of California, Santa Barbara; Norman P. Jouppi, HP Labs; Gregory Peterson, University of Tennessee; Parthasarathy Ranganathan, HP Labs; Mark Smotherman, Clemson University; Gurindar Sohi, University of Wisconsin–Madison; Mateo Valero, Universidad Politecnica de Cataluna; Sotirios G. Ziavras, New Jersey Institute of Technology.

Сотрудники лабораторий Par Lab и RAD Lab Калифорнийского университета в Беркли, которые часто рецензировали главы 1, 4, 6 и сформировали объяснение графических процессоров (GPU) и компьютеров WSC:

Krste Asanovic, Michael Armbrust, Scott Beamer, Sarah Bird, Bryan Catanzaro, Jike Chong, Henry Cook, Derrick Coetzee, Randy Katz, Yunsup Lee, Leo Meyervich, Mark Murphy, Zhangxi Tan, Vasily Volkov и Andrew Waterman.

Консультативная группа

Luiz Andre Barroso, Google Inc.; Robert P. Colwell, R&E Colwell & Assoc. Inc.; Krisztian Flautner, VP of R&D at ARM Ltd.; Mary Jane Irwin, Penn State; David Kirk, NVIDIA; Grant Martin, Chief Scientist, Tensilica; Gurindar Sohi, University of Wisconsin–Madison; Mateo Valero, Universidad Politecnica de Cataluna.

Приложения

Krste Asanovic, University of California, Berkeley (приложение G); Thomas M. Conte, North Carolina State University (приложение E); Jose Duato, Universitat Politecnica de Valencia and Simula (приложение F); David Goldberg, Xerox PARC (приложение J); Timothy M. Pinkston, University of Southern California (приложение F).

Jose Flich из Политехнического университета Валенсии (Universidad Politecnica de Valencia) внес существенный вклад в обновление приложения F.

Конкретные примеры с упражнениями

Jason D. Bakos, University of South Carolina (главы 3 и 4); Diana Franklin, University of California, Santa Barbara (глава 1 и приложение C); Norman P. Jouppi, HP

Labs (глава 2); Naveen Muralimanohar, HP Labs (глава 2); Gregory Peterson, University of Tennessee (приложение A); Parthasarathy Ranganathan, HP Labs (глава 6); Amr Zaky, University of Santa Clara (глава 5 и приложение B).

Jichuan Chang, Kevin Lim и Justin Meza помогали в разработке и тестировании примеров и упражнений к главе 6.

Дополнительный материал

John Nickolls, Steve Keckler, and Michael Toksvig, NVIDIA (глава 4 «Графические процессоры (GPU) компании NVIDIA»); Victor Lee, Intel (глава 4 «Сравнение Core i7 и GPU»); John Shalf, LBNL (глава 4 «Современные векторные архитектуры»); Sam Williams, LBNL (модель Roofline для компьютеров в главе 4); Steve Blackburn, Australian National University, и Kathryn McKinley, University of Texas at Austin (измерения производительности и мощности применительно к Intel в главе 5); Luiz Barroso, Urs Holzle, Jimmy Clidaris, Bob Felderman и Chris Johnson, Google (компьютер масштаба склада (WSC) корпорации Google в главе 6); James Hamilton, Amazon Web Services (распределение мощности и стоимостная модель в главе 6).

Jason D. Bakos из University of South Carolina сделал новые учебные слайды для этого издания.

И в заключение еще раз выражаем особую благодарность Mark Smotherman из Clemson University, который осуществил заключительное техническое прочтение нашей рукописи. Mark нашел массу ошибок и неясностей, в результате книга стала значительно качественнее.

Разумеется, эта книга не смогла бы выйти в свет без издательства. Мы хотим поблагодарить весь персонал Morgan Kaufmann / Elsevier за их работу и поддержку. За это пятое издание мы хотим особенно поблагодарить наших редакторов Nate McFadden и Todd Green, которые координировали опросы, консультативную группу, разработку конкретных примеров и упражнений, специальные рабочие группы, рецензирование рукописи и обновление приложений.

Мы должны также поблагодарить наших университетских сотрудников Margaret Rowland и Roxana Infante за бесчисленную экспресс-почту, а также за то, что они держали оборону в Стэнфорде и Беркли, пока мы работали над книгой.

В заключение мы благодарим наших жен за их страдания по утрам (все более ранним), посвященным чтению, обдумыванию и написанию.

Лица, внесшие вклад в предыдущие издания

Эксперты (рецензенты)

George Adams, Purdue University; Sarita Adve, University of Illinois at Urbana–Champaign; Jim Archibald, Brigham Young University; Krste Asanovic, Massachusetts Institute of Technology; Jean-Loup Baer, University of Washington; Paul Barr, Northeastern University; Rajendra V. Boppana, University of Texas, San Antonio; Mark Brehob, University of Michigan; Doug Burger, University of Texas, Austin; John Burger, SGI; Michael Butler; Thomas Casavant; Rohit Chandra; Peter Chen, University of Michigan; the classes at SUNY Stony Brook, Carnegie Mellon, Stanford, Clemson, и Wisconsin; Tim Coe, Vitesse Semiconductor; Robert P. Colwell; David Cummings; Bill

Dally; David Douglas; Jose Duato, Universitat Politecnica de Valencia and Simula; Anthony Duben, Southeast Missouri State University; Susan Eggers, University of Washington; Joel Emer; Barry Fagin, Dartmouth; Joel Ferguson, University of California, Santa Cruz; Carl Feynman; David Filo; Josh Fisher, Hewlett-Packard Laboratories; Rob Fowler, DIKU; Mark Franklin, Washington University (St. Louis); Kourosh Gharachorloo; Nikolas Gloy, Harvard University; David Goldberg, Xerox Palo Alto Research Center; Antonio Gonzalez, Intel and Universitat Politecnica de Catalunya; James Goodman, University of Wisconsin–Madison; Sudhanva Gurumurthi, University of Virginia; David Harris, Harvey Mudd College; John Heinlein; Mark Heinrich, Stanford; Daniel Helman, University of California, Santa Cruz; Mark D. Hill, University of Wisconsin–Madison; Martin Hopkins, IBM; Jerry Huck, Hewlett-Packard Laboratories; Wen-mei Hwu, University of Illinois at Urbana–Champaign; Mary Jane Irwin, Pennsylvania State University; Truman Joe; Norm Jouppi; David Kaeli, Northeastern University; Roger Kieckhafer, University of Nebraska; Lev G. Kirischian, Ryerson University; Earl Killian; Allan Knies, Purdue University; Don Knuth; Jeff Kuskin, Stanford; James R. Larus, Microsoft Research; Corinna Lee, University of Toronto; Hank Levy; Kai Li, Princeton University; Lori Liebrock, University of Alaska, Fairbanks; Mikko Lipasti, University of Wisconsin–Madison; Gyula A. Mago, University of North Carolina, Chapel Hill; Bryan Martin; Norman Matloff; David Meyer; William Michalson, Worcester Polytechnic Institute; James Mooney; Trevor Mudge, University of Michigan; Ramadass Nagarajan, University of Texas at Austin; David Nagle, Carnegie Mellon University; Todd Narter; Victor Nelson; Vojin Oklobdzija, University of California, Berkeley; Kunle Olukotun, Stanford University; Bob Owens, Pennsylvania State University; Greg Papadapoulous, Sun Microsystems; Joseph Pfeiffer; Keshav Pingali, Cornell University; Timothy M. Pinkston, University of Southern California; Bruno Preiss, University of Waterloo; Steven Przybylski; Jim Quinlan; Andras Radics; Kishore Ramachandran, Georgia Institute of Technology; Joseph Rameh, University of Texas, Austin; Anthony Reeves, Cornell University; Richard Reid, Michigan State University; Steve Reinhardt, University of Michigan; David Rennels, University of California, Los Angeles; Arnold L. Rosenberg, University of Massachusetts, Amherst; Kaushik Roy, Purdue University; Emilio Salgueiro, Unysis; Karthikeyan Sankaralingam, University of Texas at Austin; Peter Schnorf; Margo Seltzer; Behrooz Shirazi, Southern Methodist University; Daniel Siewiorek, Carnegie Mellon University; J. P. Singh, Princeton; Ashok Singhal; Jim Smith, University of Wisconsin–Madison; Mike Smith, Harvard University; Mark Smotherman, Clemson University; Gurindar Sohi, University of Wisconsin–Madison; Arun Somani, University of Washington; Gene Tagliarin, Clemson University; Shyamkumar Thoziyoor, University of Notre Dame; Evan Tick, University of Oregon; Akhilesh Tyagi, University of North Carolina, Chapel Hill; Dan Upton, University of Virginia; Mateo Valero, Universidad Politecnica de Cataluna, Barcelona; Anujan Varma, University of California, Santa Cruz; Thorsten von Eicken, Cornell University; Hank Walker, Texas A&M; Roy Want, Xerox Palo Alto Research Center; David Weaver, Sun Microsystems; Shlomo Weiss, Tel Aviv University; David Wells; Mike Westall, Clemson University; Maurice Wilkes; Eric Williams; Thomas Willis, Purdue University; Malcolm Wing; Larry Wittie, SUNY Stony Brook; Ellen Witte Zegura, Georgia Institute of Technology; Sotirios G. Ziavras, New Jersey Institute of Technology.

Приложения

Приложение, касающееся векторов, было переработано Krste Asanovic из Massachusetts Institute of Technology. Приложение, касающееся плавающей запятой, было изначально написано David Goldberg из Xerox PARC.

Упражнения

George Adams, Purdue University; Todd M. Bezenek, University of Wisconsin–Madison (в память своей бабушки Ethel Eshom); Susan Eggers; Anoop Gupta; David Hayes; Mark Hill; Allan Knies; Ethan L. Miller, University of California, Santa Cruz; Parthasarathy Ranganathan, Compaq Western Research Laboratory; Brandon Schwartz, University of Wisconsin–Madison; Michael Scott; Dan Siewiorek; Mike Smith; Mark Smotherman; Evan Tick; Thomas Willis.

Учебные примеры с упражнениями

Andrea C. Arpaci-Dusseau, University of Wisconsin–Madison; Remzi H. Arpaci-Dusseau, University of Wisconsin–Madison; Robert P. Colwell, R&E Colwell & Assoc., Inc.; Diana Franklin, California Polytechnic State University, San Luis Obispo; Wen-mei W. Hwu, University of Illinois at Urbana–Champaign; Norman P. Jouppi, HP Labs; John W. Sias, University of Illinois at Urbana–Champaign; David A. Wood, University of Wisconsin–Madison.

Особая благодарность

Duane Adams, Defense Advanced Research Projects Agency; Tom Adams; Sarita Adve, University of Illinois at Urbana–Champaign; Anant Agarwal; Dave Albonesi, University of Rochester; Mitch Alsup; Howard Alt; Dave Anderson; Peter Ashenden; David Bailey; Bill Bandy, Defense Advanced Research Projects Agency; Luiz Barroso, Compaq's Western Research Lab; Andy Bechtolsheim; C. Gordon Bell; Fred Berkowitz; John Best, IBM; Dileep Bhandarkar; Jeff Bier, BDTI; Mark Birman; David Black; David Boggs; Jim Brady; Forrest Brewer; Aaron Brown, University of California, Berkeley; E. Bugnion, Compaq's Western Research Lab; Alper Buyuktosunoglu, University of Rochester; Mark Callaghan; Jason F. Cantin; Paul Carrick; Chen-Chung Chang; Lei Chen, University of Rochester; Pete Chen; Nhan Chu; Doug Clark, Princeton University; Bob Cmelik; John Crawford; Zarka Cvetanovic; Mike Dahlin, University of Texas, Austin; Merrick Darley; сотрудникам DEC Western Research Laboratory; John DeRosa; Lloyd Dickman; J. Ding; Susan Eggers, University of Washington; Wael El-Essawy, University of Rochester; Patty Enriquez, Mills; Milos Ercegovac; Robert Garner; K. Gharachorloo, Compaq's Western Research Lab; Garth Gibson; Ronald Greenberg; Ben Hao; John Henning, Compaq; Mark Hill, University of Wisconsin–Madison; Danny Hillis; David Hodges; Urs Holzle, Google; David Hough; Ed Hudson; Chris Hughes, University of Illinois at Urbana–Champaign; Mark Johnson; Lewis Jordan; Norm Jouppi; William Kahan; Randy Katz; Ed Kelly; Richard Kessler; Les Kohn; John Kowaleski, Compaq Computer Corp; Dan Lambright; Gary Lauterbach, Sun Microsystems; Corinna Lee; Ruby Lee; Don Lewine; Chao-Huang Lin; Paul Losleben, Defense Advanced Research Projects Agency; Yung-Hsiang Lu; Bob Lucas, Defense Advanced Research Projects Agency; Ken Lutz; Alan Mainwaring, Intel Berkeley Research Labs; Al Marston; Rich Martin,

Rutgers; John Mashey; Luke McDowell; Sebastian Mirolo, Trimedia Corporation; Ravi Murthy; Biswadeep Nag; Lisa Noordergraaf, Sun Microsystems; Bob Parker, Defense Advanced Research Projects Agency; Vern Paxson, Center for Internet Research; Lawrence Prince; Steven Przybylski; Mark Pullen, Defense Advanced Research Projects Agency; Chris Rowen; Margaret Rowland; Greg Semeraro, University of Rochester; Bill Shannon; Behrooz Shirazi; Robert Shomler; Jim Slager; Mark Smotherman, Clemson University; исследовательская группа в области SMT в University of Washington; Steve Squires, Defense Advanced Research Projects Agency; Ajay Sreekanth; Darren Staples; Charles Stapper; Jorge Stolfi; Peter Stoll; студенты Stanford и Berkeley, которые безропотно переносили наши первые попытки создать эту книгу; Bob Supnik; Steve Swanson; Paul Taysom; Shreekant Thakkar; Alexander Thomasian, New Jersey Institute of Technology; John Toole, Defense Advanced Research Projects Agency; Kees A. Vissers, Trimedia Corporation; Willa Walker; David Weaver; Ric Wheeler, EMC; Maurice Wilkes; Richard Zimmerman.

Джон Хеннесси
Дэвид Паттерсон

ГЛАВА 1

ОСНОВЫ КОЛИЧЕСТВЕННОГО ПРОЕКТИРОВАНИЯ И АНАЛИЗА

Думаю, справедливо утверждать, что персональные компьютеры стали наиболее мощным инструментом из всех когда-либо созданных нами. Они являются средствами общения, инструментами творчества, и пользователь может изменять их.

Билл Гейтс (Bill Gates), 24 февраля 2004 г.

1.1. Введение

Компьютерная технология невероятно прогрессировала за неполные 65 лет с момента создания первого универсального электронного компьютера. Сегодня меньше чем за 500 долл. можно приобрести портативный компьютер с большей производительностью, большей оперативной памятью и большей дисковой памятью, чем у компьютера, купленного в 1985 г. за 1 млн долл. Это быстрое усовершенствование произошло как благодаря успехам в технологии, применявшейся при производстве компьютеров, так и благодаря нововведениям в проектировании компьютеров.

Несмотря на то, что технологические усовершенствования носили достаточно устойчивый характер, прогресс от улучшения компьютерных архитектур был значительно менее последовательным. В течение первых 25 лет существования электронных компьютеров обе эти движущие силы внесли основной вклад в ежегодное увеличение производительности примерно на 25 %. Конец 1970-х гг. стал свидетелем появления микропроцессора. Способность микропроцессора использовать прогресс в технологии интегральных схем привела к еще большему росту производительности — примерно 35 % в год.

Такой темп роста в сочетании с преимуществами в стоимости массово производимого микропроцессора привел к увеличению доли компьютерного бизнеса, основанного на микропроцессорах. Вдобавок два существенных изменения на компьютерном рынке способствовали небывало легкому коммерческому успеху новой архитектуры. Во-первых, фактическое исключение программирования на языке ассемблера уменьшило необходимость в совместимости по объектному коду. Во-вторых, создание стандартизованных, независимых от поставщика операционных систем, таких как UNIX и его клон Linux, понизило стоимость и риск освоения новой архитектуры.

Эти изменения в начале 1980-х гг. сделали возможной успешную разработку нового ряда архитектур с более простыми командами, названных RISC. Машины, основанные на RISC-архитектуре, сосредоточили внимание разработчиков на двух решающих методах увеличения производительности: использовании *параллелизма уровня команд* (сначала посредством конвейерной обработки, а позже посредством запуска нескольких команд в одном такте) и использовании кэшей (сначала в простых формах, а позже с использованием более сложных организаций и оптимизаций).

Компьютеры с RISC-архитектурами повысили планку производительности, обязывая предшествующие архитектуры держаться на их уровне или исчезнуть. Архитектура Vax компании Digital Equipment не смогла удержаться и была заменена RISC-архитектурой. Главным ответом компании Intel на вызов стала трансляция команд архитектуры 80x86 в RISC-подобные команды внутри процессора, что позволило принять многие инновации, впервые появившиеся в RISC-проектах. Когда в конце 1990-х гг. количество транзисторов резко возросло, аппаратные издержки на трансляцию более сложной архитектуры x86 стали незначительными. В портативных применениях, например сотовых телефонах, стоимость затрат на питание и кремний в связи с трансляцией x86 способствовала тому, что доминирующей стала RISC-архитектура ARM.

На рис. 1.1 показано, что сочетание улучшений в архитектуре и организации привело к постоянному ежегодному приросту производительности более чем на 50 % в течение 17 лет — беспрецедентный темп в истории компьютерной индустрии.

Эффект от столь впечатляющего темпа роста в XX веке имел четыре последствия. Во-первых, он существенно расширил возможности, доступные пользователям компьютеров. Для многих приложений современные наиболее производительные микропроцессоры превосходят суперкомпьютеры менее чем 10-летней давности.

Во-вторых, такое впечатляющее улучшение соотношения стоимость — производительность ведет к появлению новых классов компьютеров. Персональные компьютеры и рабочие станции появились в 1980-х гг. вследствие доступности микропроцессоров. В последнем десятилетии наблюдалось широкое распространение интеллектуальных мобильных телефонов и планшетных компьютеров, которые многие используют в качестве основных вычислительных платформ вместо персональных компьютеров. Эти мобильные клиентские устройства все больше используют Интернет для доступа к центрам обработки данных, содержащим десятки тысяч серверов и спроектированным как один гигантский компьютер.

В-третьих, продолжающееся в соответствии с законом Мура усовершенствование производства полупроводников привело к доминированию компьютеров на основе микропроцессоров во всем диапазоне компьютерных устройств. Мини-компьютеры, которые традиционно делались с применением покупной логики или вентильных матриц, были вытеснены серверами, изготовленными с использованием микропроцессоров. Все, даже большие универсальные компьютеры (mainframe) и высокопроизводительные суперкомпьютеры, представляют собой наборы микропроцессоров.

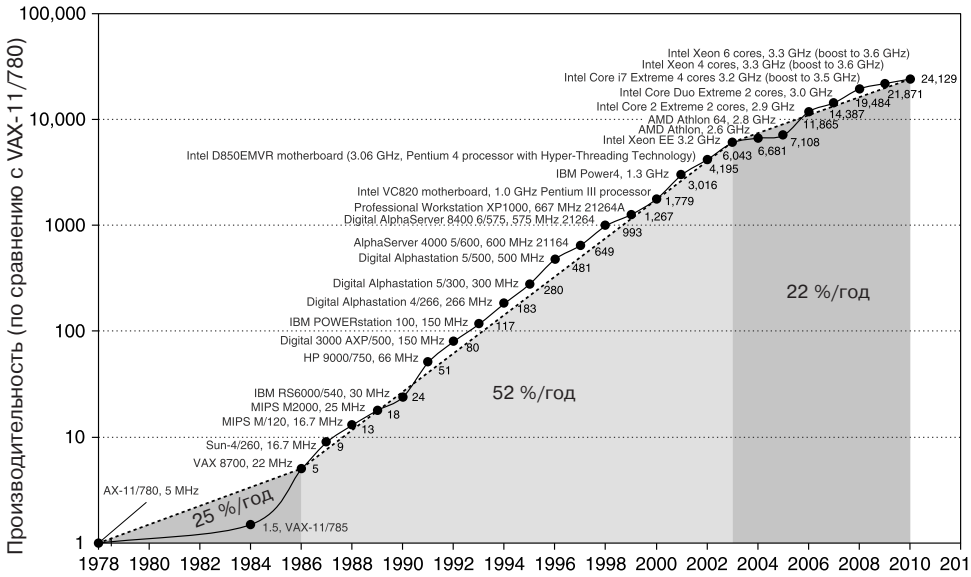

Рис. 1.1. Рост производительности процессоров с конца 1970-х годов. На этом рисунке представлен график производительности по отношению к компьютеру VAX 11/780, измеренной на тестах SPEC (Standard Performance Evaluation Corporation) (см. раздел 1.8). До середины 1980-х гг. рост производительности процессоров достигался в значительной степени за счет технологии и составлял в среднем около 25 % в год. Последующее увеличение прироста примерно до 52 % связано с более продвинутыми архитектурными и организационными идеями. К 2003 г. этот прирост привел к увеличению производительности примерно в 25 раз по сравнению с тем, какой бы она была, если бы мы продолжали оставаться в пределах 25 %. Производительность вычислений с плавающей запятой увеличивалась даже быстрее. Начиная с 2003 г. ограничения мощности и доступного параллелизма уровня команд привели к замедлению роста однопроцессорной производительности до величины, не превышавшей 22 % в год, или примерно в 5 раз медленнее, чем если бы мы продолжали при 52 % в год. (Каждый год, начиная с 2007 г., наивысшая производительность на тестах SPEC оценивалась с включенным автоматическим распараллеливанием при увеличивающемся количестве ядер на чип. Поэтому однопроцессорную скорость труднее оценить. Эти результаты ограничивались системами с одним сокетом[1] для снижения влияния автоматического распараллеливания.) На рис. 1.11 в разделе 1.5 (на стр. 59) показано увеличение тактовых частот для этих трех упомянутых отрезков времени. Поскольку с годами тесты SPEC менялись, производительность более новых машин оценивается с помощью масштабного коэффициента, который соотносит производительность для двух разных версий SPEC (например SPEC89, SPEC92, SPEC95, SPEC2000 и SPEC2006)

[1] Сокет (Socket) — модуль, содержащий один или два микропроцессора и устанавливаемый в разъем материнской платы компьютера. — *Прим. ред.*

Упомянутые выше аппаратные инновации привели к существенным изменениям в проектировании компьютеров, которое уделяло основное внимание как архитектурным новшествам, так и эффективному использованию технологического прогресса. Этот темп роста был составным, так что к 2003 г. высокопроизводительные микропроцессоры были в 7,5 раза быстрее того, что было бы достигнуто исключительно за счет технологии, включая улучшения в схемотехнике, то есть 52 % против 35 % в год.

Существенные изменения в аппаратуре привели к четвертому последствию, на этот раз касающемуся программного обеспечения. Увеличение производительности с 1978 г. в 25 тыс. раз (см. рис. 1.1) в наши дни позволило программистам разменивать производительность на продуктивность. Вместо языков, ориентированных на производительность, таких как C и C++, сегодня в программировании гораздо больше используются управляемые языки программирования, такие как Java и C#. Более того, языки сценариев, такие как Python и Ruby, еще продуктивные, приобретают все большую популярность в таких средах программирования, как Ruby_on_Rails. Чтобы поддерживать продуктивность и при этом компенсировать потерю производительности, интерпретаторы с динамическими компиляторами и компиляция на основе трассы программы замещают традиционный компилятор и линковщик прошлого. Также изменяется и распространение программного обеспечения — сервис «Программное обеспечение как коммунальная услуга» (SaaS — Software as a Service), используемый через Интернет, заменяет программное обеспечение, которое инсталлируется и затем работает на локальном компьютере.

Изменяется также и природа приложений. Речь, звук, изображения и видео становятся все более важными, одновременно растет значимость предсказуемого времени отклика, столь существенного в пользовательской практике. Вдохновляющим примером служит Google Goggles (взгляд Google). Это приложение позволяет вам, направив камеру своего мобильного телефона на какой-либо объект, переслать его изображение через Интернет в компьютер масштаба склада WSC[1], который распознает объект и сообщит вам интересную информацию о нем. Приложение может перевести текст, имеющийся на объекте, на другой язык, прочитать штрих-код на обложке книги, чтобы сообщить вам, доступна ли данная книга онлайн и ее цену, или, если вы просканируете камерой телефона панораму вокруг, сообщить вам, какие предприятия находятся неподалеку с их веб-сайтами, телефонными номерами и направлениями деятельности.

Увы, на рис. 1.1 также показано, что это 17-летнее возрождение аппаратуры закончилось. С 2003 г. прирост однопроцессорной производительности упал до величины, меньшей 22 % в год, вследствие двух препятствий: максимальной мощности рассеивания чипов с воздушным охлаждением и отсутствия большего параллелизма уровня команд, который можно эффективно использовать. Действительно, в 2004 г. компания Intel отменила проекты высокопроизводительных одноядерных процессоров и присоединилась к другим, утверждая, что путем к более высокой

[1] Компьютер WSC иначе можно называть масштабным центром обработки данных, но в соответствии с оригиналом далее будем использовать термин «компьютер WSC». — *Прим. ред.*

производительности должны быть несколько процессоров на кристалле, а не более быстрые одноядерные процессоры.

Эта веха сигнализирует об историческом переходе от использования исключительно параллелизма уровня команд (*ILP*), основной темы первых трех изданий этой книги, к использованию *параллелизма уровня данных* (*DLP*) и *параллелизма уровня потоков* (*TLP*), которые были рассмотрены в четвертом издании и более подробно рассматриваются в этом издании. В этом издании также добавлены компьютеры WSC и *параллелизм уровня запросов* (*RLP*). В то время как компилятор и аппаратура используют ILP неявным образом, не требуя внимания программиста, DLP, TLP и RLP являются параллельными явно, требующими реструктуризации приложения так, чтобы можно было использовать явный параллелизм. В некоторых случаях это просто, но в большинстве случаев является новой серьезной заботой для программистов.

Эта книга об архитектурных концепциях и сопутствующих им усовершенствованиях компиляторов, которые сделали возможным невероятный темп роста в прошлом столетии, причинах драматической перемены, проблемах и первых многообещающих подходах к архитектурным концепциям, компиляторам и интерпретаторам XXI века. В основе его лежит количественный подход к проектированию компьютеров и анализу, который использует эмпирическое изучение поведения программ, экспериментирование и моделирование в качестве инструментов. Именно такой стиль и подход к проектированию компьютеров нашли отражение в этой книге. Цель главы 1 — заложить количественную основу, на которой и будут основаны остальные главы и приложения.

Эта книга написана не только для объяснения такого стиля проектирования, но также для того, чтобы побудить вас внести свой вклад в его развитие. Мы считаем, что этот подход будет работать для компьютеров с явным параллелизмом будущего так же, как он работал для компьютеров с неявным параллелизмом прошлого.

1.2. Классы компьютеров

Эти изменения создали основу для изменения нашего взгляда на вычислительную технику, ее применение и рынки компьютеров в новом столетии. Со времени создания персонального компьютера мы не видели таких поразительных изменений в создании и использовании компьютеров. Эти изменения привели к появлению пяти различных секторов рынка вычислительной техники, каждый из которых характеризуется различными применениями, требованиями и вычислительными технологиями. На рис. 1.2 обобщаются эти основные классы компьютеров и их важные характеристики.

В 2010 г. было продано около 1,8 млрд PMD (90 % из них — мобильные телефоны), 350 млн настольных ПК и 20 млн серверов. Общее количество продаж встроенных процессоров приблизилось к 19 млрд. В общей сложности 6,1 млрд чипов на основе ARM-технологии были поставлены на рынок в 2010 г. Следует обратить внимание на широкий диапазон системной цены для серверов и встроенных систем, который начинается USB-ключами и заканчивается сетевыми маршрутизаторами.

Характеристика	Персональное мобильное устройство (PMD)	Настольный компьютер	Сервер	Кластер/WSC	Встроенные
Цены системы, долл.	100–1000	300–2500	5000–10000000	100000–200000000	10–100000
Цена микропроцессора, долл.	10–100	50–500	200–2000	50–250	0,01–100
Критические параметры при проектировании системы	Цена, энергопотребление, медиапроизводительность, время отклика	Цена, производительность, энергопотребление, графическая производительность	Скорость обработки, доступность, масштабируемость, энергопотребление	Цена, производительность, пропускная способность, пропорциональность энергопотребления	Цена, энергопотребление, прикладная производительность

Рис. 1.2. Краткие данные пяти основных классов вычислительной техники и их системные характеристики

Для серверов такой диапазон обусловлен потребностями в крупномасштабных мультипроцессорных системах для высокопроизводительной обработки транзакций.

Персональное мобильное устройство (PMD)

Персональное мобильное устройство — это название мы относим к множеству беспроводных устройств с мультимедийным пользовательским интерфейсом, таким как сотовые телефоны, планшетные компьютеры и т.п. Стоимость здесь крайне важна, учитывая, что потребительская цена всего продукта составляет несколько сотен долларов. Хотя акцент на энергетическую эффективность часто обусловлен использованием батарей, необходимость использования менее дорогого корпуса — пластик вместо керамики — и отсутствие вентилятора для охлаждения также ограничивают общее энергопотребление. Мы рассмотрим проблему энергопотребления и мощности более подробно в разделе 1.5. Приложения для PMD часто основаны на интернет-технологиях и ориентированы на среду, как в приведенном выше примере с Google Goggles. Требования к энергопотреблению и размеру определяют использование флэш-памяти вместо магнитных дисков для внешней памяти (глава 2).

Время отклика и предсказуемость являются ключевыми характеристиками для медиаприложений. Требование *производительности в реальном времени* (*Real Time Performance*) означает, что некоторый сегмент приложения имеет абсолютное максимальное время выполнения. Например, при проигрывании видео на PMD время обработки каждого видеокадра ограничено, поскольку процессор должен быстро принять и обработать следующий кадр. В некоторых приложениях существуют и более тонкие нюансы в требованиях: ограничиваются и среднее время выполнения отдельной задачи, и количество случаев, когда некоторое максимальное время превышается. Такие подходы, иногда называемые *мягким реальным временем* (*Soft*

Real-Time), возникают, когда для некоторого события можно иногда отступить от временного ограничения — лишь бы это было не слишком часто. Производительность в реальном времени обычно сильно зависит от приложения.

Другие ключевые характеристики во многих приложениях для PMD определяются необходимостью минимизировать объем памяти и эффективно использовать энергию. Энергоэффективность необходима из-за питания от батареи и рассеивания тепла. Память может составлять существенную часть стоимости системы, и в таких случаях важно оптимизировать ее объем. Ввиду важности объема памяти особое внимание придается размеру кода, поскольку размер данных диктуется приложением.

Настольная вычислительная техника

Первый и, возможно, до сих пор самый большой рынок в долларовом выражении образует настольная вычислительная техника. Настольная вычислительная техника простирается от бюджетных нетбуков стоимостью до 300 долл. до высокопроизводительных, хорошо оснащенных рабочих станций, которые могут продаваться и за 2500 долл. С 2008 г. более половины изготовлявшихся каждый год настольных компьютеров представляли собой портативные компьютеры (laptop), работающие от батарей.

Во всем этом диапазоне цен и возможностей рынок настольных систем стремится оптимизировать соотношение *цена — производительность*. Это сочетание производительности (измеряемой главным образом в терминах вычислительной и графической производительности) и цены системы и есть главное для покупателей на этом рынке и, следовательно, для разработчиков компьютеров. В результате новейшие, наиболее высокопроизводительные микропроцессоры и относительно дешевые микропроцессоры часто появляются сначала в настольных системах (см. раздел 1.6, в котором рассматриваются факторы, влияющие на стоимость компьютеров).

Существует тенденция представлять характеристики настольной вычислительной техники через приложения и тесты, хотя все большее использование веб-ориентированных интерактивных приложений создает новые проблемы в оценке производительности.

Серверы

По мере перехода к настольной вычислительной технике в 1980-х гг. возрастала роль серверов как средства обеспечения крупномасштабных и более надежных файловых и вычислительных сервисов. Заменяя традиционные большие ЭВМ, такие серверы стали основой крупномасштабной вычислительной техники на предприятиях.

Для серверов важны различные характеристики. Во-первых, важной характеристикой является доступность (см. раздел 1.7). Представим себе серверы, работающие в АТМ-машинах (Automated Teller Machine, автоматическая кассовая машина, например банкомат) для банков или в системах бронирования авиабилетов. Отказ в таких серверных системах значительно более катастрофичен, чем отказ одной настольной системы, поскольку эти серверы должны работать семь дней в неделю, 24 часа в сутки. На рис. 1.3 приведены оценки стоимости простоя для серверных приложений.

Приложение	Стоимость простоя, долл. в час	Ежегодные потери (долл.) от простоев, составляющих		
		1% (87,6 ч/год)	0,5% (43,8 ч/год)	0,1% (8,8 ч/год)
Брокерские операции	6 450 000	565 000 000	283 000 000	56 500 000
Авторизация кредитных карт	2 600 000	228 000 000	114 000 000	22 800 000
Службы доставки посылок	150 000	13 000 000	6 600 000	1 300 000
Канал покупок на дому	113 000	9 900 000	4 900 000	1 000 000
Центр продаж по каталогам	90 000	7 900 000	3 900 000	800 000
Центр бронирования авиабилетов	89 000	7 900 000	3 900 000	800 000
Служба активации мобильных телефонов	41 000	3 600 000	1 800 000	400 000
Оплата сетевых услуг онлайн	25 000	2 200 000	1 100 000	200 000
Оплата обслуживания банкоматов	14 000	1 200 000	600 000	100 000

Рис. 1.3. Показана стоимость недоступности систем, округленная до 100 000 долл., полученная на основе анализа стоимости простоя (в пересчете на непосредственно потерянный доход). При этом предполагаются три различных уровня доступности и равномерное распределение простоев. Данные получены от Kembel [2000], собраны и проанализированы службой Contingency Planning Research

Второй ключевой характеристикой серверных систем является масштабируемость. Серверные системы часто расширяются в ответ на увеличение спроса на услуги, которые они поддерживают, или на увеличение функциональных требований. Поэтому способность масштабировать вычислительную мощность, оперативную память, внешнюю память и пропускную способность ввода/вывода является для сервера крайне важной.

И наконец, серверы проектируют для эффективной пропускной способности. То есть общая производительность сервера, измеренная в транзакциях в секунду или в количестве обработанных в секунду веб-страниц, — вот что важно. Время отклика на отдельный запрос остается важным, но общая и экономическая эффективность, определенные как количество запросов, которые могут быть обработаны в единицу времени, являются ключевыми характеристиками большинства серверов. Мы вернемся к теме оценки производительности для различных типов компьютерной среды в разделе 1.8.

Кластеры / Компьютеры WSC

Развитие сервиса «Программное обеспечение как коммунальная услуга» (SaaS) для таких приложений, как поиск, социальные сети, обмен видео, многопользовательские игры, интернет-магазины и т.п., привело к появлению класса компьютеров, названных *кластерами*. Кластеры — это набор настольных компьютеров или серверов, соединенных между собой локальными сетями, чтобы работать как один компьютер большего размера. В каждом узле работает своя операционная система,

и узлы общаются между собой, используя сетевой протокол. Самые большие кластеры называются *компьютерами масштаба склада (WSC)*, поскольку они проектируются так, чтобы десятки тысяч серверов могли действовать как один компьютер. В главе 6 описывается этот класс сверхбольших компьютеров.

Соотношение цена — производительность и мощность крайне важны для компьютеров WSC, поскольку они так велики. Как будет показано в главе 6, 80 % расходов на компьютер WSC стоимостью 90 млн долл. связаны с энергопотреблением и охлаждением составляющих его компьютеров. Сами компьютеры и сетевое оборудование стоят еще 70 млн долл., и их следует менять каждые несколько лет. Когда вы приобретаете так много вычислительной техники, вам надо делать это с умом, так как 10%-е улучшение соотношения цена — производительность означает экономию в 7 млн долл. (10 % от 70 млн долл.).

Компьютеры WSC похожи на серверы, для которых доступность является важной характеристикой. Компания Amazon.com, к примеру, получила 13 млрд долл. от продаж в четвертом квартале 2010 г. Поскольку в одном квартале примерно 2200 часов, средний доход в час составил почти 6 млн долл. На рождественских продажах в час пик потенциальные потери от простоя были бы во много раз больше. Как будет показано в главе 6, компьютеры WSC отличаются от серверов тем, что в качестве конструктивных, компоновочных блоков используют недорогие компоненты с избыточностью, используя программное обеспечение, чтобы выявить и изолировать многие отказы, которые будут иметь место в вычислительной технике такого масштаба. Заметим, что масштабируемость компьютеров WSC обеспечивается локальной сетью, соединяющей компьютеры, а не встроенным оборудованием компьютера, как в случае серверов.

Суперкомпьютеры похожи на компьютеры WSC в том, что они так же дороги и стоят сотни миллионов долларов, однако суперкомпьютеры отличаются ориентацией на производительность с плавающей запятой и выполнение больших с интенсивным обменом циклических программ, которые могут выполняться несколько недель за один запуск. Такое плотное взаимодействие ведет к использованию значительно более быстрых внутренних сетей. Компьютеры WSC, напротив, ориентированы на интерактивные приложения, большую внешнюю память, системную надежность и высокую пропускную способность Интернета.

Встроенные компьютеры

Встроенные компьютеры присутствуют в повседневных машинах. Микроволновые печи, стиральные машины, большинство принтеров, большинство сетевых коммутаторов и все автомобили содержат простые встроенные микропроцессоры.

Процессоры в PMD часто рассматриваются как встроенные компьютеры, однако мы относим их к отдельной категории, потому что PMD представляют собой платформы, которые могут выполнять программное обеспечение сторонних производителей, и у них есть много характеристик, присущих настольным компьютерам. Другие встроенные устройства больше ограничены уровнем сложности аппаратуры и программного обеспечения. Мы используем способность выполнять программное обеспечение сторонних производителей как разделительную линию между невстроенными и встроенными компьютерами.

Встроенные компьютеры имеют наибольший разброс в вычислительной мощности и стоимости. Они включают 8- и 16-разрядные процессоры, которые могут стоить меньше 10 центов, 32-разрядные микропроцессоры, которые выполняют 100 млн команд в секунду и стоят в пределах 5 долл., а также высокопроизводительные процессоры для сетевых коммутаторов, которые стоят 100 долл. и могут выполнять миллиарды команд в секунду. Хотя диапазон вычислительной мощности на рынке встроенной вычислительной техники очень велик, цена является ключевым фактором при разработке компьютеров для этой области. Конечно, в действительности существуют требования к производительности, но главной целью часто является скорее достижение необходимой производительности при минимальной цене, а не достижение более высокой производительности при более высокой цене.

Большая часть этой книги относится к проектированию, использованию и производительности встроенных процессоров, как имеющихся в продаже микропроцессоров, так и микропроцессорных ядер, объединяемых с другой специальной аппаратурой. Так, третье издание данной книги содержало примеры встроенных систем для иллюстрации идей каждой главы.

К сожалению, большинство читателей сочли эти примеры неудовлетворительными, так как данные, на которых построены количественное проектирование и оценка других классов компьютеров, до сих пор не были в достаточной степени распространены на встроенные системы (см., например, проблемы с EEMBC в разделе 1.8). Следовательно, в настоящее время для них имеются только качественные описания, которые не сочетаются с остальной частью книги. Поэтому в этом и предыдущем изданиях материал по встроенным системам объединен в приложении E. Мы полагаем, что отдельное приложение улучшает изложение идей в тексте и позволяет читателям понять, как различные требования влияют на встроенные системы.

Классы параллелизма и параллельные архитектуры

В настоящее время параллелизм на различных уровнях является движущей силой проектирования компьютеров во всех четырех классах, при этом главными ограничениями являются потребление энергии и стоимость. В принципе имеются два вида параллелизма в приложениях.

1. *Параллелизм уровня данных* (*DLP*) возникает потому, что имеется много элементов данных, которые могут быть обработаны одновременно.
2. *Параллелизм уровня задач* (*TLP*) возникает потому, что создаются задачи, которые могут выполняться независимо и в значительной степени параллельно.

В свою очередь, аппаратура компьютера может использовать эти два вида параллелизма в приложениях четырьмя основными способами.

1. *Параллелизм уровня команд* (*ILP*) использует параллелизм уровня данных на нижних уровнях с помощью компилятора, применяя конвейерную обработку, и на средних уровнях, применяя спекулятивное выполнение.
2. *Векторные архитектуры и графические процессоры* (*GPUs*) используют параллелизм уровня данных, выполняя одну команду над набором данных параллельно.

3. *Параллелизм уровня потоков (TLP*) использует либо параллелизм уровня данных, либо параллелизм уровня задач в сильно связанной аппаратной модели, которая делает возможным взаимодействие параллельных потоков.
4. *Параллелизм уровня запросов* (*RLP*) использует параллелизм в основном несвязанных задач, определяемый программистом или операционной системой.

Эти четыре способа поддержки аппаратурой параллелизма уровня данных и уровня задач известны уже 50 лет. Когда Майкл Флинн (Michael Flynn) [1966] изучал работы 1960-х гг., посвященные аппаратуре для параллельных вычислений, он нашел простую классификацию, сокращенные названия которой мы по-прежнему используем в наши дни. Он рассмотрел параллелизм в потоках команд и параллелизм данных, вызываемых командами, в наиболее ограниченном узле мультипроцессора и разделил все компьютеры по четырем категориям.

1. *Один поток команд, один поток данных* (*SISD — Single Instruction Stream, Single Data Stream*) — к этой категории относится однопроцессорная система. Программист считает его стандартным последовательным компьютером, однако однопроцессорная система может использовать параллелизм уровня команд. В главе 3 рассматриваются SISD-архитектуры, которые используют ILP-методы, такие как суперскалярное и спекулятивное выполнение.
2. *Один поток команд, несколько потоков данных* (*SIMD — Single Instruction Stream, Multiple Data Streams*) — одна и та же команда выполняется несколькими процессорами, использующими разные потоки данных. SIMD-компьютеры используют *параллелизм уровня данных*, применяя одни и те же операции ко многим элементам данных параллельно. Каждый процессор имеет свою собственную память данных (отсюда MD в SIMD), но имеются только одна память команд и один управляющий процессор, который выбирает и выдает команды. В главе 4 рассматривается параллелизм уровня данных DLP и три различные архитектуры, которые его используют: векторные архитектуры, мультимедийные расширения стандартных систем команд и графические процессоры GPU.
3. *Несколько потоков команд, один поток данных (MISD — Multiple Instruction Streams, Single Data Stream*s) — к настоящему моменту не было построено ни одного коммерческого мультипроцессора такого типа, однако эта категория делает простую классификацию полной.
4. *Несколько потоков команд, несколько потоков данных* (*MIMD — Multiple Instruction Streams, Multiple Data Streams*) — каждый процессор выбирает свои собственные команды и выполняет операции со своими собственными данными, и он нацелен на параллелизм уровня задач. В общем, категория MIMD является более гибкой, чем SIMD, и поэтому применяется более широко, однако она, по существу, дороже, чем SIMD. Например, MIMD-компьютеры также могут использовать параллелизм уровня данных, хотя накладные расходы, вероятно, будут выше, чем при использовании SIMD-компьютеров. Такие накладные расходы означают, что для эффективного использования параллелизма размер структурных единиц (зерно) должен быть достаточно большим. В главе 5 рассматриваются сильно связанные MIMD-архитектуры, которые используют *параллелизм уровня потоков*, поскольку многие взаимодействующие потоки работают параллельно. В главе 6

рассматриваются слабосвязанные MIMD-архитектуры, а именно *кластеры* и *компьютеры WSC*, с параллелизмом *уровня запросов*, где многие независимые задачи могут выполняться параллельно естественным образом, имея малую потребность в обмене данными или синхронизации.

Эта классификация является грубой, поскольку многие параллельные процессоры являются гибридами SISD-, SIMD- и MIMD-классов. Тем не менее полезно ввести разграничения в той области проектирования компьютеров, которая будет рассмотрена в этой книге.

1.3. Определение компьютерной архитектуры

Задача, с которой сталкивается разработчик компьютера, сложна: определить, какие параметры важны для нового компьютера, а затем спроектировать компьютер, добиваясь максимальной производительности и энергетической эффективности, при этом оставаясь в пределах ограничений на стоимость, мощность и доступность. Эта задача имеет множество аспектов, включая разработку системы команд, функциональную организацию, логическое проектирование и реализацию. Реализация может включать проектирование интегральной схемы, корпуса, питания и охлаждения. Оптимизация проекта требует знания очень широкого диапазона технологий, от компиляторов и операционных систем до логического проектирования и корпусирования.

Несколько лет назад термин «*компьютерная архитектура*» часто относился только к разработке системы команд. Другие аспекты проектирования компьютера назывались *реализацией*, намекая, что реализация является неинтересной или менее сложной.

Мы полагаем, что такая точка зрения неверна. Работа архитектора или разработчика включает в себя много больше, чем разработка системы команд, и технические трудности в других частях проекта, вероятно, являются более сложными, чем встретившиеся при разработке системы команд. Рассмотрим кратко архитектуру системы команд, прежде чем приступить к описанию более серьезных проблем, стоящих перед компьютерным архитектором.

Архитектура системы команд: ограниченный взгляд на компьютерную архитектуру

В этой книге используется термин «*архитектура системы команд*» (*ISA —Instruction Set Architecture*) для обозначения реальной, видимой программисту системы команд. ISA служит границей между программным обеспечением и аппаратурой. В этом кратком обзоре ISA будут использованы примеры из систем команд 80x86, ARM и MIPS для иллюстрации семи характеристик ISA. В приложениях A и K приводятся более подробные описания этих трех ISA.

1. *Класс ISA.* Почти все ISA сегодня классифицируются как архитектуры с регистрами общего назначения, в которых операндами являются либо регистры, либо ячейки памяти. Архитектура 80x86 имеет 16 регистров общего назначения и 16 регистров, способных хранить данные с плавающей запятой, тогда как в архитектуре MIPS имеются 32 регистра общего назначения и 32 регистра для хранения данных с плавающей запятой (см. рис. 1.4). Двумя популярными предста-

Имя	Номер	Назначение	Сохраняется ли между вызовами процедур
$zero	0	Постоянное значение 0	Неприменимо
$at	1	Временное значение ассемблера	Нет
$v0–$v1	2—3	Значения результатов функций и вычисления выражений	Нет
$a0–$a3	4—7	Аргументы	Нет
$t0–$t7	8—15	Временные значения	Нет
$s0–$s7	16—23	Сохраненные временные значения	Да
$t8–$t9	24—25	Временные значения	Нет
$k0–$k1	26—27	Зарезервированы для ядра ОС	Нет
$gp	28	Глобальный указатель	Да
$sp	29	Указатель стека	Да
$fp	30	Указатель фрейма	Да
$ra	31	Адрес возврата	Да

Рис. 1.4. Регистры MIPS и их назначение. В дополнение к 32 регистрам общего назначения (R0—R31) в архитектуре MIPS имеются 32 регистра с плавающей запятой (F0—F31), каждый из которых может содержать или 32-разрядное число одинарной точности, или 64-разрядное число двойной точности

вителями этого класса являются: ISA типа *регистр-память*, такая как архитектура 80x86, в которой доступ в память возможен как часть выполнения многих команд, и ISA типа *считывание-запись*, такие как архитектуры ARM и MIPS, в которых доступ в память возможен только командами «загрузить» и «сохранить». Все недавние ISA являются архитектурами типа считывание-запись.

2. *Адресация памяти*. Фактически все настольные компьютеры и серверы, в том числе с архитектурами 80x86, ARM и MIPS, используют байтовую адресацию для обращения к операндам в памяти. Некоторые архитектуры, подобные ARM и MIPS, требуют, чтобы объекты были *выровнены*. Доступ к объекту размером s байтов по байтовому адресу A считается выровненным, если A mod s = 0 (см. рис. A.5 на с. 597). Архитектура 80x86 не требует выравнивания, однако обращения в память обычно быстрее, если операнды выровнены.
3. *Режимы адресации*. Кроме указания регистров и констант в качестве операндов, режимы адресации задают адрес объекта в памяти. Режимами адресации в архитектуре MIPS являются регистровый, непосредственный (для констант) и смещения, в котором для формирования адреса в памяти к содержимому регистра прибавляется постоянное смещение. Архитектура 80x86 поддерживает эти три базовых режима и еще имеет три варианта смещения: без регистра (абсолютная адресация), два регистра (база с индексом и со смещением) и два регистра, когда один из регистров (индекс) умножается на размер элемента в байтах (база со сдвинутым индексом и со смещением). Также из базовых вариантов для смещения можно получить производные варианты адресации путем отбрасывания

смещения: адресация по базовому регистру (относительная), по индексному регистру (умноженный индексный регистр) и их комбинации (базовый регистр со сдвинутым индексом). Архитектура ARM имеет три режима адресации архитектуры MIPS и еще адресацию относительно счетчика команд PC (program counter), сумму двух регистров и сумму двух регистров с умножением содержимого одного из регистров на значение операнда в байтах. Имеются также автоинкрементная и автодекрементная адресации, при которых вычисленный адрес заменяет содержимое одного из регистров, использовавшихся для формирования адреса.

4. *Типы и размеры операндов.* Подобно большинству ISA, архитектуры 80x86, ARM и MIPS поддерживают размеры операндов 8 бит (символ ASCII), 16 бит (символ Unicode или полуслово), 32 бита (целое или слово), 64 бита (двойное слово или длинное целое) и числа с плавающей запятой стандарта IEEE 754 — 32 бита (одинарная точность) и 64 бита (двойная точность). Архитектура 80x86 поддерживает также 80-битовое число с плавающей запятой (расширенная двойная точность).
5. *Операции.* Основными категориями операций являются пересылка данных, арифметические и логические, управление (рассмотрено дальше) и операции с плавающей запятой. Архитектура MIPS имеет простую и легко конвейеризуемую архитектуру системы команд, которая и является представителем RISC-архитектур, применявшихся в 2011 г. На рис. 1.5 представлена сводка MIPS ISA. Архитектура 80x86 имеет значительно более представительный набор операций (см. приложение K).
6. *Команды потока управления* — практически все ISA, включая и эти три, поддерживают условные передачи управления, безусловные передачи управления, вызовы процедур и возвраты. Все три используют адресацию относительно программного счетчика PC, при которой адрес перехода получается сложением содержимого поля адреса с содержимым PC. Существуют и небольшие отличия. В архитектуре MIPS команды условной передачи (BE, BNE и т. д.) проверяют содержимое регистров, в то время как в архитектурах 80x86 и ARM передачи управления проверяют разряды условного кода, установленные как побочный эффект арифметико-логических операций. При вызове процедуры в архитектурах ARM и MIPS адрес возврата помещается в регистр, в то время как вызов процедуры в архитектуре 80x86 (CALLF) помещает адрес возврата в стек в памяти.
7. Кодировка ISA. Существуют два основных варианта кодировки: *с фиксированной длиной* и *с переменной длиной*. Все команды в архитектурах ARM и MIPS имеют длину 32 бита, что упрощает их дешифрацию. На рис. 1.6 показаны форматы команд MIPS. Кодировка в архитектуре 80x86 имеет переменную длину в диапазоне от 1 до 18 байт. Команды с переменной длиной могут занимать меньше места, чем команды с фиксированной длиной, поэтому программа, скомпилированная для архитектуры 80x86, обычно меньше, чем та же программа, скомпилированная для архитектуры MIPS. Отметим, что упомянутые выше варианты влияют на то, как команды кодируются в двоичном представлении. Например, количество регистров и количество режимов адресации сильно

Тип команды/ код операции	Описание
Пересылки данных	*Пересылка данных между регистрами и памятью либо между целыми и FP или специальными регистрами; единственным режимом адресации памяти является режим с 16-битным смещением + содержимое GPR*
LB, LBU, SB	Загрузить байт, загрузить беззнаковый байт, записать байт (в/из целых регистров)
LH, LHU, SH	Загрузить полуслово, загрузить беззнаковое полуслово, записать полуслово (в/из целых регистров)
LW, LWU, SW	Загрузить слово, загрузить беззнаковое слово, записать слово (в/из целых регистров)
LD, SD	Загрузить двойное слово, записать двойное слово (в/из целых регистров)
L.S, L.D, S.S, S.D	Загрузить SP с плавающей запятой, загрузить DP с плавающей запятой, записать SP с плавающей запятой, записать DP с плавающей запятой
MFC0, MTC0	Копировать из/в GPR в/из специальный регистр
MOV.S, MOV.D	Копировать один SP- или DP FP-регистр в другой FP-регистр
MFC1, MTC1	Копировать 32 бита в/из FP-регистры из/в целых регистров
Арифметические/ логические	Операции над целыми или логическими данными в *GPR; арифметическое прерывание при переполнении со знаком*
DADD, DADDI, DADDU, DADDIU	Сложить, сложить непосредственное (все непосредственные операнды — 16-битовые); знаковое и беззнаковое
DSUB, DSUBU	Вычесть знаковое и беззнаковое
DMUL, DMULU, DDIV, DDIVU, MADD	Умножить и делить, знаковое и беззнаковое; умножить-сложить; все операции принимают и выдают 64-битовые значения
AND, ANDI	И, И непосредственное
OR, ORI, XOR, XORI	ИЛИ, ИЛИ непосредственное, исключающее ИЛИ, исключающее или непосредственное
LUI	Загрузить непосредственное в старшие разряды; загружает непосредственное в разряды регистра с 32 по 47, затем размножает знак
DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV	Сдвиги: непосредственный (DS__) и в переменной форме (DS__V); сдвиги — логический влево, логический вправо, арифметический вправо
SLT, SLTI, SLTU, SLTIU	Установить меньше, установить меньше, чем непосредственное, знаковые и беззнаковые

Рис. 1.5. Подмножество команд в MIPS64. SP — одинарная точность, DP — двойная точность. В приложении А система команд MIPS64 представлена более подробно. Для данных номер наиболее значимого бита — 0, наименее значимого — 63

Тип команды/ код операции	Описание
Управление	*Условные и безусловные передачи управления; относительно счетчика команд (СК) или с использованием регистра*
BEQZ, BNEZ	*Передача управления* по GPR, равным/неравным нулю; 16-битовое смещение по отношению к PC + 4
BEQ, BNE	*Передача управления* по равенству/неравенству двух GPR; 16-битовое смещение по отношению к PC + 4
BC1T, BC1F	Проверить бит сравнения в регистре FP-статуса и *передача управления* (в зависимости от его значения); 16-битовое смещение по отношению к PC + 4
MOVN, MOVZ	Скопировать GPR в другой GPR, если третий GPR отрицателен, равен нулю
J, JR	Безусловные *передачи управления*: 26-битовое смещение относительно PC + 4 (J) или значения регистра (JR)
JAL, JALR	*Передача управления* с возвратом: сохранить PC + 4 в R31, адрес перехода — относительно PC (JAL) или регистра (JALR)
TRAP	*Передача управления* на код операционной системы по адресу из вектора прерывания
ERET	Возврат в код пользователя из прерывания; восстановить режим пользователя
Плавающая запятая	Операции с плавающей запятой *в форматах DP и SP*
ADD.D, ADD.S, ADD.PS	Сложить числа DP, SP и пары чисел SP
SUB.D, SUB.S, SUB.PS	Вычесть числа DP, SP и пары чисел SP
MUL.D, MUL.S, MUL.PS	Перемножить DP, SP с плавающей запятой и пары чисел SP
MADD.D, MADD.S, MADD.PS	Умножить-сложить DP, SP с плавающей запятой и пары чисел SP
DIV.D, DIV.S, DIV.PS	Поделить DP, SP с плавающей запятой и пары чисел SP
CVT._._	Команды преобразования типов: CVT.x.y преобразует из типа x в тип y, где x и y являются L (64-битовыми целыми), W (32-битовыми целыми), D (DP) или S (SP). Оба операнда — регистры с плавающей запятой FPR
C.__.D, C.__.S	Сравнения чисел DP и SP: «__» = LT,GT,LE,GE,EQ,NE; устанавливает бит в регистре FP статуса

Рис. 1.5. Окончание

Рис. 1.6. Форматы команд архитектуры MIPS64. Все команды имеют длину 32 бита. Формат R используется для целых операций регистр-регистр, таких как DADDU, DSUBU и т.п. Формат I используется для пересылки данных, условных передач управления и команд с непосредственным значением, таких как LD, SD, BEQZ и DADDI. Формат J используется для безусловных передач управления, формат FR — для операций с плавающей запятой и формат FI — для условных передач управления с плавающей запятой

влияют на размер команд, поскольку поля адреса регистра и режима адресации могут появляться много раз в одной команде. (Архитектуры ARM и MIPS позже выпустили расширения, в которых вводились команды длиной 16 бит в целях уменьшения размера программ, названные Thumb или Thumb-2 и MIPS16 соответственно.)

Другие проблемы, стоящие перед компьютерным архитектором после разработки ISA, особенно важны сегодня, когда различия между системами команд невелики и существуют различные области приложений. Поэтому, начиная с последнего издания, основная часть материала по системе команд, за исключением данного краткого обзора, находится в приложениях (см. приложения A и K).

В этой книге мы используем подмножество системы команд MIPS64 в качестве примера ISA, так как MIPS64 преобладает в сетях и является наглядным примером упомянутых ранее RISC-архитектур, наиболее популярным примером которых является ARM. ARM-процессоры были в 6,1 млрд чипов, поставленных на рынок в 2010 г., что примерно в 20 раз больше чипов, поставленных с процессорами 80x86.

Истинная архитектура компьютеров: проектирование организации и аппаратуры, соответствующих целям и функциональным требованиям

Реализация компьютера имеет два компонента: структуру и аппаратуру. Термин «*организация*» включает высокоуровневые аспекты устройства компьютера,

такие как система памяти, подключение памяти, устройство внутреннего процессора или CPU, в котором реализованы арифметика, логика, передача управления и пересылка данных. Вместо термина «организация» используется также термин «микроархитектура». Например, AMD Opteron и Intel Core i7 — это два процессора с одинаковыми архитектурами системы команд, но с различными организациями. Оба процессора реализуют систему команд x86, но организации конвейера и кэша у них существенно различаются.

Переход к нескольким процессорам в микропроцессоре привел к тому, что термин «ядро» применяется также для обозначения процессора. Вместо выражения «многопроцессорный микропроцессор» употребляется термин «*многоядерный*» (*multicore*). Поскольку фактически все чипы имеют несколько процессоров, термин «центральный процессор» или CPU теряет популярность.

Термин «*аппаратура*» (*hardware*) относится к особенностям компьютера, включая детальное логическое устройство и технологию компоновки (конструктивного оформления) компьютера. Часто семейство компьютеров содержит компьютеры с идентичными архитектурами системы команд и организациями, но они различаются в детальной аппаратной реализации. Например, микропроцессоры Intel Core i7 (см. главу 3) и Intel Xeon 7560 (см. главу 5) почти идентичны, имеют разные частоты синхронизации и разные системы памяти, что делает микропроцессор Xeon 7560 более эффективным для серверных компьютеров.

В этой книге слово «*архитектура*» обозначает все три аспекта проектирования компьютеров — архитектуру системы команд, организацию, то есть микроархитектуру, и аппаратуру.

Компьютерные архитекторы должны проектировать компьютер в соответствии с функциональными требованиями, а также целями, определенными для цены, мощности, производительности и доступности. На рис. 1.7 показаны требования, которые следует рассматривать при разработке нового компьютера. Часто архитекторы также должны определить функциональные требования, которые могут быть главной задачей. Требования могут быть конкретными характеристиками, обусловленными рынком. Прикладное программное обеспечение часто управляет выбором конкретных функциональных требований, определяемых тем, как будет использоваться компьютер. Если существует большой объем программного обеспечения для некоторой архитектуры системы команд, то архитектор может решить, что новый компьютер должен реализовать существующую систему команд. Наличие широкого рынка для какого-либо конкретного класса приложений может побудить разработчиков включить требования, которые должны сделать компьютер конкурентоспособным на этом рынке. В последующих главах многие из этих требований и особенностей рассматриваются более глубоко.

Архитекторы должны также знать важные тенденции в технологии и использовании компьютеров, так как эти тенденции влияют не только на будущую стоимость, но и на долговечность архитектуры.

Функциональные требования	Требуемые или поддерживаемые типичные свойства
Область применения	*Требования к компьютеру*
Персональное мобильное устройство	Производительность в реальном времени для диапазона задач, включая интерактивную производительность для графики, видео и аудио; энергетическая эффективность (главы 2—5; приложение A)
Универсальный настольный компьютер	Сбалансированная производительность для диапазона задач, включая интерактивную производительность для графики, видео и аудио (главы 2—5; приложение A)
Серверы	Поддержка баз данных и обработки транзакций; улучшения надежности и доступности; поддержка масштабируемости (главы 2, 5; приложения A, D, F)
Кластеры / компьютеры WSC	Пропускная способность для многих независимых задач; коррекция ошибок в памяти; энергетическая пропорциональность (главы 2, 6; приложение F)
Встроенная вычислительная техника	Часто требуется специальная поддержка графики или видео (или другого расширения, зависящего от приложения); могут потребоваться ограничения мощности или управление мощностью; ограничения реального времени (главы 2, 3, 5; приложения A, E)
Уровень совместимости программного обеспечения	*Определяет количество существующего программного обеспечения для компьютера*
В языке программирования	Наиболее гибкий для разработчика; необходим новый компилятор (главы 3, 5; приложение A)
Объектный код или двоичная совместимость	Архитектура системы команд полностью определена — небольшая гибкость, но не требуются инвестиции в программное обеспечение или в перенос программ (приложение A)
Требования операционной системы	*Необходимые свойства для поддержки выбранной ОС (глава 2; приложение B)*
Размер адресного пространства	Очень важная особенность (глава 2); может ограничивать приложения
Управление памятью	Требуется для современной ОС; может быть страничным или сегментным (глава 2)
Защита	Требуется различным ОС и приложениям: выбор между страничной и сегментной организацией; виртуальные машины (глава 2)

Рис. 1.7. Перечень некоторых из наиболее важных требований, стоящих перед архитектором. В левой колонке описывается класс требований, а в правой приводятся конкретные примеры. Правая колонка также содержит ссылки на главы и приложения, в которых рассматриваются конкретные проблемы

Функциональные требования	Требуемые или поддерживаемые типичные свойства
Стандарты	*Рынок может потребовать определенных стандартов*
Плавающая запятая	Формат и арифметика: стандарт IEEE 754 (приложение J), специальная арифметика для графики и обработки сигналов
Интерфейсы ввода/вывода	Для устройств ввода/вывода: интерфейсы Serial ATA, Serial Attached SCSI, PCI Express (приложения D, F)
Операционные системы	UNIX, Windows, Linux, CISCO IOS
Сети	Поддержка, требуемая для различных сетей: Ethernet, Infiniband (приложение F)
Языки программирования	Языки (ANSI C, C++, Java, Fortran) влияют на систему команд (приложение A)

Рис. 1.7. Окончание

1.4. Тенденции в развитии технологий

Если предполагается, что архитектура системы команд будет успешной, она должна быть спроектирована так, чтобы пережить быстрые изменения в компьютерной технологии. Ведь успешная новая архитектура системы команд может жить несколько десятилетий, например, ядро центрального процессора IBM используется около 50 лет. Архитектор должен планировать изменения в технологии, которые могут увеличить срок службы успешного компьютера.

Чтобы планировать развитие своего компьютера, разработчик должен иметь представление о быстрых изменениях в технологии реализации. Пять технологий реализаций, которые меняются с колоссальной скоростью, являются крайне важными для современных реализаций.

- *Технология интегральных логических схем*. Плотность транзисторов возрастает примерно на 35 % в год, то есть приблизительно в 4 раза за четыре года. Рост размеров кристалла менее предсказуем, медленнее и колеблется от 10 до 20 % в год. Совместный эффект приводит к росту числа транзисторов на кристалл приблизительно на 40—50 % в год, то есть удваивается каждые 18—24 месяца. Эта тенденция широко известна как закон Мура. Как будет показано ниже, скорость устройства изменяется более медленно.
- *Полупроводниковая технология памяти DRAM*. Сейчас, когда большинство микросхем DRAM поставляется преимущественно в составе модулей DIMM (Dual In-line Memory Module, двухсторонний модуль памяти), стало труднее отследить объем микросхемы, так как производители DRAM обычно предлагают продукты разного объема одновременно, чтобы обеспечить нужный объем DIMM. Объем микросхем DRAM в последнее время увеличивался на 25—40 % в год, приблизительно удваиваясь каждые два-три года. На этой технологии основывается

оперативная память, она будет рассмотрена в главе 2. Обратите внимание, что, как показано на рис. 1.8, темп совершенствования памяти непрерывно снижался в течение всего времени выхода изданий этой книги. Есть даже опасение, не остановится ли этот рост к середине текущего десятилетия из-за возрастающей трудности эффективного производства еще меньших ячеек DRAM [Ким (Kim) 2005]. В главе 2 упоминаются несколько других технологий, которые могут заменить технологию DRAM, если она столкнется с пределом по объему.

Издание CA:AQA	Год	Скорость роста DRAM, %/год	Характеристика влияния на объем DRAM
1	1990	60	Рост в 4 раза каждые три года
2	1996	60	Рост в 4 раза каждые три года
3	2003	40—60	Рост в 4 раза каждые три-четыре года
4	2007	25—40	Удвоение каждые два года
5	2011	25—40	Удвоение каждые два-три года

Рис. 1.8. Изменение темпа роста объема DRAM с течением времени. В первых двух изданиях даже называли этот темп DRAM Growth Rule of Thumb (практическое правило роста объема DRAM), поскольку именно так обстояло дело начиная с 1977 г. с DRAM объемом 16 Кбит по 1996 г. с DRAM объемом 64 Мбит. В настоящее время из-за проблем в производстве трехмерных ячеек DRAM стоит вопрос — можно ли вообще улучшить объем DRAM в течение пяти-семи лет [Kim 2005]

- *Полупроводниковая флеш-память* (электрически стираемая перепрограммируемая постоянная память). Эта энергонезависимая полупроводниковая память является стандартным запоминающим устройством в PMD, и ее быстро возрастающая популярность стимулирует быстрый рост объема. В последнее время объем кристалла флэш-памяти увеличивался примерно на 50—60 % в год, удваиваясь приблизительно каждые два года. В 2011 г. флэш-память стоила в расчете на бит в 15—20 раз дешевле, чем DRAM. Флэш-память описывается в главе 2.
- *Технология магнитных дисков*. До 1990 г. плотность увеличивалась примерно на 30 % в год, удваиваясь за три года. После этого ежегодное увеличение достигло 60 % и выросло до 100 % в 1996 г. С 2004 г эта скорость упала примерно до 40 % в год, иначе говоря, плотность удваивалась каждые три года. Диски в 15—20 раз дешевле в пересчете на бит, чем флэш-память. При замедлившемся темпе роста DRAM диски в настоящее время в 300—500 раз дешевле, чем DRAM, в пересчете на бит. Эта технология является основной для внешней памяти серверов и компьютеров WSC, и подробное обсуждение тенденций в этой области проводится в приложении D.
- *Сетевые технологии*. Производительность сетей зависит как от производительности коммутаторов, так и от передающей системы. Тенденции развития сетевых технологиях обсуждаются в приложении F.

Эти быстро изменяющиеся технологии формируют устройство компьютера так, что с учетом улучшений в скорости и технологии компьютер может иметь срок службы от трех до пяти лет. Ключевые технологии, такие как DRAM, Flash и дисков, меняются настолько сильно, что разработчик должен планировать эти изменения. В самом деле, разработчики часто проектируют в расчете на следующую технологию, зная, что к моменту массового выхода данного продукта на рынок эта следующая технология может оказаться наиболее рентабельной или иметь преимущества по производительности. Обычно стоимость понижается примерно с той же скоростью, с какой повышается плотность.

Несмотря на то, что технология улучшается непрерывно, влияние этих усовершенствований может проявляться в форме отдельных скачков, возникающих по достижении некоторого порога, который открывает дорогу какой-либо новой возможности. Например, когда в начале 1980-х гг. технология MOS (МОП-технология) достигла точки, в которой от 25 000 до 50 000 транзисторов можно было поместить в одну микросхему, стало возможным построение однокристального 32-разрядного микропроцессора. К концу 1980-х гг. в микросхеме смогли поместиться и кэши первого уровня. Размещение процессора и кэша внутри одного чипа сделало возможным впечатляющее улучшение соотношений стоимость — производительность и энергопотребление — производительность. Такая конструкция была просто неосуществима, пока технология не достигла определенной точки. С появлением многоядерных микропроцессоров и увеличением числа ядер в каждом новом поколении, даже в серверах, все больше прибегают к размещению всех процессоров в одной микросхеме. Такие технологические пороги нередки и оказывают значительное влияние на широкий спектр проектных решений.

Тенденции изменения производительности: пропускная способность против задержки

Как будет показано в разделе 1.8, *пропускная способность* (*bandwidth или throughput*) — это общее количество работы, проделанной в заданное время, например количество мегабайт в секунду при обмене с диском. В отличие от этого *задержка* (*latency*) или *время отклика* (*response time*) — это время между началом и завершением события, например количество миллисекунд при обращении к диску. На рис. 1.9 приведен график относительного улучшения пропускной способности и задержки микропроцессоров, памяти, сетей и дисков в соответствии с технологическими этапами. На рис. 1.10 примеры и технологические этапы показаны более подробно.

Производительность — главная отличительная характеристика для микропроцессоров и сетей, поэтому для них виден наибольший прирост: пропускная способность увеличилась в 10 000—25 000 раз, задержка — в 30—80 раз. Для памяти и дисков объем, как правило, важнее производительности. Поэтому больше всего увеличился объем, тем не менее улучшение пропускной способности в 300—1200 раз все равно много больше улучшения задержки в 6—8 раз.

Ясно, что пропускная способность при этих технологиях обогнала задержку и, вероятно, так будет и дальше. Простое практическое правило (правило, основанное на практическом опыте) заключается в том, что пропускная способность растет,

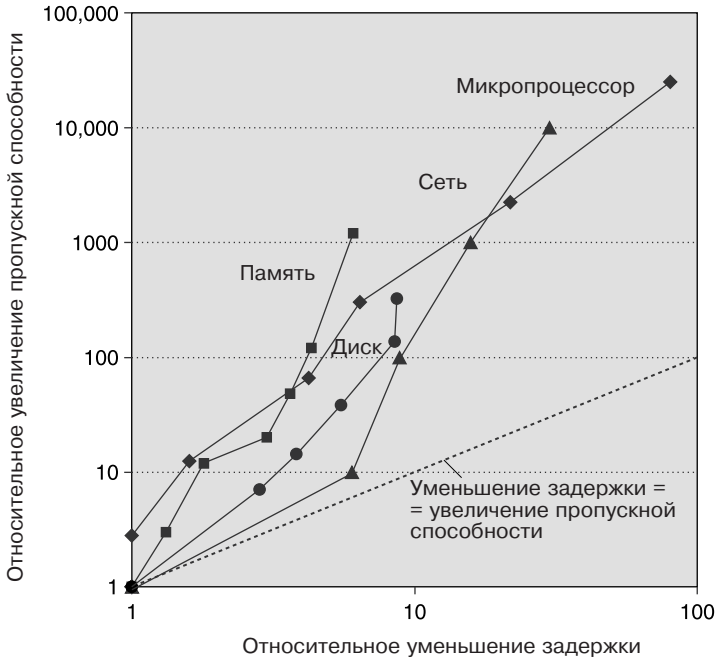

Рис. 1.9. График поэтапного изменения пропускной способности и задержки из рис. 1.10, заданных относительно показателей первого этапа (двойной логарифмический масштаб). Отметим, что задержка улучшилась в 6—80 раз, а пропускная способность — примерно в 300—25 000 раз. Откорректированные данные из Patterson [2004]

по крайней мере, квадратично по отношению к улучшению задержки. Разработчики компьютеров должны учитывать эту закономерность.

Масштабирование производительности транзисторов и проводников

Процессы производства интегральных схем характеризуются *минимальным топологическим размером* (*feature size*), который является минимальным размером транзистора или проводника по любой из осей *x* или *y*. Минимальный топологический размер уменьшился с 10 мк в 1971 г. до 0,032 мк в 2011 г.; фактически изменились единицы измерения, поэтому продукция 2011 г. обозначается «32 нм» и на подходе кристаллы с 22 нм. Так как количество транзисторов на 1 мм^2 кристалла определяется размером поверхности одного транзистора, плотность транзисторов возрастает квадратично при линейном уменьшении минимального топологического размера.

Однако рост производительности транзисторов определяется более сложной зависимостью. С сокращением минимального топологического размера площадь устройства сокращается квадратично, так как размеры устройства сокращаются в горизонтальном и вертикальном направлениях. Сокращение вертикального размера требует уменьшения рабочего напряжения для поддержания нормальной

Микропроцессор	16-битовые адрес и шина, микрокодирование	32-битовые адрес и шина, микрокодирование	5-стадийный конвейер, кэши команд и данных на кристалле, FPU	Суперскаляр 2 команды за такт, 64-битовая шина	Внеочередное выполнение команд, суперскаляр 3 команды за такт	Внеочередное выполнение команд, суперконвейер, кэш второго уровня на кристалле	Многоядерный, внеочередное выполнение команд, 4 команды за такт, кэш третьего уровня на кристалле, турборежим
Модель	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Год	1982	1985	1989	1993	1997	2001	2010
Размер кристалла (мм^2)	47	43	81	90	308	217	240
Количество транзисторов	134000	275000	1200000	3100000	5500000	42000000	1170000000
Процессоров на чип	1	1	1	1	1	1	4
Количество выводов	68	132	168	273	387	423	1366
Задержка (тактов)	6	5	5	5	10	22	14
Ширина шины (биты)	16	32	32	64	64	64	196
Тактовая частота (МГц)	12,5	16	25	66	200	1500	3333
Пропускная способность (MIPS)	2	6	25	132	600	4500	50.000
Задержка (нс)	320	313	200	76	50	15	4

Рис. 1.10. Изменение производительности по этапам в течение 25—40 лет для микропроцессоров, памяти, сетей и дисков. Для микропроцессоров этапы представлены несколькими поколениями процессоров IA-32, прошедших путь от микропрограммируемого микропроцессора 80826 с 16-разрядной шиной до многоядерного суперконвейерного микропроцессора Core i7 с 64-разрядной шиной и внеочередным (out-of-order) выполнением команд. Этапы для модуля памяти отражают переход от одностороннего модуля DRAM с 16-разрядной шиной к синхронному модулю DRAM с 64-разрядной шиной с удвоенной частотой передачи (DDR) версии 3. Ethernet продвинулся от 10 Мбит/с до 100 Гбит/с. Этапы для диска основаны на скорости вращения, улучшаясь от 3600 до 15000 об/мин. В каждом примере представлена максимальная пропускная способность, а задержка — это время простой операции, если нет других указаний. Скорректированные данные из Patterson [2004]

Микропроцессор	16-битовые адрес и шина, микро-кодирование	32-битовые адрес и шина, микро-кодирование	5-стадийный конвейер, кэши команд и данных на кристалле, FPU	Суперскаляр 2 команды за такт, 64-битовая шина	Внеочередное выполнение команд, суперскаляр 3 команды за такт	Внеочередное выполнение команд, супер-конвейер, кэш второго уровня на кристалле	Многоядерный, внеочередное выполнение команд, 4 команды за такт, кэш третьего уров-ня на кристалле, турборежим
Модуль памяти	DRAM	Страничная DRAM	Быстрая страничная DRAM	Быстрая страничная DRAM	Синхронная DRAM (SDRAM)	SDRAM удво-енной скорости (DDR SDRAM)	DDR3 SDRAM
Шина модуля (биты)	16	16	32	64	64	64	64
Год	1980	1983	1986	1993	1997	2000	2010
Мбит/DRAM чип	0,06	0,25	1	16	64	256	2048
Размер кристалла (мм^2)	35	45	70	130	170	204	50
Количество выводов на чип DRAM	16	16	18	20	54	66	134
Пропускная способность (Мбайт/с)	13	40	160	267	640	1600	16.000
Задержка (нс)	225	170	125	75	62	52	37
Локальная сеть	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet	100 Gigabit Ethernet		
Стандарт IEEE	802.3	803.3u	802.3ab	802.3ac	802.ba		
Год	1978	1995	1999	2003	2010		
Пропускная способность (Мбит/с)	10	100	1000	10 000	100 000		
Задержка (мкс)	3000	500	340	190	100		

Рис. 1.10. Продолжение

Микропроцессор	16-битовые адрес и шина, микро-кодирование	32-битовые адрес и шина, микро-кодирование	5-стадийный конвейер, кэши команд и данных на кристалле, FPU	Суперскаляр 2 команды за такт, 64-битовая шина	Внеочередное выполнение команд, суперскаляр 3 команды за такт	Внеочередное выполнение команд, супер-конвейер, кэш второго уровня на кристалле	Многоядерный, внеочередное выполнение команд, 4 команды за такт, кэш третьего уров-ня на кристалле, турборежим
Жесткий диск	3600 об/мин	5400 об/мин	7200 об/мин	10 000 об/мин	15 000 об/мин	15 000 об/мин	
Модель	CDC WrenI 94145-36	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	Seagate ST3600057	
Год	1983	1990	1994	1998	2003	2010	
Объем (Гбайт)	0,03	1,4	4,3	9,1	73,4	600	
Форм-фактор	5,25 дюйма	5,25 дюйма	3,5 дюйма	3,5 дюйма	3,5 дюйма	3,5 дюйма	
Диаметр носителя	5,25 дюйма	5,25 дюйма	3,5 дюйма	3,0 дюйма	2,5 дюйма	2,5 дюйма	
Интерфейс	ST-412	SCSI	SCSI	SCSI	SCSI	SAS	
Пропускная способность (Мбайт/с)	0,6	4	9	24	86	204	
Задержка (мс)	48,3	17,1	12,7	8,8	5,7	3,6	

Рис. 1.10. Окончание

работы и надежности транзисторов. Эта комбинация масштабирующих факторов приводит к сложной взаимозависимости между производительностью транзистора и минимальным топологическим размером. В первом приближении производительность транзистора растет линейно с уменьшением минимального топологического размера.

Тот факт, что количество транзисторов растет квадратично с линейным ростом их производительности, одновременно является и проблемой, и возможностью, для которых и появились компьютерные архитекторы! В начале эпохи микропроцессоров повышенный темп увеличения плотности был использован для быстрого перехода от 4-разрядных к 8-, 16-, 32- и 64-разрядным микропроцессорам. В последнее время увеличение плотности способствовало размещению нескольких процессоров в кристалле, появлению более широких устройств SIMD и многим нововведениям в спекулятивном выполнении и кэшах, о которых можно прочитать в главах 2—5.

Несмотря на то, что производительность транзисторов обычно увеличивается с уменьшением минимального топологического размера, для проводников в интегральной схеме это не так. В частности, задержка сигнала в проводнике растет пропорционально произведению его сопротивления на его емкость. Конечно, при сокращении минимального топологического размера проводники становятся короче, но сопротивление и емкость на единицу длины ухудшаются. Эта зависимость сложна, поскольку как сопротивление, так и емкость зависят от особенностей технологического процесса, геометрии проводника, нагрузки на проводник и даже от соседства с другими структурами. Порой происходят усовершенствования технологического процесса, такие как введение меди, которое дает одноразовое уменьшение задержки в проводнике.

Тем не менее, как правило, задержка проводника масштабируется слабее по сравнению с производительностью транзистора, что создает дополнительные трудности для разработчика. В последние несколько лет вдобавок к ограничению по рассеиванию мощности задержка проводника стала основным конструктивным ограничением для больших интегральных схем, и часто она является более критичной, чем задержка переключения транзистора. Все большие и большие доли такта синхронизации расходуются на задержку распространения сигналов по проводникам, но роль, которую сейчас играет мощность, все же значительнее, чем влияние этой задержки.

1.5. Тенденции потребления мощности и энергии интегральных схем

В настоящее время мощность является самой большой проблемой для разработчика компьютера почти любого класса. Во-первых, питание должно быть доставлено внутрь микросхемы и распределено по ней, и современные микропроцессоры используют сотни выводов и несколько слоев внутренних соединений только для питания и земли. Во-вторых, мощность рассеивается в виде тепла и его нужно отводить.

Мощность и энергопотребление: системная точка зрения

Как архитектор системы или пользователь должен размышлять о производительности, мощности и энергопотреблении? С точки зрения разработчика системы, существуют три главные проблемы.

Во-первых: какая максимальная мощность вообще требуется процессору? Удовлетворение этого требования может быть важным для обеспечения правильной работы. Например, если процессор пытается потреблять больше мощности, чем обеспечивает система питания (потребляя больше тока, чем система может обеспечить), результатом обычно является падение напряжения, которое может привести к неисправности устройства. В современных процессорах потребление мощности с большими пиковыми токами может изменяться в широких пределах, поэтому они используют методы изменения напряжения, которые позволяют замедлять процессор и регулировать напряжение в более широких границах. Очевидно, что при этом снижается производительность.

Во-вторых: каково потребление энергии в установившемся режиме? Эта характеристика часто называется *величиной отвода тепловой мощности* (*TDP — Thermal Design Power*), так как она определяет требования к охлаждению. TDP не является ни пиковой мощностью, которая часто бывает в 1,5 раза больше, ни фактической средней мощностью, которая будет потребляться в течение данного вычисления и, вероятно, будет еще меньше. Типичное питание для системы обычно делается превышающим TDP, а система охлаждения обычно проектируется так, чтобы соответствовать или превосходить TDP. Неспособность обеспечить адекватное охлаждение позволит температуре перехода в процессоре превысить свое максимальное значение, что приведет к сбою устройства и, возможно, его неустранимому повреждению. Современные процессоры имеют две конструктивные особенности, способствующие регулированию теплового режима, так как максимальная мощность (и, следовательно, рост тепловыделения и температуры) может превосходить долгосрочное среднее значение, определенное TDP. Первая состоит в том, что по мере приближения тепловой температуры к температурному пределу перехода уменьшается тактовая частота интегральной схемы, а следовательно, и мощность. Если же это не приносит успеха, то активируется вторая защита от тепловой перегрузки — снижается напряжение питания микросхемы.

Третьей проблемой, которую разработчики и пользователи должны иметь в виду, является энергопотребление и энергетическая эффективность. Вспомним, что мощность — это просто энергия в единицу времени: 1 Вт = 1 Дж/с. Какая метрика является правильной для сравнения процессоров: энергопотребление или мощность? Вообще говоря, энергопотребление всегда лучше потому, что оно привязано к конкретной задаче и ко времени, требуемому для ее выполнения. В частности, энергия, необходимая для выполнения некой работы, равна средней мощности, умноженной на время выполнения работы.

Таким образом, если нужно знать, который из двух процессоров более эффективен для данной задачи, следует сравнить энергопотребление (не мощность) при выполнении задачи. Например, процессор A может иметь на 20 % более высокое потребление мощности, чем процессор B, но если A выполняет задачу только за

70 % времени, необходимого B, его энергопотребление составит 1,2 × 0,7 = 0,84, что, конечно, лучше.

Могут возразить, что в большом сервере или «облаке»[1] достаточно рассматривать только среднюю мощность, так как рабочая нагрузка часто считается бесконечной во времени, но это заблуждение. Если бы наше «облако» состояло из процессоров B, а не процессоров A, то оно выполнило бы меньше работы, потребив то же самое количество энергии. Использование энергопотребления для сравнения этих альтернатив позволяет избежать такого заблуждения. Всякий раз, когда у нас имеется фиксированная рабочая нагрузка, будь то «облако» размером с компьютер WSC или смартфон, сравнение энергопотребления будет правильным способом сравнения процессорных альтернатив, поскольку как счет за электричество, потребленное «облаком», так и время жизни аккумулятора смартфона определяются потребленной энергией.

Когда потребляемая мощность является полезной мерой? В основном ее законно использовать в качестве ограничения, например мощность микросхемы может быть ограничена величиной 100 Вт. Ее можно использовать как меру, если рабочая нагрузка фиксированна, но тогда она становится лишь разновидностью истинной характеристики — энергопотребления на задачу.

Энергопотребление и мощность в микропроцессоре

Для КМОП-кристаллов основное потребление энергии традиционно относится к переключению транзисторов, так называемому *динамическому энергопотреблению* (*dynamic energy*). Энергия, приходящаяся на один транзистор, пропорциональна произведению емкостной нагрузки транзистора на квадрат напряжения:

$$\text{Энергопотребление}_{\text{динамическое}} \sim \text{Емкостная нагрузка} \times \text{Напряжение}^2.$$

Это уравнение энергии импульса при логических переключениях $0 \rightarrow 1 \rightarrow 0$ или $1 \rightarrow 0 \rightarrow 1$. Отсюда энергия одного переключения ($0 \rightarrow 1$ или $1 \rightarrow 0$):

$$\text{Энергопотребление}_{\text{динамическое}} \sim 1/2 \times \text{Емкостная нагрузка} \times \text{Напряжение}^2.$$

Мощность, приходящаяся на транзистор, представляет собой всего лишь произведение энергии переключения на частоту переключений:

$$\text{Мощность}_{\text{динамическая}} \sim 1/2 \times \text{Емкостная нагрузка} \times \\ \times \text{Напряжение}^2 \times \text{Частота переключения}.$$

Для фиксированной задачи снижение тактовой частоты уменьшает мощность, но не энергопотребление.

Ясно, что динамические мощность и энергопотребление существенно снижаются при уменьшении напряжения, поэтому за 20 лет напряжения снизились с 5 В до величины, немного меньшей 1 В. Емкостная нагрузка является функцией от количества транзисторов, соединенных с нагрузкой, и технологии, которая определяет емкость проводников и транзисторов.

[1] «Облако» — обозначение крупномасштабного компьютера, предназначенного для решения задач удаленных пользователей (см. раздел 6.5). — *Прим. ред.*

Пример В наши дни некоторые микропроцессоры спроектированы так, что напряжение их питания можно регулировать, при этом 15%-е уменьшение напряжения может привести к 15%-му уменьшению частоты. Как это отразится на динамическом энергопотреблении и динамической мощности?

Ответ Поскольку емкость не изменяется, энергопотребление определяется отношением напряжений:

$$\frac{\text{Энергопотребление}_{\text{новое}}}{\text{Энергопотребление}_{\text{старое}}} = \frac{(\text{Напряжение} \times 0{,}85)^2}{\text{Напряжение}^2} = 0{,}85^2 = 0{,}72$$

и, следовательно, энергопотребление снижается примерно до 72 % по сравнению с исходным.

Для мощности добавляем отношение частот:

$$\frac{\text{Мощность}_{\text{новая}}}{\text{Мощность}_{\text{старая}}} = 0{,}72 \times \frac{(\text{Частота переключения} \times 0{,}85)}{\text{Частота переключения}} = 0{,}61,$$

сокращая мощность примерно до 61 % от исходной.

По мере того как мы переходим от одного технологического процесса к следующему, увеличение количества переключающихся транзисторов и частота их переключения перевешивают уменьшение емкости нагрузки и напряжения, что приводит к общему росту мощности и энергопотребления. Первые микропроцессоры потребляли менее 1 Вт, а первые 32-разрядные микропроцессоры (такие как микропроцессор Intel 80386) использовали около 2 Вт, в то время как микропроцессор Intel Core i7 с его тактовой частотой 3,3 ГГц потребляет 130 Вт. Поскольку это тепло необходимо рассеивать с кристалла со стороной около 1,5 см, мы достигли предела того, что может быть охлаждено воздухом.

Учитывая приведенное выше уравнение, можно было бы ожидать, что рост тактовой частоты уменьшится, если мы не сможем уменьшить напряжение или увеличить мощность кристалла. На рис. 1.11 показано, что так оно и было с 2003 г., даже для микропроцессоров, приведенных на рис. 1.1, которые были самыми быстрыми в каждом году. Отметим, что этот период горизонтального участка на кривой тактовых частот соответствует периоду медленного увеличения производительности на рис. 1.1.

Распределение мощности, отвод тепла и предотвращение появления точек перегрева стали проблемами все возрастающей сложности. Сейчас мощность — основное ограничение в использовании транзисторов; в прошлом самым слабым звеном был кремний. В связи с этим в современных микропроцессорах предлагается много способов, чтобы попытаться увеличить энергетическую эффективность, несмотря на «горизонтальные» тактовые частоты и неизменяющееся напряжение питания.

1. *Лучше ничего не делать (Do nothing well)*. В настоящее время в большинстве микропроцессоров отключаются тактовые сигналы в неактивных модулях для экономии энергии и динамической мощности. Например, если не выполняются

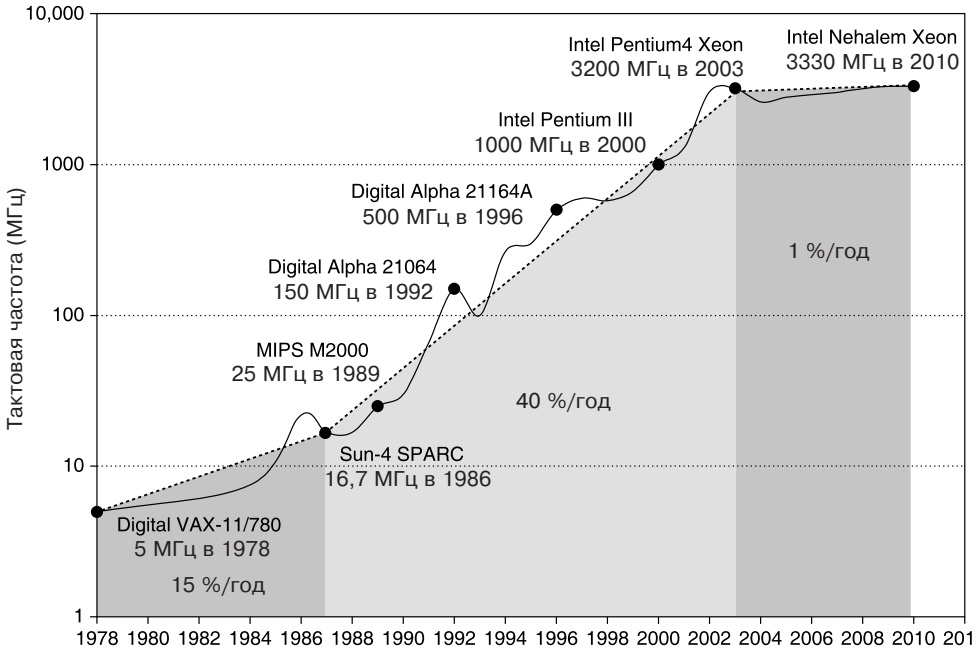

Рис. 1.11. Рост тактовой частоты микропроцессоров, представленных на рис. 1.1. Между 1978 и 1986 гг. тактовая частота увеличивалась менее чем на 15 % в год, тогда как производительность увеличивалась на 25 % в год. Во время периода «возрождения» между 1986 и 2003 гг. производительность увеличивалась на 52 % в год, тактовые частоты «подскакивали» почти на 40 % ежегодно. С тех пор при росте менее 1 % в год кривая была почти горизонтальной, в то время как производительность одного процессора росла менее чем на 22 % в год

команды с плавающей запятой, то отключаются тактовые сигналы в устройстве с плавающей запятой. Если некоторые ядра простаивают, их тактовые сигналы отключаются.

2. *Динамическое масштабирование напряжения и частоты (DVFS — Dynamic Voltage-Frequency Scaling)*. Второй метод прямо следует из приведенных выше формул. У персональных мобильных устройств, ноутбуков и даже у серверов случаются периоды низкой активности, когда нет нужды работать на максимальных тактовой частоте и напряжениях. Современные микропроцессоры обычно предлагают несколько тактовых частот и напряжений для работы, при которой используются меньшие мощность и энергопотребление. На рис. 1.12 показана потенциальная экономия мощности, получаемая на сервере от DVFS, вследствие сокращения рабочей нагрузки при трех различных тактовых частотах: 2,4; 1,8 и 1 ГГц. Общая экономия мощности для сервера составляет примерно от 10 до 15 % на каждом из этих двух шагов.
3. *Проектирование для типового случая*. Поскольку устройства PMD и ноутбуки часто простаивают, для оперативной и внешней памяти предлагаются режимы пониженной мощности, чтобы сэкономить энергопотребление. Например, модули

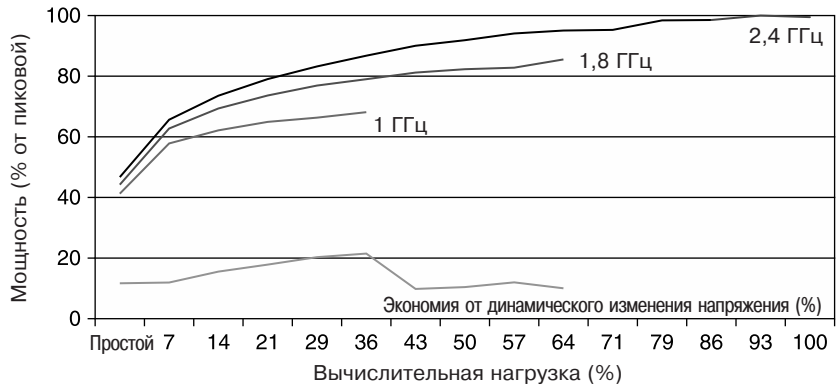

Рис. 1.12. Экономия энергопотребления для сервера с микропроцессором AMD Opteron, памятью DRAM емкостью 8 Гбайт и диском ATA. На частоте 1,8 ГГц сервер может обработать только до двух третей рабочей нагрузки, не вызывая нарушений уровня обслуживания; а на частоте 1,0 ГГц сервер может безопасно обработать только одну треть рабочей нагрузки (рис. 5.11 у Barroso and Hölzle [2009])

DRAM имеют серию режимов снижения мощности для продления срока службы батареи в PMD и ноутбуках; предлагались также диски с замедленным вращением при простоях, чтобы сэкономить мощность. К сожалению, в этих режимах к памяти или дискам обращаться нельзя, поэтому необходимо вернуться в режим полной активности для чтения или записи, независимо от того, насколько редки такие обращения. Как упоминалось выше, микропроцессоры для персональных компьютеров были разработаны, в отличие от этого, для более типичного случая интенсивной работы при высоких рабочих температурах, используя датчики температуры в микросхеме для определения момента, когда активность должна быть автоматически уменьшена во избежание перегрева. Такое «аварийное замедление» дает возможность производителям проектировать, ориентируясь на более типичный случай, и применять этот механизм безопасности, если кто-то действительно запускает программы, потребляющие большую мощность, чем типовая.

4. *Разгон (Overclocking).* В 2008 г. фирма Intel начала предлагать *режим Turbo*, при котором микросхема решает, что она может безопасно работать короткое время на повышенной частоте, возможно, на нескольких ядрах, пока температура не начнет подниматься. Например, Core i7 с тактовой частотой 3,3 ГГц в течение коротких интервалов может работать на частоте 3,6 ГГц. Действительно, все наиболее производительные микропроцессоры, упомянутые на рис. 1.1, каждый год начиная с 2008 г. предлагают временный разгон примерно на 10 % сверх номинальной тактовой частоты. Для программы с одним потоком эти микропроцессоры могут отключить все ядра, кроме одного, и работать даже на еще большей тактовой частоте. Отметим, что с того времени, когда операционная система может отключать режим Turbo, отсутствуют сообщения о моменте его включения, поэтому программисты могут удивиться, обнаружив, что производительность их программ изменяется из-за комнатной температуры!

Хотя динамическую мощность традиционно считают главным источником мощности рассеивания в КМОП, возрастает важность статической мощности из-за тока утечки, который существует, даже когда транзистор выключен:

$$\text{Мощность}_{\text{статическая}} \approx \text{Ток}_{\text{статический}} \times \text{Напряжение}.$$

То есть статическая мощность пропорциональна количеству устройств.

Таким образом, с увеличением количества транзисторов растет мощность, даже если транзисторы простаивают, при этом ток утечки возрастает в процессорах с меньшими размерами транзисторов. По этой причине в очень маломощных системах даже выключают питание (*power gating*) неактивных модулей, чтобы управлять потерями из-за утечки. В 2011 г. уровнем для тока утечки было 25 % от общего потребления энергии, причем утечка в высокопроизводительных системах иногда значительно превосходила этот уровень. Для таких кристаллов утечка может достигать 50 % частично из-за больших кэшей на базе статической памяти SRAM, которые расходуют мощность для хранения данных. (Буква S в аббревиатуре SRAM означает «static» — статическая.) Единственная надежда остановить утечку — отключить питание части устройств кристалла.

И наконец, поскольку процессор потребляет лишь часть всей энергии системы, имеет смысл использовать более быстрый и менее энергоэффективный процессор, чтобы дать возможность остальной части системы перейти в спящий режим. Такая стратегия известна как *«гонка к остановке»* (*race-to-halt*).

Из-за важности мощности и энергопотребления повысилось внимание к эффективности инноваций, поэтому сейчас главной оценкой стало количество задач на джоуль или производительность на ватт, а не производительность на квадратный миллиметр кремния. Эта новая характеристика влияет на подход к параллелизму, что будет рассмотрено в главах 4 и 5.

1.6. Тенденции изменения стоимости

Несмотря на то что в некоторых компьютерных проектах — особенно это касается суперкомпьютеров — фактор стоимости обычно бывает менее важным, растет значимость проектов, для которых стоимость важна. И действительно, в течение последних 30 лет использование технологических улучшений для снижения стоимости и увеличения производительности было главной темой в компьютерной индустрии.

В учебниках стоимостная составляющая общего отношения «стоимость — производительность» часто игнорируется, потому что стоимость со временем изменяется, оставляя содержание книг привязанным к определенным датам, а также потому, что эти вопросы трудноуловимые и различаются в разных сегментах промышленности. В то же время понимание стоимости и ее факторов является существенным для компьютерных архитекторов при принятии правильных продуманных решений, включать или нет новое свойство в проекты, где стоимость существенна. (Представьте себе архитекторов, проектирующих небоскребы и не имеющих никакой информации о стоимости стальных балок и бетона!)

В этом разделе обсуждаются основные факторы, влияющие на стоимость компьютера, и то, как эти факторы меняются с течением времени.

Влияние времени, объема выпуска и превращения в товар широкого потребления

Стоимость изготовленного компонента компьютера со временем уменьшается, даже без больших усовершенствований в базовой технологии реализации. В основе снижения стоимости лежит принцип *кривой освоения производства* (*learning curve*): со временем стоимость производства снижается. Сама кривая освоения производства лучше всего представляется изменением *выхода годной продукции* (*yield*), то есть процентом изготовленных устройств, выдержавших процедуру тестирования. Разработка, у которой выход годных в два раза больше, будет иметь половинную стоимость независимо от того, будь то микросхема, плата или система.

Понимание того, как кривая освоения производства улучшает выход годной продукции, является решающим для планирования стоимости в течение всего срока службы продукта. Одним из примеров является цена мегабайта DRAM, которая снижалась в течение долгого времени. Поскольку микросхемам DRAM свойственна тесная взаимосвязь цены и стоимости, за исключением периодов, когда наблюдается нехватка или избыток, цена и стоимость DRAM изменяются согласованно.

Цены на микропроцессоры тоже снижаются (с течением времени, но, поскольку они менее стандартизованы, чем микросхемы DRAM, связь между ценой и стоимостью у них более сложная). В периоды значительной конкуренции цена близко подходит к стоимости, хотя производители микропроцессоров, вероятно, редко продают себе в убыток.

Вторым ключевым фактором при определении стоимости является объем выпуска. Увеличение объемов влияет на стоимость несколькими путями. Во-первых, оно уменьшает время, необходимое для того, чтобы спуститься вниз по кривой освоения производства, которая отчасти пропорциональна количеству изготовленных систем (или микросхем). Во-вторых, рост объема выпуска снижает стоимость, так как увеличивает объем закупок и эффективность производства. На основании практического опыта некоторые разработчики оценивают, что стоимость снижается примерно на 10 % при каждом удвоении объема. Более того, объем уменьшает величину стоимости разработки, которая должна амортизироваться каждым компьютером, таким образом позволяя сблизить стоимость и цену продажи.

Товары широкого потребления — это продукты, которые продаются многими поставщиками в больших объемах и, в сущности, идентичны. Практически все продукты, продаваемые в магазинах, являются товарами широкого потребления, такими как стандартные DRAM, флэш-память, диски, мониторы и клавиатуры. За последние 25 лет большая часть индустрии персональных компьютеров превратилась в бизнес товаров широкого потребления, сосредоточенный на изготовлении настольных компьютеров и ноутбуков, работающих под управлением операционной системы Microsoft Windows.

Поскольку многие фирмы поставляют практически одинаковые продукты, конкуренция на данном рынке весьма высока. Разумеется, эта конкуренция уменьшает разрыв между стоимостью и продажной ценой, но она еще уменьшает и стоимость. Это происходит потому, что товарному рынку свойственны и объем, и ясное определение реализуемого на нем продукта, что позволяет многим поставщикам соревноваться в производстве компонентов для товаров широкого потребления. В результате общая стоимость продукции снижается из-за конкуренции

среди поставщиков компонентов и эффективности объема производства, которой поставщики могут достичь. Такая конкуренция привела к тому, что в секторе младших моделей компьютеров удалось достичь лучшего соотношения цена — производительность, и был продемонстрирован больший рост, чем в других секторах, хотя и с очень ограниченными прибылями (что вообще характерно для любого бизнеса в области товаров широкого потребления).

Стоимость интегральной схемы

Зачем книге по компьютерной архитектуре раздел, посвященный стоимости интегральных схем? На рынке компьютеров со все более растущей конкуренцией, где стандартные компоненты — диски, флэш-память, DRAM и т.п. — становятся значительной частью общей стоимости любой системы, стоимость интегральных схем становится большей частью общей стоимости, которая варьируется от компьютера к компьютеру, особенно в высокообъемном и чувствительном к стоимости сегменте этого рынка. Действительно, с нарастающей тенденцией применения полных *систем на кристалле* (*SOC — Systems On a Chip*) в персональных мобильных устройствах стоимость интегральной схемы становится большей частью стоимости PMD. Следовательно, разработчики компьютеров, чтобы разбираться в стоимости современных компьютеров, должны разбираться в стоимости микросхем.

Хотя стоимости интегральных схем упали экспоненциально, базовый процесс кремниевого производства не изменился. Пластину, как и раньше, тестируют и разрезают на кристаллы, которые потом помещают в корпуса (см. рис. 1.13—1.15). Таким образом, стоимость корпусированной интегральной схемы равна

$$\text{Стоимость интегральной схемы} = \frac{\text{СК} + \text{СТ} + \text{СТК}}{\text{Выход годных после окончательного тестирования}},$$

где *СК* — стоимость кристалла; *СТ* — стоимость тестирования; *СТК* — стоимость тестирования и корпусирования.

В данном разделе основное внимание уделяется стоимости кристаллов, но в конце мы кратко рассмотрим наиболее важные вопросы тестирования и корпусирования.

Чтобы предсказать количество годных кристаллов на пластине, требуется сначала узнать, сколько кристаллов на ней размещается, а затем — как предсказать процент тех из них, которые будут работать. Исходя из этого стоимость предсказать просто:

$$\text{Стоимость кристалла} = \frac{\text{Стоимость пластины}}{\text{Количество кристаллов на пластине} \times \text{Выход годных кристаллов}}.$$

Наиболее интересным свойством первого члена в знаменателе формулы является его зависимость от размера кристалла, рассмотренная ниже.

Количество кристаллов на пластине приблизительно равно площади пластины, деленной на площадь кристалла. Более точно его можно вычислить по формуле

$$\text{Количество кристаллов на пластине} = \frac{\pi \times (\text{Диаметр пластины}/2)^2}{\text{Площадь кристалла}} - \frac{\pi \times \text{Диаметр пластины}}{\sqrt{2 \times \text{Площадь кристалла}}}.$$

Первый член — отношение площади пластины (πr^2) к площади кристалла. Второй член компенсирует проблему «квадратного гвоздя в круглом отверстии» —

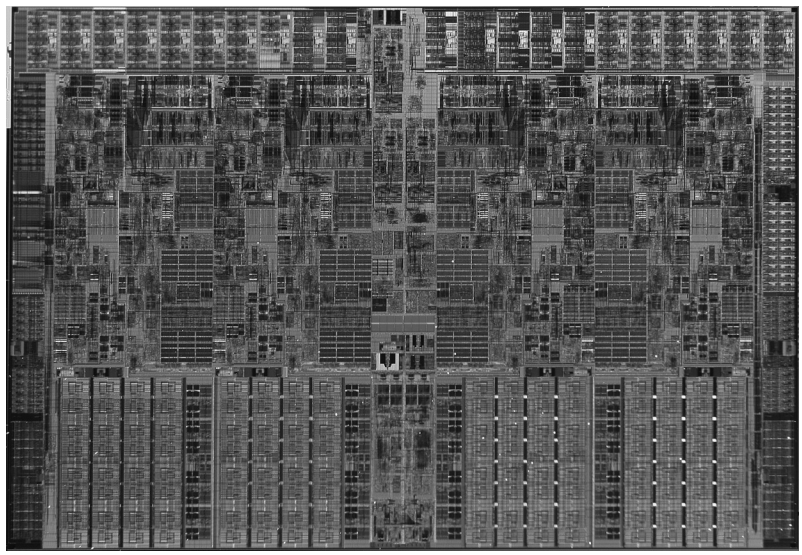

Рис. 1.13. Фотография кристалла микропроцессора Intel Core i7, который анализируется в главах 2—5. Его размеры 18,9 × 13,6 мм (257 мм^2) для процесса 45 нм (с разрешения Intel)

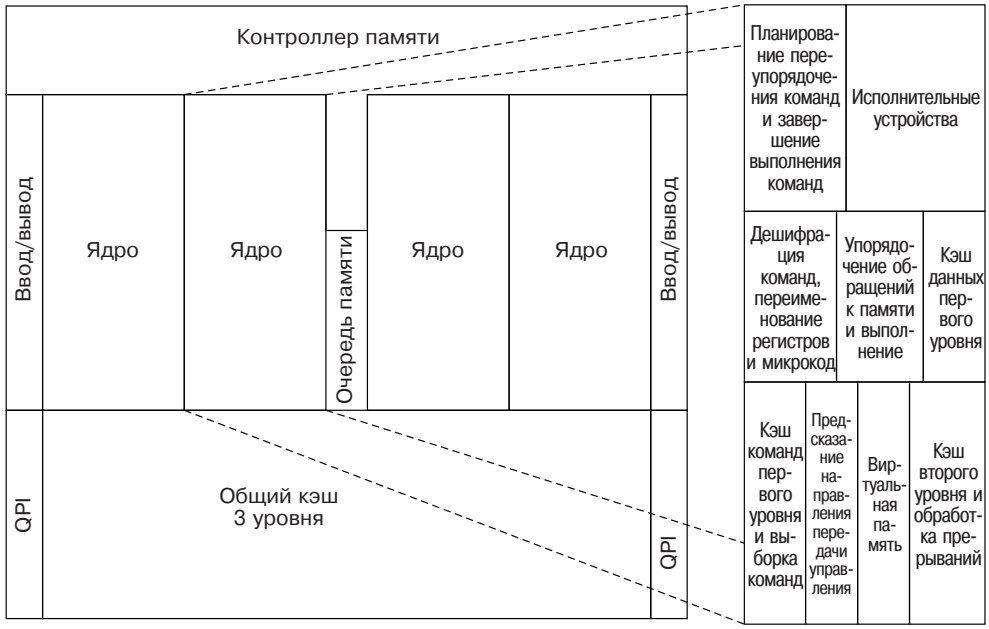

Рис. 1.14. В левой части — компоновочный план кристалла Core i7 с рис. 1.13, в правой — увеличенный компоновочный план второго ядра

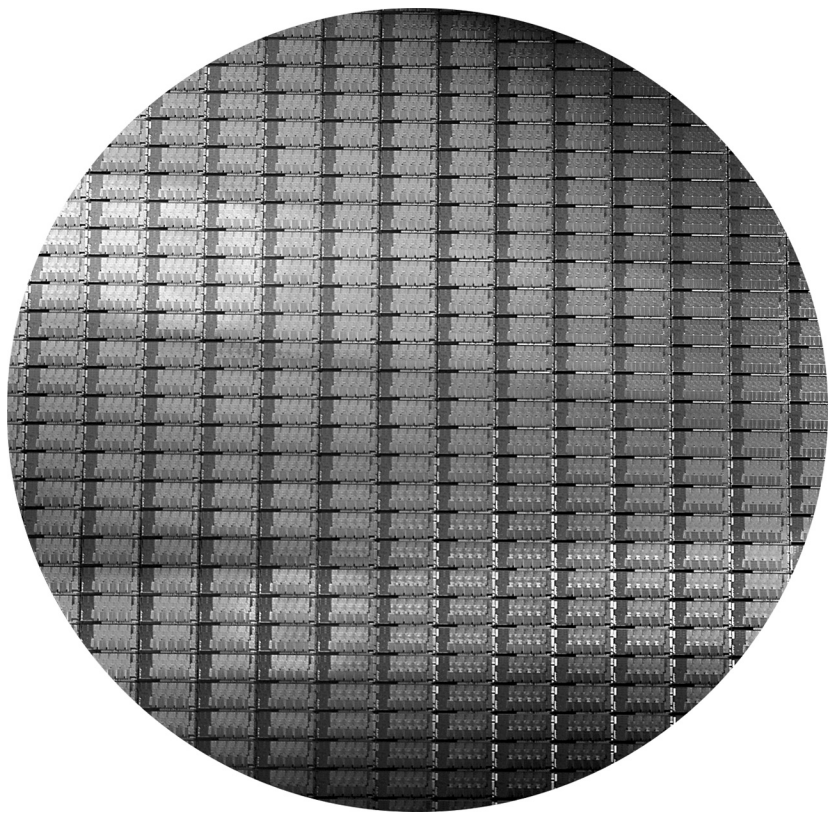

Рис. 1.15. Эта 300-мм пластина содержит 280 полных кристаллов Sandy Bridge, каждый размером 20,7 × 10,5 мм при процессе 32 нм. (Sandy Bridge — следующая после Nehalem архитектура Intel, использованная в Core i7.) При площади 216 мм^2 формула для количества кристаллов на пластину дает число 282 (с разрешения Intel)

прямоугольных кристаллов около периферии круглых пластин. Частное от деления длины окружности пластины (πd) на диагональ квадратного кристалла примерно равно количеству кристаллов, расположенных вдоль края.

Пример Найдем количество кристаллов на пластине диаметром 300 мм (30 см) для кристалла со стороной 1,5 и 1,0 см.

Ответ При площади кристалла 2,25 см^2:

$$\text{Количество кристаллов на пластине} = \frac{\pi \times (30/2)^2}{2{,}25} - \frac{\pi \times 30}{\sqrt{2 \times 2{,}25}} = \frac{706{,}9}{2{,}25} - \frac{94{,}2}{2{,}12} = 270.$$

Поскольку площадь большего кристалла в 2,25 раза больше, кристаллов меньшего размера на пластине помещается примерно в 2,25 раза больше:

$$\text{Количество кристаллов на пластине} = \frac{\pi \times (30/2)^2}{1{,}00} - \frac{\pi \times 30}{\sqrt{2 \times 1{,}00}} = \frac{706{,}9}{1{,}00} - \frac{94{,}2}{1{,}41} = 640.$$

Однако эта формула дает лишь максимальное количество кристаллов на пластине. Важнейший вопрос состоит в том, какова доля годных кристаллов на пластине или каков *выход годных кристаллов*. Простая модель выхода годных для интегральных схем, которая предполагает, что дефекты распределяются по пластине случайным образом и что выход годных обратно пропорционален сложности производственного процесса, приводит к следующему:

$$\text{Выход годных кристаллов} = \text{Выход годных пластин} \times \frac{1}{(1 + \text{КД} \times \text{Площадь кристалла})^N},$$

где КД — количество дефектов на единицу площади.

Эта формула Бозе — Эйнштейна представляет собой эмпирическую модель, полученную посредством наблюдения за выходом годных многих производственных линий [Sydow 2006]. *Выход годных пластин* учитывает полностью негодные пластины, тестировать которые нет необходимости. Для простоты мы будем считать, что выход годных пластин равен 100 %. Дефекты на единицу площади есть мера возникающих при производстве случайных дефектов. В 2010 г. эта величина обычно составляла 0,1—0,3 дефекта на квадратный дюйм, или 0,016—0,057 дефекта на квадратный сантиметр для процесса 40 нм, так как она зависит от зрелости технологического процесса (вспомните кривую освоения производства, упомянутую раньше). И наконец, N — параметр, называемый фактором сложности процесса, — мера трудности изготовления. Для процесса 40 нм в 2010 г. параметр N составлял 11,5—15,5.

Пример Найдем выход годных для кристаллов со сторонами 1,5 и 1,0 см, принимая плотность дефектов равной 0,031 на квадратный сантиметр и N равным 13,5.

Ответ Общие площади кристаллов составляют 2,25 и 1,00 см^2. Для большего кристалла выход годных составит:

$$\text{Выход годных кристаллов} = 1/(1 + 0{,}031 \times 2{,}25)^{13{,}5} = 0{,}40.$$

Для кристалла меньшего размера выход годных составит:

$$\text{Выход годных кристаллов} = 1/(1 + 0{,}031 \times 1{,}00)^{13{,}5} = 0{,}66.$$

То есть годных больших кристаллов меньше половины, тогда как меньшего размера — две трети.

Нижняя строка в формуле стоимости кристалла (см. стр. 63) является количеством годных кристаллов на пластине и получается умножением количества кристаллов на пластине на выход годных, чтобы учесть влияние дефектов. Приведенные выше примеры предсказывают примерно 109 годных кристаллов площадью 2,25 см^2 и 424 кристалла площадью 1,00 см^2 на пластине диаметром 300 мм. Многие микропроцессоры попадают в диапазон, задаваемый этими двумя размерами. Встроенные 32-битовые процессоры невысокой производительности иногда не превышают 0,10 см^2, а процессоры, используемые для встроенного управления (в принтерах, микроволновках и т.д.), часто даже меньше 0,04 см^2.

Учитывая огромное ценовое давление на товары широкого потребления, такие как микросхемы памяти DRAM и SRAM, разработчики используют избыточность

как способ увеличения выхода годных. В течение ряда лет в DRAM постоянно включалось некоторое количество избыточных ячеек памяти, чтобы можно было компенсировать определенное количество дефектов. Разработчики применяют подобные методы и в стандартных SRAM, и в больших массивах SRAM, используемых в кэшах внутри микропроцессоров. Очевидно, что наличие избыточных элементов позволяет значительно увеличить выход годных.

В 2010 г. производство пластины диаметром 300 мм (12 дюймов) по передовой технологии стоило от 5000—6000 долл. Если предположить, что стоимость готовой пластины равна 5500 долл., то стоимость кристалла площадью 1,00 см^2 составит примерно 13 долл. В то же время стоимость кристалла площадью 2,25 см^2 будет уже около 51 долл., или почти в четыре раза больше стоимости кристалла, который почти в два раза меньше.

Что же о стоимостях кристаллов следует помнить разработчику компьютера? Производственный процесс определяет стоимость пластины, выход годных пластин и дефекты на единицу площади, так что единственное, чем он может управлять, — это площадь кристалла. На практике, поскольку количество дефектов на единицу площади мало, количество годных кристаллов на пластине, а следовательно, и стоимость кристалла растут примерно как квадрат площади кристалла. Разработчик компьютера влияет на размер кристалла и тем самым на его стоимость, включая или исключая те или иные функции, а также посредством количества контактов ввода/вывода.

Прежде чем мы получим партию интегральных схем, готовых к использованию в компьютере, кристалл необходимо протестировать (для отделения годных от негодных), корпусировать и после этого снова протестировать. Все это значительно повышает стоимость.

Приведенный выше анализ был посвящен изменяющимся стоимостям изготовления функционального кристалла применительно к интегральным схемам, выпускаемым в больших количествах. Однако есть одна очень важная составляющая фиксированных стоимостей, которая может значительно влиять на стоимость интегральных схем, выпускаемых в небольших количествах (партии менее 1 млн штук), а именно стоимость набора масок. Каждый шаг в процессе изготовления интегральной схемы требует отдельной маски. Поэтому для современных высокоплотных процессов производства с 4—6 слоями металла стоимость масок превышает 1 млн долл. Очевидно, что на эту стоимость влияют создание прототипа и отладка, а это может составить значительную часть себестоимости для продукции с небольшими объемами выпуска. Так как стоимости масок, вероятно, продолжат увеличиваться, разработчики могут встраивать реконфигурируемую логику для улучшения приспособляемости части интегральной схемы, или предпочесть использовать матрицы логических элементов (имеющих меньше слоев заказных масок) и тем самым уменьшить стоимостное влияние масок.

Стоимость и цена

С превращением компьютеров в товар широкого потребления разница между стоимостью производства товара и его рыночной ценой уменьшалась. Эта разница образуется расходами компании на исследования и разработку (R&D), маркетинг,

продажи, обслуживание производственного оборудования, аренду зданий, стоимостью финансирования, предналоговой прибылью и налогами. Многие инженеры с удивлением узнают, что большинство компаний тратят только от 4 % (в сегменте производства персональных компьютеров) до 12 % (в сегменте производства старших моделей серверов) своего дохода на исследования и разработку (R&D), что включает в себя всю инженерию.

Стоимость производства и стоимость эксплуатации

В первых четырех изданиях этой книги под стоимостью понималась стоимость создания компьютера, а под ценой — цена покупки компьютера. С появлением компьютеров WSC, содержащих десятки тысяч серверов, стоимость эксплуатации компьютеров стала существенной прибавкой к стоимости покупки.

Как показано в главе 6, амортизированная цена покупки серверов и сетей составляет чуть более 60 % от месячной стоимости эксплуатации компьютера WSC в предположении, что срок службы IT-оборудования составляет три-четыре года. Около 30 % стоимости месячных эксплуатационных расходов приходится на энергопотребление и амортизацию инфраструктуры для распределения электропитания и охлаждения IT-оборудования, несмотря на то что эта инфраструктура амортизируется в течение 10 лет. Таким образом, чтобы снизить стоимости эксплуатационных расходов компьютеров WSC, компьютерные архитекторы должны использовать энергию эффективно.

1.7. Системная надежность

Исторически, интегральные микросхемы были одними из самых надежных компонентов компьютера. Хотя их выводы могут быть уязвимыми, а в каналах связи могут возникать сбои, частота ошибок внутри микросхемы оставалась очень низкой. Это общеизвестное представление изменяется по мере продвижения к минимальным топологическим размерам 32 нм и меньше[1], ибо сбои и отказы становятся более обычными, так что архитекторы должны проектировать системы с учетом этих проблем. В данном разделе приводится краткий обзор вопросов системной надежности (dependability), а формальное определение терминов и подходов приводится в разделе D.3 «Приложение D».

Компьютеры проектируются и строятся на различных уровнях абстракции. Мы можем рекурсивно просматривать структуру компьютера, начиная с компонентов, являющихся полными подсистемами, пока не достигнем отдельных транзисторов. Хотя некоторые сбои, такие как нарушение питания, оказывают обширное воздействие, многие из них могут воздействовать только на один из компонентов внутри модуля. Таким образом, полный отказ модуля на одном уровне может рассматриваться всего лишь как ошибка некоторого компонента в модуле более высокого

[1] Научно-техническая литература, посвященная вопросам надежности интегральных схем, по состоянию на начало 2014 г. не содержит данных об ухудшении надежности микросхем с минимальными топологическими размерами 32 нм и меньше. — *Прим. ред.*

уровня. Такое различие полезно при поиске путей к построению надежных компьютеров.

Одна из трудностей заключается в решении вопроса, работает ли система должным образом. Этот философский вопрос стал конкретным в связи с популярностью интернет-сервисов. Инфраструктурные провайдеры начали предлагать *соглашения об уровне обслуживания* (*SLAs — Service Level Agreements*) или о *цели уровня обслуживания* (*SLOs — Service Level Oobjectives*), которые должны гарантировать, что их сетевые услуги или услуги по обеспечению производительностью будут надежными. Например, если их услуги не соответствовали соглашению более нескольких часов в месяц, то потребителю должен быть уплачен штраф. Таким образом, соглашение SLA могло бы использоваться для решения вопроса о работоспособности системы.

В соответствии с соглашением SLA системы могут находиться в одном из двух состояний:

1) *успешное обслуживание* (*service accomplishment*), при котором обслуживание соответствует своей спецификации;
2) *прерывание обслуживания* (*service interruption*), если предоставленная услуга не соответствует SLA.

Переходы между этими состояниями вызываются *отказами* (*failures*) (из состояния 1 в состояние 2) или *восстановлениями* (*restorations*) (из состояния 2 в состояние 1). Количественное представление этих переходов приводит к двум главным показателям системной надежности:

- *надежность модуля* (*module reliability*) — показатель непрерывного успешного обслуживания (или, что эквивалентно, время до момента отказа), начиная с определенной начальной точки. Следовательно, *наработка до отказа*, или *время безотказной работы* (*MTTF — Mean Time To Failure*), является показателем надежности. Величина, обратная MTTF, — частота отказов. Обычно эта величина определяется как количество отказов на миллиард часов работы и обозначается как *FIT* (*Failures In Time — количество отказов в заданное время*). 1 FIT = 10^{-9}/час. Таким образом, показателю MTTF, равному 1 000 000 часов, соответствует частота отказов $10^9/10^6$, или 1000 FIT. Показателем прерывания обслуживания является *среднее время восстановления* (*MTTR — Mean Time To Repair*). *Среднее время наработки между отказами* (*MTBF — Mean Time Between Failures*) — это просто сумма MTTF + MTTR. Хотя широко используется показатель MTBF, часто более подходящим показателем является MTTF. Если набор модулей имеет экспоненциально распределенные сроки службы (а это означает, что возраст модуля не влияет на вероятность отказа), то общая частота отказов набора является суммой частот отказов модулей;
- *доступность модуля* (*module availability*) — этот показатель является отношением времени успешного обслуживания к общему времени работы модуля (сумма времен успешного обслуживания и прерывания обслуживания). Для нерезервированных систем с восстановлением доступность модуля равна:

$$\text{Доступность модуля} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}.$$

Обратите внимание, что надежность и доступность теперь являются количественными показателями, а не просто синонимами системной надежности. С помощью этих определений мы можем количественно оценить системную надежность, если сделаем некоторые предположения о надежности компонентов и о том, что отказы независимы друг от друга.

Пример Рассмотрим дисковую подсистему со следующими компонентами и значениями MTTF:

- 10 дисков, MTTF каждого — 1 000 000 часов,
- 1 ATA-контроллер, MTTF — 500 000 часов,
- 1 источник питания, MTTF — 200 000 часов,
- 1 вентилятор, MTTF — 200 000 часов,
- 1 ATA-кабель, MTTF — 1 000 000 часов.

Положив для простоты, что сроки службы компонентов распределены экспоненциально и отказы независимы друг от друга, рассчитаем MTTF всей системы.

Ответ Сумма частот отказов равна:

$$\text{Частота отказов}_{\text{системы}} = 10 \times \frac{1}{1\,000\,000} + \frac{1}{500\,000} + \frac{1}{200\,000} + \frac{1}{200\,000} + \frac{1}{1\,000\,000} =$$

$$= \frac{10 + 2 + 5 + 5 + 1}{1\,000\,000} = \frac{23}{1\,000\,000} = \frac{23\,000}{1\,000\,000\,000 \text{ часов}}$$

или 23 000 FIT. MTTF системы является величиной, обратной частоте отказов, то есть

$$\text{MTTF}_{\text{системы}} = \frac{1}{\text{частота отказов}_{\text{системы}}} = \frac{1\,000\,000\,000}{23\,000} = 43\,500 \text{ часов}$$

или чуть менее пяти лет.

Основным способом борьбы с отказами является избыточность — либо во времени (повторение операции, чтобы проверить, является ли она по-прежнему ошибочной), либо в ресурсах (наличие других компонентов, которые возьмут на себя функции отказавшего). После замены компонента и полного восстановления системы системная надежность системы предполагается такой же, как и в самом начале работы. Для количественной оценки полезности избыточности рассмотрим пример.

Пример Чтобы улучшить системную надежность, дисковые подсистемы часто применяют избыточные источники питания. Используя компоненты и значения MTTF из предыдущего примера, определим надежность избыточных источников питания. Полагаем, что для работы дисковой подсистемы достаточно одного источника питания и что мы добавили еще один избыточный источник питания.

Ответ Нам нужна формула, показывающая, что следует ожидать при отказе, продолжая обслуживание. Для упрощения расчетов полагаем, что сроки службы компонентов распределены экспоненциально, а отказы независимы друг от друга. Значение MTTF для наших избыточных источников питания равно среднему времени до отказа одного из источников питания, деленному на вероятность того, что второй источник откажет до замены первого. Таким образом, если вероятность второго отказа до восстановления мала, то значение MTTF пары источников велико.

Поскольку мы имеем два источника питания и независимые отказы, среднее время до отказа одного диска равно значению $\text{MTTF}_{\text{источника питания}}/2$. Хорошим приближением вероятности второго отказа является значение MTTR, деленное на среднее время до отказа второго источника питания. Отсюда приемлемым приближением значения MTTF пары источников питания является:

$$\text{MTTF}_{\text{пары источников}} = \frac{\text{MTTF}_{\text{источника}}/2}{\frac{\text{MTTR}_{\text{источника}}}{\text{MTTF}_{\text{источника}}}} = \frac{\text{MTTF}^2_{\text{источника}}/2}{\text{MTTR}_{\text{источника}}} = \frac{\text{MTTF}^2_{\text{источника}}}{2 \times \text{MTTR}_{\text{источника}}}.$$

Используя значения MTTF из предыдущего примера и предполагая, что оператору нужно в среднем 24 часа для обнаружения отказа и замены источника питания, получим надежность устойчивой к отказам пары источников питания, равную

$$\text{MTTF}_{\text{пары источников}} = \frac{\text{MTTF}^2_{\text{источника}}}{2 \times \text{MTTR}_{\text{источника}}} = \frac{200000^2}{2 \times 24} \approx 830\ 000\ 000,$$

что превосходит надежность одного источника питания примерно в 4150 раз.

Оценив количественно стоимость, мощность и системную надежность технологии компьютера, мы готовы перейти к количественной оценке производительности.

1.8. Измерение, отчетность и обобщение показателей производительности

Что мы имеем в виду, когда говорим, что один компьютер быстрее другого? Пользователь настольного компьютера может сказать, что компьютер быстрее тогда, когда программа выполняется за меньшее время, в то время как, по мнению администратора Amazon.com, более быстрый компьютер выполняет больше транзакций за час. Пользователя компьютера интересует уменьшение *времени отклика* (*response time*), то есть времени между началом и окончанием события, которое также называют *временем выполнения* (*execution time*). Оператора компьютера WSC может интересовать увеличение *пропускной способности* (*throughput*), то есть общее количество работы, сделанной за определенное время.

При сравнении проектных альтернатив часто нужно сопоставить производительность двух различных компьютеров, например X и Y. Фраза «X быстрее, чем Y» означает, что для данной задачи время отклика или время выполнения на компьютере X меньше, чем на компьютере Y. В частности, «X в *n* раз быстрее, чем Y» будет означать

$$\frac{\text{Время выполнения}_Y}{\text{Время выполнения}_X} = n.$$

Поскольку время выполнения является величиной, обратной производительности, то имеет место соотношение

$$\frac{\text{Время выполнения}_Y}{\text{Время выполнения}_X} = \frac{\frac{1}{\text{Производительность}_Y}}{\frac{1}{\text{Производительность}_X}} = \frac{\text{Производительность}_X}{\text{Производительность}_Y}.$$

Фраза «пропускная способность компьютера X в 1,3 раза больше, чем пропускная способность компьютера Y» означает, что компьютер X выполняет в 1,3 раза больше задач в единицу времени, чем компьютер Y.

К сожалению, при сравнении производительности компьютеров время используется не всегда. Мы считаем, что единственной достоверной и надежной мерой производительности является время выполнения реальных программ, а все предложенные альтернативы времени в качестве характеристики и реальным программам как объектам изменений в конечном счете приводят к неверным утверждениям и даже ошибкам при проектировании компьютеров.

Даже время выполнения может быть определено различными способами в зависимости от того, что мы рассчитываем. Наиболее простым определением времени являются так называемые *настенное время* (*wall-clock time*), *время отклика* (*response time*) или *затраченное время* (*elapsed time*), которое представляет собой задержку до завершения задачи, включая обращения к дискам, обращения к памяти, операции ввода/вывода, работу операционной системы, то есть все. В случае мультипрограммирования процессор, ожидая завершения ввода/вывода при выполнении одной программы, выполняет другую программу и поэтому может и не минимизировать время выполнения отдельной программы. Следовательно, нам нужен термин, принимающий во внимание такое взаимодействие. *Процессорное время* (*CPU time*) учитывает данное различие и означает время, на протяжении которого процессор непосредственно занят вычислениями, и не включает ожидание завершения ввода/вывода или выполнение других программ. (Понятно, что с точки зрения пользователя временем отклика является все время выполнения данной программы, а не время работы центрального процессора.)

Пользователи, обычно запускающие на исполнение одни и те же программы, были бы прекрасными кандидатами для оценки нового компьютера. Чтобы оценить новую систему, они просто сравнивали бы времена выполнения их *рабочих нагрузок* (*workloads*) — смеси программ и команд операционной системы, которые пользователи запускают на компьютере. Однако таких счастливчиков немного. Большинство должны полагаться на другие методы оценки компьютеров и зачастую на других оценщиков, надеясь, что эти методы предскажут производительность в случае использования нового компьютера.

Тесты (benchmarks)

Лучшим вариантом тестов для измерения производительности являются реальные приложения, такие как Google Goggles, описанный в разделе 1.1. Попытки выполнения программ, значительно более простых, чем реальное приложение, приводили к недостоверным оценкам производительности. Примерами таких программ являются:

- *ядра* (*kernels*), являющиеся небольшими ключевыми фрагментами реальных приложений;
- *программы-игрушки* (*toy programs*), представляющие собой учебные программы длиной в 100 строк, такие как быстрая сортировка;
- *синтетические тесты* (*synthetic benchmarks*), которые представляют собой псевдопрограммы, придуманные, чтобы попытаться имитировать свойства и поведение реальных программ. Таков, например, тест Dhrystone.

Все три типа программ сегодня скомпрометированы обычно из-за того, что разработчик компилятора и архитектор могут сговориться сделать так, чтобы на этих эрзац-программах компьютер показывал более высокую производительность, чем на реальных приложениях. В четвертом издании данной книги ее авторы не отметили это заблуждение в отношении применения синтетических программ для оценки производительности, ибо полагали, что архитекторы компьютеров признали их использование неприличным. Но вот что огорчает — синтетический тест Dhrystone до сих пор является наиболее цитируемым тестом для встраиваемых процессоров!

Другой проблемой являются условия выполнения тестов. По сей день один из способов улучшить производительность тестовой программы связан со специфическими для теста флагами, которые часто вызывают преобразования, запрещенные для многих программ или снижающие производительность других. Чтобы ограничить этот процесс и поднять значимость результатов, разработчики тестов часто требуют от поставщика использовать один и тот же компилятор и набор флагов для всех программ на определенном языке (C или Си++). Помимо флагов компилятора, существует еще вопрос о допустимости модификаций исходного кода. Возможны три различных подхода к этому вопросу.

1. Модификации исходного кода не допускаются.
2. Модификации исходного кода допускаются, но неосуществимы на практике. Например, тесты для баз данных, основанные на стандартных программах баз данных, которые содержат десятки миллионов строк кода. Весьма маловероятно, чтобы компании, разрабатывающие эти программы, вносили в них изменения для улучшения производительности одного конкретного компьютера.
3. Модификации исходного кода допускаются при условии, что модифицированная версия выдает тот же результат.

Ключевой вопрос, с которым встречаются разработчики тестов, рассматривая допустимость модификации исходного кода, состоит в том, будет ли такая модификация отражать реальность и будет ли полезной пользователям, или такая модификация только уменьшит точность тестов как предсказателей реальной производительности.

Чтобы не класть слишком много яиц в одну корзину, широко используются измерения производительности процессоров на различных приложениях с помощью наборов тестовых приложений, так называемых *пакетов тестов* (*benchmark suites*). Разумеется, качество таких *пакетов* зависит от достоинств входящих в них тестов. Но все же важным преимуществом таких *пакетов* является то, что слабость одного теста менее существенна из-за наличия других. Назначение *пакета* тестов состоит в том, что он будет оценивать относительную производительность двух компьютеров, в частности и для отсутствующих в *пакете* программ, которые, возможно, будут выполнять потребители.

Предостерегающим примером являются тесты от консорциума Electronic Design News Embedded Microprocessor Benchmark Consortium или EEMBC, что произносится как «embassy». Эта система из 41 ядра используется для предсказания производительности встроенных применений в различных областях: автомобильной промышленности, сфере потребления, компьютерных сетях, для автоматизации офисов и телекоммуникации. EEMBC показывает производительность без модификаций и производительность «на всю катушку», когда задействуются почти все ресурсы. В силу того, что в этих тестах используются ядра и задаются указанные выше опции, EEMBC не имеет репутации хорошего предсказателя относительной производительности различных встроенных компьютеров в этой области. Вот поэтому все еще используется тест Dhrystone, который пробовали заменить тестами EEMBC.

Одна из наиболее успешных попыток создать стандартизированные пакеты прикладных программ для тестирования принадлежит корпорации SPEC, которая в конце 1980-х гг. начала разрабатывать лучшие тесты для рабочих станций. По мере развития компьютерной индустрии росла и потребность в различных пакетах тестов, так что сегодня тесты SPEC охватывают многие классы приложений. Все пакеты тестов SPEC и их опубликованные результаты можно найти на сайте *www.spec.org*.

Хотя во многих следующих разделах основное внимание уделено тестам SPEC, много тестов было разработано также для персональных компьютеров с операционной системой Windows.

Тесты для настольных компьютеров

Тесты для настольных компьютеров делятся на два больших класса, один из которых рассчитан на интенсивную работу процессора, другой — на большой объем графических операций, хотя во многих тестах для графики также интенсивно используется процессор. Корпорация SPEC первоначально разработала пакет тестов для оценки производительности процессоров (исходное название SPEC89), который эволюционировал, и его пятому поколению SPEC CPU2006 предшествовали SPEC2000, SPEC95, SPEC92 и SPEC89. SPEC CPU2006 содержит 12 целочисленных тестов (CINT2006) и 17 тестов для плавающей запятой (CFP2006). На рис. 1.16 приведено описание всех версий тестов SPEC.

Тесты SPEC представляют собой реальные программы, модифицированные, чтобы обеспечить переносимость и минимизировать влияние ввода/вывода на производительность.

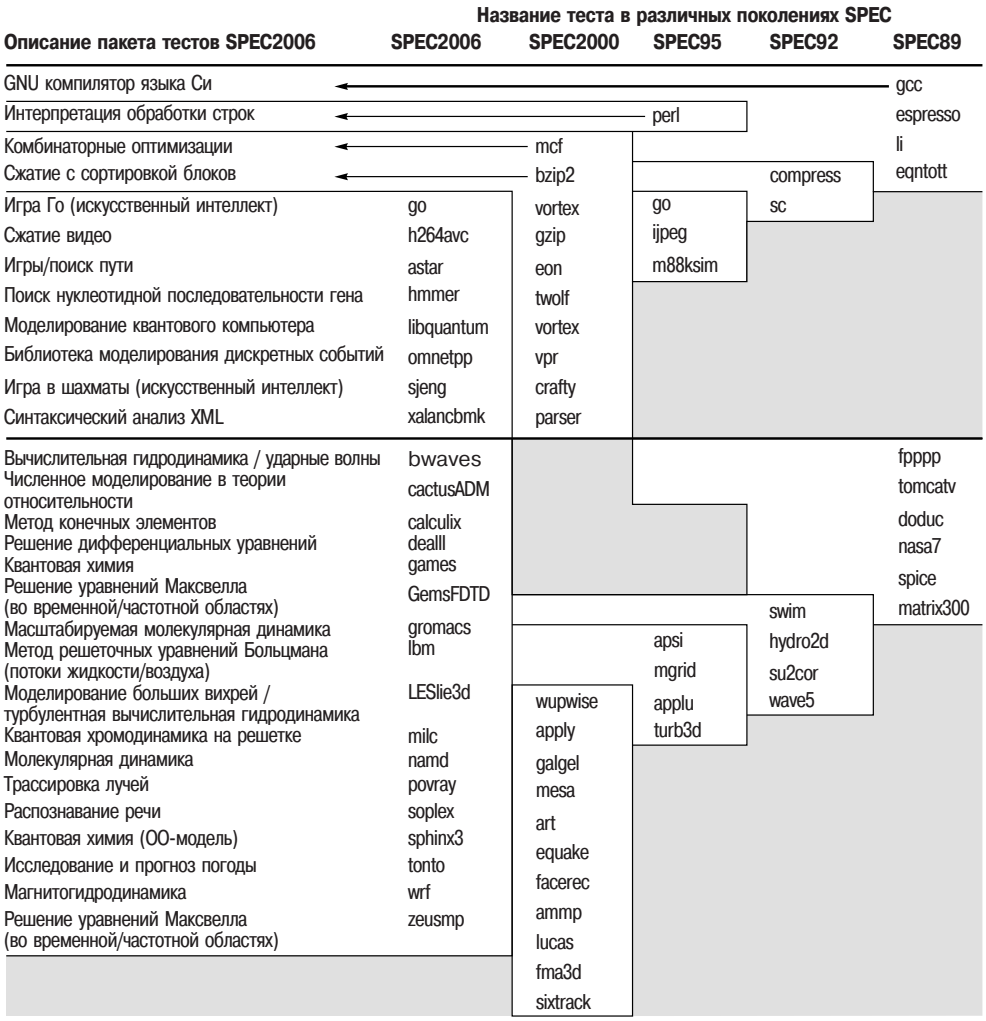

Рис. 1.16. Программы SPEC2006 и эволюция тестов SPEC во времени. Программы целочисленных вычислений представлены выше линии, вычислений с плавающей запятой — ниже ее. Из 12 целочисленных программ SPEC2006 9 программ написаны на C, а остальные — на C++. Среди программ для плавающей запятой шесть написаны на Фортране, четыре — на C++, три — на C и три — на смеси C и Фортрана. На рисунке показаны все 70 программ в реализациях 1989, 1992, 1995, 2000 и 2005 гг. Описания тестов слева даны только для SPEC2006 и неприменимы к более ранним версиям. Программы различных поколений SPEC, указанные в одном ряду, в общем, не связаны, например, fpppp не является тестом типа теста bwaves для вычислительной гидродинамики / ударных волн. Старше всех в группе тест gcc. Только три целочисленные программы и три программы для плавающей запятой «выжили» в трех и более поколениях. Отметим, что все программы для плавающей запятой в SPEC2006 новые. Хотя несколько программ переносятся от поколения к поколению, версия такой программы обновляется, а входные данные или размер теста часто меняются, чтобы увеличить время ее выполнения и избежать неточности измерения или доминирования выполнения какого-либо фактора в общем времени, отличного от времени работы центрального процессора

Целочисленные тесты существенно различаются — от фрагмента компилятора с языка C до шахматной программы и моделирования квантового компьютера. Тесты для плавающей запятой включают ряд программ, применяемых при моделировании, например построение структурированных сеток для метода конечных элементов, метод частиц в молекулярной динамике и методы линейной алгебры с разреженными матрицами для динамики жидкостей. Пакет SPEC CPU полезен при измерении производительности процессора в настольных системах и однопроцессорных серверах. Далее будут представлены данные по многим программам пакета. Тем не менее отметим, что эти программы имеют мало общего с языками программирования, средами исполнения и приложением Google Goggles, описанным в разделе 1.1. Семь из них используют C++, восемь — язык C и девять — Фортран! Мало того, что они компонуются статически, сами приложения не представляют собой ничего интересного. Вообще неясно, что же примечательного из вычислений XXI века входит в SPEC CINT2006 и SPEC CFP2006.

В разделе 1.11 представлены просчеты, допущенные при разработке пакета тестов SPEC, а также проблемы поддержки полезного и предсказывающего набора тестов.

SPEC CPU2006 оценивает производительность процессоров, но корпорация SPEC предлагает и много других тестов.

Тесты для серверов

Так как функции серверов многообразны, для них разработано много типов тестов. Возможно, самый простой тест — это тест оценки пропускной способности процессора. SPEC CPU2000 использует тесты SPEC CPU в качестве простого теста оценки пропускной способности, в котором скорость работы мультипроцессора может быть измерена путем прогона нескольких (по количеству процессоров) копий тестов SPEC CPU и преобразования времени работы процессора в скорость. Это приводит к показателю, названному SPECrate, который оценивает параллелизм уровня запросов, введенный в разделе 1.2. Для измерения параллелизма уровня потоков SPEC предлагает то, что корпорация называет тестами высокопроизводительных вычислений для OpenMP (Open Multi-Processing, открытый стандарт для распараллеливания программ) и MPI (Message Passing Interface, интерфейс передачи сообщений).

Помимо показателя SPECrate, многие серверные приложения и тесты имеют значительный объем операций ввода/вывода, вызванный дисковым или сетевым трафиком. К ним относятся тесты для файловых серверов, веб-серверов, систем баз данных и обработки транзакций. Корпорация SPEC предлагает тесты для файлового сервера (SPECSFS) и веб-сервера (SPECWeb). Тест SPECSFS предназначен для измерения производительности сетевой файловой системы NFS (Network File System) с использованием сценария (script) запросов к файловому серверу; он тестирует производительность подсистемы ввода/вывода (дисковой или сетевой) и процессора. Тест SPECSFS ориентирован на пропускную способность, но с учетом существенных требований ко времени отклика. (В приложении D подробно обсуждаются некоторые тесты для файловой подсистемы и

подсистемы ввода/вывода.) Тест SPECWeb предназначен для веб-серверов, он имитирует запросы к статическим и динамическим страницам сервера от множества клиентов, а также отправку данных клиентов на сервер. Тест SPECjbb измеряет производительность веб-приложений, написанных на языке Java. Последним по времени тестом корпорации SPEC является тест SPECvirt_Sc2010, оценивающий полную (end-to-end) производительность виртуализированных серверов в центрах обработки данных, в том числе аппаратуру, уровень виртуальной машины и виртуализированную «гостевую» операционную систему. Еще один недавний тест корпорации SPEC, рассматриваемый в разделе 1.10, измеряет мощность.

Тесты обработки транзакций (TP — Transacion-Processing) оценивают способность системы обрабатывать транзакции, состоящие из доступа к базе данных и ее обновления. Системы бронирования авиабилетов и банковские ATM-системы (банкоматы) — это типичные простые примеры TP-систем, а в более сложные TP-системы входят базы данных и процедуры принятия решений. В середине 1980-х гг. группа заинтересованных инженеров сформировала независимый от поставщиков совет (TPC — Transaction Processing Council), чтобы попытаться создать реалистичные и беспристрастные тесты для TP. Тесты совета TPC описаны на сайте *www.tpc.org*.

Первый вариант теста TPC под названием TPC-A опубликован в 1985 г., и с тех пор на смену ему пришло несколько других, более совершенных. Созданный в 1992 г. тест TPC-C имитирует сложную среду запросов. Тест TPC-H моделирует поддержку специальных решений, при этом запросы не связаны друг с другом и прошлые запросы не могут быть использованы для оптимизации будущих. Тест TPC-E — это новый вариант, рассчитанный на онлайн-обработку транзакций (OLTP — On-Line Transaction Processing), он имитирует счета клиентов брокерской конторы. Последним достижением является тест TPC Energy, который добавляет показатели потребляемой энергии во все существующие TPC-тесты.

Все тесты TPC измеряют производительность количеством транзакций в секунду. Более того, в них включены требования ко времени отклика, так что измерение пропускной способности выполняется, только когда время отклика не превышает заданного предела. При моделировании реальных систем более высокие скорости транзакций также ассоциируются с системами большего размера (в терминах пользователей и базы данных), в которых выполняются транзакции. И наконец, стоимость системы также должна быть задана для тестовой системы, что позволяет точно сравнивать соотношение стоимость — производительность. Совет TPC изменил свою ценовую политику так, что теперь имеется одна спецификация для всех тестов TPC и можно проверить цены, которые TPC публикует.

Отчетность по результатам измерений производительности

Руководящим принципом отчетности при измерениях производительности должна быть *воспроизводимость* (*reproducibility*) — список всего, что может потребоваться другому экспериментатору, чтобы повторить результаты. Отчет по тесту SPEC требует расширенного описания компьютера и флагов компилятора, а также публикации базовых и оптимизированных результатов. Помимо описаний

аппаратуры, программного обеспечения и параметров настройки базового режима, отчет SPEC содержит представленные в таблицах и на графиках действительные времена, характеризующие производительность. Отчет по тесту TPC еще более полный, так как он должен содержать результаты проверки тестирования и информацию о стоимости. Эти отчеты являются превосходными источниками для определения реальных стоимостей вычислительных систем, поскольку производители конкурируют в достижении высоких показателей производительности и соотношения стоимость — производительность.

Обобщение показателей производительности

При практической разработке компьютера необходимо оценить много проектных вариантов на предмет их количественных преимуществ с помощью набора тестов, которые считаются подходящими. Аналогично потребители, пытаясь выбрать компьютер, будут полагаться на измерения производительности на тестах, которые, хочется надеяться, похожи на приложения пользователя. В обоих случаях полезно иметь измерения на наборе тестов, чтобы производительность важных приложений была схожа с одним или несколькими тестами из набора и чтобы разница в производительности могла быть понятна. В идеальном случае такой набор напоминает статистически значимую выборку в прикладной области, однако подобная выборка требует больше тестов, чем обычно присутствует в большинстве наборов, а также случайной выборки, которая, по сути, в наборе тестов не применяется.

Если уж решено измерять производительность с помощью набора тестов, то нам хотелось бы свести результаты оценки производительности к одному числу. Прямым подходом к вычислению обобщенного результата могло быть сравнение средних арифметических значений времен выполнения программ в наборе. Увы, некоторые программы SPEC выполняются в четыре раза дольше других, поэтому эти программы больше других повлияли бы на производительность, если бы одним числом, используемым для обобщения производительности, было бы среднее арифметическое. Альтернативой могло бы быть введение весового множителя для каждого теста и использование взвешенного среднего арифметического для обобщения производительности. Но в таком случае возникает проблема подбора весов: поскольку SPEC является консорциумом конкурирующих компаний, каждая компания предлагала бы предпочтительную для нее систему весов и достичь консенсуса было бы затруднительно. Есть еще подход с использованием весов, которые выравнивают время выполнения программ на некотором эталонном компьютере, но это смещает результаты к характеристикам производительности эталонного компьютера.

Вместо подбора весов мы могли бы нормализовать времена выполнения по отношению к эталонному компьютеру, поделив время на эталонном компьютере на время оцениваемого компьютера, и получить отношение, пропорциональное производительности. Этот подход использует SPEC, а данное отношение называется SPECRatio. Он полезен особенно благодаря тому, что совпадает с методом, используемым при сравнении производительности компьютеров во всей этой книге, а именно сопоставлением отношений производительностей. Например,

предположим, что значение SPECRatio компьютера A на некотором тесте в 1,25 раза больше, чем у компьютера B, тогда получим:

$$1{,}25 = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\dfrac{\text{Время}_{\text{вып}_{эк}}}{\text{Время}_{\text{вып}_A}}}{\dfrac{\text{Время}_{\text{вып}_{эк}}}{\text{Время}_{\text{вып}_B}}} = \frac{\text{Время}_{\text{вып}_B}}{\text{Время}_{\text{вып}_A}} = \frac{\text{Производительность}_A}{\text{Производительность}_B}.$$

Обратите внимание, что в формуле значения времен выполнения (время$_{\text{вып}}$) на эталонном компьютере сокращаются. Когда при сравнении используется отношение времен выполнения, выбор эталонного компьютера становится несущественным, именно такой подход принят далее в книге. На рис. 1.17 приведен пример.

Поскольку SPECRatio является отношением, а не абсолютным временем выполнения, то и среднее надо рассчитывать, используя *среднее геометрическое*. (Так как

Тесты	Ultra 5 время (с)	Opteron время (с)	SPECRatio	Itanium 2 время (с)	SPECRatio	Opteron/ Itanium время (с)	Itanium/ Opteron SPECRatios
wupwise	1600	51,5	31,06	56,1	28,53	0,92	0,92
swim	3100	125,0	24,73	70,7	43,85	1,77	1,77
mgrid	1800	98,0	18,37	65,8	27,36	1,49	1,49
applu	2100	94,0	22,34	50,9	41,25	1,85	1,85
mesa	1400	64,6	21,69	108,0	12,99	0,60	0,60
galgel	2900	86,4	33,57	40,0	72,47	2,16	2,16
art	2600	92,4	28,13	21,0	123,67	4,40	4,40
equake	1300	72,6	17,92	36,3	35,78	2,00	2,00
facerec	1900	73,6	25,80	86,9	21,86	0,85	0,85
ammp	2200	136,0	16,14	132,0	16,63	1,03	1,03
lucas	2000	88,8	22,52	107,0	18,76	0,83	0,83
fma3d	2100	120,0	17,48	131,0	16,09	0,92	0,92
sixtrack	1100	123,0	8,95	68,8	15,99	1,79	1,79
apsi	2600	150,0	17,36	231,0	11,27	0,65	0,65
Геометрическое среднее			20,86		27,12	1,30	1,30

Рис. 1.17. Времена выполнения SPECfp2000 (в секундах) для Sun Ultra 5 — эталонного компьютера SPEC2000, а также времена выполнения и значения SPECratio для AMD Opteron и Intel Itanium 2. (SPEC2000 умножает отношение времен выполнения на 100, чтобы удалить из результата десятичную запятую, поэтому 20,86 представляется как 2086.) Последние две колонки показывают отношения времен выполнения и значений SPECRatio. Здесь демонстрируется несущественность выбора эталонного компьютера для относительной производительности. Отношение времен выполнения идентично соответствующему отношению из колонки SPECRatio, а отношение средних геометрических (27,12/20,86 = 1,30) идентично среднему геометрическому отношений (1,30)

значения SPECRatios не имеют единиц измерения, их арифметическое сравнение лишено смысла.) Формула для среднего геометрического ($\text{Геом}_{\text{ср}}$) имеет вид:

$$\text{Геом}_{\text{ср}} = \sqrt[n]{\prod_{i=1}^{n} sample_i}.$$

Применительно к SPEC величина $sample_i$ соответствует SPECRatio для программы *i*. Использование среднего геометрического гарантирует два важных свойства:

1) среднее геометрическое отношений равно отношению средних геометрических;
2) отношение средних геометрических равно среднему геометрическому отношений производительности, благодаря чему выбор эталонного компьютера несуществен.

Таким образом, мы имеем основания для использования среднего геометрического, особенно если при сравнении применяются отношения производительности.

Пример Покажем, что отношение средних геометрических равно среднему геометрическому отношений производительности и что выбор эталонного компьютера для SPECRatio не имеет значения.

Ответ Допустим, что для каждого из двух компьютеров A и B существуют свои SPECRatio.

$$\frac{\text{Геом}_{\text{ср}_\text{A}}}{\text{Геом}_{\text{ср}_\text{B}}} = \frac{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio}_{\text{Ai}}}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio}_{\text{B}_\text{i}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{SPECRatio}_{\text{A}_\text{i}}}{\text{SPECRatio}_{\text{B}_\text{i}}}} =$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\dfrac{\text{Время}_{\text{вып}_\text{эк}}}{\text{Время}_{\text{вып}_{\text{A}_\text{i}}}}}{\dfrac{\text{Время}_{\text{вып}_\text{эк}}}{\text{Время}_{\text{вып}_{\text{B}_\text{i}}}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Врем}_{\text{вып}_{\text{B}_\text{i}}}}{\text{Врем}_{\text{вып}_{\text{A}_\text{i}}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Производительность}_{\text{A}_i}}{\text{Производительность}_{\text{B}_i}}}$$

Таким образом, отношение средних геометрических SPECRatio для компьютеров A и B равно среднему геометрическому отношений производительности для A и B, полученных для всех тестов пакета. Рис. 1.17 подтверждает это соотношение примерами от SPEC.

1.9. Количественные принципы проектирования компьютеров

Теперь, когда мы рассмотрели, как определять, измерять и обобщать производительность, стоимость, системную надежность, энергию и мощность, можно перейти к изучению рекомендаций и принципов, полезных при проектировании и анализе компьютеров. В этом разделе приводятся важные сведения из практики проектирования, а также два уравнения для оценки альтернатив.

Используйте преимущество параллелизма

Использование преимущества параллелизма — это один из самых важных методов повышения производительности. В каждой главе книги есть пример того, как благодаря использованию параллелизма увеличивается производительность. Здесь представлены три кратких примера, которые будут подробно объяснены в последующих главах.

В нашем первом примере используется параллелизм на системном уровне. Чтобы улучшить пропускную способность на типичном серверном тесте, таком как SPECWeb или TPC-C, могут быть использованы несколько процессоров и дисков. Тогда рабочая нагрузка обработки запросов может быть распределена между процессорами и дисками, что приводит к улучшению пропускной способности. Способность наращивать объем памяти, а также количество процессоров и дисков называется *масштабируемостью* (*scalability*), и это ценное свойство серверов. Распределять данные на многих дисках для параллельного чтения и параллельной записи позволяет параллелизм уровня данных. Тест SPECWeb также полагается на параллелизм уровня запросов, чтобы использовать несколько процессоров, в то время как тест TPC-C использует параллелизм уровня потоков для более быстрой обработки запросов к базе данных.

На уровне отдельного процессора решающим для достижения высокой производительности является использование преимущества параллелизма команд. Одним из простейших способов является конвейерная обработка (pipelining). (Более подробно она объясняется в приложении C, и ей уделено основное внимание в главе 3.) Основная идея конвейерной обработки состоит в совмещении выполнения команд, чтобы сократить общее время выполнения последовательности команд. Ключевой момент, который позволяет конвейерной обработке работать, состоит в том, что не каждая команда зависит от команды, непосредственно ей предшествующей, и поэтому возможно полное или частичное параллельное выполнение команд. Конвейерная обработка является самым известным примером параллелизма уровня команд.

Параллелизм можно также применять на уровне отдельных устройств. Например, частично-ассоциативные кэши используют несколько банков памяти, которые, как правило, просматриваются параллельно при поиске нужного элемента. В современных АЛУ (арифметико-логических устройствах) применяется перенос с предварительным просмотром, который использует параллелизм для ускорения суммирования за счет перехода от линейной к логарифмической зависимости от числа битов в операнде. Это другие примеры параллелизма уровня данных.

Принцип локальности

При исследовании свойств программ получены важные фундаментальные результаты. Наиболее значимый из них и систематически используемый — *принцип локальности*. Программам присуще повторно использовать данные и команды, только что бывшие в употреблении. Широко используемое эмпирическое правило заключается в том, что 90 % времени выполнения программы приходится только на 10 % кода. Следствием локальности является возможность предсказывать

с приемлемой точностью, какие команды и данные программа будет использовать в ближайшем будущем, основываясь на обращениях программы в недавнем прошлом. Принцип локальности применим также при обращениях к данным, хотя и не в такой степени, как при обращениях к коду.

Выявлены два различных типа локальности. *Временная локальность* (*temporal locality*) устанавливает, что к объектам недавно состоявшегося доступа, вероятно, состоится доступ и в ближайшем будущем. *Пространственная локальность* (*spatial locality*) говорит о том, что ссылки на смежно расположенные объекты будут большей частью смежными и во времени. Применение этих принципов рассматривается в главе 2.

Предпочтение общего случая

Вероятно, наиболее важный и распространенный принцип проектирования компьютеров — это предпочтение общего случая частному при проектном выборе. Этот принцип используется, когда определяют, как расходовать ресурсы, так как влияние определенного улучшения выше при более частом его проявлении.

Предпочтение общего случая действенно как для мощности, так и для распределения ресурсов и производительности. Устройство выборки и дешифрации команд процессора может использоваться чаще, чем устройство умножения, поэтому оптимизируется в первую очередь. То же относится и к системной надежности. Если сервер базы данных имеет 50 дисков на каждый процессор, системная надежность внешней памяти будет доминировать в общей системной надежности.

Вдобавок частый случай в большинстве случаев проще и может быть сделан более быстрым, чем редкий. Например, при сложении двух чисел в процессоре можно ожидать, что переполнение будет возникать редко, а потому можно улучшать производительность путем оптимизации более общего случая отсутствия переполнения. Такое решение может замедлить работу при переполнении, но, если оно возникает редко, общая производительность возрастет за счет оптимизации нормального случая.

В этой книге можно увидеть много проявлений данного принципа. Применяя этот простотой принцип, необходимо решить, какой случай является частым и насколько можно улучшить производительность, если сделать этот случай более быстрым. Для количественной оценки этого принципа может быть использован фундаментальный закон, названный *законом Амдаля* (*Amdahl's Law*).

Закон Амдаля

Увеличение производительности, которое может быть получено благодаря улучшению какой-либо части компьютера, можно рассчитать, используя закон Амдаля. Закон Амдаля устанавливает, что улучшение производительности, получаемое при некотором более быстром режиме выполнения, ограничивается отрезком времени, в течение которого этот более быстрый режим может быть использован.

Закон Амдаля определяет *ускорение* (*speedup*), которое может быть достигнуто при использовании конкретного свойства. Что такое ускорение? Предположим,

можно добавить в компьютер некоторое улучшение, которое, когда используется, увеличивает производительность. Ускорение определяется отношением

$$\text{Ускорение} = \frac{\begin{array}{c}\text{Производительность для всей задачи при}\\ \text{использовании улучшения по возможности}\end{array}}{\begin{array}{c}\text{Производительность для всей задачи}\\ \text{без использования улучшения}\end{array}}$$

или

$$\text{Ускорение} = \frac{\begin{array}{c}\text{Время выполнения для всей задачи}\\ \text{без использования улучшения}\end{array}}{\begin{array}{c}\text{Время выполнения для всей задачи при}\\ \text{использовании улучшения по возможности}\end{array}}.$$

Ускорение говорит, насколько быстрее будет выполняться задача на улучшенном компьютере по сравнению с исходным компьютером.

Закон Амдаля дает быстрый способ найти ускорение благодаря некоторому улучшению, которое зависит от двух факторов:

1) *доли времени счета на исходном компьютере, который может быть изменен, чтобы реализовать преимущества улучшения.* Например, если в программе, выполняемой за 60 с, улучшение можно использовать в течение 20 с, то эта доля равна 20/60. Эта величина, которую мы будем называть $\text{Доля}_{\text{улучшенная}}$ ($\text{Fraction}_{\text{enhanced}}$), всегда меньше или равна 1;
2) *выигрыша от улучшенного режима выполнения, то есть насколько быстрее могла бы выполниться задача, если этот режим использовать во всей программе.* Это величина равна отношению времени выполнения в исходном режиме ко времени выполнения в улучшенном режиме. Если, скажем, счет в улучшенном режиме занимает для некоторой части программы 2 с, а в прежнем режиме — 5 с, то выигрыш равен 5/2. Мы будем называть эту величину, которая всегда больше 1, $\text{Ускорение}_{\text{при улучшении}}$.

Время выполнения при использовании исходного компьютера с улучшенным режимом будет равно времени, затраченному на часть счета в прежнем режиме, плюс время работы с использованием улучшения.

$$\begin{array}{c}\text{Время}\\ \text{выполнения}_{\text{новое}}\end{array} = \begin{array}{c}\text{Время}\\ \text{выполнения}_{\text{старое}}\end{array} \times \left((1 - \text{Доля}_{\text{улучшенная}}) + \frac{\text{Доля}_{\text{улучшенная}}}{\text{Ускорение}_{\text{при улучшении}}} \right).$$

Общее ускорение есть отношение времен выполнения:

$$\text{Ускорение}_{\text{общее}} = \frac{\begin{array}{c}\text{Время}\\ \text{выполнения}_{\text{старое}}\end{array}}{\begin{array}{c}\text{Время}\\ \text{выполнения}_{\text{новое}}\end{array}} = \frac{1}{(1 - \text{Доля}_{\text{улучшенная}}) + \dfrac{\text{Доля}_{\text{улучшенная}}}{\text{Ускорение}_{\text{при улучшении}}}}.$$

Пример Предположим, что мы хотим улучшить процессор, используемый для веб-сервиса. При выполнении некоторого приложения веб-сервиса новый процессор работает в 10 раз быстрее, чем прежний. Предположим, что исходный процессор занят вычислениями 40 % времени, а 60 % времени находится в ожидании ввода/вывода. Каково будет общее ускорение при введении улучшения?

Ответ $\text{Доля}_{\text{улучшенная}} = 0{,}4;\ \text{Ускорение}_{\text{при улучшении}} = 10;$

$$\text{Ускорение}_{\text{общее}} = \frac{1}{0{,}6 + \frac{0{,}4}{10}} = \frac{1}{0{,}64} \approx 1{,}56.$$

Закон Амдаля выражает закон убывающей доходности: прирост в ускорении, полученный при усовершенствовании некоторой части вычислений, уменьшается по мере добавления других усовершенствований. Важным следствием закона Амдаля является то, что если определенное усовершенствование действует только для части задачи, то нельзя ускорить задачу более, чем на величину, обратную разности 1 и этой части задачи.

Общая ошибка при применении закона Амдаля состоит в смешении понятий «доля времени, потраченная *с использованием* усовершенствования» и «доля времени *после начала использования* усовершенствования». Если вместо измерения времени, в течение которого мы *могли бы* использовать данное усовершенствование при вычислении, измеряется время *после* того, как усовершенствование начало работать, результаты будут некорректными!

Закон Амдаля может послужить руководством при определении того, насколько усовершенствование увеличит производительность и как распределить ресурсы, чтобы улучшить соотношение стоимость — производительность. Очевидно, что цель заключается в расходовании ресурсов там, где расходуется время. В частности, закон Амдаля применим для сравнения общей системной производительности в случае двух альтернатив, но, как показывает последующий пример, его также можно использовать для сравнения двух альтернатив при проектировании процессора.

Пример Извлечение квадратного корня является обычным преобразованием в графических процессорах. Его реализации в варианте с плавающей запятой (FP — Floating-Point) значительно отличаются производительностью, особенно среди процессоров, спроектированных для графики. Предположим, что на FP-операцию вычисления квадратного корня (FPSQR — Floating Point SQuare Root, вычисление квадратного корня с плавающей запятой) приходится 20 % времени вычислений в важном тесте для графики. Первое предложение состоит в том, чтобы усовершенствовать оборудование для операции FPSQR и ускорить эту операцию в 10 раз. По другому варианту надо просто попытаться ускорить все команды с плавающей запятой в графическом процессоре в 1,6 раза, на них приходится половина общего времени выполнения для данного приложения. Разработчики полагают, что они смогут ускорить все FP-команды в 1,6 раза, приложив те же усилия, что потребуются для быстрого извлечения квадратного корня. Сравним эти два проектных варианта.

Ответ Мы можем сравнить эти две альтернативы путем сравнения ускорений:

$$\text{Ускорение}_{\text{FPSQR}} = \frac{1}{(1-0{,}2)+\frac{0{,}2}{10}} = \frac{1}{0{,}82} = 1{,}22;$$

$$\text{Ускорение}_{\text{FP}} = \frac{1}{(1-0{,}5)+\frac{0{,}5}{1{,}6}} = \frac{1}{0{,}8125} = 1{,}23.$$

Улучшение производительности всех FP-операций немного лучше вследствие более частого использования.

Закон Амдаля применим не только к производительности. Давайте повторим пример по теме надежности на странице 70, считая, что надежность системы питания, измеренная в MTTF, в результате резервирования улучшилась с 200 000 до 830 000 000 часов, то есть в 4150 раз.

Пример Вычисление частоты отказов дисковой подсистемы дает

$$\text{Частота отказов}_{\text{системы}} = 10 \times \frac{1}{1\,000\,000} + \frac{1}{500\,000} + \frac{1}{200\,000} + \frac{1}{200\,000} + \frac{1}{1\,000\,000} =$$

$$= \frac{10+2+5+5+1}{1\,000\,000} = \frac{23}{1\,000\,000 \text{ часов}}.$$

Поэтому доля частоты отказов, которая могла бы быть улучшена, равна 5 на миллион из 23 на миллион для всей системы, или 0,22.

Ответ Улучшение надежности составило бы:

$$\text{Улучшение}_{\text{пара источников питания}} = \frac{1}{(1-0{,}22)+\frac{0{,}22}{4150}} = \frac{1}{0{,}78} = 1{,}28.$$

Несмотря на впечатляющее улучшение надежности одного модуля в 4150 раз, с системной точки зрения это изменение дает заметный, но небольшой выигрыш.

В приведенных выше примерах нам нужна была доля времени для новой и улучшенной версий; зачастую измерить эти времена напрямую трудно. В следующем разделе будет показан иной способ выполнения таких сравнений, основанный на использовании уравнения, которое разделяет время работы центрального процессора CPU на три отдельных компонента. Если известно, насколько некая альтернатива влияет на эти три компонента, можно определить ее общую производительность. Более того, часто можно построить модели, измеряющие эти компоненты, до того как аппаратура фактически спроектирована.

Уравнение производительности процессора

По существу, все компьютеры конструируются с использованием сигналов синхронизации (тактовых сигналов), подаваемых с постоянной частотой. Эти дискретные временные события называются синхроимпульсами, или *тактовыми сигналами* (*ticks, clocks, clock ticks*); периодами синхронизации, или циклами, или тактовыми

циклами, или тактами (*clock periods,cycles, clock cycles*). Разработчики компьютеров используют длительность периода сигналов синхронизации (*время такта*, то есть длительность такта, например 1 нс) или его частоту (например 1 ГГц). Время работы CPU для программы может быть выражено двумя способами:

$$\text{CPUвремя} = \text{Количество тактов CPU на программу} \times \text{Время такта}$$

или

$$\text{CPUвремя} = \frac{\text{Количество тактов CPU на программу}}{\text{Частота тактовых сигналов}}.$$

В дополнение к количеству тактов, необходимых для выполнения программы, можно также подсчитать количество выполненных команд — *длину пути команд* (*instruction path length*) или количество команд *IC* (*instruction count*). Если мы знаем количество тактов и количество команд, то можно вычислить среднее *количество тактов на команду* (*CPI — Clock cycles Per Instruction*). Ввиду того, что с этим параметром легче работать, для рассматриваемых в данной главе простых процессоров будет использоваться CPI. Иногда разработчики также используют обратную величину — *количество команд на тактовый цикл* (*IPC — Instructions Per Clock*).

Количество тактов на команду CPI вычисляется как

$$\text{CPI} = \frac{\text{Количество тактов CPU для программы}}{\text{Количество команд}}.$$

Эта характеристика качества процессора обеспечивает понимание различных типов систем команд и реализаций и будет широко использоваться в следующих четырех главах.

При подстановке количества команд в приведенную выше формулу количество тактов для программы может быть определено как IC × CPI. Это позволяет использовать CPI в формуле времени выполнения (CPU время):

$$\text{CPUвремя} = \text{Количество команд} \times \text{Количество тактов на команду} \times \text{Время такта}.$$

Добавление в первую формулу единиц измерения показывает, как эти части сочетаются вместе:

$$\frac{\text{Команды}}{\text{Программа}} \times \frac{\text{Тактовые циклы}}{\text{Команды}} \times \frac{\text{Секунды}}{\text{Тактовый цикл}} = \frac{\text{Секунды}}{\text{Программа}} = \text{CPUвремя}$$

Как показано в этой формула, производительность процессора зависит от трех характеристик: такта, то есть периода синхронизации (или частоты), количества тактов на команду и количества команд в программе. Более того, CPUвремя зависит от этих трех характеристик *равным образом*, например, 10%-е улучшение любой их них ведет к 10%-му улучшению времени CPU.

К сожалению, трудно изменить один параметр отдельно от других, ибо базовые технологии, связанные с изменением каждой характеристики, взаимозависимы:

- *время такта* — технология и организация аппаратуры;
- *CPI* — организация и архитектура системы команд;
- *количество команд* — архитектура системы команд и технология компилятора.

К счастью, многие потенциальные методы улучшения производительности совершенствуют один компонент производительности процессора при небольшом или предсказуемом влиянии на другие два.

Иногда при проектировании процессора полезно подсчитывать общее количество его тактов как

$$\text{Количество тактов процессора} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i,$$

где IC_i — количество исполнений команды *i* в программе, а CPI_i представляет среднее количество тактов для команды *i*. Эту форму можно использовать, чтобы выразить процессорное время CPU как

$$\text{CPUвремя} = \left(\sum_{i=1}^{n} IC_i \times CPI_i \right) \times \text{Время такта},$$

а обобщенное количество тактов на команду CPI — как

$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{Количество команд}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{Количество команд}} \times \text{CPI}_i.$$

Последняя форма вычисления CPI использует каждое отдельное CPI_i и долю появления этой команды в программе (то есть отношение IC_i / Количество команд). Значения CPI_i следует измерять, а не просто подсчитывать из табличных данных справочного руководства, так как этот параметр должен учитывать конвейерные эффекты, промахи в кэше и любые другие проявления неэффективности системы памяти.

Рассмотрим наш пример оценки производительности на стр. 84, модифицированный, чтобы использовать измерения частоты команд и значения CPI команд, которые на практике получаются моделированием или аппаратными замерами.

Пример Предположим, выполнены следующие измерения:

Частотность FP-операций = 25 %
Среднее значение CPI для FP-операций= 4,0
Среднее значение CPI для других команд = 1,33
Частотность FPSQR = 2 %
CPI для FPSQR= 20

Пусть имеются две проектные альтернативы: уменьшить до 2 значение CPI для операции FPSQR или уменьшить среднее CPI всех FP-операций до 2,5. Сравним эти две проектные альтернативы, используя уравнение производительности процессора.

Ответ Сначала рассмотрим только изменения CPI, при этом тактовая частота и количество команд остаются исходными. Начнем с нахождения исходного значения CPI ($\text{CPI}_{\text{исх}}$) без улучшения:

$$\text{CPI}_{\text{исх}} = \sum_{i=1}^{n} \text{CPI}_i \times \left(\frac{\text{IC}_i}{\text{Количество команд}} \right) = (4 \times 25\ \%) + (1{,}33 \times 75\ \%) = 2{,}0.$$

Можем вычислить CPI улучшенной команды FPSQR ($CPI_{newFPSQR}$) вычитанием циклов, сэкономленных в исходном CPI:

$$CPI_{new\ FPSQR} = CPI_{исх} - 2\ \% \times (CPI_{oldFPSQR} - CPI_{только\ новой\ FPSQR}) =$$
$$= 2{,}0 - 2\ \% \times (20 - 2) = 1{,}64.$$

Можем вычислить CPI для варианта с улучшением всех FP-команд с плавающей запятой (CPI_{newFP}) тем же способом или суммируя значения CPI для FP- и не FP-команд. Суммирование значения CPI для FP- и не FP-команд дает:

$$CPI_{newFP} = (75\ \% \times 1{,}33) + (25\ \% \times 2{,}5) = 1{,}625.$$

Так как значение CPI при общем улучшении команд с плавающей запятой (FP) несколько меньше, то производительность будет лишь немного лучше. Отметим, что ускорение за счет общего улучшения FP-команд ($Ускорение_{newFP}$) равно:

$$Ускорение_{newFP} = \frac{CPU\ врем_{исх}}{CPU\ врем_{newFP}} =$$
$$= \frac{IC \times Тактовый\ цикл \times CPI_{исх}}{IC \times Тактовый\ цикл \times CPI_{newFP}} = \frac{CPI_{исх}}{CPI_{newFP}} = \frac{2{,}0}{1{,}625} = 1{,}23.$$

Замечательно, что мы получили такое же ускорение, используя закон Амдаля на странице 83.

Зачастую можно измерить составляющие части уравнения производительности процессора. В предыдущем примере это является ключевым преимуществом использования данного уравнения вместо закона Амдаля. В частности, может быть трудно измерить такие значения, как доля времени выполнения некоторого набора команд. На практике, вероятно, это подсчитывалось бы путем суммирования произведений количества команд на CPI для каждой из команд данного набора. Так как здесь подсчет обычно начинается с измерений количества команд определенного типа и CPI, уравнение производительности процессора чрезвычайно полезно.

Чтобы использовать уравнение производительности процессора как инструмент проектирования, необходимо уметь измерять различные факторы. Для существующего процессора легко получить время выполнения с помощью измерения, кроме того, мы по умолчанию знаем значение тактовой частоты. Проблема состоит в определении количества команд или CPI. Большинство новых процессоров содержат счетчики выполненных команд и тактов синхронизации. Осуществляя периодический мониторинг этих счетчиков, можно также привязать время выполнения и количество команд к сегментам данного кода, что может быть полезным программистам, пытающимся понять и настроить производительность приложения. Часто разработчик или программист хочет понять производительность на более детальном уровне, чем тот, которого можно достичь с использованием аппаратных счетчиков. К примеру, они захотят понять, почему CPI имеет именно такое значение. В таких случаях используются методы моделирования, схожие с теми, что применялись при проектировании процессоров.

Методы, которые могут помочь с энергоэффективностью, такие как динамическое масштабирование напряжения — частоты (DVFS) и разгон (см. раздел 1.5), затрудняют использование этого уравнения, ибо тактовая частота при замерах на определенной программе может меняться. Простым подходом является отключение этих свойств для воспроизводимости результатов. К счастью, производительность и энергоэффективность зачастую взаимосвязаны (сокращение времени выполнения программы обычно сберегает энергию), и поэтому можно, наверное, спокойно рассматривать производительность, не заботясь о влиянии DVFS или разгона на результаты.

1.10. Соединяем все вместе: производительность, цена и мощность

В разделах «Соединяем все вместе», которые вводятся ближе к концу каждой главы, представлены реальные примеры на основе положений главы. В этом разделе рассматриваются измерения производительности и соотношения мощность — производительность в небольших серверах с помощью теста SPECPower.

На рис. 1.18 представлены три оцениваемых многопроцессорных сервера, а также их стоимости. Ради корректности сравнения все они являются серверами Dell PowerEdge. Первый — это PowerEdge R710 на базе микропроцессора Intel Xeon X5670 с тактовой частотой 2,93 ГГц. В отличие от микропроцессора Intel Core i7, рассматриваемого в главах 2—5, имеющего четыре ядра и L3 кэш объемом 8 Мбайт, этот микропроцессор фирмы Intel имеет шесть ядер и L3 кэш объемом 12 Мбайт, при этом сами ядра идентичны. Мы выбрали систему с двумя сокетами (socket) и памятью DDR3 DRAM, с защитой с помощью кода исправления ошибок (ECC[1]-protected) и объемом 12 Гбайт и частотой 1333 МГц. Следующий сервер — PowerEdge R815 на базе микропроцессора AMD Opteron 6174. Микропроцессор имеет шесть ядер, L3 кэш объемом 6 Мбайт и работает на частоте 2,2 ГГц, но AMD размещает два таких микропроцессора в один сокет. Следовательно, сокет имеет 12 ядер и два L3 кэша объемом по 6 Мбайт. Наш второй сервер имеет два сокета с 24 ядрами и память DDR3 DRAM с защитой с помощью кода исправления ошибок (ECC-protected) объемом 16 Гбайт и частотой 1333 МГц, а наш третий сервер (также PowerEdge R815) имеет четыре сокета с 48 ядрами и память DRAM объемом 32 Гбайт. Всеми управляют JVM (Java Virtual Machine, виртуальная машина Java) IBM J9 и операционная система Microsoft Windows 2008 Server Enterprise x64 Edition.

Отметим, что благодаря тестированию (см. раздел 1.11) это необычно сконфигурированные серверы. Системы на рис. 1.18 имеют небольшую память по отношению к объему вычислений и крошечный твердотельный диск объемом 50 Гбайт. Добавление ядер недорого, если при этом вам не нужно соразмерно увеличивать оперативную и внешнюю память!

Вместо статически связанных программ на языке C теста SPEC CPU тест SPECpower использует более современный программный стек, написанный на

[1] ECC — Error Correction Code, код исправления ошибок.

Компоненты	Система 1		Система 2		Система 3	
		Стоимость (% от стоимости), долл.		Стоимость (% от стоимости), долл.		Стоимость (% от стоимости), долл.
Серверная платформа	PowerEdge R710	653 (7 %)	PowerEdge R815	1437 (15 %)	PowerEdge R815	1437 (11 %)
Мощность	570 Вт		1100 Вт		1100 Вт	
Процессор	Xeon X5670	3738 (40 %)	Opteron 6174	2679 (29 %)	Opteron 6174	5358 (42 %)
Частота синхронизации	2,93 ГГц		2,20 ГГц		2,20 ГГц	
Общее количество ядер	12		24		48	
Количество сокетов	2		2		4	
Количество ядер/сокет	6		12		12	
Память DRAM	12 Гбайт	484 (5 %)	16 Гбайт	693 (7 %)	32 Гбайт	1386 (11 %)
Интерфейс Ethernet	2 канала 1-Gbit	199 (2 %)	2 канала 1-Gbit	199 (2 %)	2 канала 1-Gbit	199 (2%)
Диск	50 Гбайт SSD	1279 (14 %)	50 Гбайт SSD	1279 (14 %)	50 Гбайт SSD	1279 (10 %)
Операционная система Windows		2999 (32 %)		2999 (33 %)		2999 (24 %)
Всего		9352 (100 %)		9286 (100 %)		12,658 (100 %)
Максимальное количество ssj_ops	910 978		926 676		1 840 450	
Максимальное количество ssj_ops/долл.	97		100		145	

Рис. 1.18. Три сравниваемых сервера Dell PowerEdge и их цены в августе 2010 г. Стоимость процессоров рассчитана путем вычитания стоимости второго процессора. Аналогично общая стоимость памяти рассчитана исходя из стоимости дополнительной памяти. Следовательно, стоимость серверной платформы[1] определена вычитанием сметной стоимости используемых по умолчанию процессора и памяти. В главе 5 описывается, как эти многосокетные системы соединяются вместе

[1] Серверная платформа (base server) — это корпус сервера вместе с материнской платой без съемных модулей — *Прим. ред.*

языке Java. Он основан на тесте SPECjbb и представляет собой серверную сторону бизнес-приложений с производительностью, измеряемой количеством транзакций в секунду и называемой *ssj_ops — количество Java операций в секунду для серверной стороны*. Он тестирует не только процессор сервера, как это делает SPEC CPU, но также кэши, систему памяти и даже систему межсоединений мультипроцессора. Кроме того, он тестирует виртуальную машину Java (JVM), в том числе динамический компилятор JIT и сборщик мусора, а также компоненты находящейся ниже операционной системы.

Как показано в последних двух строках на рис. 1.18, наилучшие производительность и соотношение стоимость — производительность имеет сервер PowerEdge R815 с четырьмя сокетами и 48 ядрами. Сервер PowerEdge R815 достигает 1,8 млн ssj_ops и наивысшего ssj_ops на доллар, равного 145. Удивительно, что этот компьютер с наибольшим количеством ядер является самым эффективным по стоимости. На втором месте двухсокетный сервер R815 с 24 ядрами, а сервер R710 с 12 ядрами на последнем месте.

Хотя большинство тестов (и большинство архитекторов компьютеров) ориентированы только на производительность при пиковой нагрузке, компьютеры редко работают при пиковой нагрузке. Действительно, на рис. 6.2 в главе 6 представлены результаты измерения нагрузки десятков тысяч серверов в течение шести месяцев в Google. Менее 1 % из них работали при средней нагрузке 100 %. Большинство имеют нагрузку от 10 до 50 %. Таким образом, тест SPECpower измеряет мощность при изменении рабочей нагрузки с 10 %-ми интервалами от пиковой нагрузки вплоть до нагрузки, равной 0 %, которая называется режимом активного ожидания.

На рис. 1.19 представлены диаграммы ssj_ops (ssj_оп/с) на ватт и средней мощности при изменении целевой нагрузки от 100 до 0 %. Сервер Intel R710 всегда имеет самую низкую мощность и наилучший показатель ssj_ops/Вт при всех уровнях рабочей нагрузки. Одной из причин этого является гораздо большая потребляемая мощность у сервера R815 — 1100 Вт против 570 Вт у сервера R710. Как показано в главе 6, эффективность потребляемой мощности очень важна для эффективности общей потребляемой мощности компьютера. Так как Вт = Дж/с, то эта величина пропорциональна ssj операциям на джоуль:

$$\frac{\text{ssj\_ операции/с}}{\text{Вт}} = \frac{\text{ssj\_ операции/с}}{\text{Дж/с}} = \frac{\text{ssj\_ операции}}{\text{Дж/с}}.$$

Для вычисления одного числа, используемого при сравнении эффективности систем, тест SPECpower использует

$$\text{Среднее значение ssj\_ ops/Вт} = \frac{\sum \text{ssj\_ ops}}{\sum \text{мощности}}.$$

Среднее значение ssj_ops/Вт у трех серверов равно: 3034 для Intel R710, 2357 для двухсокетного AMD R815 и 2696 для четырехсокетного AMD R815. Следовательно, Intel R710 имеет наилучшее соотношение мощность — производительность. Разделив на цену серверов, получим значения ssj_ops/Вт/1000 долл., равные

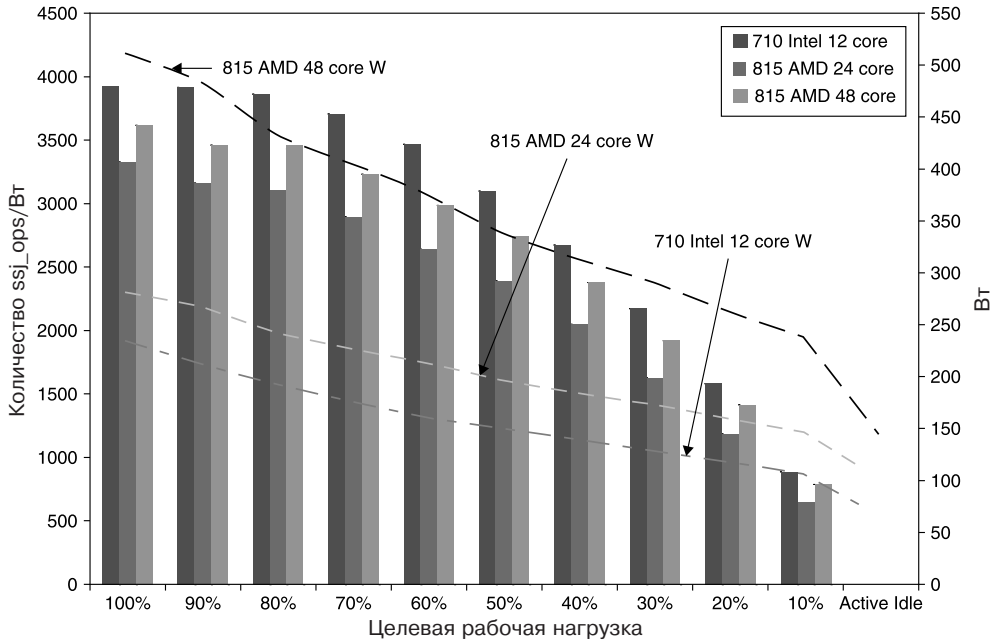

Рис. 1.19. Соотношение мощность — производительность для трех серверов, представленных на рис. 1.18. Значения ssj_ops/Вт указываются на левой оси с относящимися к ним столбцами, а ватты приводятся на правой оси с тремя относящимися к ним линиями. Горизонтальная ось показывает целевую рабочую нагрузку при ее изменении от 100 % до режима активного ожидания. Сервер R715 на базе Intel имеет наилучшее отношение ssj_ops/Вт на каждом уровне рабочей нагрузки и также потребляет самую низкую мощность на каждом уровне

324 для Intel R710, 254 для двухсокетного AMD R815 и 213 для четырехсокетного AMD R815. Таким образом, учет мощности полностью изменяет результаты соревнования цена — производительность, и первенство в соотношении цена / мощность / производительность переходит к Intel R710, а 48-ядерный R815 переходит на последнее место.

1.11. Заблуждения и просчеты

Цель данного раздела, который можно увидеть в каждой главе, состоит в том, чтобы объяснить некоторые распространенные ошибочные мнения и просчеты, которых вам следует избегать. Мы называем такие ошибочные мнения *заблуждениями*. При обсуждении заблуждения мы стараемся приводить контрпример. Обсуждаем также *просчеты*, то есть легко совершаемые ошибки. Часто просчетами являются обобщения принципов, которые истинны в ограниченном контексте. Цель таких

разделов состоит в том, чтобы помочь вам избежать этих ошибок в компьютерах, которые вы проектируете.

Заблуждение: *мультипроцессоры — это серебряная пуля.*

Переход к нескольким процессорам на кристалле приблизительно в 2005 г. не был неким прорывом, который чрезвычайно упростил параллельное программирование или облегчил создание многоядерных компьютеров. Это изменение было вызвано тем, что не было другой возможности из-за ограничений ILP и мощности. Несколько процессоров на кристалле не гарантируют меньшего потребления мощности, но, вне всякого сомнения, можно спроектировать многоядерный кристалл, который потребляет большую мощность. В принципе можно продолжать увеличивать производительность посредством замещения неэффективного ядра с высокой тактовой частотой несколькими эффективными ядрами с более низкой тактовой частотой. Развитие технологии уменьшения размеров транзисторов позволило немного уменьшить емкость и напряжение питания, так что можно получить скромное увеличение количества ядер в каждом поколении. Например, за последние несколько лет фирма Intel добавляла по два ядра в каждом поколении.

Как будет показано в главах 4 и 5, производительность теперь является бременем программиста. Эпоха ленивых программистов, не желающих ударить палец о палец, чтобы сделать свои программы более быстрыми, и оставляющих это разработчикам аппаратных средств, официально закончилась. Если программисты хотят, чтобы их программы становились быстрее в каждом поколении, они должны увеличивать параллелизм программ.

Популярная версия закона Мура — увеличение производительности в каждом поколении технологии — теперь относится и к программистам.

Просчет: *стать жертвой душераздирающего закона Амдаля.*

Фактически каждый практикующий разработчик архитектуры компьютера знает закон Амдаля. Тем не менее почти все иногда тратят огромные усилия, оптимизируя некоторое свойство, прежде чем оценить его полезность. Только когда общее ускорение не оправдывает надежд, вспоминают, что прежде чем тратить так много усилий на совершенствование данного свойства, сначала следовало бы произвести его оценку!

Просчет: *единственная точка отказа.*

Расчеты улучшения надежности с применением закона Амдаля в разделе 1.9 (на стр. 85) показывают, что системная надежность не выше надежности самого слабого звена в цепи. Неважно, насколько надежными будут источники питания, как в нашем примере, единственный вентилятор будет ограничивать надежность дисковой подсистемы. Этот вывод из закона Амдаля привел к эмпирическому правилу для отказо-устойчивых систем — необходимо убедиться в том, что каждый компонент избыточен настолько, чтобы отказ одного компонента не мог нарушить работу всей системы. В главе 6 показано, как на уровне программного обеспечения избегают единственных точек отказа в компьютерах WSC.

Заблуждение: *усовершенствование аппаратуры для увеличения производительности улучшает энергоэффективность или, в худшем случае, оставляет ее прежней.*

Esmaeilzadeh и др. [2011] измерили на тесте SPEC2006 только одно ядро микропроцессора Intel Core i7 с частотой 2,67 ГГц, используя режим Turbo (раздел 1.5). Производительность выросла в 1,07 раза при увеличении тактовой частоты до 2,94 ГГц (или в 1,10 раза), но микропроцессор i7 потреблял в 1,37 раза больше джоулей и в 1,47 раза больше ватт-часов!

Заблуждение: *достоверность тестов бессрочна.*

Некоторые факторы влияют на полезность теста как предсказателя реальной производительности и некоторые из них изменяются во времени. Важным фактором, влияющим на полезность теста, является его устойчивость к «доработкам» (benchmark engineering) или «настройкам» (benchmarketing). Как только тест становится стандартным и популярным, предпринимаются очень настойчивые попытки улучшить его производительность путем целенаправленных оптимизаций или агрессивных интерпретаций правил выполнения теста. Небольшие ядра или программы, выполняющие большую часть времени небольшую часть своего кода, особенно уязвимы.

Например, несмотря на благие намерения, в первоначальный пакет программ для тестирования SPEC89 входило небольшое ядро matrix300, которое состояло из восьми различных умножений матриц размером 300 × 300. В этом ядре 99 % времени тратилось на выполнение единственной строки (см. SPEC [1989]). Когда компилятор IBM оптимизировал этот внутренний цикл (используя концепцию разбиения на *блоки*, обсуждаемую в главах 2 и 4), производительность выросла в девять раз по сравнению с предыдущей версией компилятора! Этот тест проверял настройки компилятора и, разумеется, не был ни хорошим показателем общей производительности, ни типичным значением данной оптимизации.

Спустя длительное время такие изменения могут превратить даже хорошо подобранные тесты в устаревшие; Gcc — единственный оставшийся из SPEC89. На рис. 1.16 в разделе 1.8 (с. 75) приведены сведения обо всех 70 тестах из различных версий SPEC. Удивительно, что почти 70 % всех программ из SPEC2000 или более ранних версий были удалены из последующей версии.

Заблуждение: *нормативная наработка до отказа дисков (MTTF) составляет 1 200 000 часов, или почти 140 лет, то есть диски никогда не отказывают.*

Существующие маркетинговые практики производителей дисков могут вводить пользователей в заблуждение. Как такие MTTF рассчитываются? На начальной стадии производители устанавливают тысячи дисков в помещении, запускают их в течение нескольких месяцев и подсчитывают количество неисправных. Они вычисляют значение MTTF как общее количество часов, которое диски работали в совокупности, деленное на количество отказавших дисков.

Одна из проблем здесь в том, что это число существенно превышает срок службы диска, который обычно принимается равным пяти годам, или 43800 часам. Чтобы такие большие MTTF имели хоть какой-то смысл, производители

дисков доказывают, что модель соответствует пользователю, который покупает диск и затем меняет его каждые пять лет — плановый срок службы диска. Утверждается, что если многие покупатели (и их правнуки) делали бы это в течение следующего столетия, в среднем до отказа они бы заменяли диск 27 раз в течение примерно 140 лет.

Более полезным показателем была бы доля неисправных дисков. Предположим, что имеем 1000 дисков с MTTF, равным 1 000 000 часов, и что диски используются 24 часа в сутки. Если отказавшие диски заменялись новыми с теми же характеристиками надежности, то количество отказавших дисков в год (8760 часов) равно:

$$\text{Отказы дисков} = \frac{\text{Количество дисков} \times \text{Время}}{\text{MTTF}} = \frac{1000 \text{ дисков} \times 8760 \text{ часов/дисковод}}{1000000 \text{ часов/отказ}} = 9.$$

Другими словами, в год отказывало бы 0,9 % дисков, или 4,4 % за пятилетний срок службы.

Более того, ссылки на эти высокие числа предполагают ограниченные диапазоны температуры и вибрации, а при их превышении обязательства не соблюдаются. Обзор по дисководам для реальных условий [Gray and van Ingen 2005] показал, что в год отказывают от 3 до 7 % дисководов с MTTF, равным примерно 125 000—300 000 часов. Даже расширенное исследование показало, что отказы дисков составляют от 2 до 10 % в год [Pinheiro, Weber, and Barroso 2007]. Таким образом, в реальном мире MTTF в 2—10 раз хуже, чем MTTF производителей.

Заблуждение: *пиковая производительность отслеживает реальную производительность.*

Единственное универсально истинное определение пиковой производительности таково: уровень производительности компьютера, который гарантированно не будет превышен. На рис. 1.20 показана производительность четырех мультипроцессоров на четырех программах в процентах от пиковой производительности. Она изменяется от 5 до 58 %. Так как отличие от пиковой производительности настолько велико и может значительно изменяться в зависимости от теста, пиковая производительность, в общем, бесполезна для предсказания реальной производительности.

Просчет: *обнаружение неисправностей может ухудшить доступность.*

Этот, очевидно, иронический просчет вызван тем, что аппаратура компьютера имеет изрядное количество состояний, которые не всегда могут быть критическими для правильной работы. Например, не фатально, если ошибка возникает в предсказателе передач управления, поскольку при этом может пострадать только производительность.

В процессорах, которые пытаются агрессивно использовать параллелизм уровня команд, не все операции необходимы для правильного выполнения программы. Mukherjee и др. [2003] обнаружили, что менее 30 % операций в тестах SPEC2000 потенциально определяли производительность микропроцессора Itanium 2.